FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO
Mestrado em Engenharia Electrotécnica e de Computadores



Redes de Comunicação Industriais do Tipo *Field Bus*, Integração em Ambiente CIM

Dissertação de Mestrado de
Eduardo Manuel de Médicis Tovar

Dezembro de 1994

Elect.

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO
Mestrado em Engenharia Electrotécnica e de Computadores

Redes de Comunicação Industriais do Tipo *Field Bus*, Integração em Ambiente CIM

UNIVERSIDADE DO PORTO
Faculdade de Engenharia
BIBLIOTECA
N.º 20467-01
CDU 004(043)
Data 5 / 5 / 1995
JIN. 34805

Dissertação de Mestrado de
Eduardo Manuel de Médicis Tovar

Sob a Orientação do
Professor Doutor Artur Agostinho Capelo Cardoso

043M
671m
X.3

Dezembro de 1994

Aos meus Pais

Resumo

Esta dissertação analisa a interligação de uma rede do tipo *Field Bus*, vocacionada para o nível mais baixo da hierarquia CIM (*Computer Integrated Manufacturing*), numa infraestrutura global de comunicações industriais. O objectivo da interligação é permitir a integração, no ambiente CIM, das funcionalidades e actividades cobertas por esse tipo de redes. A rede de *Field Bus* considerada para o efeito é uma rede especialmente vocacionada para aplicações de tempo crítico.

O texto está dividido em três partes.

Na primeira parte, designada por 'Redes de Comunicações Industriais Sob a Perpectiva CIM', começa-se por fazer uma abordagem geral ao conceito da produção integrada por computador, definido como visando a integração e coordenação global de todas as actividades em ambiente de fabrico. Conclui-se àcerca do papel crucial desempenhado pelas redes de comunicação industriais para atingir este objectivo.

O equipamento utilizado no ambiente industrial é caracterizado por apresentar grande heterogeneidade, quer ao nível de fabricantes, quer ao nível de gerações. Salienta-se por isso, no capítulo 2, o trabalho desenvolvido no âmbito da ISO para a regulamentação de normas abertas (OSI - *Open Systems Interconnection*), bem como o esforço paralelo desenvolvido em torno da especificação da norma de comunicações para ambiente industrial MAP (*Manufacturing Automation Protocol*).

No capítulo 3, abordam-se as redes de comunicação industriais numa perspectiva hierarquica, referindo-se a necessidade da coexistência de diferentes arquitecturas, uma vez que uma só solução não preenche os requisitos de todas as áreas no ambiente de fabrico. Nesse mesmo capítulo são analizadas mais aprofundadamente as redes de comunicação industriais para os níveis inferiores da hierarquia CIM, em particular, os requisitos genéricos que este tipo de redes devem preencher. Conclui-se, no entanto, que mesmo a este nível, é necessário considerar um conjunto distinto de soluções, dado que este tipo de redes reivindica a cobertura de aplicações onde os requisitos são muito díspares.

Finalmente, e ainda na primeira parte, é abordada uma filosofia aqui designada por 'Sistemas Federadores de Subsistemas de Comunicação Heterogéneos'. Trata-se de uma abordagem recente por parte dos fabricantes no sentido da integração global. Basicamente, são arquitecturas que integram simultaneamente os diferentes tipos de redes de comunicação industriais e diferentes pacotes da produção assistida por computador. São disso exemplo as propostas BASESTAR, PYRAMID e OpenCIM.

A segunda parte denomina-se 'Das Arquitecturas Consideradas Para a Interligação'. No capítulo 5, é feita uma descrição funcional da rede de *Field Bus* considerada para o estudo da interligação, o FILBUS da GESPAC. Realça-se a sua elevada funcionalidade e flexibilidade. A interligação desta rede é considerada em termos da arquitectura MAP 3.0, e em particular com o o MMS (*Manufacturing Message Specification*), uma vez que a interligação deverá ser considerada a nível da camada de aplicação. Mais importante do que a consideração genérica da interligação com o MAP, salienta-se o MMS como entidade federadora. Isto é tanto mais importante uma vez que o MMS é suportado como camada de aplicação por outras redes de valência hierarquica intermédia, como o MiniMAP, o PROFIBUS ou mesmo o FIP.

Assim, no capítulo 6, são introduzidos os modelos e conceitos do MMS. É dado algum relevo ao esclarecimento das correspondências entre o MMS e o modelo de referência OSI, questão muitas vezes pouco clara e por isso objecto de muitas discussões em diferentes foros.

Por fim, na terceira parte, designada por 'Do Dispositivo de Interligação' é estudada, analizada e proposta uma arquitectura para um dispositivo que permita a interligação entre esta rede particular de *Field Bus* e a arquitectura MAP 3.0.

A integração desta rede de *Field Bus* no ambiente MAP 3.0 implica, dadas as características genéricas de ambas as arquitecturas, fazer a interligação das duas redes ao nível da camada de aplicação. A tarefa é tanto mais difícil quanto os serviços da camada de aplicação da rede de *Field Bus* considerada não se aproximam de todo aos serviços disponibilizados pela arquitectura MAP, em particular aos serviços MMS. Isto é válido quer no que se refere aos modelos implementados (orientado aos objectos no MMS) quer no que se refere à sua sintaxe e semântica.

O MMS é um protocolo e um conjunto de serviços extremamente completo e complexo. O FILBUS é, acima de tudo, extremamente flexível em termos de funcionalidade. A combinação destas duas ilações faz antever as inúmeras opções de modelização que poderão ser consideradas para este dispositivo de interligação. Em grande medida, estas opções irão ser analizadas à luz de três ou quatro perspectivas, que no seu conjunto formaram as fronteiras da análise feita no capítulo 7. São exemplo dessa fronteiras, as características técnicas e funcionais, e a tipologia desta rede de *Field Bus*, o tipo de aplicações para as quais a rede de *Field Bus* está vocacionada, bem como potenciais limitações impostas pela plataforma base do dispositivo de interligação.

Por fim, no capítulo 8, serão abordadas questões relacionadas com a implementação da arquitectura proposta.

Agradecimentos

Estou em dívida para com muitas pessoas que me aconselharam e me deram assistência.

Em especial, o Prof. Artur Cardoso pela orientação mas, acima de tudo, pela magnifica camaradagem e relação de trabalho. Foi meio caminho andado.

O Prof. Adriano Carvalho, pelo interesse e acompanhamento sempre manifestados. O Prof. Franquelim Ferreira. O Raúl Oliveira, pelas magnificas discussões tidas. Todos os colegas do perfil de mestrado, a Cristina Russo, apesar do 'preço', o Miguel Correia, o Mário Alves e os outros dois mosqueteiros, o Zé Carlos e o Rui Pedro. Sem eles não tinha tido graça.

Os familiares, amigos e a Tuna de Engenharia, por me aturarem.

A Conceição Neves e o João Rocha, como meus coordenadores no ISEP.

Os Profs. António Quintas e Francisco Restivo, enquanto coordenadores dos projectos ESPRIT FICIM - *Fieldbus Integration into CIM* - e CNMA - *Communications Network for Manufacturing Applications*, e directores do Centro de CIM do Porto.

O Eng. Eduardo Prata da Fatrónica, o Michel Gouget da GESPAC S.A., e diversas pessoas da EFACEC S.A..

A D. Elisabete e o Vitor Monteiro do CCP.

Os outros.

Mais do que todos, os meus Pais.

Conteúdo

PRIMEIRA PARTE

Redes de Comunicação Industriais Sob a Perpectiva CIM

Capítulo 1 - A Produção Integrada Por Computador 3

1.1. Automatização Industrial: Motivação 3

1.2. A Flexibilidade e os Sistemas de Fabrico Flexível 5

1.2.1. Célula de Fabrico Flexível 6

1.2.2. Sistema de Fabrico Flexível 6

1.3. O CIM e Tecnologias CIM 7

1.3.1. A Integração CAD/CAE/CAM 8

1.3.2. Gestão dos Recursos de Fabrico (MRP II) 9

1.4. Sistemas Industriais 9

1.5. Organização e Processamento de Informação em Ambiente de Fabrico 11

1.5.1. Funções das Operações de Fabrico 11

1.5.2. Actividades de Organização, Geração e Processamento da Informação de Fabrico 13

1.6. Referências e Bibliografia 15

Capítulo 2 - Fluxos de Informação em Ambiente CIM e Redes de Comunicação Industriais 17

2.1. Aspectos da Integração e Interligação em Ambiente Industrial 17

2.2. Redes de Comunicação Industriais 20

2.3. Modelo de Referência OSI 21

2.4. MAP e TOP no Modelo de Referência OSI 23

2.5. Referências e Bibliografia 25

Capítulo 3 - Redes de Comunicação Industriais Para os Níveis Inferiores da Hierarquia CIM - Field Bus 27

3.1. Introdução 27

3.2. As Normas e o Mercado 30

3.2.1. Redes de Fábrica 30

3.2.2. Redes de Célula 31

3.2.3. Redes de Campo 31

3.3. Arquitectura de Redes de Campo 31

3.3.1. Camada Física e hardware 32

3.3.2. Camada de Ligação de Dados 33

3.3.3. Camada de Aplicação 33

3.4. Implementações de Redes de Campo 34

3.5. Conclusões 36

3.6. Referências e Bibliografia 36

Capítulo 4 - Sistemas Federadores de Subsistemas de Comunicações Heterogéneos 39

4.1. Introdução 39

4.2. O BASESTAR 40

4.2.1. Funcionalidades e Abrangência da Proposta 40

4.2.2. Arquitectura do BASESTAR 41

4.3. O Pyramid 42

4.4. O PFS-DAE 42

4.5. O OpenCIM 44

4.6. Dispositivos Para a Interligação de Redes de Comunicação 44

4.6.1. Repetidores 45

4.6.2. Pontes (Bridges) 45

4.6.3. Routers 46

4.6.4. Gateways 47

4.7. Referências e Bibliografia 47

SEGUNDA PARTE

Das Arquitecturas Consideradas Para a Interligação

Capítulo 5 - Características e Funcionalidades da Rede de Field Bus Considerada 51

5.1. Introdução 51

5.2. Sinais do Barramento 52

5.3. O Protocolo SDLC 53

5.3.1. Tramas Não Numeradas 53

5.3.2. Tramas de Supervisão 54

5.3.3. Tramas de Informação 54

5.4. O mecanismo de Polling da Rede 55

5.5. Utilização da Rede de Field Bus 55

5.5.1. Aplicações do Pacote de Software 55

5.6. Quadros Resumo das Mensagens de Aplicação 57

5.6.1. Mensagens Gerais da Rede 57

5.6.2. Mensagens Particulares dos Módulos 58

5.6.3. Mensagens de Entrada e Saída de Dados 59

5.7. Estrutura de Algumas das Mensagens de Aplicação 60

5.8. Funcionalidade dos Módulos de Entrada e Saída Digitais 60

5.8.1. O Módulo de Entradas Digitais FILINP-1648 60

5.8.2. O Módulo de Saídas Digitais FILOUT-1648 65

5.9. Referências e Bibliografia 66

Capítulo 6 - O MMS Como Elemento Federador na Arquitectura MAP 3.0 67

6.1. Alguns Elementos do Modelo de Referência OSI 67

6.1.1. Nomeação e Endereçamento em Ambiente OSI 68

6.1.2. Conexões 69

6.2. A Estrutura da Camada de Aplicação 69

6.3. O MMS no Ambiente OSI 70

6.4. Modelo Cliente/Servidor 71

6.5. Dispositivo Virtual de Fabrico (VMD) 72

6.6. Objectos MMS 73

6.7. Serviços MMS 75

6.7.1. Serviços de Gestão do Ambiente MMS 75

6.7.2. Serviços de Suporte de VMD 77

6.7.3. Serviços de Gestão de Domínios 77

6.7.4. Serviços de Gestão de Instâncias de Programa 79

6.7.5. Serviços de Acesso a Variáveis 81

6.7.6. Serviços de Gestão de Semáforos 82

6.7.7. Serviços de Comunicação com o Operador 83

6.7.8. Serviços de Gestão de Acontecimentos 83

6.7.9. Serviços de Gestão de Jornais 86

6.7.10. Serviços de Gestão de Ficheiros 87

6.8. Normas Companheiras 87

6.9. Mapeamento do MMS no Modelo OSI 88

6.10. Referências e Bibliografia 89

TERCEIRA PARTE

Do Dispositivo de Interligação

Capítulo 7 - Análise Funcional e Proposta de Arquitectura 93

7.1. Introdução 93

7.2. Mapeamento da Rede de Field Bus em Objectos Abstractos MMS 95

7.2.1. Do Mapeamento em Termos de VMDs e Endereçamento OSI 95

7.2.2. Da Opção Por um Só VMD 96

7.2.3. Da Estrutura do VMD em Termos de Domínios 103

7.2.4. Da Estrutura do VMD em Termos de Objectos de Acesso a Variáveis 104

7.3. Manipulação dos Objectos MMS do Servidor 110

7.3.1. Da Coerência entre VMD e Dispositivos Reais 110

7.3.2. Da Utilização do MMS em Modo Assíncrono 118

7.3.3. Da Manipulação dos Parâmetros de Configuração das E/S dos Módulos 120

7.3.4. Da Manipulação de Variáveis de Entradas dos Módulos da Rede de Field Bus 129

7.3.5. Da Manipulação das Variáveis de Saída dos Módulos da Rede de Field Bus 136

7.3.6. Mapeamento Vertical de Eventos Gerados Pela Rede de Field Bus 142

7.3.7. De Outros Mapeamentos 144

7.4. Caracterização Geral da Arquitectura Proposta 149

7.4.1. Das Variáveis MMS 149

7.4.2. Atendimento de Indicações de MMS Read em 1ª Etapa 150

7.4.3. Atendimento de Indicações de MMS Write em 1ª Etapa 153

7.4.4. Processamento das Indicações Recebidas do Field Bus 155

7.4.5. Aspectos de Conformidade do Servidor de MMS 156

7.5. Referências e Bibliografia 157

Capítulo 8 - Apontamentos de Implementação 159

8.1. Introdução 159

8.1.1. Considerações Genéricas Sobre o Hardware e Software MAP 3.0 160

8.1.2. Desenvolvimento de Aplicações para a Rede de Field Bus 161

8.1.3. Utilização da Interface de Desenvolvimento do MMS-EASE 162

8.1.4. Exemplo de Aplicação 164

8.2. Da Configuração do Servidor MMS 166

8.2.1. Instalação das Cartas de Comunicação 166

8.2.2. Criação e Utilização da Base de Dados da Rede de Field Bus 166

8.2.3. Informação de Directório da Stack OSI 167

8.2.4. Ficheiro de Informação de Directório OSI do Dispositivo de Interligação 171

8.2.5. Registo da Aplicação Local Utilizando a Interface MMS-EASE 172

8.2.6. Configuração das Estruturas de Dados de Sistema do MMS-EASE 175

8.2.7. Configuração das Estruturas de Dados da Máquina Virtual do MMS-EASE 180

8.2.8. Geração dos Objectos Abstractos MMS no Servidor 182

8.2.9. Função de Inicialização da Máquina MMS 193

8.2.10. Gestão de Associações de Aplicação 194

8.2.11. Serviços de Suporte de VMD 199

8.2.12. Paragem da Máquina MMS 200

8.2.13. Inicialização da Rede de Field Bus 201

8.3. Suporte de Comunicações Verticais 201

8.3.1. Estruturas de Dados Auxiliares 201

8.3.2. Atendimento de Indicações MMS 202

8.3.3. Processamento de Indicações MMS em Modo Assíncrono 206

8.4. Especificação das Funções de Mapeamento Vertical de Comunicações 207

8.4.1. Sobre a Especificação das Funções 207

8.4.2. Função de Atendimento Multiplexado de Indicações 208

8.4.3. Atendimento de Indicações MMS Read 209

8.4.4. Atendimento de Indicações MMS de Write 218

8.4.5. Atendimento de 'Indicações' da Rede de Field Bus 223

8.5. Referências e Bibliografia 225

Capítulo 9 - Considerações Finais e Conclusões 227

Indice de Figuras 233

Índice Analítico 235

PRIMEIRA PARTE

Redes de Comunicação Industriais Sob a Perpectiva CIM

Capítulo 1

A Produção Integrada Por Computador

Capítulo 2

Fluxos de Informação em Ambiente CIM e Redes de Comunicação Industriais

---

Capítulo 3

Redes de Comunicação Industriais Para os Níveis Inferiores da Hierarquia CIM - Field Bus

---

Capítulo 4

Sistemas Federadores de Subsistemas de Comunicações Heterogéneos

---

Capítulo 1

A Produção Integrada Por Computador

Nos últimos anos, o conceito CIM (*Computer Integrated Manufacturing*), sobressai no âmbito das estratégias de produção. Basicamente, esse conceito preconisa a unificação, num só processo global, de todos os subsistemas de produção auxiliada por computador: CAD, CAE, CAM, MRP, etc. No entanto, não existe uma fórmula para a aplicação do CIM. Mais ainda, a sua aplicabilidade e validade pode ser muitas vezes questionada. Que as áreas do fabrico serão de integrar? Em que tipos de indústrias? Vale pena substituir? Estas são questões pertinentes, e que têm levado ao aparecimento de uma área de investigação cujo objectivo é o de avaliar os benefícios da introdução do CIM num potencial cliente. Este capítulo aborda genericamente a automatização industrial, alguns dos tradicionais conceitos de sistemas de ajuda por computador, bem como alguns aspectos relacionados com a organização e a tipologia industrial.

1.1. Automatização Industrial: Motivação

Nos últimos anos, os esquemas de produção têm mudado drasticamente. Isto deve-se, por um lado à crescente competição entre as empresas, e por outro, ao enorme desenvolvimento das tecnologias que envolvem microprocessadores, *robots*, redes de comunicação, inteligência artificial, máquinas de controlo numérico, etc.

Em abstracto, todas as estratégias de produção visam um conjunto de objectivos. Na realidade, e se for entendido nas suas diversas vertentes (ver *Fig. 1.1*, adaptada de [BOU - 87]), esse conjunto de objectivos corresponde a um só mais lato: o aumento da competitividade.

É comum o erro de confundir os objectivos com os meios. Sendo o objectivo o de aumento da competitividade (em termos de custos, qualidade, disponibilidade e inovação), os meios são, eventualmente automatizar, informatizar, robotizar, equipar, racionalizar, organizar, etc. Vem isto a propósito do facto de muitas vezes, o comportamento natural de um especialista (p. ex. em automação industrial) ser o de considerar a sua técnica com sendo objectivo principal, quando na realidade ela é um meio ao serviço do objectivo.

Dê-se para já de barato, que por automatização industrial se entende o conjunto de tecnologias relacionadas com a aplicação de equipamentos (mecânicos, electrónicos, baseados em computador, etc) em sistemas industriais de produção, e que com ela é conseguido um aumento de competitividade. Nesse sentido, e em relação à *Fig. 1.1*,, poder-se-ão tecer algumas considerações.

Por exemplo, a redução de custos de pessoal pode ser conseguida com a automatização das máquinas (tem o seu custo...), com a automatização do controlo e planeamento de produção, com a automatização da engenharia de projecto e de concepção, etc. De notar que na maioria dos casos a qualificação humana também terá de ser maior (isto também tem os seus custos...).

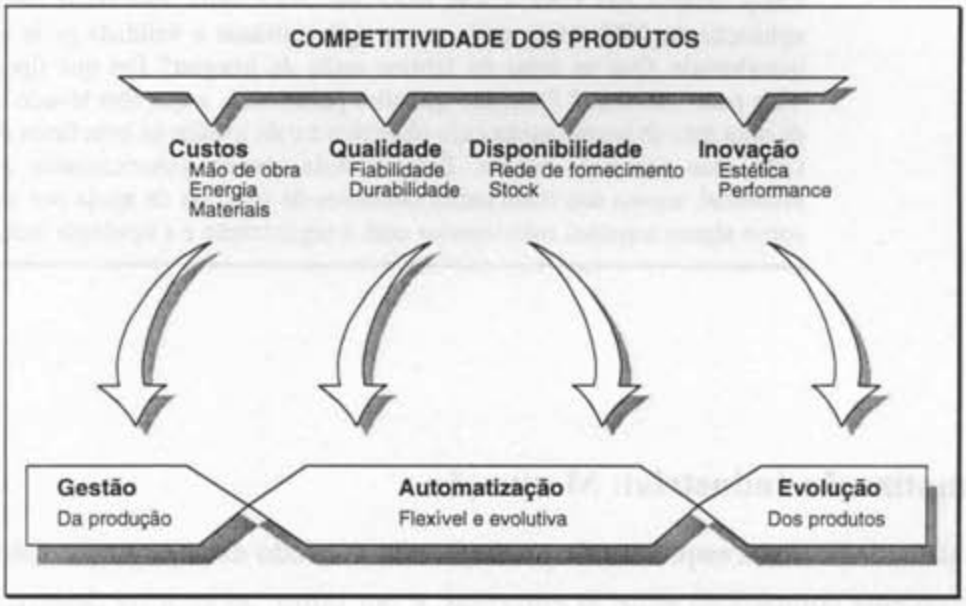


***Figura 1.1** - O Ojectivo Competitividade*

Um planeamento e controlo da produção eficiente deverá possibilitar a redução de custos de *stock*. Este é geralmente um aspecto importante, uma vez que os *stocks* intermédios, finais ou de matéria prima, constituem investimentos imobilizados. A propósito desta questão, é de referir o JIT (*Just in Time*), que, apesar do crescente número de opiniões em desfavor, tem sido desde à uns anos uma filosofia muito badalada. O JIT visa essencialmente um objectivo que é o de *stock* zero e um outro não menos importante que á o de atraso zero (disponibilidade).

A maior disponibilidade dos produtos consegue-se com a redução do número de avarias, o que aconselha a utilização de técnicas de manutenção preventiva e um melhor aproveitamento das máquinas recorrendo a planeamento e controlo de produção para cada máquina, bem como a sistemas distribuidos de aquisição de dados.

No que se refere à qualidade dos produtos, não é difícil considerar os benefícios decorrentes da utilização de, por exemplo, máquinas de controlo numérico mais precisas, com melhores características de repetibilidade, com uma garantia de qualidade constante e independente de turnos particulares.

Por fim, uma nota àcerca da evolutibilidade. Um sistema industrial é evolutivo se tiver a capacidade de se adaptar, no menor tempo necessário, ao projecto e planeamento de fabrico de novos produtos. Para esse efeito, a existência de máquinas programáveis aptas a desempenhar diferentes operações e sistemas de engenharia e concepção automatizados, são obviamente dois requisitos importantes.

Convém fazer a distinção entre o conceito de evolutibilidade e o conceito de flexibilidade. Um sistema de fabrico diz-se flexível, se, por exemplo, apresenta uma resposta rápida às solicitações do mercado, se se adapta a situações de avaria ou se é capaz de operar sobre diferentes variantes de produtos, quer simultaneamente quer por campanhas.

1.2. A Flexibilidade e os Sistemas de Fabrico Flexível

As questões técnicas relacionadas com a flexibilidade de um sistema produtivo, constituem um conjunto de requisitos que vão influenciar, sobremaneira, os sistemas de informática industrial suporte de um sistema de produção, em particular no que se refere à transferência e integração de informação.

A flexibilidade pode ser considerada sob diferentes âmbitos. Diversa bibliografia, p. ex. [GRO - 86], refere três tipos de flexibilidade num sistema de fabrico: de volume, de encaminhamento e de produto.

Por flexibilidade de volume, deve entender-se a capacidade de lidar com variações quer no volume de produção global quer no volume de componentes ou de um determinado produto. A flexibilidade de encaminhamento, diz respeito à capacidade que um sistema tem de encaminhar dinamicamente os componentes através do sistema, por forma a fazer face a eventuais avarias de máquinas, diferentes ferramentas necessárias, etc. Por fim, de referir a flexibilidade de produto. Esta diz respeito à possibilidade de lidar com uma variedade de produtos, utilizando os mesmos recursos de fabrico. Por exemplo, a produção de diferentes modelos de um mesmo produto.

1.2.1. Célula de Fabrico Flexível

Uma célula fabrico flexível, pode ser constituída, p. ex., pela associação de uma ou várias máquinas de controlo numérico com um *robot* manipulador de peças, para carga, descarga, e transferência. A *Fig. 1.2* ilustra este exemplo.

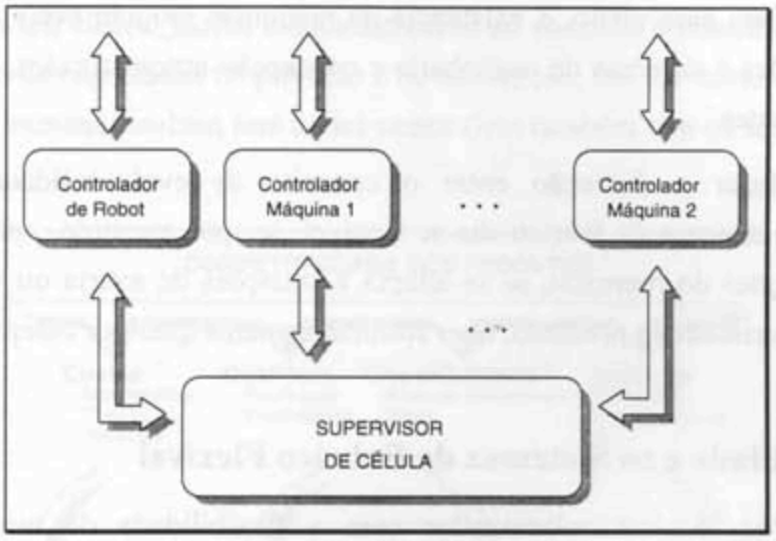


Figura 1.2. *- Célula de Fabrico Flexível*

Imagine-se uma célula de maquinação flexível, cujo objectivo é o da maquinação de peças metálicas. O centro de maquinação pode incluir as seguintes funcionalidades: capacidade para realizar múltiplas operações de maquinação, mudança automática de ferramenta, incluir um sistema de sensorização que vai monitorizando algumas variáveis relacionadas com a manutenção preventiva das máquinas, ter um sistema de transporte interno ou mesas de transferência, etc. Todo este equipamento será controlado pelo supervisor de célula.

1.2.2. Sistema de Fabrico Flexível

De uma forma simples, um sistema de fabrico flexível (SFF) pode ser considerado como sendo um conjunto de células flexíveis, interligadas por um sistema automático de transporte e armazenamento. Um SFF deverá, potencialmente, ser capaz de corresponder aos diferentes tipos de flexibilidade abordados anteriormente.

A *Fig. 1.3*, ilustra o fluxo de materiais num potencial SFF. O sistema da referida figura integra os diferentes equipamentos de controlo das máquinas com as operações físicas e com o sistema de planeamento e controlo de produção. O SFF está apto a produzir produtos através de diferentes sequências de operações mediante entrada de ordens de fabrico aleatórias. O escalonamento das diferentes operações para as diferentes máquinas, é definido por um sistema

central. As ferramentas e programas das máquinas ferramenta são carregados automaticamente, em função da operação a efectuar.

Num sistema deste género, a interligação e coordenação de todos os equipamentos quer de produção quer de processamento de informação, tem obviamente de recorrer sistemas computarizados e a redes de comunicações.

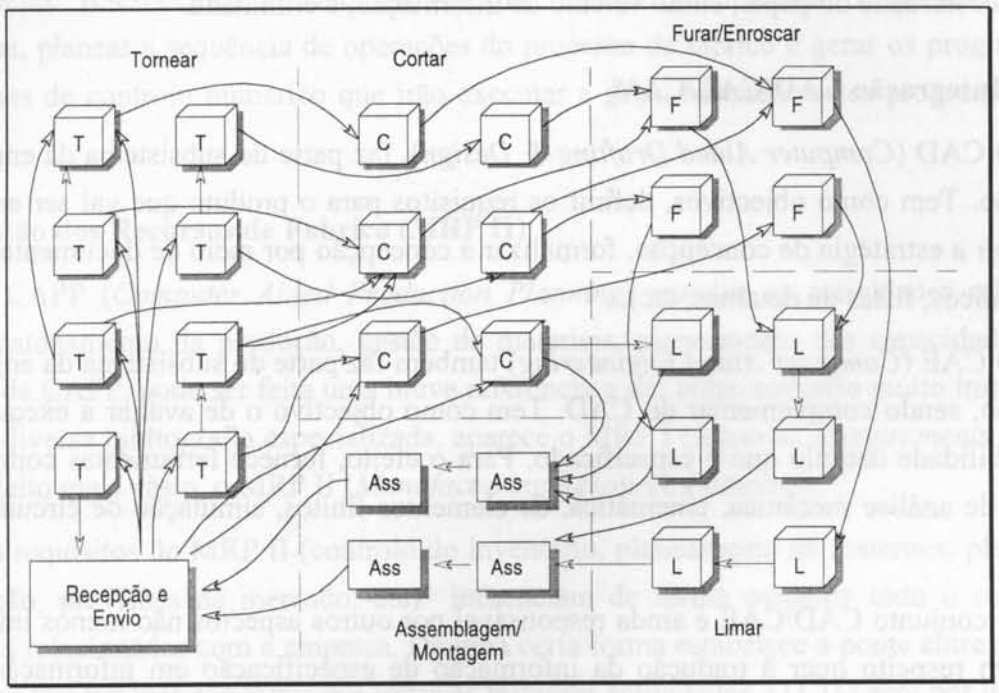


***Figura 1.3.** - Sistema de Fabrico Flexível*

As redes de comunicação não só são responsáveis pela transferência de informação, por exemplo programas, entre as estações de processamento, como também suporta a coordenação, monitorização controlo e gestão de todo o sistema (não confundir suportar com ser responsável por...).

1.3. O CIM e Tecnologias CIM

A expressão 'produção automatizada' (do inglês *automated manufacturing*) refere-se a uma unidade de produção com um elevado grau de automatização. Em diversa literatura [GRO - 86], [KOR - 85], [SIE - 91], [RAN - 83] são utilizadas outras expressões que descrevem conceitos similares: o já falado FMS (*Flexible Manufacturing Systems*), o CIM (*Computer Integrated Manufacturing*), ou mesmo FOF (*Factory Of the Future*).

O conceito de CIM é no entanto um conceito mais abrangente, já que tipicamente envolve não só os aspectos relacionados com as actividades de produção (fabrico) mas também as questões de *marketing*, vendas, gestão de *stocks*, finanças, pessoal, etc.

O CIM é a designação utilizada para descrever a completa automatização da fábrica, na qual todos os processos e actividades são controladas por computador, e onde a informação que circula é, tratando-se de completa integração, exclusivamente na forma digital. No ambiente CIM, a necessidade do papel, como veículo de informação, é eliminada.

1.3.1. A Integração CAD/CAE/CAM

O CAD (*Computer Aided Drafting & Design*), faz parte do subsistema da engenharia de concepção. Tem como objectivos, definir os requisitos para o produto que vai ser especificado, determinar a estratégia de concepção, formalizar a concepção por meio de documentação diversa (esquemáticos, listas de detalhes, etc).

O CAE (*Computer Aided Engineering*) também faz parte do subsistema da engenharia de concepção, sendo complementar do CAD. Tem como objectivo o de avaliar a exequibilidade e funcionabilidade daquilo que é especificado. Para o efeito, fornece ferramentas computacionais como as de análise mecânica, cinemática, de elementos finitos, simulação de circuitos digitais, etc.

O conjunto CAD/CAE é ainda responsável por outros aspectos não menos importantes e que dizem respeito quer à tradução da informação de especificação em informação utilizável pelos diferentes processos de fabrico quer à definição das ferramentas e da sequência de processos necessários.

Mais recentemente apareceu o conceito de DFM (*Design For Manufacturing*), que, como o próprio nome indica, visa uma especificação que facilite ao máximo as actividades de produção e de gestão. Por exemplo, reduzido o número de tipos de parafusos de um automóvel ter-se-ão menos complicações a nível de gestão e transporte de materiais. Soluções que apresentem as mesmas funcionalidades mas que se sejam menos problemáticas para o fabrico (encaixes, maquinações exequíveis, etc) deverão der previlegiadas.

Quais as vantagens de ter o processo de especificação automatizado e integrado? O CAD/CAE reduz o tempo que decorre desde o início da especificação, até ao início da produção, baixa os custos de especificação e desenvolvimento (o custo dos sistemas de CAD/CAE têm vindo a baixar relativamento ao custo dos engenheiros de concepção e desenvolvimento) e consegue-se maior precisão e qualidade nas especificações e desenhos. Os produtos são cada vez mais complexos e cada vez mais sujeitos a modificações. Os sistemas de CAD/CAE facilitam a manutenção e actualização da documentação e podem obviar à necessidade da existência de

protótipos de teste e simulação. Os sistemas de CAD/CAE possibilitam, às empresas, ultrapassar esforços de concepção e especificação cada vez mais prolongados e complexos face a produtos com ciclos de vida cada vez mais curtos.

Para simplificar, considere-se a produção de uma peça mecânica. A integração dos sistemas CAD/CAE/CAM torna possível, a partir do esquema ou desenho da peça: obter o modelo bi ou tri-dimensional da peça, proceder a estudos de engenharia sobre o modelo e eventualmente re-especificar o modelo, estimar custos de produção e analisar soluções alternativas, planear a sequência de operações do processo de fabrico e gerar os programas para as máquinas de controlo numérico que irão executar a peça, transferir estes programas para as máquinas.

1.3.2. Gestão dos Recursos de Fabrico (MRP II)

O CAPP (*Computer Aided Production Planning*) envolve as actividades relacionadas com o escalonamento da produção, gestão de materiais, planeamento das capacidades, etc. A propósito de CAPP, pode ser feita uma breve referência a um outro conceito muito importante: o MRP. Na diversa bibliografia especializada, aparece o MRP I (*Material Requirements Planning*) e um conceito mais vasto, o MRP II (*Manufacturing Resource Planning*).

Os requisitos do MRP II (controlo do inventário, planeamento de materiais, planeamanto da produção, previsões de mercado, etc) infuenciam de forma evidente todo o conjunto de actividades relacionadas com a empresa. De uma certa forma estabelece a ponte entre aquilo que se designou por funções económicas e todas as restantes actividades. O CIM será por isso melhor definido se for apresentado como sendo:

$$CIM = (CAD + CAE + CAM + CAPP) * MRP$$

1.4. Sistemas Industriais

Antes de abordar as funções envolvidas na actividade de fabrico (entendida no sentido CIM) e o tipo de informação que circula entre as suas diferentes entidades, é razoável fazer a distinção entre alguns tipos básicos de indústria. Esta distinção é importante, uma vez que as solicitações técnicas são diferentes consoante os tipos de indústria.

Convencionou-se a divisão em dois grupos principais, ditos da Indústria de Manufactura (ou Indústria de Fabrico Discreto) e da Indústria de Processos. Assim, dentro das indústrias de fabrico discreto podem dar-se como exemplos a Indústria Electrónica e de Computadores, a Indústria Automóvel, a Indústria Aeronáutica, a Indústria Têxtil e a Indústria de Moldes.

Exemplos para a indústria de processos podem ser a Indústria Petroquímica, a Indústria de Produção de Energia, a Indústria de Refinação, a Indústria Farmacêutica e Química.

Se na primeira, o fabrico de objectos físicos discretos (separados) é típico, já certo tipo de indústria de processos se pode caracterizar pela produção de substâncias sólidas, líquidas ou gasosas, com um fluxo contínuo de materiais. Sobressaem imediatamente diferentes requisitos numa e noutra.

Ainda ao nível das distinções genéricas de tipos de indústria, é possível descrever subtipos, por exemplo, ao nível da indústria de manufactura. Neste caso, a classificação tem a haver com as quantidades de produtos produzidos. Um exemplo, é o tipo de Produção em Massa ou Larga Escala. Neste caso, o sistema produtivo está exclusivamente dedicado a um produto (eventualmente alguns) particular. Durante o ciclo de vida da fábrica apenas são possíveis pequenas modificações. Alteração na especificação do produto implica reorganizações na fábrica. Na Produção em Pequena Escala, um pequeno número de unidades é produzido simultaneamente. Neste caso, o sistema produtivo está apto a adaptar-se a contínuas alterações das tarefas a executar. Isto tem a haver com as características de flexibilidade já referidas anteriormente. Um exemplo extremo em termos de exigência de flexibilidade é a indústria de moldes (uma das mais importantes a nível nacional). Este é um dos tipos de indústria que se aproxima mais do conceito de indústria de protótipos (cada produto final é um produto específico). Obviamente que a produção em pequena escala exige grandes recursos ao nível da engenharia de especificação e concepção, controlo e planeamento da produção (produzir em paralelo diferentes produtos utilizando recursos de produção concorrentes é uma tarefa complicada ...).

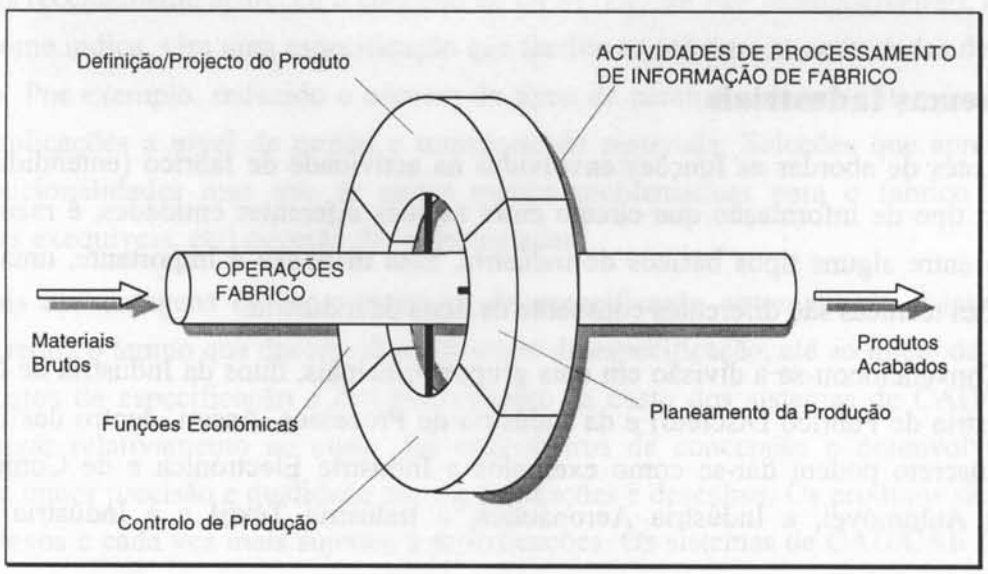


Figura 1.4 *- Organização das Actividades de Fabrico*

A análise que se segue, aborda as diferentes funções existentes no âmbito das actividades de fabrico. Pretendendo ser tão genérica quanto o possível, reflecte melhor as características inerentes à indústria de manufactura.

1.5. Organização e Processamento de Informação em Ambiente de Fabrico

Muitas companhias fazem centenas de produtos diferentes, cada um constituído por inúmeros componentes individuais, talvez na casa dos milhares. A tarefa de coordenar todas as actividades individuais requeridas para executar os componentes, assemblar e entregar o produto ao cliente, é sem dúvida complexa.

Quando se pretende fazer enquadramento desta situação com o do objectivo competitividade, o modelo de fabrico e o tipo de informação envolvida resultam evidentemente complexos.

A figura *Fig. 1.4* descreve os fluxos de informação e as funcionalidades tipicamente envolvidas nas actividades de fabrico. O cilindro interno representa as operações ou processos de fabrico propriamente dito.

1.5.1. Funções das Operações de Fabrico

1 - Processamento de materiais

As operações de processamento transformam o produto de um estado de acabamento para um outro estado de acabamento. Nenhuns materiais ou componentes são adicionados ou assemblados para atingir essa transformação. É sim utilizada energia (mecânica, calorífica, eléctrica, química, etc) para mudar a forma do componente, remover material, alterar as propriedades físicas, etc. Exemplos de componentes da automação usados para estas operações: máquinas ferramenta (quinadoras, frezadoras, máquinas de corte, etc), reactores químicos.

2 - Assemblagem dos materiais

A assemblagem é o segundo tipo de operações de fabrico mais efectuadas e consiste em juntar um ou mais componentes discretos. São incluídos neste grupo, as operações de juntar, usando parafusos ou afins, rebites, ou processos de aquecimento, abrasamento, soldadura, etc. Exemplos de componentes da automação usados: manipuladores, *robots*, mesas de montagem (ex. na indústria de electrónica).

3 - Transporte e armazenamento dos materiais

Os sistemas de transporte e armazenamento são muito críticos na indústria de fabrico discreto. Alguns autores, [TEI - 89], apontam tempos de 70% para este tipo de tarefas. Na maioría dos sistemas de fabrico tradicionais, os materiais passam mais tempo a serem transportados e armazenados do que a ser processados. Em muitos casos, os custos laborais de uma fábrica são resultantes, em grande medida, da movimentação e transporte de materiais. É importante que esta função seja especificada com eficiência. Exemplos de componentes da automação usados: AGV (*Automated Guided Veichles*), tapetes rolantes, armazéns automáticos, cadeias de transporte, etc.

4 - Inspecção e teste

A inspecção e teste está associada ao controlo de qualidade. O objectivo é o de verificar e determinar se um determinado produto obedece às especificações previstas. A informação de teste (que especificações são para cumprir, em que pontos da sequência de produção se deve fazer o teste, ...) é gerada ao nível das actividades de processamento de informação de fabrico. Exemplos de componentes de automação usados: barramentos de instrumentação de laboratório, como o GPIB, com esquemas de teste automático, sistemas de visão artificial (por exemplo na indústria têxtil e corticeira) que utilizam técnicas de processamento e análise de imagem.

5 - Controlo do fabrico

O controlo de fabrico, inclui a regulação de processos individuais e das operações de assemblagem, bem como a gestão de todas as actividades da fábrica, sendo o ponto mais importante de contacto entre as actividades físicas do fabrico e as de processamento de informação de fabrico. Exemplos: controlo de movimento, controlo de posição, controlo de processos químicos, rotulação e identificação, aquisição de dados. Componentes: sensores, actuadores controlados por dispositivos inteligentes (controladores baseados em microprocessadores, PLC, NC, micro, mini ou computadores *mainframe*).

De notar a hierarquia existente ao nível do controlo. Por exemplo, numa célula de processamento constiuida por duas máquinas ferramenta, por uma mesa de assemblagem que inclui um manipulador (*robot*) e sistemas de transferência para entrada e saída da célula, existem muitos controladores: diversos controladores ao nível do *robot*, controlo numérico (NC - *Numerical Control*) ao nível das máquinas ferramenta, por exemplo um PLC a comandar e supervisionar o sistema de transferência, um micro ou minicomputador a coordenar toda a actividade da célula.

1.5.2. Actividades de Organização, Geração e Processamento da Informação de Fabrico

1 - Funções económicas

As funções económicas são o principal meio de comunicação com o cliente e são por isso o princípio e o fim da cadeia de organização, geração e processamento da informação de fabrico. São de extrema relevância, para a actividade de fabrico, variáveis como a procura do produto (em quantidade e qualidade), gostos do cliente, prazos de entrega, etc. Estas variáveis refletem-se em aspectos tão importantes como, por exemplo, o escalonamento e planeamento da produção ou as especificações do produto.

É crescente a quantidade de informação recolhida pelo departamento de *Marketing:* resultante de pesquisas do mercado, preços da concorrência, situação económica nacional/ internacional, características dos clientes, etc. É cada vez mais usual a utilização de sistemas periciais de apoio às decisões de gestão (aplicação de conceitos do domínio da inteligência artificial) por forma a que elas sejam tomadas em tempo real. Essas decisões são de capital importância para o controlo e planeamento da produção bem como para a engenharia de concepção de produtos.

O departamento de compras é também essencial nas modernas estratégias de produção. Os grandes objectivos são a redução de *stocks* (quer de matérias brutas quer de produtos), a capacidade de entrega dentro dos prazos, etc. Isto implica decisões em tempo real na gestão de materiais, ferramentas e *stocks*. Um desempenho pouco eficiente destas actividades pode comprometer gravemente o plano de produção.

2 - Definição/Projecto do Produto

Se o produto é da responsabilidade da empresa, ela é responsável pela sua especificação e desenvolvimento. O conjunto de iniciativas que dão origem a esta fase provém, usualmente, dos departamentos de vendas e de *marketing*. Departamentos envolvidos: investigação e desenvolvimento, *design engineering*, *drafting*, elaboração de protótipos. É óbvio que se trata de um processo iterativo, na medida em que os estudos de engenharia, com base nos modelos resultantes de especificação prévia, poderão ser utilizados para alterar essas mesmas especificações. As especificações podem também ser alteradas por análise de viabilidade (técnica e financeira).

É preciso seleccionar os materiais e os métodos de fabrico, fazer a análise estrutural, especificar projecto de sistemas (p. ex., electrónico, pneumático, mecânico... no caso de um automóvel), etc.

3 - Planeamento da Produção

A documentação que constitui a especificação do produto é utilizada como *input* para a função de Planeamento da Produção. Os departamentos usualmente envolvidos são os de : engenharia de produção, engenharia industrial, de planeamento e controlo da produção, *marketing*, etc. O planeamento da produção inclui o planeamento de processos e o planeamento de capacidades.

O objectivo do planeamento de processos é o de determinar a sequência de processamentos individuais e operações de assemblagem. O planeamento da produção também é responsável pela afectação dos recursos de fabrico às necessidades de produção.

O Capacity Planning, ou planeamento das capacidades, está relacionado com as necessidades em termos de recursos humanos e materiais.

4 - Controlo da produção

Está relacionado com a gestão e supervisão das operações físicas da fábrica. O fluxo de informação vem do planeamento da produção. A informação flui nos dois sentidos com as operações da fabrico. Este tipo de controlo não é o mesmo do anteriormente referido como sendo o controlo de fabrico. Alguns dos aspectos relacionados com o controlo da produção:

Shop floor control - monitorizar e controlar o progresso do produto que está a ser processado, assemblado, transportado ou inspecionado na fábrica.

Inventory control - em certa medida está relacionado com o *Shop floor control.* O objectivo é manter no mínimo o inventário ou seja, o somatório de *stocks* (de matérias primas, produtos acabados ou semi-acabados) com o WIP (*Work In Progress*) que diz respeito a componentes em processamento, fazendo um balanço entre o limite perigoso de ter pouco *inventory* (por exemplo, a possibilidade de ficar sem *stocks*) e custo monetário de ter muito *inventory*.

Quality control - mecanismos que possibilitam garantir a entrega de produtos dentro das especificações de qualidade desejadas.

A implementação de todas as funcionalidades atràs descritas, torna evidente o grau de integração necessário em termos de sistemas de informática industrial e realça a importância dos sistemas de comunicação (redes de comunicação industriais) como suporte tecnológico para a grandes quantidades de informação transferidas entre as diversas entidades cooperantes.

1.6. Referências e Bibliografia

[RAN - 83] **Paul Ranky**, *The design and the Operation of FMS*, North-Holland, 1983.

[KOR - 85] **Hans Körner**, **Dietrich Reisch**, *CIM - The Key to the Automated Factory,* SIEMENS Power Engineering & Automation VII, No. 5, 1985, pp. 315-319.

[WAL - 85] **Siegfried Waller**, *Strategies of Factory Automation,* SIEMENS Power Engineering & Automation VII, No. 2, 1985, pp. 50-55.

[GRO - 86] **Mikell P. Groover**, *Automation Production Systems, and Computer Integrated Manufacturing*, Prentice-Hall International Editions.

[BOU - 87] **Daniel Bouteille**, **Jean-Paul Frachet**, *Les Automatismes Programables*, Cepadues-Editions, 1987.

[GUT - 87] **W. Gutshke**, **K. Mertins**, *CIM: Competitive Edge in Manufacturing*, Robotics & Computer Integrated Manufacturing, Vol. 3, No. 1, Pergamon Journals Ltd 1987, pp. 77-87.

[KUS - 88] **Andrew Kusiak**, **Sunderesh S. Heragu**, *Computer Integrated Manufacturing: a Structural Overview,* IEEE Network, Vol. 2, No. 3, Maio 1988, pp. 14-22.

[BIS - 89] **Tony Bishop**, **Norman Schofield**, *Unlocking the Potencial of CIM - A management Guide*, PA Consulting Services Ltd, 1989.

[KOR - 89] **Hans Körner**, *New Horizons in Industrial Automation,* SIEMENS Energy & Automation X, No. 1, 1989, pp. 4-7.

[TEI - 89] **Eric Teicholz**, **Joel N. Orr**, *Computer Integrated Manufacturing Handbook*, McGraw-Hill International Editions, Second Edition, 1989.

[SIE - 91] *A Factory With a Future,* SIEMENS Energy & Automation XIII, No. 2, 1991, pp. 22-29.

[STE - 91] **Roland Stender**, **Klaus Winkler**, *Automation Efficiency Upgrading,* SIEMENS Energy & Automation XIII, No. 5, 1991, pp. 4-7.

[BRA - 92] **Marco Leite Brandão**, *Rumo à Fábrica Automatizada: Os Sistemas Flexíveis de Fabricação*, Electricidade, No. 288, Abril 1992, pp. 149-155.

Capítulo 2

Fluxos de Informação em Ambiente CIM e Redes de Comunicação Industriais

O CIM engloba todas as actividades no âmbito do fabrico, não só aquelas relacionadas com as operações de fabrico propriamente ditas. A coordenação e cooperação entre os diferentes subsistemas relacionados com as actividades de fabrico traduz-se pela transferência, armazenamento e processamento de informação. Nesse sentido, têm papel preponderante os sistemas de comunicação de dados. Sendo os ambientes de fabrico tipicamente heterogéneos, do ponto de vista dos equipamentos utilizados, torna-se crucial a definição de normas multivendedor de comunicações.

2.1. Aspectos da Integração e Interligação em Ambiente Industrial

Todos os aspectos referidos no capítulo anterior têm grandes implicações na concepção dos sistemas informáticos para as respectivas indústrias. A *Fig. 2.1* ilustra o tipo de coordenação existente entre as diferentes entidades.

A automação não é nada de novo na indústria. Tem sido uma longa tradição a sua crescente aplicação ao nível da automatização local (*factory floor automation*), nomeadamente em máquinas ferramenta, certos processos de assemblagem, cadeias de transporte, etc.

Com a crescente disponibilização, material e financeira, dos computadores, o âmbito da automação alarga-se, tendendo a abranger aspectos administrativos e tarefas de processamento, organização e geração de informação em ambiente de fabrico. Estas tarefas mais "intelectuais" da automatização industrial são, por exemplo, as de controlo e planeamento da produção, gestão de materiais e recursos em ambiente de fabrico, engenharia de especificação de produto, controlo de qualidade. Novos desafios aparecem como resultado da aplicação da filosofia do fabrico flexível, dada a complexidade requerida ao nível dos sistemas de transporte automáticos e armazenamento automático.

A integração, significa a possibilidade dos subsistemas da empresa poderem interactuar entre si. De uma forma ou de outra está implicada a interligação. Erradamente, quando se fala de CIM tende-se a sobrevalorizar o C e o M, em detrimento do I.

A complexidade crescente dos sistemas bem como o *state of art* levou os engenheiros a concentrarem os seus esforços naquilo que facilmente podia ser isolado - dividir para conquistar.

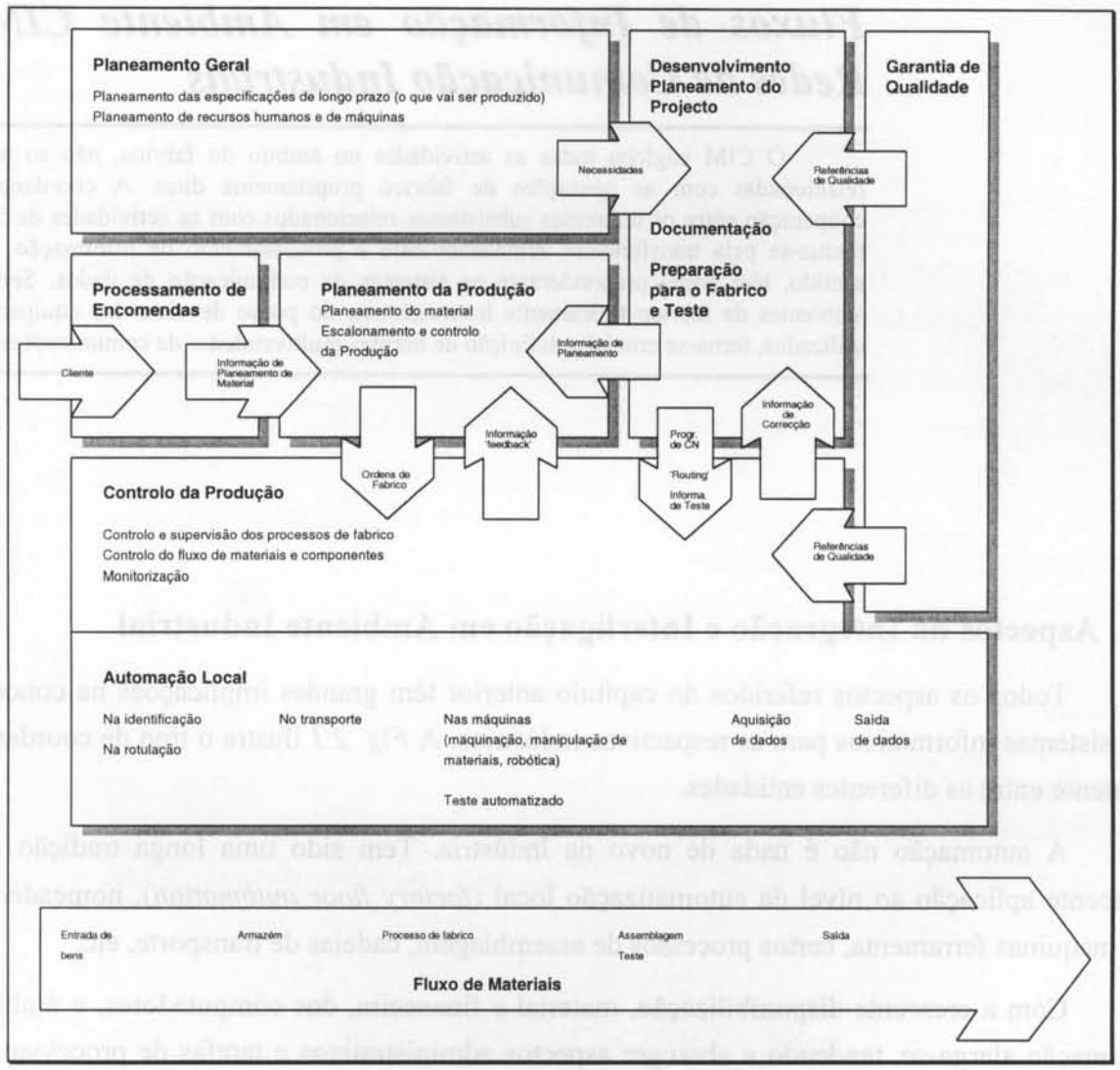


Figura 2.1. *- Fluxos de Informação em Ambiente de Fabrico*

De facto, é muito mais simples conceber um sistema de CAD cujo único objectivo é o de simplesmente facilitar a tarefa de desenho, descurando a preocupação paralela de como fazer a interface com o processo de fabrico/assemblagem. É muito menos problemático reunir dados estatísticos resultantes do controlo de qualidade e imprimir relatórios finais estatísticos, do que

reagir de imediato aos primeiros sinais de irregularidade, ou mesmo, de uma forma automática diagnosticar a causa.

A integração trata de reunir as soluções isoladas: eliminar os fossos entre as *Automated Islands* (fazer a ponte entre as diferentes ilhas CAx) e ligá-las às partes administrativas.

A integração implica cooperação entre subsistemas, cada um encarregue de uma determinada tarefa no processo industrial. Cooperação implica comunicações. Comunicações requerem infraestruturas técnicas (*software+hardware*). As comunicações requerem também regras (protocolos), regras essas, condicionadas não apenas por aspectos técnicos mas também pela funcionalidade exigida.

Quando se fala em comunicações fala-se na capacidade de tornar acessível informação, de uma localização geográfica para a outra, e, por outro lado, na capacidade de transferir informação de um ponto para o outro com eventual necessidade de armazenamento de informação por períodos definidos ou indefinidos de tempo.

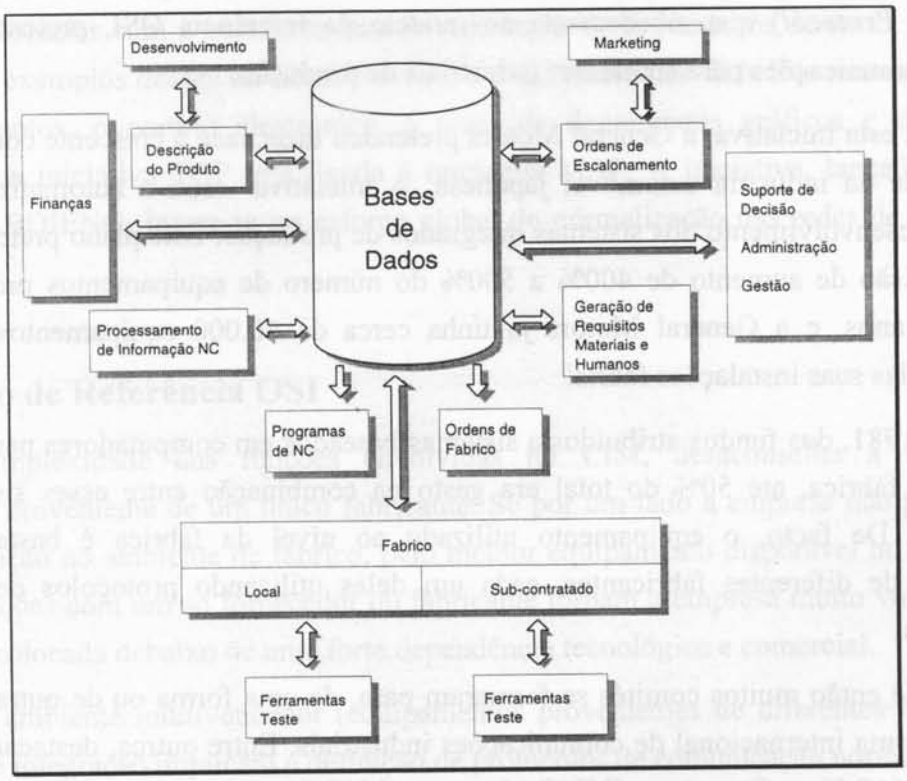


***Figura 2.2.** - Bases de Dados Comuns*

A integração significa assim a possibilidade de os subsistemas da empresa poderem interactuar entre si através de: sistemas de comunicações de dados e bases de dados comuns. No

que concerne a estas últimas, não confundir bases de dados comuns com base de dados centralizada (fisicamente), como pode aparentar a figura *Fig. 2.2*.

O modelo de fabrico pode ser visto numa perspectiva de bases de dados. Na realidade, a utilização de diferentes bases de dados em sistemas industriais de fabrico é actualmente profusa. A figura *Fig. 2.2* pretende mostrar a interacção existente entre as bases de dados localizadas em diferentes subsistemas de fabrico e uma base de dados comum. Por exemplo, uma base de dados contendo informação relativamente a ferramentas pode ser utilizada por diferentes subsistemas, como sejam: o planeamento de produção, o controlo de processos, os programas de controlo numérico, a preparação de ferramentas e manutenção de ferramentas, *robots* para a manipulação de ferramentas (colocação e substituição de ferramentas nas máquinas), e o controlo de *stocks* e armazenamento de materiais.

2.2. Redes de Comunicação Industriais

Nos Estados Unidos, a General Motors lançou em 1980 a iniciativa MAP (*Manufacturing Automation Protocol*) que, obedecendo ao modelo de referência OSI, pretende definir uma norma de comunicações para ambientes industriais de produção.

Com esta iniciativa, a General Motors pretendeu fazer face à crescente competitividade e agressividade da indústria automóvel japonesa. A iniciativa visou a automatização em larga escala e o desenvolvimento dos sistemas integrados de produção. Este plano pretendia fazer face a uma previsão de aumento de 400% a 500% do número de equipamentos programáveis em cerca de 5 anos, e a General Motors já tinha cerca de 40.000 equipamentos programáveis dispersos pelas suas instalações fabris!

Em 1981, dos fundos atribuídos a sistemas baseados em computadores para a automação ao nível da fábrica, até 50% do total era gasto na combinação entre esses sistemas e os já existentes!!! De facto, o equipamento utilizado ao nível da fábrica é bastante diverso e proveniente de diferentes fabricantes, cada um deles utilizando protocolos de comunicação proprietários.

Desde então muitos comités se formaram para, de uma forma ou de outra, contribuirem para uma norma internacional de comunicações industriais. Entre outros, destacam-se o EMUG (*European MAP User Group*) e o JMUG (*Japanese MAP User Group*). Neste momento, a norma internacional MAP está mais ou menos estabilizada, e corresponde à versão MAP 3.0 SP91 de 1991.

Paralelamente, na europa, através do programa ESPRIT, foi lançado o projecto CNMA (*Computer Networks for Manufacturing Applications*), com os seguintes objectivos: contribuir

para a definição, adopção e promoção de normas de âmbito universal para as comunicações industriais, realização de projectos piloto em ambientes reais e desenvolvimento de ferramentas de *software* para teste.

O CNMA contribuiu de forma decisiva para a especificação MAP, entre outros aspectos, nos serviços da camada de aplicação (MMS - *Manufacturing Message Specification*), nos serviços de gestão de rede (NMT - *Network ManagemenT*), e nos serviços de acesso a bases de dados remotas (RDA - *Remote Database Access*).

Refira-se a importância política da maior abrangência possível deste tipo de consórcios. Note-se que qualquer norma é sempre uma solução de compromisso.

Paralelamente à iniciativa MAP, e também nos Estados Unidos, um consórcio liderado pela BOEING lançou a iniciativa TOP (*Technical Office Protocol*), com o mesmo objectivo do MAP, mas especialmente vocacionado para as àreas técnicas e administrativas da empresa.

Evidentemente que as opções para as diferentes camadas do modelo OSI refletem os diferentes requisitos das aplicações MAP e das aplicações TOP. O TOP fornece essencialmente serviços relacionados com a transferência e interpretação de documentos de texto, de gráficos e técnicos. São exemplos desses serviços, o acesso e transferência de ficheiros remotos, o acesso a terminais remotos, o correio electrónico, a troca de documentos gráficos e de texto, etc. Actualmente, a iniciativa TOP está ligada à iniciativa MAP. A iniciativa, lançada em 1985 e liderada pela BOIENG, insere-se no esforço global de normalização das redes de comunicação de dados.

2.3. Modelo de Referência OSI

A complexidade das funções envolvidas no CIM, desaconselha a utilização de equipamento proveniente de um único fabricante. Se por um lado a empresa não podería optar, para cada função no ambiente de fabrico, pelo melhor equipamento disponível no mercado, por outro, as relações com um só fornecedor ou fabricante tornam a empresa muito vulnerável, uma vez que fica colocada debaixo de uma forte dependência tecnológica e comercial.

Num ambiente multivendedor (equipamentos provenientes de diferentes fabricantes), a interligação e integração implicam a definição de protocolos de comunicação normalizados.

A ISO (*International Organization for Standardization*) definiu o modelo de referência OSI (*Open Systems Interconnection*) [ISO/7498-84], com o objectivo de promover o aparecimento de normas na área das comunicações entre computadores, equivalente ao que na altura se verificava já para as comunicações telefónicas, definidas no âmbito da CCITT (*International Telegraph and Telephone Consultive Committee*).

O termo 'aberto' foi escolhido para denotar que, sendo conforme com normas OSI, um sistema está aberto a comunicar com qualquer outro que obedeça às mesmas normas.

Atente-se que o MR (modelo de referência) OSI não especifica, por si, as normas de comunicação. O seu propósito é o de simplesmente fornecer uma arquitectura que sirva de base ao desenvolvimento de normas para sistemas de interconecção. As diferentes normas encaixam-se na arquitectura OSI.

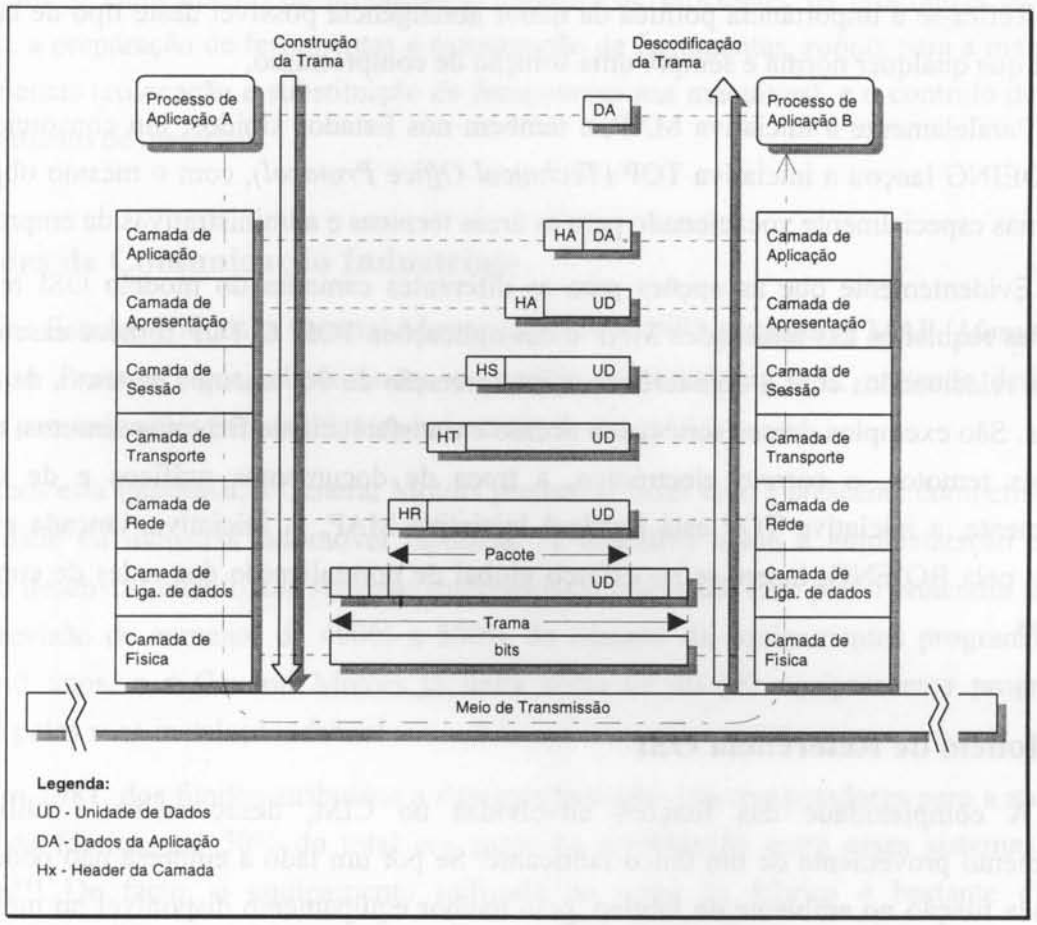


***Figura 2.3** - Arquitectura do MR-OSI*

O modelo de referência OSI define 7 camadas, conforme se indica na *Fig. 2.3* (adaptada de [PIM-90]). O âmbito de definição é o que, de uma forma não detalhada, se apresenta a seguir.

Camada Física - Especifica a ligação ao meio físico de transmissão. Especifica os aspectos relacionados com o próprio meio físico, como por exemplo as suas características eléctricas e mecânicas. Especifica o tipo de codificação e/ou modulação do sinal.

Camada de Ligação de Dados - Gestão das comunicações entre os sistemas ligados ao mesmo meio físico (controlo do acesso ao meio) e controlo e detecção de erros ocorridos na transmissão, entre outros aspectos.

Camada de Rede - Responsável pelo encaminhamento de mensagens através da rede (comutação de percursos, etc) ou entre redes lógicas (interligação de redes do mesmo tipo, mas autónomas em termos de endereçamento). Esta camada é na maioría das redes locais (LANs) quase nula, quando comparada com, por exemplo, as redes públicas de dados.

Camada de Transporte - Garante a transmissão eficiente e fiável entre dois sistemas finais: funções de controlo de fluxo e controlo de erro numa base *end-to-end*. Faz também segmentação.

Camada de Sessão - Organização e gestão da sessão de diálogo entre entidades identificadas por processos, independentemente da estação, ao contrário da camada de transporte que é orientada à estação. Processos a comunicarem utilizando a camada de sessão podem ser descritos (apenas como analogia) como sendo pessoas a comunicar através de assistentes que por sua vez utilizam o sistema telefónico.

Camada de Apresentação - Representação consistente e mutuamente aceite do formato de dados transmitidos: não basta haver ligação física, poder estabelecer a ligação, manter um diálogo, se as duas entidades não falam a mesma linguagem. Diferentes máquinas podem utilizar diferentes codificações para a representação de informação (ASCII ou EBDDIC), para só falar num exemplo. Outras das funções da camada de apresentação dizem respeito à compressão de informação e encriptação.

Camada de Aplicação - definição do significado das mensagens transmitidas. Constitui a interface entre as aplicações propriamente ditas e o sistema de comunicações. Para esta camada aparecem as mais variadas normas, consoante a aplicação em causa. Alguns exemplos de normas para a camada de aplicação, são as de transferência de ficheiros, de teletexto e videotexto e por exemplo protocolos especiais para máquinas de automação industrial (ex. *robots*, máquinas de controlo numérico) que têm de ter funcionalidades específicas. A norma internacional da camada de aplicação para troca de mensagens de fabrico é o MMS, norma ISO 9506.

2.4. MAP e TOP no Modelo de Referência OSI

Em abstracto, dois sistemas poderão comunicar entre si se a respectiva arquitectura de comunicações for equivalente em todos os aspectos. A *Fig. 2.4* ilustra as arquitecturas MAP e TOP, e as diferenças pontuais entre elas.

Incialmente o MAP só suportava o *Token Bus* como método de controlo do acesso ao meio. Essencialmente esse facto devia-se ao carácter determinístico do *Token Bus* (no sentido de se poder prever o máximo intervalo de tempo entre acessos consecutivos de uma estação) e porque as aplicações industriais são em grande parte determinísticas. Posteriormente o MAP passou também a suportar o CSMA/CD.

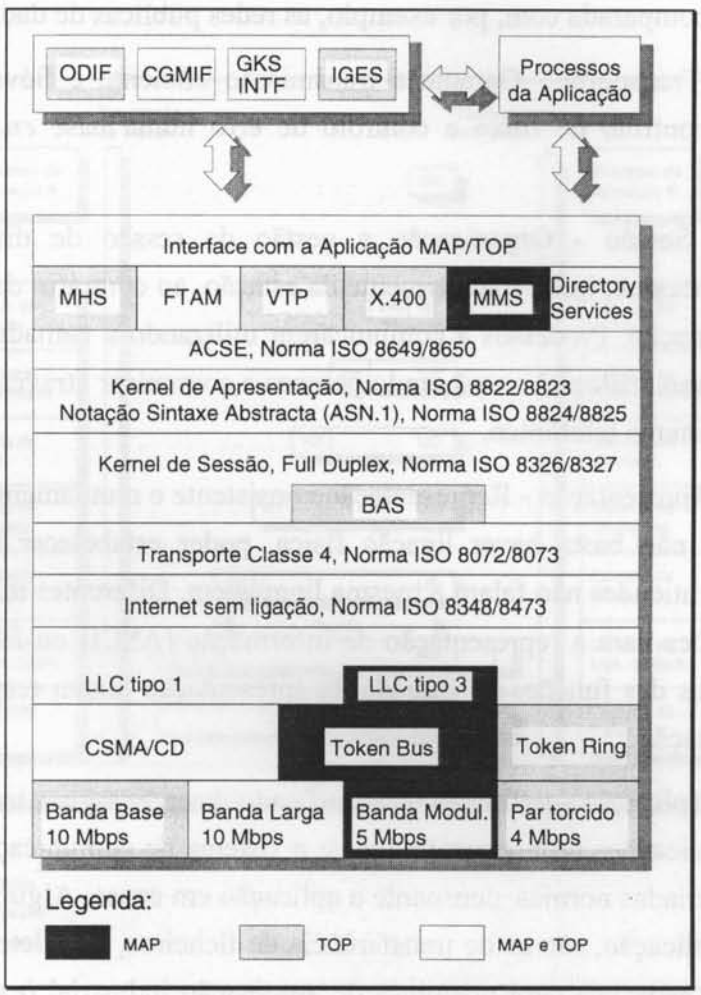


***Figura 2.4** - Arquitecturas MAP/TOP*

A diferenças mais substanciais residem, como é evidente, nos diferentes tipos de serviços fornecidos pela camada de aplicação: ambos suportam serviços de acesso a directorias e ambos suportam serviços de transferência de ficheiros. O TOP suporta o MHS (*Message Handling Specification*), uma norma específica de transferência de mensagens e o VTP, que é uma norma de emulação de terminais virtuais e serviços de correio electrónico, norma X-400.

ODIF - *Office Document Interchange Format*: especificação usada pelo TOP para a representação de documentos a transferir, usando os serviços FTAM ou MHS.

IGES - *Initial Graphics Exchange Specification*: especifica um formato para a transferência de informação de especificação de um produto (proveniente do CAD).

CGMIF - *Computer Graphics Metafile Interchange Format*: especifica um formato comum para a representação de informação gráfica. A informação pode ser transmitida e reconstituida para imagem, em qualquer sistema final TOP.

GKS INTF - *Computer Graphics Applications Interface*: fornece à aplicação uma interface normalizada com os serviços de computação gráfica.

2.5. Referências e Bibliografia

[IEEE - 83] *Special Issue on Open Systems Interconnection*, Proceedings of the IEEE, Vol. 71, No. 12, Dezembro 1983.

[GEKO - 84] **Chris Georgopoulos, Kristos Koukourlis**, *Fiber Optic Link Design Considerations for Applications in Noisy Industrial Environments*, IEEE Transactions on Industrial Electronics, Vol. 1E-31, No. 3, Agosto 1984, pp. 209-215.

[SCST - 85] **Armin Schwartz, Herbert Stier**, *Ethernet and ISO Transport Layer Protocol - The basis for the Solution of Automation Problemes of the Future,* SIEMENS Power Engineering & Automation VII, No. 3, 1985, pp. 188-192.

[WSG - 86] **Richard H. Weston, Chris M. Sumpter, Jack D. Gascoigne**, *Industrial Computer Networks and the Role of Map, Part 1,* Microprocessors and Microsystems, nº 7, Set 1986, pp. 363-370.

[ERWA - 87] **Dietrich Ernst, Siegfried Waller**, *Communications in Automation,* SIEMENS Energy & Automation IX, No. 1, 1987, pp. 2-7.

[WSG - 87] **Richard H. Weston, Chris M. Sumpter, Jack D. Gascoigne**, *Industrial Computer Networks and the Role of Map, Part 2,* Microprocessors and Microsystems, nº 1, Jan. 1987, pp. 21-34.

[CAT - 88] **J. M. Catalfo, G. Cancelo**, *LAN Node for Industrial Environments,* Microprocessors and Microsystems, Vol. 12, No. 7, Setembro 1988, pp. 383-390.

[CIDE - 88] **L. Ciminiera, C. Demartini**, *Performance Analysis of Type 3 LLC in Industrial 802.5 Networks*, Vol. 2, No. 3, Maio 1988, pp. 90-96.

[CJT- 88] **Prasad Chintameneni, Panjak Jalote, S. Tripathi**, *On Fault Tolerance in Manufacturing Systems,* IEEE Network, Vol. 2, No. 3, Maio 1988, pp. 32-39.

[DSP - 88] **J. N. Daigle, A. Seidmann, J. R. Pimentel**, *Communications for Manufacturing,* IEEE Network, Vol. 2, No. 3, Maio 1988, pp. 6-13.

[HACO - 88] **W. Hatfield, M. Coden**, *Fiber Optic LANs for Manufacturing Environment*, Vol. 2, No. 3, Maio 1988, pp. 70-74.

[LEGE - 88] **Arthur Lessard, Mario Gerla**, *Wireless Communications in the Automated Factory Environment*, Vol. 2, No. 3, Maio 1988, pp. 64-69.

[MAS - 88] **Madhav Marathe, Robin Smith**, *Performance of a MAP Network Adapter*, Vol. 2, No. 3, Maio 1988, pp. 82-89.

[MRS - 88] **L. J. McGuffin, L. Reid, S. Sparks**, *MAP/TOP in CIM, Distributed Computing,* IEEE Network, Vol. 2, No. 3, Maio 1988, pp. 23-31.

[RAY - 88] **Asok Ray**, *Networking for Computer Integrated Manufacturing,* IEEE Network, Vol. 2, No. 3, Maio 1988, pp. 40-47.

[SIE4 - 88] *The Marriage of ISDN and Factory Automation,* SIEMENS Energy & Automation X, No. 4, 1988, pp. 44.

[SITA - 88] **Leo Sintonen, Tapio Virvalo**, *The Hierarchy of Communication Networks,* IEEE Network, Vol. 2, No. 3, Maio 1988, pp. 48-54.

[STWE - 88] **W. Strayer, Alfred Weaver**, *Performance Measurement of Data Transfer Services in MAP*, Vol. 2, No. 3, Maio 1988, pp. 75-81.

[TAN - 88] **Andrew S. Tanenbaum**, *Computer Networks*, Prentice-Hall International Editions, Second Edition, 1988.

[SHWU - 89] **Holger Shulze, Klau Wucherer**, *Integrating Open Communications into the Automation Environment,* SIEMENS Energy & Automation X, No. 1, 1989, pp. 26-29.

[MIL - 90] **Mark A. Miller**, *Lan Protocol Handbook*, Prentice-Hall, 1990.

[PIM - 90] **Juan R. Pimentel**, *Communications Networks for Manufacturing*, Prentice-Hall International Editions, 1990.

[RAN - 90] **Paul Ranky**, *Computer Networks for World Class Cim Systems*, CIMware Limited, 1990.

[STA - 92] **Gerhard Stauss**, *Industrial Communications Networks: the Key to Integrated Manufacturing,* SIEMENS - Review R&D Special , Primavera 1992, pp. 12-15.

Capítulo 3

Redes de Comunicação Industriais Para os Níveis Inferiores da Hierarquia CIM - Field Bus

Sistemas de comunicações industriais, quer baseados em normas (nacionais ou internacionais) quer soluções proprietárias, começam a estar disponíveis para todos os âmbitos do ambiente de fabrico. Do ponto de vista do utilizador, o problema deixa de ser a eventual inexistência de soluções mas a crescente sobreposição das suas valências. É precisamente nos níveis inferiores da hierarquia CIM que este problema assume maior relevo. Este capítulo aborda as diferentes soluções, tentando justificar a necessidade de diferentes arquitecturas de comunicações com particular relevo para as arquitecturas do tipo *Field Bus*. Este capítulo corresponde sensivelmente ao conteúdo do artigo referênciado por [CAR-94].

3.1. Introdução

O CIM (*Computer Integrated Manufacturing*), visa a cooperação entre os diferentes sistemas intervenientes no ambiente de fabrico, e engloba todas as actividades com ele relacionadas, desde o projecto e desenvolvimento até ao *marketing* e vendas, passando pelo controlo do fabrico.

De acordo com a ISO (*International Organization for Standardization*), os sistemas de fabrico automatizados podem ser hierarquizados em seis níveis. Mesmo que visando essencialmente as indústrias de fabrico discreto, o modelo, ilustrado na *Fig. 3.1,* pode também ser aplicado ao controlo de processos.

A subdivisão em níveis é baseada, entre outros aspectos, nos tipos de actividades realizadas na empresa e leva, geralmente, ao uso de diferentes tipos de redes de comunicações nos vários níveis. A *Fig. 3.2* representa um exemplo de uma arquitectura possível para a hierarquia de comunicações dentro da empresa.

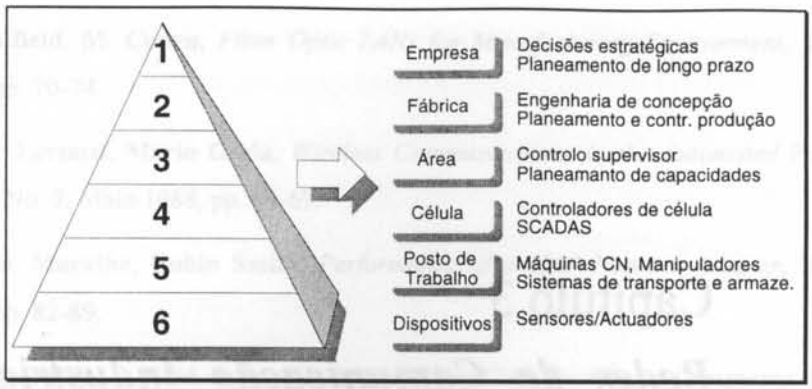


Figura 3.1 *- Modelo Hierárquico de Fabrico*

Em áreas tipicamente relacionadas com o planeamento ou com a engenharia de concepção (CAD/CAE), as comunicações podem ser caracterizadas como correspondendo à troca de grandes quantidades de informação, que tem de ser processada durante períodos relativamente longos mas com uma frequência relativamente baixa.

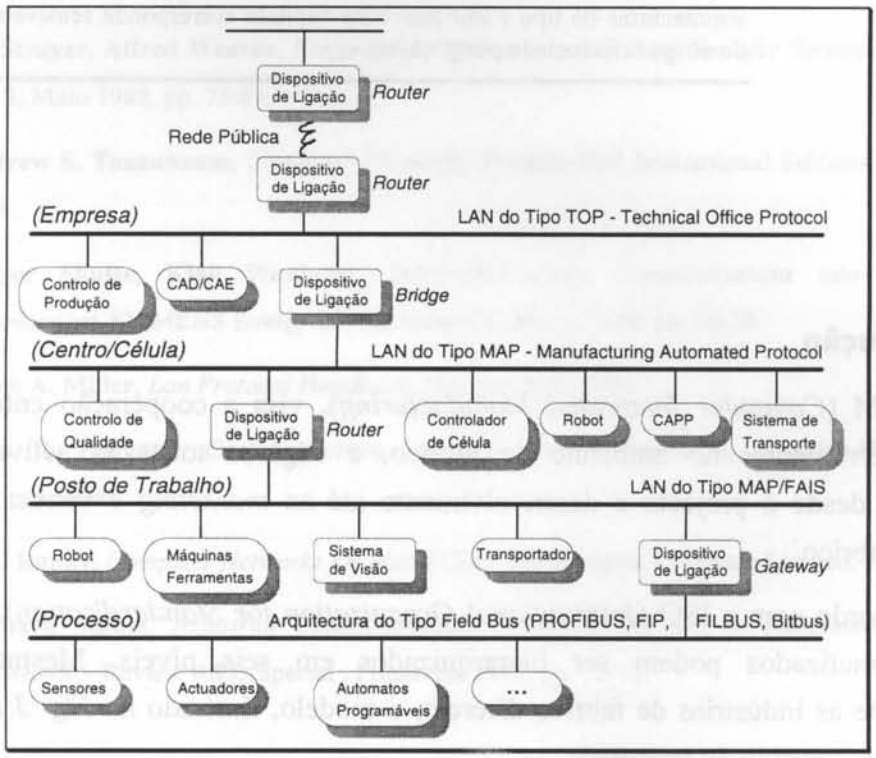


Figura 3.2 *- Hierarquia de Automação e de Comunicações*

Ao contrário, nos níveis inferiores da hierarquia, pequenas quantidades de informação precisam de ser processadas de uma forma rápida, com o objectivo de controlar processos

industriais de tempo crítico. Este tipo de transacções tem normalmente uma periodicidade cíclica e frequência relativamente elevada. A *Fig. 3.3* ilustra a caracterização dos vários tipos de dados referidos atrás.

Por razões óbvias, e que são melhor analisadas em §3.3., não é possível satisfazer todos estes requisitos de transferência de dados com um só tipo de rede, pese embora a vantagem que isso teria para o utilizador.

No nível mais baixo da hierarquia CIM existem redes de comunicações fortemente vocacionadas para a interligação de sensores/actuadores (redes normalmente designadas por redes de campo - *Field Bus*). Contudo, alguns dos tipos de redes para a interligação deste tipo de equipamento abrangem também sistemas finais de níveis superiores.

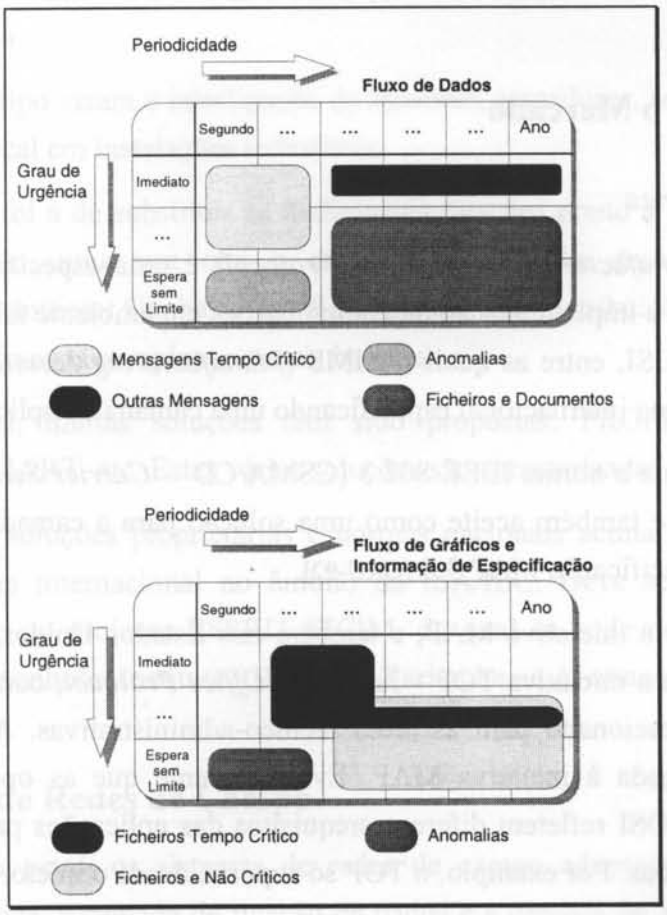


***Figura 3.3** - Caracterização de Tráfego de Comunicações*

O mesmo se pode dizer em relação a redes para os níveis mais elevados da pirâmide CIM, na medida em que poderão englobar sistemas finais de níveis intermédios: a versão MiniMAP do MAP teve esse propósito, ao procurar compatibilizar a performance do sistema de

comunicação com os requisitos de tempos de resposta mais exigentes de determinadas aplicações.

Pode assim propor-se uma classificação hierarquica das comunicações industriais em três grandes grupos. São eles, e alguns dos seus mais importantes exemplos:

Redes de Fábrica: MAP - *Manufacturing Automation Protocol;* CNMA - *Communications Network for Manufacturing Applications*; Ethernet TCP/IP; TOP - *Technical and Office Protocol*; FDDI - *Fiber Distributed Data Interface.*

Redes de Célula: MiniMAP; FAIS - *Factory Automation Interconnection System*; FIP - *Factory Instrumentation Protocol*; PROFIBUS - *PROcess FIeld BUS.*

Redes de Campo: PROFIBUS; FIP; BITBUS; CAN - *Controller Area Network*; HART - *Highway Addressable Remote Transducer*; FILBUS; InterBUS-S; Modulink; RACKBUS.

3.2. As Normas e o Mercado

3.2.1. Redes de Fábrica

O MAP (*Manufacturing Automation Protocol*) é uma especificação, patrocinada pela General Motors, para a implementação de comunicações em ambiente fabril, baseada nas normas ISO para o modelo OSI, entre as quais o MMS (*Manufacturing Message Specification*) que é, desde 1990, uma norma internacional especificando uma camada de aplicação muito poderosa.

É de referir que a norma IEEE 802.3 (CSMA/CD - *Carrier Sense Multiple Access with Collision Detection*) é também aceite como uma solução para a camada de ligação de dados e camada física na especificação MAP 3.0 [GRU-92].

Paralelamente à iniciativa MAP, e também nos Estados Unidos, um consórcio liderado pela BOEING lançou a iniciativa TOP - *Technical Office Protocol*, com o mesmo objectivo do MAP, mas mais vocacionado para as áreas técnico-administrativas. Actualmente a iniciativa TOP decorre interligada à inciativa MAP. Evidentemente que as opções para as diferentes camadas do modelo OSI refletem diferentesrequisitos das aplicações para as quais o MAP e o TOP estão vocacionadas. Por exemplo, o TOP só suporta, desde o início, a norma IEEE 802.3.

Apesar de o MAP ser uma norma internacionalmente aceite, as suas aplicações industriais mais importantes têm-se circunscrito à industria automóvel (GM, Mercedes, Renault). É de destacar ainda o sucesso de instalações piloto inseridas no projecto ESPRIT CNMA, entre os quais a nova fábrica de transformadores da EFACEC [QUI-92].

3.2.2. Redes de Célula

Existem vários tipos de redes no mercado que reivindicam cobrir esta área de aplicações: o MAP, especialmente a sua versão MiniMAP, a norma japonesa FAIS (uma implementação MiniMAP), o PROFIBUS [GOD-90] e com algumas restrições [AUT-92] [LET-91], o FIP.

De notar que a nível das redes de célula o suporte de MMS é essencial para a integração com os níveis superiores da hierarquia de comunicações. Qualquer das normas acima referidas suporta MMS ou sub-conjuntos MMS.

O MiniMAP é baseado em normas internacionais, o que não acontece com o FIP ou o PROFIBUS que tem especificações não normalizadas para as diferentes camadas. O FIP e o PROFIBUS apresentam a seu favor uma real penetração no mercado.

3.2.3. Redes de Campo

As redes deste tipo visam a interligação de sensores, actuadores, *racks* de entrada/saída e sistemas de controlo local em instalações industriais.

A ideia inicial foi a de substituir as tradicionais ligações ponto a ponto por um meio de comunicação partilhado, uma vez que o embaratecimento das tecnologias baseadas em microprocessador o permitiam. Ganha-se em flexibilidade, em distribuição da inteligência pelo campo, nas cablagens (menos complexas e mais baratas).

Para este nível, muitas soluções têm sido propostas: PROFIBUS, FIP, BITBUS, INTERBUS-S, CAN, HART, etc. Estas várias soluções são caracterizadas abaixo (§3.4.).

Para além das soluções proprietárias e normas nacionais acima mencionadas, está em preparação uma norma internacional no âmbito da ISA/IEC. Deve ser realçado o trabalho relevante desenvolvido no projecto ESPRIT FICIM, do qual se pode destacar uma instalação piloto localizada no Fraunhofer Institute (IITB) de Karlsruhe na Alemanha [KYM-92].

3.3. Arquitectura de Redes de Campo

De uma forma geral, os sistemas de redes de campo adoptam arquitecturas de três camadas: a camada física, a camada de ligação de dados e a camada de aplicação do conhecido modelo de referência OSI.

Arquitecturas de comunicações de três camadas foram concebidas como sendo sistemas compactos e mais baratos capazes de fornecer tempos de resposta rápidos requeridos em aplicações de tempo crítico.

A arquitectura de 3 camadas deriva das de 7 camadas através da eliminação das camadas intermédias do modelo OSI (*Fig. 3.4*). Contudo, esta ideia, bastante difundida, é um pouco ilusória. Por um lado, existem implementações da arquitectura MAP completa que dispensam de todo a funcionalidade de uma camada, por exemplo, a de rede. Por outro lado, as implementações ditas de 3 camadas acabam por incorporar na forma de sub-camadas, algumas das funcionalidades que nominalmente pertencem às camadas intermédias suprimidas.

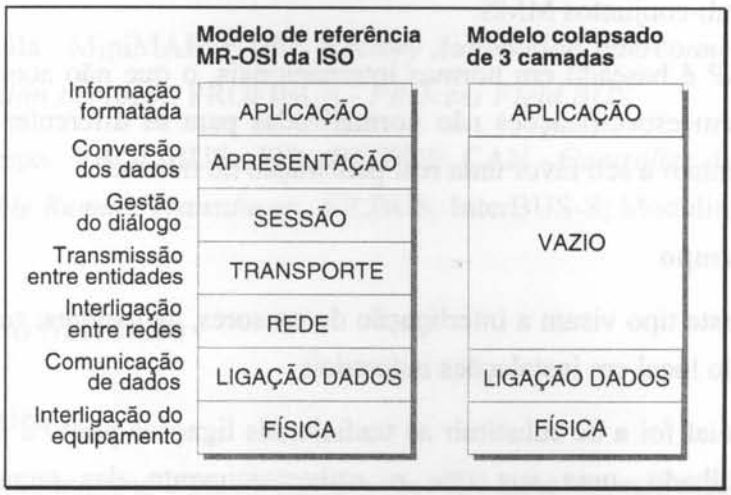


***Figura 3.4** - Modelo de Referência OSI e Modelo Colapsado de 3 Camadas*

Caracterizam-se a seguir vários condicionalismos genéricos de cada uma das três camadas, tal com são implementadas nas redes de campo.

3.3.1. Camada Física e *hardware*

Muitas aplicações, por exemplo, complexos petroquímicos, exigem redes de campo capazes de abranger grandes áreas. Este aspecto poderá resultar, uma vez que as soluções técnicas são sempre um compromisso, por exemplo em velocidades de transmissão relativamente baixas. A largura de banda pode ainda ser restringida por forma a limitar o consumo de energia, uma vez que microprocessadores e dispositivos electrónicos mais rápidos tenderão a exigir mais potência. Isto é tanto mais crítico na medida em que por razões económicas a alimentação de cada um dos nós pode em alguns casos ser feita através do barramento de dados.

Em certa medida a utilização de fibra óptica nas redes de campo poderá atenuar as limitações deste compromisso.

As restrições de potência têm também implicações a nível do protocolo. Protocolos complexos exigem *hardware* mais complexo (em quantidade e qualidade), logo mais caro e mais exigente em termos de consumo.

3.3.2. Camada de Ligação de Dados

As aplicações de redes de campo caracterizam-se por três aspectos que determinam fortemente a camada de ligação de dados e que são: o carácter determinístico da grande maioría das transacções, o número elevado de nós e a elevada frequência de transacções.

Relativamente à subcamada MAC (*Medium Access Control*), claramente o mecanismo aleatório do CSMA/CD especificado na norma IEEE802.3 não é muito apropriado para utilização em redes de campo, uma vez que as redes baseadas em CSMA/CD são tipicamente caracterizadas por um desempenho pobre quando a carga é relativamente elevada e existem muitas estações na rede. Além disso o CSMA/CD não pode assegurar qualquer tipo de escalonamento temporal nas transacções. Isto é tanto mais grave quando é pretendido conciliar-se o tráfego periódico e aperiódico.

Um método de controlo distribuido como o *Token Bus*, especificado na norma IEEE 802.4, apesar de ser quase determinístico (quase porque a manutenção e reconfiguração do anel implicam actividade aleatória no anel), também não se mostra especialmente adequado. O comportamento deste método é relativamente pobre quando, por exemplo, existem muitas estações na rede porque aumenta muito o tempo de rotação do *token*. A manutenção do anel e os processos de recuperação de situações de erro implicam algorítmos complexos que implicam *hardware* mais sofisticado. Em [GHO-89] são propostas algumas alterações ao IEEE 802.4, por forma a minorar algumas das suas desvantagens.

Quanto à sub-camada LLC, e do ponto de vista das redes de campo, normalmente o LLC tipo 1 (serviços não confirmados) será adequado para as transacções cíclicas, já que as variáveis são actualizadas periodicamente. Sendo assim, não é normalmente necessário retransmitir tramas erradas, sendo mais eficiente esperar pelo próximo ciclo de actualização.

Mesmo para as transacções não periódicas, pode, através de serviços da camada de aplicação, ultrapassar-se situações de erros de transmissão.

3.3.3. Camada de Aplicação

A especificação dos serviços deverá ter em conta as necessidades de simplificação quer dos pontos de vista de tempo consumido, quer do ponto de vista de *hardware*.

A camada de aplicação deverá suprir o processo de aplicação com ferramentas eficientes que possibilitem a manipulação e acesso a variáveis.

As redes de campo deverão fornecer serviços do tipo MMS, tais como serviços de acesso a variáveis, de gestão de eventos e de gestão de programas. Contudo, com vista a operações puramente cíclicas, serviços de refrescamento de bases de dados, deverão ser considerados.

A utilização de subconjuntos da norma MMS poderá trazer vantagens do ponto de vista de uma integração na hierarquia CIM. Por outro lado, a utilização de MMS implica o suporte de representação e codificação de informação de acordo com as normas ISO/8824 (ASN.1) e ISO/8825 (ASN.1-BER). Contudo, esta codificação implica uma grande complexidade e um *overhead* inadequado às características das redes de campo [FIC-91] [CAR-92].

3.4. Implementações de Redes de Campo

Neste momento existem no mercado várias propostas diferentes de redes de campo, o que se justifica por vários factores condicionantes. Um dos mais importantes é a existência de muitos campos de aplicação diferentes, que vão desde o controlo de processos até às máquinas de controlo numérico [BAG-91] ou braços de *robots*, passando por cadeias de transporte ou de assemblagem.

Tem-se assistido nos últimos anos à introdução de sistemas quer proprietários quer baseados em normas nacionais, que procuram satisfazer os requisitos de campos de aplicações mais ou menos especializados.

No quadro da *Fig. 3.5* estão sumariadas algumas das características das várias camadas de algumas das implementações que nos parecem mais relevantes.

O BITBUS, foi introduzido pela Intel em 1984, e impôs-se como uma norma de facto, tendo vendido vários milhões de dispositivos [GES]. Algumas das suas características mais importantes são a sua simplicidade e o seu baixo custo. Vários fabricantes propõem sistemas compatíveis com o BITBUS, como é o caso da GESPAC com o FILBUS [GES].

O FIP e o PROFIBUS são duas implementações mais complexas. Ambas suportam, em paralelo com serviços especialmente vocacionados para aplicações de redes de campo, sub-conjuntos MMS. Contemplam, dessa forma, as necessidades de aplicações com tempos de resposta mais exigentes e as necessidades de integração com níveis superiores ou mesmo de preenchimento das necessidades de níveis superiores. Ambos apresentam soluções muito particulares no que se refere aos métodos de acesso ao meio. O FIP, utilizando um controlo centralizado mostra-se no entanto mais adaptado aos requisitos de transferências cíclicas.

Nos guias de produtos do PROFIBUS e do FIP, centenas de produtos e serviços são listados. Dos respectivos consórcios fazem parte muitas dezenas de fabricantes, utilizadores e institutos de investigação. Um aspecto relevante é a participação nestes consórcios de entidades de países que não o de origem das normas.

Apesar disto, nenhum dos sistemas acima mencionados, apesar da sua implantação real, se conseguiu impor como norma internacional. A dificuldade em chegar a um consenso quanto a

uma norma internacional tem levado ao aparecimento de novos consórcios. É o caso do ISP (*Interoperable Systems Project*) em que estão presentes apenas quatro mas importantes companhias: Yokogawa (Japão), Rosemount (EUA), Siemens (Alemanha) e Fisher-Control (EUA). O perfil iminentemente intercontinental poderá ter o peso político capaz de impor a sua proposta, baseada no PROFIBUS, mas com extensões que deixam de limitar a performance deste em alguns aspectos.

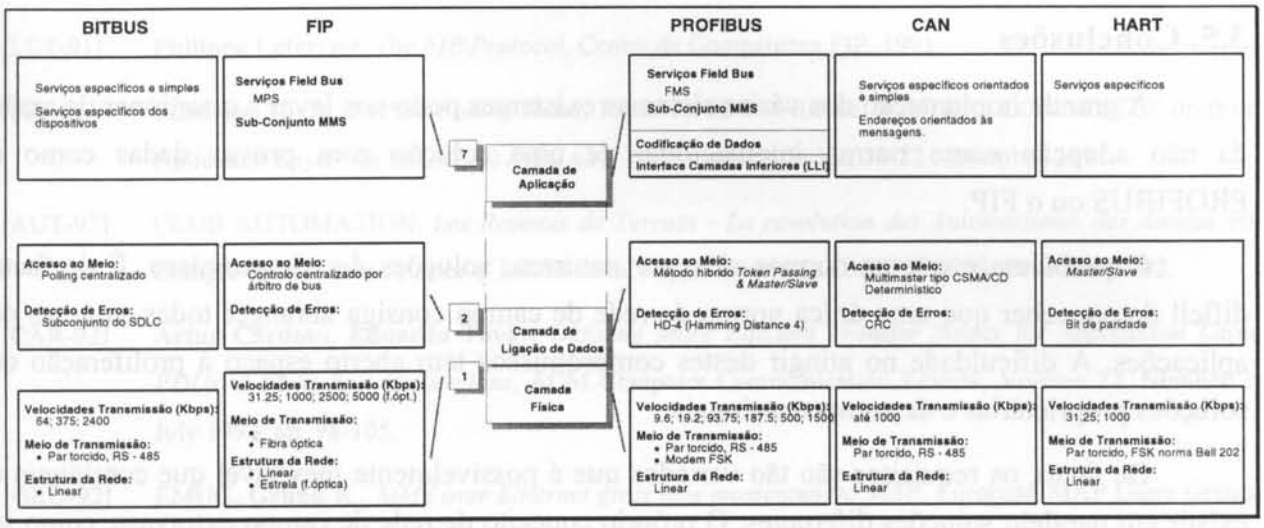


***Figura 3.5** - Súmula das Principais Características de Algumas Redes de Campo*

Posteriormente, e como resposta ao aparecimento do ISP, surgiu o consórcio WorldFIP, alargando o ClubFIP aos japoneses da FUJI, Mitsubishi e Hitachi e aos americanos da Allen Bradley, Honeywell e Exxon.

Fora das grandes lutas, o CAN [MAR-93] tem vindo, sem grandes alaridos, a ganhar importância. Originalmente o CAN foi desenvolvido para comunicações entre componentes nos automóveis. A sua aceitabilidade e implantação alargou o leque de aplicações para as máquinas ferramenta e o equipamento de controlo e instrumentação. Algumas das suas características atraentes são: o seu baixo custo de conexão, devido, entre outros aspectos, à quantidade de dispositivos produzidos (peso da indústria automóvel); a sua utilização e configuração simples (JPI - *Just Plug In*); a sua alta performance com baixo nível de *overhead* no protocolo.

O CAN utiliza um mecanismo de acesso ao meio do tipo *multimaster* de acesso aleatório. De facto, começaram a aparecer recentemente redes para ambiente industrial que utilizam aquilo a que se poderá chamar CSMA/CD determinístico ou como em alguma bibliografia [PIM-90], [BOU-87] CSMA/DCR (*Carrier Sense Multiple Access with Deterministic Collision Resolution*).

O CAN, assim como outros [WEI], combina as vantagens do CSMA/CD com as vantagens dos métodos de *token-passing* .

O HART tem actualmente uma forte penetração, principalmente na América do Norte (mais de 70 vendedores, mais de 500,000 instrumentos HART vendidos) [HAR-93]. Como característica mais relevante é de referir a possibilidade de utilização simultânea de ligações ponto a ponto 4-20mA e de comunicações digitais.

3.5. Conclusões

A grande implantação dos vários sistemas existentes pode-nos levar a questionar da razão da não adopção como norma internacional de uma solução com provas dadas como o PROFIBUS ou o FIP.

O problema é que as normas são, por natureza, soluções de compromisso. É de facto difícil de conceber que uma única norma de rede de campo consiga abranger todas as áreas de aplicações. A dificuldade no atingir destes compromissos tem aberto espaço à proliferação de soluções proprietárias e às normas nacionais.

De facto, os requisitos são tão variados que é possivelmente inevitável que continuem a existir em paralelo soluções diferentes. O próprio conceito de rede de campo extravasa, como se viu atrás, o ambiente de CIM. As aplicações do tipo "fechado" como por exemplo em automóveis, têm obviamente requisitos muito diferentes das aplicações onde é necessária a integração numa hierarquia de comunicações.

" *Réseau tout terrain?... Réseau de terrain ... et ... réseau de cellule...*"

em Revista MESURES 6/92

3.6. Referências e Bibliografia

[BOU-87] **J. Boundenant**, **B. Feydel**, *LYNX: an IEEE 802.3 Compatible Deterministic Protocol*, INFOCOM'87, São Francisco, Abril 1987, pp. 573-579.

[GHO-89] **Somnath Ghosh**, **David W. Pritty**, *Flexinet: a Token Bus Protocol for Local Area Networks Interconnecting Low Cost Data Communicating Devices*, Journal of Microcomputer Applications, No. 12, 1989, pp. 319-332.

[GOD-90] **J. Godertz**, *PROFIBUS*, Klockner-Moeller, Tecnical Publication VER 27-759 GB, Dez. 1990.

[KAT-90] **M. Katz, G. Biwer, Bjorn Raad**, *The PROFIBUS Application Layer. A New Approach to Open Communications in the Field Bus Area,* Forschungszentrun Informatik an der Universität Karlsruhe (FZI) Report, Outubro 1990.

[PIM-90] **Juan R. Pimentel**, *Communications Networks for Manufacturing*, Prentice-Hall International Editions, 1990.

[BAG-91] **J.-P. Baguet et Al.**, *Field Bus Requirements and Architecture for CNC*, Proceedings European Control Conference-91 (ECC-91), Grenoble, Julho 1991.

[LET-91] **Philippe Leterrier**, *The FIP Protocol,* Centre de Compétence FIP, 1991.

[FIC-91] **Eduardo Tovar**, *FICIM Coding Rules*, Paper 91-98-014-RE do "EDITING COMMITTEE draft on Aplication Layer" da ISA SP 50 / IEC 65C WG6, Friburg, Suiça, 18-22 Novembro 1991.

[AUT-92] CLUB AUTOMATION, *Les Reseaux de Terrain - La revolution des Automatismes des Annees 90*, Compte-Rendu de la Journee d' Information, Amphithéatre du CNRS, PARIS, Fevereiro 1992.

[CAR-92] **Artur Cardoso, Eduardo Tovar**, *Defining More Efficient Transfer Syntax for Application Layer PDUs in Field Bus Applications*, ACM Computer Communication Review, Volume 22, Number 3, July 1992, pp. 98-105.

[GRU-92] EMUG, **Grund K.**, *MMS over Ethernet gives new momentum to MAP*, European MAP Users Group, October 1992.

[KYM-92] **Kym Watson**, *Overview of FICIM*, Proceedings of the FICIM Workshop, Karlsruhe, Novembro 1992.

[PEI-92] **Jean-François Peirucat**, *Réseaux de Terrain: Ne Reportez Pas au Lendemain*, MESURES, No. 646, Junho 1992, pp. 24-29.

[QUI-92] **A. Quintas, F. Restivo, A. Carvalho, R. Oliveira**, *Soluções CNMA para os requisitos do Piloto EFACEC*, Proceedings of CNMA CEC, 1992.

[STA-92] **Gerhard Stauss**, *Industrial Communications Networks: the Key to Integrated Manufacturing,* SIEMENS - Review R&D Special , Primavera 1992, pp. 12-15.

[WEI-92] *PROFIBUS CPU,* Weidmüller Technische Dokumentation, Fevereiro 1992.

[HAR-93] **HUG - HART User's Group**, HART Field Communications Protocol, USA, Maio 1993.

[MAR-93] **Kurt Marquardt, Horst Müller**, *Peripherals for automotive and industrial electronics,* SIEMENS Components No. 6, Dec. 1993, pp. 12-15.

[CAR-94] **Artur Cardoso, Eduardo Tovar**, *Redes de Comunicação Industriais Para os Níveis Inferiores da Hierarquia CIM*, Livro do 1° Encontro Nacional do Colégio de Engenharia Electrotécnica, 12-13 Maio de 1994, pp. 107-112.

[GES] *FILBUS: Le reseaux de terrain,* GESPAC.

[MOD] *Serial Field Bus MODULINK,* Weidmüller Technical Information.

Capítulo 4

Sistemas Federadores de Subsistemas de Comunicações Heterogéneos

A necessidade de interligar sistemas provenientes de diferentes fabricantes é bem evidente no ambiente de fabrico. No entanto, quase sempre se depara com diferentes arquitecturas de comunicações, o que constitui um forte obstáculo à integração. Recentemente, alguns dos mais importantes fabricantes começaram a introduzir no mercado propostas globais constituíndo um esforço válido no sentido da integração total. Essas soluções, baseadas quase sempre em plataformas de arquitectura proprietária, tiram partido de diferentes acordos entre os diferenetes fabricantes, o que permite ofertas abrangentes e coordenadas para todos os níveis da hierarquia CIM.

4.1. Introdução

Os obstáculos à integração são numerosos. Desde o simples facto de os equipamentos utilizados serem de diversas gerações até ao de nem todos permitirem comunicar através de redes de comunicação normalizadas, passando por bases de dados que têm diferentes formas de acesso, inúmeros são os exemplos de dificuldades que podem surgir no âmbito da integração. Emerge assim, um problema de heterogeneidade que não é trivial.

A integração passa por fazer comunicar entre si, de uma forma harmoniosa, os diferentes níveis da hierarquia CIM. Não sendo possível cobrir todos esses níveis com uma mesma arquitectura de comunicações, e visando obstar às dificuldades naturais da integração de sitemas heterogéneos, os grandes fabricantes começaram a apresentar arquitecturas que se podem designar de federadoras. Podem destacar-se a DIGITAL com a sua proposta *BASESTAR* [BAS - 93], a IBM com o PFS-DAE [PFS - 93], a BULL com o *DCM-Manufacturing* [DCM - 93], a Allen-Bradley com a proposta *PYRAMID* e por fim, a Hewlett-Packard com o seu sistema designado *OpenCIM* [OPE - 93].

De comum, as diferentes filosofias tiram partido de uma espinha dorsal, que funciona como plataforma federadora de outros sistemas. A DIGITAL utiliza a sua conhecida *DECNet* como espinha dorsal do sistema de comunicações e plataformas VAX quer com VMS quer com ULTRIX. A IBM tem no seu *Token-Ring* a base da sua proposta. A HP serve-se da sua série 9000 e da sua linha de comunicações MAP [PATA - 90], [LCB - 90], [JOH - 90]. Basicamente, estes sistemas resultam de acordos feitos com terceiros. Por exemplo, o *BASESTAR* comunica com os equipamentos através de RS-232 ou *DECNet* (*Ethernet*). Por outro lado, são também feitos acordos quanto à utilização de pacotes de supervisão, controlo e planeamento. A DIGITAL, p.ex., tem uma gama que designou de *DECFactory*, juntamente com o gerador de aplicações *CimFast*. O trunfo reside em oferecer uma gama tanto mais alargada de equipamentos e sistemas possíveis de interligar.

4.2. O BASESTAR

4.2.1. Funcionalidades e Abrangência da Proposta

O *BASESTAR* funciona num ambiente distribuído e consiste numa ferramenta de comunicações que federa o sistema CIM, contendo uma base de dados relacional de tempo real, onde são definidos os sistemas, o equipamento e a lógica. Os dados poderão ser guardados na base de dados relacional (BDR) para a gestão de historiais, tendências, análise estatística, etc. Os sistemas de controlo e supervisão acedem à base de dados. O construtor integra na sua arquitectura diferentes tipos de redes de comunicação normalizadas, como o MAP e a *Ethernet*, que permite a ligação a equipamentos heterogéneos não normalizados, como a LacFactor ou mesmo proprietários como o *UniTelway*. Graças a um acordo com a Cegelec, o FIP faz também parte do pacote de ofertas.

No que diz respeito a equipamentos, o *BASESTAR* suporta diversos autómatos programáveis provenientes de diversos fabricantes de que são exemplo a ABB, AEG-Modicon, Allen-Bradley, April, Cegelec, GE-Fanuc, Klöckner-Moeller, Mitsubishi, Siemens, Telemecanique e Texas Instruments. A ligação é feita através quer através de *Ethernet*, quer através de cartas de comunicação com o formato *QBus* (encontrado nos VAX) ou ainda por ligação directa no próprio VAX, como é o caso do autómato Q-984 da AEG-Modicon. É de referir que os VAX são desde à longa data utilizados por fabricantes de sistemas de controlo numérico como a Honeywell, a Rosemount e a Bailey Controls. Por outro lado, e mais importante ainda, é de relembrar que a DIGITAL foi um dos três 'pais' da *Ethernet*, juntamente com a Intel e a Xerox.

No que se refere a pacotes de *software*, a oferta é também extremamente vasta. Alguns exemplos: aquisição de dados provenientes de leitura de códigos de barras da Burr-Brown e da Intermec, gerador de interfaces gráficas da VI, sistemas de tratamento de alarmes da Sferca, sistemas de manutenção da Champs Software, sistemas de controlo de qualidade da GE-Fanuc, Heuristics, Intelution, Setpoint, Sferca e da Sligos, sistemas de acompanhamento da produção da Consilium, geradores de sistemas periciais da Gensym e da Neuron Data e por fim, construtores de bases de dados a Ingres e Informix.

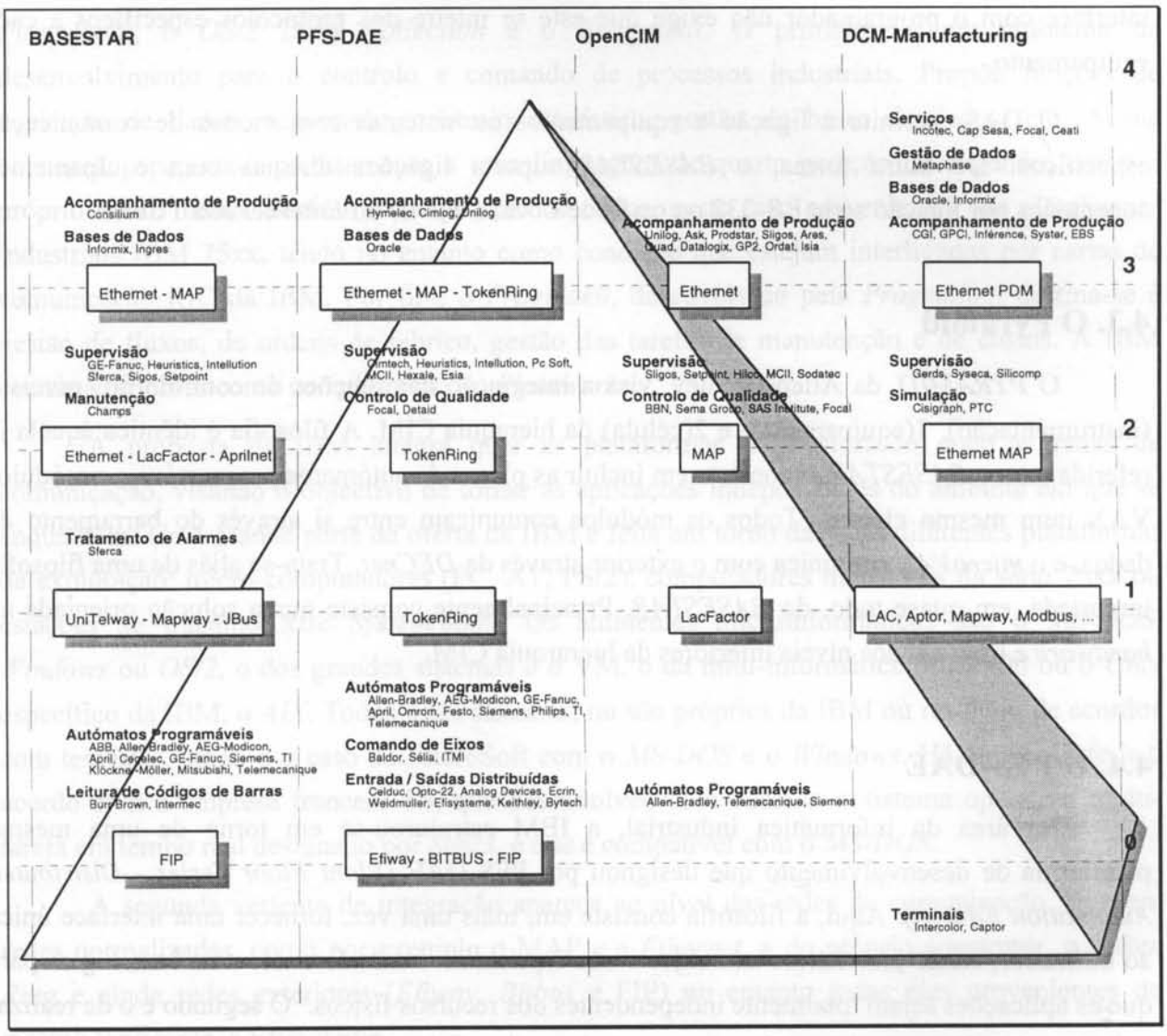


Figura 4.1 *- Comparação das Arquitecturas de Alguns Sistemas Federadores*

4.2.2. Arquitectura do BASESTAR

O *BASESTAR* é composto por três partes fundamentais: o núcleo, o DCM (*Device Connect Management*) e o DAS (*Device Access Services*).

O núcleo é responsável pela sincronização dos sistemas de controlo e supervisão da aplicação e por gerar os dados, as mensagens e as interfaces operadoras. O princípio base pelo qual se rege é o de que quem está a desenvolver a aplicação não conhece (nem quer conhecer) os tipos de comunicação existentes entre os diferentes equipamentos, nem sequer o tipo de equipamento a que está ligado.

O DCM é o supervisor que faz a ligação ao equipamento de automação. Permite a passagem de elementos lógicos a elementos físicos. Por exemplo, faz o mapeamento da variável lógica que corresponde a ligar/desligar um motor, e o comando de um determinado autómato. A interface com o programador não exige que este se inteire dos protocolos específicos a cada equipamento.

O DAS, permite a ligação a equipamentos ou sistemas com modos de comunicação específicos. De outra forma, o *BASESTAR* suporta ligações directas com equipamentos conectados por ligação série RS-232 ou ou Rede Local *DECnet* ou *Ethernet* sob TCP/IP.

4.3. O Pyramid

O *PYRAMID*, da Allen-Bradley, visa a integração das funções de controlo dos níveis 0 (instrumentação), 1(equipamento) e 2(célula) da hieraquia CIM. A filosofia é idêntica àquela já referida para o *BASESTAR*, e consiste em incluir as placas de autómatos programáveis e módulos VAX num mesmo chassis. Todos os módulos comunicam entre si através do barramento de dados, e o *microVAX* comunica com o exterior através da *DECnet*. Trata-se aliás de uma filosofia importada, em quase tudo, do *BASESTAR*. Principalmente consiste numa solução orientada ao *hardware* e limitada aos níveis inferiores da hierarquia CIM.

4.4. O PFS-DAE

Na área da informática industrial, a IBM estruturou-se em torno de uma mesma plataforma de desenvolvimento que designou por PFS-DAE (*Plant Floor Series - Distributed Automation Edition*). Aqui, a filosofia consiste em, mais uma vez, fornecer uma interface única ao utilizador, sendo prioritários três objectivos. O primeiro é o de fornecer uma base lógica para que as aplicações sejam totalmente independentes dos recursos físicos. O segundo é o de realizar a integração desse conjunto de recursos, sejam eles físicos, como por exemplo autómatos programáveis, comandos numéricos, ou recursos informáticos como sejam uma base de dados, um programa, um ecran ou um ficheiro. Os programas de aplicação do PFS-DAE vão invocar estes recursos sem saber onde eles se encontram na rede de comunicações ou mesmo que tipo de protocolos são utilizados para os aceder. O terceiro e último objectivo consiste em propiciar uma

visão única do sistema aos engenheiros de desenvolvimento. Para este efeito, a IBM propõe o PFS-ACE (*Plant Floor Series - Automation Control Edition*) destinado ao nível 1 da hierarquia CIM e para os níveis superiores uma série de ferramentas de geração de aplicações.

O PFS-ACE baseia-se numa linguagem de programação orientada ao objecto e de tempo real, a AML/2, cujos primórdios remontam já a 13 anos atrás. Uma aplicação escrita em AML/2 pode carregar dinamicamente um programa escrito em C. Mais do que uma simples linguagem, o PFS-ACE é uma ambiente de desenvolvimento que permite simular e executar aplicações.

Para os níveis 2 e 3 da hierarquia CIM a IBM disponibiliza três geradores de aplicação: o *Plantworks*, o *OS/2 Data Collection* e o *Pragmakit*. O primeiro, é um ambiente de desenvolvimento para o controlo e comando de processos industriais. Propõe funções de aquisição de dados, supervisão, controlo estatístico, gestão de bases de dados, etc. A sua concepção aparenta-o precisamente a uma linguagem de quarta geração. O segundo, como o próprio nome indica, destina-se à aquisição de dados, neste caso provenientes das plataformas industriais IBM 75xx, tendo no entanto como condição que estejam interligadas por cartas de comunicação RIC da IBM. Por fim, o *Pragmakit*, desenvolvido pela *Pragmalog*, destina-se à gestão de fluxos, de ordens de fabrico, gestão das tarefas de manutenção e de custos. A IBM costuma propô-lo como complemento do *Plantworks*.

A proposta da IBM integra quer as plataformas de exploração quer as redes de comunicação, visando o objectivo de tornar as aplicações independentes do ambinte em que se enquadram. Uma grande parte da oferta da IBM é feita em torno das suas diferentes plataformas de exploração: micro-computadores (PC, AT, PS/2), computadores industriais da série 75xx ou estações de trabalho *Risc System/6000*. Os ambientes microinformáticos são o *MS-DOS*, *Windows* ou *OS/2*, o dos grandes sistemas é o VM, o da mini-informática o *OS/400* ou o *Unix* específico da IBM, o *AIX*. Todos estes sistemas, ou são próprios da IBM ou resultam de acordos com terceiros, como é o caso da MicroSoft com o *MS-DOS* e o *Windows*. Há ainda a referir o acordo com a empresa francesa 3S, que desenvolveu com sucesso o sistema operativo multi-tarefa em tempo real designado por *Sim/3*, e que é compatível com o *MS-DOS*.

A segunda vertente de integração aparece ao nível das redes de comunicação. Existem redes normalizadas, como por exemplo o MAP e a *Ethernet*, a do próprio construtor, o *Token Ring* e ainda redes exteriores (*Efiway*, *Bitbus* e FIP) no entanto todas elas provenientes do mesmo fabricante, a Efisystème.

Os acordos feitos com diferentes fabricantes para os níveis 0 e 1 da hierarquia CIM aparecem referênciados na *Fig. 4.1*. Nos níveis superiores aparecem pacotes como o *CimView* da Cimtech, o *Onspec* da Heuristics, *The FIX* da Intellution, etc.

A IBM propõe ainda sistemas de 'chave na mão' para determinadas indústrias: o POMS (*Process Operations Management System*) para a indústria alimentar ou química e o PMW (*Paperless Manufacturing Workplace*), para a transmissão de informação entre os diferentes níveis da hierarquia CIM.

4.5. O OpenCIM

A Hewlett-Packard designa a sua oferta por *OpenCim*. A estratégia é no entanto ligeiramente diferente, e em grande medida porque a HP optou desde à algum tempo por fazer abordagens indústria a indústria. De facto, a HP desenvolveu centros europeus especializados em tipos de indústrias, como são exemplos as indústrias automóvel/mecânica, a indústria electrónica, a indústria de defesa, a indústria aeroespacial, a indústria agro-alimentar e a indústria química e farmacêutica. Esta abordagem vai ainda mais longe, uma vez que os acordos são feitos país a país.

O *OpenCim* baseia-se em duas grandes famílias de ferramentas complementares. As ditas ferramentas de base são desenvolvidas pela própria HP. São exemplo o *NewWave* que é uma interface de comunicação em rede. Outro exemplo das ferramentas de base é o *HP Sockets*, que permite a aplicações das mais diversas origens, comunicarem entre elas em ambientes físicos heterogéneos. A segunda família de ferramentas são usualmente complementares destas e resultam acordos com terceiros.

A aplicação *HP-Sockets* tem como principal objectivo fazer com que duas aplicações desenvolvidas independentemente possam entender-se e não somente trocar dados entre si. Basicamente o *socket* corresponde a um codificador/descodificador de dados visando compatibilizar as estruturas de dados. Basta escrever uma dúzia de linhas de programa em cada aplicação que façam a sua 'socketização'.

O parente pobre da oferta *OpenCim* é por enquanto ao nível dos equipamentos, como se deduz da *Fig. 4.1*. No entanto é de referir que a aplicação *HP-Sockets* funciona nos sistemas HP, BULL, IBM ou DIGITAL. Nos níveis superiores da hierarquia CIM surgem inúmeras possibilidades.

4.6. Dispositivos Para a Interligação de Redes de Comunicação

Quando dois sistemas não estão ligados directamente um ao outro, e a conectividade é conseguida através de um intermediário, ou dispositivos de interligação, esse dispositivo de interligação é conhecido, na terminologia OSI, por *relay*. Quando um *relay* partilha uma camada

A IBM propõe ainda sistemas de 'chave na mão' para determinadas indústrias: o POMS (*Process Operations Management System*) para a indústria alimentar ou química e o PMW (*Paperless Manufacturing Workplace*), para a transmissão de informação entre os diferentes níveis da hierarquia CIM.

4.5. O OpenCIM

A Hewlett-Packard designa a sua oferta por *OpenCim*. A estratégia é no entanto ligeiramente diferente, e em grande medida porque a HP optou desde à algum tempo por fazer abordagens indústria a indústria. De facto, a HP desenvolveu centros europeus especializados em tipos de indústrias, como são exemplos as indústrias automóvel/mecânica, a indústria electrónica, a indústria de defesa, a indústria aeroespacial, a indústria agro-alimentar e a indústria química e farmacêutica. Esta abordagem vai ainda mais longe, uma vez que os acordos são feitos país a país.

O *OpenCim* baseia-se em duas grandes famílias de ferramentas complementares. As ditas ferramentas de base são desenvolvidas pela própria HP. São exemplo o *NewWave* que é uma interface de comunicação em rede. Outro exemplo das ferramentas de base é o *HP Sockets*, que permite a aplicações das mais diversas origens, comunicarem entre elas em ambientes físicos heterogéneos. A segunda família de ferramentas são usualmente complementares destas e resultam acordos com terceiros.

A aplicação *HP-Sockets* tem como principal objectivo fazer com que duas aplicações desenvolvidas independentemente possam entender-se e não somente trocar dados entre si. Basicamente o *socket* corresponde a um codificador/descodificador de dados visando compatibilizar as estruturas de dados. Basta escrever uma dúzia de linhas de programa em cada aplicação que façam a sua 'socketização'.

O parente pobre da oferta *OpenCim* é por enquanto ao nível dos equipamentos, como se deduz da *Fig. 4.1*. No entanto é de referir que a aplicação *HP-Sockets* funciona nos sistemas HP, BULL, IBM ou DIGITAL. Nos níveis superiores da hierarquia CIM surgem inúmeras possibilidades.

4.6. Dispositivos Para a Interligação de Redes de Comunicação

Quando dois sistemas não estão ligados directamente um ao outro, e a conectividade é conseguida através de um intermediário, ou dispositivos de interligação, esse dispositivo de interligação é conhecido, na terminologia OSI, por *relay*. Quando um *relay* partilha uma camada

desemcapsulamento de tramas, já que elas são diferentes nos dois casos. A *bridge*, no lado *token-bus* participa na rotação do *token* como se fosse uma estação.

As arquitecturas pertencentes aos dois segmentos interligados pela *Bridge* deverão ser iguais, à excepção das especificações da camada de ligação de dados e da camada física.

4.6.3. Routers

Um *router* possibilita a interligação de duas redes lógicas (em termos de endereçamento) independentes. Duas estações em dois segmentos de rede diferentes interligados por um *router* podem ter endereços idênticos. O endereço das redes (relacionado com um mínimo de protocolo de camada de rede suportado pela maioría das LANs) é que é diferente.

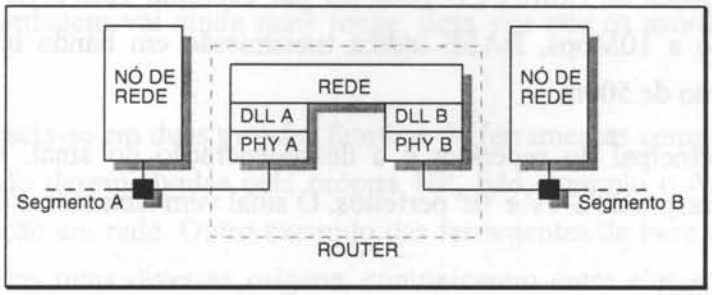


Figura 4.3 *- Contexto de Utilização de um Router*

As estações interligadas por *Routers* têm de ter protocolos das camadas 4 a 7 iguais. A interligação é feita ao nível da camada de rede, pelo que o *router* poderá interligar dois segmentos com camada de ligação de dados e camada física diferentes.

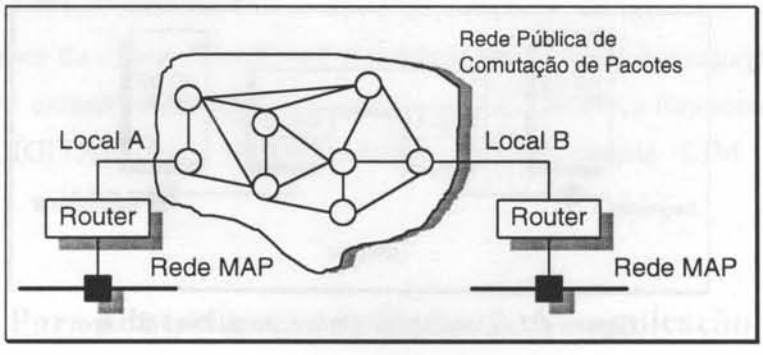


Figura 4.4 *- Outro Contexto de Utilização de um Router*

No caso de interligação de estações MAP e *MiniMAP* poderão ser utilizados *routers* para interligar redes lógicas distintas. Par o *MiniMAP* faz o *bypass* do protocolo das camadas 4, 5, e 6

proveniente de estações MAP e acrescenta os respectivos *headers* quando quer enviar para estações MAP. Ao nível da rede *MiniMAP* circulam exclusivamente tramas *MiniMAP*.

Os *routers* são também muito utilizados para fazer a interligação entre redes idênticas, situadas em pontos geográficos distintos correspondentes a, por exemplo, duas fábricas de uma mesma empresa, utilizando para tal, por exemplo, a rede pública de dados.

Neste caso, o *router* utiliza a trama MAP como "dado da aplicação" (do ponto de vista da rede pública) e acrescenta-lhe os protocolos de rede necessários.

4.6.4. Gateways

A *gateway* interliga ao nível das camadas superiores, redes de protocolos diferentes. Por isso pode ser usada para interligar duas arquitecturas de comunicações completamente diferentes.

Em particular as *gateways* servem para interligar redes de comunicações proprietárias (quase todas as do tipo *Field Bus*) e redes baseadas no modelo de referência OSI (por exemplo o MAP).

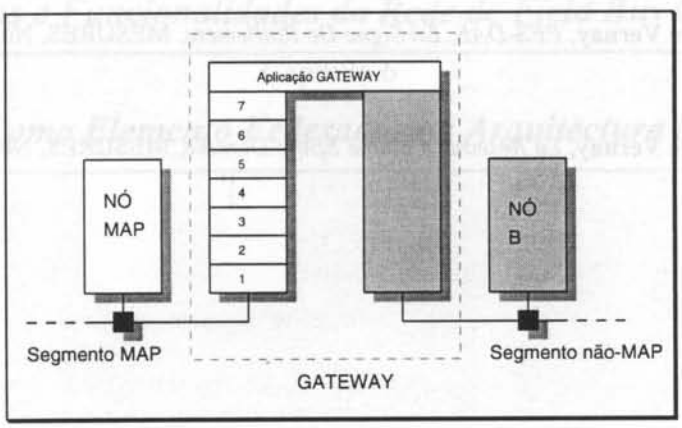


Figura 4.5 *- Contexto de Utilização de uma Gateway*

4.7. Referências e Bibliografia

[ALL - 86] **Bernie Allenstein**, *Gateway Transparency*, Systems International, Julho 1986, pp. 40-41.

[CAAM - 88] **Lilian Cassel**, **Paul Amner**, *Management of Distributed Measurement Over Interconnected Networks*, IEEE Network, Vol. 2, No. 2, Março 1988, pp. 50-56.

[CDF - 88] **Jeffrey Case**, **Janes Davin**, **Mark Fedor**, *Introducing to the Simple Gateway Monitoring Protocol*, IEEE Network, Vol. 2, No. 2, Março 1988, pp. 43-49.

[CHA - 88] **A. Lyman Chapin**, *Standards for Bridges and Gateways*, IEEE Network, Vol. 2, No. 1, Janeiro 1988, pp. 90-91.

[PHV - 88] **Radia Perleman, Arthur Harvey, George Varghese**, *Choosing the Appropriate ISO Layer for LAN Interconnection*, IEEE Network, Vol. 2, No. 1, Janeiro 1988, pp. 81-86.

[LCB - 90] **Peter Lagoni, Chritopher Crall, Thomas Bartz** *HP MAP3.0 Manufacturing Message Specification/800*, Hewlett-Packard Journal, Agosto 1990, pp. 31-39.

[JOH - 90] **William Johnson**, *An Overview of the HP OSI Express Card*, Hewlett-Packard Journal, Fevereiro 1990, pp. 6-8.

[PATA - 90] **Colin Park, Bruce Talley**, *HP Manufacturing Automation Protocol*, Hewlett-Packard Journal, Agosto 1990, pp. 6-10.

[RAN - 90] **Paul Ranky**, *Computer Networks for World Class Cim Systems, Cap. 5 - Gateways and Routers: OSI MAP/TOP Networking*, CIMware Limited, 1990.

[OPE - 93] **Jean-Pierre Vernay**, *L'Intégration Des Logiciels Avant Tout,* MESURES, No. 655, MAI 1993, pp. 75-77.

[DCM - 93] **Jean-Pierre Vernay**, *D'Abord Le Logiciel pour Le Manufacturier,* MESURES, No. 658, OCTOBRE 1993, pp. 77-79.

[PFS - 93] **Jean-Pierre Vernay**, *PFS-DAE, En Signe De Ralliement,* MESURES, No. 654, AVRIL 1993, pp. 70-72.

[BAS - 93] **Jean-Pierre Vernay**, *Le Réseaux Comme Épine Dorsale,* MESURES, No. 653, MARS 1993, pp. 44-46.

SEGUNDA PARTE

Das Arquitecturas Consideradas Para a Interligação

Capítulo 5

Características e Funcionalidades da Rede de Field Bus Considerada

---

Capítulo 6

O MMS Como Elemento Federador na Arquitectura MAP 3.0

Capítulo 5

Características e Funcionalidades da Rede de Field Bus Considerada

O FILBUS é, desde meados de 1992, uma proposta completa e extremamente funcional de *Field Bus*. O aliciante da consideração desta rede para o estudo da integração e interligação, reside no facto de se tratar de uma rede não muito sofisticada ao nível de arquitectura, em particular no que se refere à sua camada de aplicação. Apresenta-se, no entanto, como sendo representativa de muitas outras de índole proprietária.

5.1. Introdução

O FILBUS é uma proposta da empresa Suiça GESPAC, para o âmbito das redes de comunicação industrial do tipo *Field Bus*. Trata-se de uma rede de topologia em bus, com um método de acesso ao meio do tipo centralizado.

Em termos comerciais, o FILBUS ofereçe uma gama completa de módulos de entradas/saídas. Cada módulo contém todo o *software* e *hardware* necessário à implementação da *stack* de comunicações (camada física, de ligação de dados e camada de aplicação) bem como o necessário ao suporte de entradas e saídas propriamente ditas. Existe ainda outra componente de *software* que diz respeito àquilo a que se pode chamar de *software* da "camada do utilizador". Essas funções são algumas das tarefas do *Kernel* do FILBUS, relacionadas com a gestão da memória local e a gestão das entradas e saídas e dos parâmetros que regem a sua actuação, leitura, etc.

Algumas características importantes desta rede são a sua autonomia em termos de controlo, traduzida na possibilidade de acção reflexa entre módulos, a possibilidade de quase todos os aspectos da rede (parâmetros ao nível dos módulos ou mesmo reformulação da rede) poderem ser alterados com a rede em exploração, e por fim, o seu mecanismo de autoendereçamento, que evita qualquer tipo de mecanismo físico de endereçamento (p. ex., *dip-*

switches nos módulos). O método de endereçamento, exige no entanto linhas suplementares no barramento.

O FILBUS está optimizado para a transferência rápida de mensagens curtas. Uma rede FILBUS poderá ter até 250 módulos, podendo estes pertencer a um dos seguintes tipos: módulos de controlo, capazes de tomar o controlo do barramento quando em função de mestre, e módulos de entrada/saída, geralmente escravos na rede.

5.2. Sinais do Barramento

O protocolo no barramento de dados utiliza uma versão simplificada do SDLC (*Synchronous Data-Link Control*), um protocolo para o controlo da ligação lógica proposto pela IBM, ao contrário o HDLC (*High-Level Data Link Control*), que é uma norma internacional ISO. De notar que os módulos FILBUS são implementados em torno do µProcessador da INTEL *i*8044, e que este micro implementa uma versão simplificada do protocolo SDLC.

A estrutura das tramas que circulam na rede, é a que é apresentada na *Fig. 5.1*. O campo de endereçamento contém o endereço do módulo escravo envolvido na transferência, quer seja no envio quer seja na recepção de e a partir de um escravo. O campo de controlo indicará o tipo de trama SDLC utilizada. O campo de informação contém a mensagem FILBUS, que tem sempre a estrutura que é indicada na *Fig. 5.1*, se exceptuarmos a questão relacionada com o endereçamento extendido.

O campo de detecção de erro implementa um CRC (*Cyclic Random Check*) gerado a partir do polinómio $x^{16} + x^{12} + x^5 + 1$.

O código de linha utilizado é o NRZI (a transmissão de um zero modifica a saída) com *bitstuffing* de um '0' quando cinco '1s' são enviados, o que permite conjuntamente com o DPLL (*Digital Phase Locked Loop*) implementado pelo *i*8044 veicular a necessária informação de sincronismo.

Um outro par de linhas implementa uma conexão do tipo *daisy-chain*, utilizada meramente na inicialização da rede. Os módulos são interrogados em sequência física sendo a linha de *daisy-chain* habilitada pelo controlador. Depois de um módulo ser endereçado, o controlador pode enviar um comando para que esse módulo propague o *daisy-chain* para o que está fisicamente colocado a seguir.

Poderão existir alguns problemas neste endereçamento. No entanto, o mestre utiliza o tipo do módulo (existe uma mensagem para o efeito) e o mecanismo de propagação do *daisy-chain* para verificar se existe algum erro na ligação dos módulos. Este processamento pode não cobrir outro tipo de situações de erro.

Os sinais eléctricos do barramento de dados são de acordo com a norma RS-485, com uma velocidade de transmissão de 375 KBits/s.

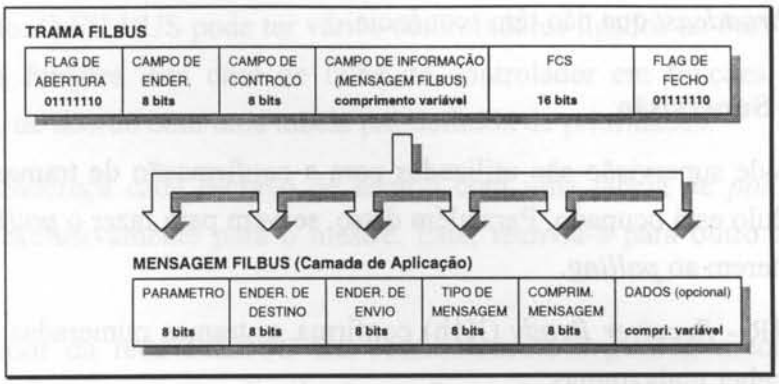


Figura 5.1 *- Estrutura da Trama FILBUS*

5.3. O Protocolo SDLC

Na trama que é apresentada na *Fig. 5.1*, pode ser omitido o campo de informação. A existir, o campo de informação terá no máximo 32 *bytes*. Por isso, no seu total, as tramas FILBUS têm entre 6 e 38 *bytes* de comprimento.

O SDLC prevê 3 tipos de tramas: tramas não numeradas, tramas de supervisão e tramas de informação.

5.3.1. Tramas Não Numeradas

No caso do FILBUS, as tramas não numeradas são essencialmente utilizadas para a sincronização dos escravos, aquando da conexão e/ou após uma troca não correcta de tramas.

A trama UP - *Unnumbered Poll* (3Fh), é enviada pelo mestre em cada segundo, implementando assim um *watchdog* da linha de barramento. Este *watchdog* é utilizado quando existem eventuais mestres redundantes.

A trama não numerada DISC - *Disconnect* (53h) é enviado pelo mestre para a re-sincronização de um escravo.

A trama UA - *Unnumbered Acknowledge* (73h) é utilizada como resposta de um escravo após um pedido de SNRM feito pelo mestre.

A trama FRMR - *Frame Reject* (87h) é enviada por um escravo, quando recebe uma trama não válida (por exemplo por causa do um Nr errado).

A trama SNRM - *Set Normal Response Mode* (93h) é enviada pelo mestre a um novo módulo.

Finalmente, a trama UI - *Unnumbered Information* (13h) é utilizada pelo mestre para enviar tramas de *broadcast* que não têm sequência.

5.3.2. Tramas de Supervisão

As tramas de supervisão são utilizadas para a confirmação de tramas recebidas ou para indicar que o módulo está ocupado. Para além disso, servem para fazer o *polling* de um escravo e para estes responderem ao *polling*.

A trama RR - *Receiver Ready* (X1h) confirma as tramas numeradas até à Nr-1 e indica que está apta a receber mais tramas.

A trama RNR - *Receiver Not Ready* (X5h) indica que não está apto a receber mais tramas devido, por exemplo, a ter o *buffer* de recepção cheio. Para além disso confirma as tramas recebidas até à Nr-1.

CAMPO DE CONTROLO

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0		
*	*	*	*	*	*	1	1	**Tramas Não Numeradas**	
1	0	0	1	0	0	1	1	Set Normal Response Mode (SNRM)	**Comandos**
0	1	0	1	0	0	1	1	Disconnect (DISC)	
0	0	1	1	1	1	1	1	Unnumbered Poll (UP)	
0	0	0	1	0	0	1	1	Unnumbered Info (UI)	
0	1	1	1	0	0	1	1	Unnumbered Acknowledge (UA)	**Respostas**
1	0	0	0	0	1	1	1	Frame Reject (FRMR)	
*	*	*	*	*	*	0	1	**Tramas de Supervisão**	
Nr2	Nr1	Nr0	p/f	0	0	0	1	Receiver Ready (RR)	
Nr2	Nr1	Nr0	p/f	0	1	0	1	Receiver Not Ready (RNR)	
*	*	*	*	*	*	*	0	**Tramas de Informação**	
Nr2	Nr1	Nr0	p/f	Ns2	Ns1	Ns0	0		

Figura 5.2 *- Campo de Controlo da Trama SDLC*

5.3.3. Tramas de Informação

As tramas de informação são utilizadas para a transferência de dados. O campo de controlo contém o número de tramas enviadas e o número de tramas recebidas. O *bit* de P/F (*Poll/Final*) é enviado pelo mestre a um escravo, indicando a este que pode enviar. O *Final bit* é enviado por um escravo ao mestre para indicar que se trata ou não da última trama. Se é a última, vai a '0'. Em mensagens curtas vai a '0' em mensagens longas vai a '1' até à última trama.

5.4. O mecanismo de Polling da Rede

A rede FILBUS é, do ponto de vista do controlo de acesso ao meio, uma rede do tipo *polling* centralizado. O FILBUS pode ter vários controladores ligados ao *bus*. No entanto, só um deles exerce essas funções. Em caso de falha do controlador em funções, outro controlador assume esse papel, de acordo com uma tabela pré-definida de prioridades.

O mestre endereça cada escravo de acordo com uma tabela de *polling*. Cada escravo envia mensagens exclusivamente para o mestre. Este, reenvia-a para outro escravo, ou para o *host*.

O controlador da rede FILBUS faz continuamente o *polling* a toda a rede, com o objectivo de manter um mapa actualizado da configuração da rede e assegurar comunicações. Este *polling* pode ser de dois tipos: síncrono ou assíncrono.

O ciclo síncrono é um múltiplo do valor de 10ms. O ciclo pode ser de 10, 20, 30, 40, etc (ms). O *polling* a cada módulo leva alguns milisegundos, dependendo da classe, conforme especificado em [HFSM - 92]. O escalonamento dos módulos nas tabelas de *polling* bem como o período para o *polling* síncrono têm de ser equacionados em função da quantidade de módulos e da aplicação em causa.

5.5. Utilização da Rede de Field Bus

Genericamente uma rede pode ser constituída por diferentes módulos. A rede considerada no estudo tem um só controlador, que acumula com as funções de mestre. Esse controlador é específico de plataformas PC, sendo instalada no barramento do PC. O pacote de *software* considerado é para *MS-DOS*.

5.5.1. Aplicações do Pacote de Software

Existem cinco utilitários disponíveis. São eles o CFG.EXE, o SETPARM.EXE, o FILRESET.EXE, o FDBM.EXE e o DOWNLOAD.EXE.

O utilitário CFG.EXE (aplicação de configuração de rede), permite ao utilizador definir a(s) rede(s) e as operações nos diferentes módulos. A aplicação CFG.EXE cria uma base de dados para armazenar toda a informação relacionada com a configuração do sistema. Esta base de dados caracteriza-se por uma organização hierarquica em cinco níveis.

O nível 1 diz respeito à lista de sistemas FILBUS. A informação está contida no ficheiro FILBUS.DOC e poderá conter até 19 sistemas. O nível 2 especifica a lista de redes de cada

sistema declarado no nível anterior. Os ficheiros são do tipo <nome_sys>.sys. Podem existir até 255 redes por sistema. O nível 3, especifica os módulos de cada rede. Essa informação reside em ficheiros do tipo <nome_rede>.net e pode haver 250 módulos por rede. Para cada módulo existe um ficheiro <nome_modu>.md onde está a lista de E/S de cada módulo. Trata-se do nível 4, e o número de E/S é evidentemente dependente do tipo de módulo. Por fim, o nível 5 diz respeito aos parâmetros de operação para cada E/S de cada módulo. A constituição destes parâmetros é dependente do tipo de módulo. Estão em ficheiros do tipo <nome_es>.io. A *Fig. 5.3* ilustra a estrutura desta base de dados. A estrutura de cada um dos tipos de ficheiros está detalhada em [SSMS - 92].

A aplicação FILRESET.EXE reinicializa a rede a partir da informação dos parâmetros e endereços a partir da base de dados.

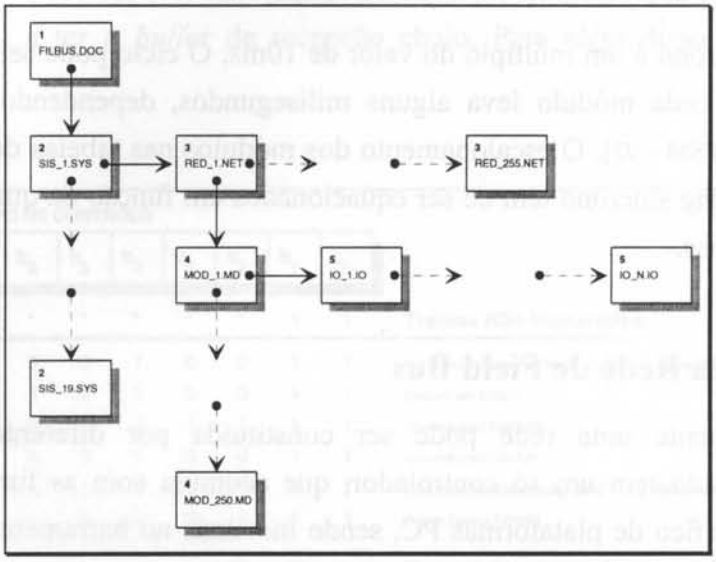


***Figura 5.3** - Base de Dados Criada e Utilizada Pelo Utilitário CFG.EXE*

A aplicação SETPARAM.EXE actualiza ou modifica todos os parâmetros de todas as E/S de um módulo específico.

A aplicação FDBM.EXE, consiste numa interface interactiva que permite ver as mensagens trocadas entre o controlador e o computador. Permite ao utilizador enviar mensagens (comandos FILBUS).

A aplicação DOWNLOAD.EXE, permite ao utilizador programar as *Eproms* do tipo *Flash* dos diferentes módulos. Os dados colocados nestas memórias não serão apagados aquando dos *resets* ou remoção dos módulos. É possível fazer o *download* de parâmetros dos módulos

(ficheiros .md e .io), de programas do utilizador, valores de defeito para as E/S e de *firmware* para os módulos, o que é particularmente útil para fazer actualizações das versões.

A aplicação FDBM.EXE fornece serviços de inicialização de rede e permite suportar dinamicamente a conecção e disconexão de módulos e reinicialização e *reset* de módulos. Os serviços utilizados para o efeito estão em FILMON.OBJ, que pode ser incluido no *linking* a qualquer *software* desenvolvido pelo utilizador.

5.6. Quadros Resumo das Mensagens de Aplicação

No FILBUS circulam três tipos de mensagens de aplicação. São eles, as mensagens gerais da rede, as mensagens particulares dos módulos e as mensagens de entrada/saída de dados.

Nos quadros seguintes são apresentados os serviços FILBUS, com uma breve descrição. Neles, as siglas H representa o HOST, C o controlador, M um mestre e IO um módulo de entradas/saídas. De notar que normalmente o mestre coincide com o controlador.

5.6.1. Mensagens Gerais da Rede

As mensagens gerais da rede correspondem aos códigos 00h a 3Fh.

Mensagem	Nome	Despoletar	Descrição	Circulação
DEF (04h)	Default	Evento	Assinala a conecção ou desconecção de um módulo. Informação recolhida pelo controlador ao fazer o polling ao módulo	C → H
NER (05h)	NetworkError	Evento	Assinala que ocorreram mais de 256 erros em pelo menos um módulo. Envia 256 bytes, com o número de erros para cada módulo.	C → H
DER (06h)	DemandError	Comando	Pergunta quantos erros ocorreram na rede	H → M
DMA (07h)	DemandMaster	Comando	Pergunta quem é o mestre	H → C
DRE (08h)	DemandNetwork	Comando	Pergunta que módulos estão ligados à rede	H → M
DFX (09h)	DemandFlow	Comando	Pergunta o estado dos buffer dos módulos de entrada/saída	H → M
MSS (14h)	MasterStartStop	Evento	O mestre notifica ao host que se ligou ou desligou.	M → H
AER (16h)	PassError	Resposta a DER	Resposta de formato e conteúdo idêntico ao da mensagem NER	M → H
MAS (17h)	Master	Resposta a DMA	Responde com o endereço do mestre, o status da rede e o endereço da rede.	C → H
RES (18h)	Network	Resposta a DRE	Responde com 250 bytes, em que cada byte indica o estado do módulo (não ligado, modo síncrono, assíncrono, não está na tabela, modo download).	M → H
RFX (19h)	Flow	Resposta a DFX	Responde com o status de todos os módulos (250) da rede. O status é 0 (não aceita mais mensagens - buffer cheio) ou 1 (aceita mensagens)	M → H

Tabela 5.1 *- Tabela de Mensagens Gerais do FILBUS*

5.6.2. Mensagens Particulares dos Módulos

As mensagens particulares dos módulos correspondem aos códigos 40h até 7Fh.

Mensagem	Nome	Despoletar	Descrição	Circulação
RUN (42h)	Start	Comando	Pede a um módulo para entrar em modo run e autoriza a propagação de daisy chain.	H → IO
STP (43h)	Stop	Comando	Pede a um módulo para passar ao estado de stop e para bloquear a propagação do daisy chain.	H → IO
CLR (45h)	Clear	Comando	Reset do módulo, apaga todos os parâmetros e o endereço do módulo, a menos que estes estejam na EPROM de Flash. Desabilita a propagação do daisy chain.	H → IO
FLG (46h)	Flag	Evento	Indica ao host e a todos os módulos que o seu buffer está cheio ou vazio. Pode ser utilizado no controlo de fluxo de mensagens.	M → ALL
EVO (47h)	StartEvolution	Command	Pede ao módulo para enviar a evolução de uma saída/entrada sempre que acontece uma mudança de estado ou para enviar contagens.	H → IO
SEV (48h)	StopEvolution	Comando	Pede ao módulo que pare a função de evolução.	H → IO
AFX (49h)	StopFlow	Comando	Pede ao módulo para não transmitir o buffer status (mensagem FLG). Isto não se aplica entre o host e o controlador.	H → IO
DYP (4Ah)	DemandType	Comando	Pede o tipo do módulo.	H → IO
LOP (4Ch)	SoftwareSync.	Comando	Sincronização de mensagens entre o host e o módulo. O módulo responde com o mesmo comando para indicar que não há outras mensagens na fila do módulo.	H ↔ IO
MAH (52h)	SetTime	Comando	Fixa a data e a hora.	H → IO
TYP (5Ah)	Type	Resposta a DYP	Retorna o tipo ou a versão.	IO → H
DAL (60h)	DemandAlarm	Comando	Pede a um módulo de entradas digitais para ler as suas entradas (todas ou só uma específica).	H → IO
DEA (61h)	DemandStatus	Comando	Pede a um módulo de entradas digitais para ler as suas entradas (todas ou só uma específica).	H → IO
DCP (62h)	DemandCounter	Comando	Pede a um módulo de entradas para lêr o status dos timers, contadores ou entradas analógias de todos ou de um só canal.	H → IO
DRL (65h)	MemoValues	Comando	Pede a um módulo de entrada para enviar os valores dos contadores ou valores analógicos previamente colocados no buffer de saída.	H → IO
DRY (66h)	DemandOutput	Comando	Pede a um módulo de saída para enviar o status de uma ou de todas as saídas.	H → IO
RCL (75h)	RemoteControl	Comando	Mudança do estado saídas de um módulo de saidas digitais.	H/IO → IO

Tabela 5.2 *- Tabela de Mensagens Particulares do Módulos*

5.6.3. Mensagens de Entrada e Saída de Dados

As mensagens de entrada e saída de dados correspondem aos códigos 80h até FFh.

Mensagem	Nome	Despoletar	Descrição	Circulação
DAT (80h)	Data	Comando	Envia dados para os módulos série.	H/IO → IO
PAR (81h)	Parameter	Comando	Transfere parâmetros para qualquer tipo de módulos.	H/IO → IO
ACK (91h)	Acknowledge	Resposta a PAR.		IO → H
WCI (A0h)	WritePulse	Comando	Envia valor para um contador de impulsos num módulo de entrada.	H → IO
WCT(A1h)	WriteTimeCntr	Comando	A mesma coisa mas para um countador de tempo.	H → IO
ALM (B0h)	Alarm	Evento	Notifica o host de que aconteceu uma condição de curto circuito numa saída ou um alarme numa entrada.	IO → H
STA (B1h)	Status	Evento	Notificação de uma mudança de nível numa entrada.	IO → H
CPI (B2h)	Pulse counter	Evento	Retorna o valor do contador de impulsos para uma dada entrada numa situação de leitura automática a partir de uma data ou após recepção do comando DRN.	IO → H
CPT (B3h)	Timer counter	Evento	A mesma coisa que o anterior.	IO → H
ANA (B4h)	AnalogValue	Evento	Retorna o valor analógico de uma dada entrada.	IO → H
RAN (B6h)	RelayOn	Comando	Põe a saída digital a valor lógico '1'.	H → IO
ORR (B7h)	RelayOff	Comando	Põe a saída digital a valor lógico '0'.	H → IO
RXO (B8h)	RelayFlipFlop	Comando	Complementa uma determinada saída digital.	H → IO
DIG (B9h)	DigitalIO	Evento	Resposta a evolution, sempre que uma saída é alterada.	IO → H
LAN (BAh)	AnalogLevel	Comando	Envia nível analógico de saída.	H → IO
VAN (BBh)	AnalogValue	Comando	Envia valor analógico de saída.	H → IO
PAL (C0h)	ReturnAlarm	Resposta a DAL	Resposta ao comando DAL.	IO → H
PST (C1h)	ReturnStatus	Resposta a DEA	Resposta ao comando DEA.	IO → H
PCI (C2h)	ReturnPulseCntr	Resposta a DCP	Resposta ao comando DCP ou Enviado porque o módulo de entradas está programado para enviar evolution reports.	IO → H
PCT (C3h)	ReturnTimerCntr	Resposta a DCP	Resposta ao comando DCP ou Enviado porque o módulo de entradas está programado para enviar evolution reports.	IO → H
PAM (C4h)	ReturnAnalog	Resposta a DCP		IO → H
PRY (C5h)	ReturnOutput	Resposta a DRY		IO → H
RPA (CFh)	ReturnParam	Resposta a DPA		IO → H
DRN (D4h)	StartStatement	Comando	Pede a uma entrada tipo contador ou a uma entrada analógica para iniciar um leitura automáticas.	H → IO
DPA (DFh)	DemandParam	Comando	Pede a leitura dos parametros do modulo.	H → IO

***Tabela 5.3** - Tabela de Mensagens de Entrada e Saída de Dados*

5.7. Estrutura de Algumas das Mensagens de Aplicação

Estrutura das mensagens:

mm = B0, B1, C0, C1

PAR	DA	SA	mm	0Ch	NUM	QUA	VAL	Y	M	D	WD	H	M	S	1/100	250µ

mm = B2, C2

PAR	DA	SA	mm	0Eh	NUM	QUA	MSB	...	...	LSB	Y	M	D	WD	H	M	S	1/100

mm = B3, C3

PAR	DA	SA	mm	19h	NUM	QUA	VAL	Ds M	Ds L	Hs	Ms	Ss	1/100	1/250	Ds M	Ds L	Hs	Ms	Ss
1/100	1/250	Y	M	D	WD	H	M	S	1/100										

5.8. Funcionalidade dos Módulos de Entrada e Saída Digitais

Aqui serão analizadas em maior detalhe as funcionalidades supridas pelos módulos de E/S digitais.

Os módulos de entrada e saída de dados do tipo digital, poderão ser só de saídas, só de entradas ou mistos. A única grande diferença reside no facto de as mensagens reflexas nos módulos mistos (designados pela GESPAC por FILCIO-xxxx) não circularem na rede quando dirigidas a saídas dos próprios módulos. Faz-se a seguir uma descrição das funcionalidades previstas para os módulos de entradas digitais, neste caso concreto o FILINP-1648 e os módulos de saídas digitais, em particular o FILOUT-1648.

5.8.1. O Módulo de Entradas Digitais FILINP-1648

O módulo tem 16 entradas que podem ser programadas de forma independente. Cada entrada pode ser de um dos cinco tipos seguintes: não usada, do tipo alarme, do tipo estado, do tipo contador de impulsos e por fim do tipo contador de tempo.

Cada entrada tem associada uma tabela de parâmetros que define as suas funções, incluindo a programação de períodos de actividade, definição dos estados, notificação periódica das contagens, programação de mensagens reflexas, entre outras.

Os parâmetros possíveis de definir são os que se indicam na *Tab. 5.6*. Para cada entrada existe no módulo uma tabela idêntica. Existe ainda uma tabela com os parâmetros gerais do módulo. Essa tabela corresponde à entrada '0'. O *byte* '0' dessa tabela corresponde ao endereço do módulo e o byte '1' ao endereço de rede.

Posição na tabela	Definição
0	Parâmetros gerais das entradas
1	Validação de alarmes múltiplos
2	Janela de amostragem (MSB)
3	Janela de amostragem (LSB)
4-59	Periodos de actividade
60-66	Data de início de leituras automáticas
67-68	Intervalo entre leituras
69-72	Reservado
73	Numero de leituras automáticas
74-75	Contagem máxima para nível '0'
76-77	Contagem mínima para nível '1'
78-79	Contagem máxima para nível '1'
80-81	Contagem mínima para nível '2'
82-83	Contagem máxima para nível '2'
84-85	Contagem mínima para nível '3'
86-89	Valor limite de contagem
90-92	Reservado
93-96	Valor limite de contagem
97-115	Reservado
116-128	Equações boleanas
129-132	Intervalos entre relatórios de evolução
133-159	Reservado
160-255	Mensagens

***Tabela 5.4** - Parâmetros de Cada Entrada de um Módulo FILINP-1648*

A figura *Fig. 5.5* ilustra a codificação dada ao parâmetro geral de cada entrada. O parâmetro de validação de múltiplos alarmes especifica quando é que o módulo envia mensagens nas transições entre os diferentes níveis (0-1, 1-2, 2-3, 1-0, 2-1, 3-2). Especifica também se,

quando uma entrada volta a um estado inactivo, se deve ou não transmitir. A janela de amostragem corresponde a um valor em décimos de segundo. Vinte e oito períodos de actividade podem ser programados para cada entrada do módulo. O início de cada período de actividade/inactividade é codificado em dois *bytes*, como se indica na *Fig. 5.5*.

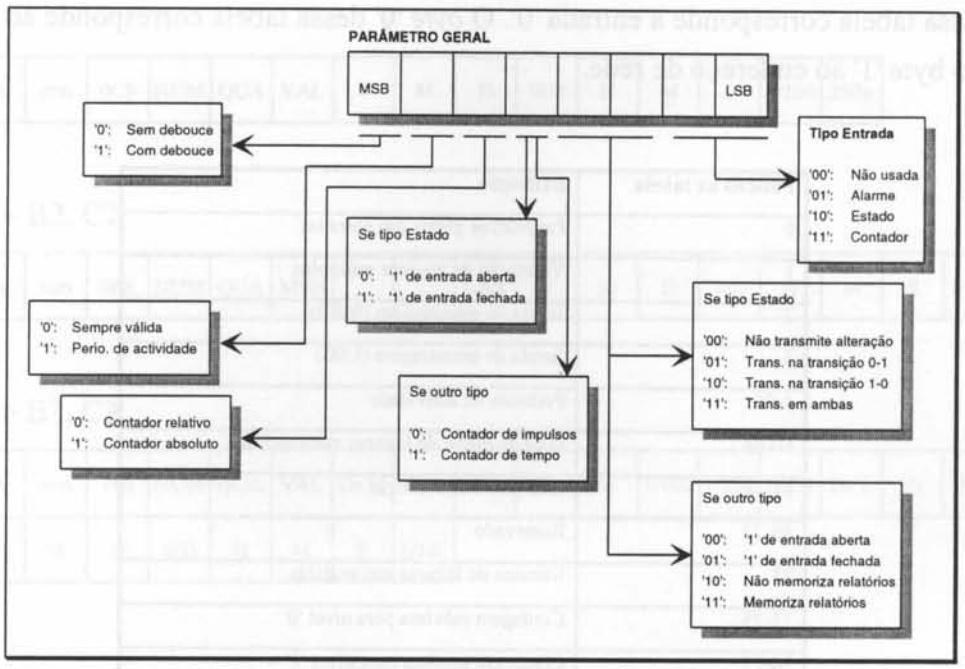


Figura 5.4 *- Codificação do Parâmetro Geral da Entrada*

Para cada entrada do módulo que esteja a funcionar como contador (de impulsos ou de tempo) pode ser programada uma data de início de leituras automáticas. Nesse caso o módulo envia as mensagens *CPI_B2* ou *CPT_B3*. O número de leituras automáticas que é feito corresponde ao parâmetro que ocupa o *byte* 73. O intervalo entre leituras é dado em segundos, a parte mais significativa em 67 e a menos significativa em 68. A data de início de leituras automáticas obedece ao seguinte formato, respectivamente dos *bytes* 60 a 66: ano (00h = 1900 ... 5Fh = 1995 ...), mês (01h = janeiro ... 0Ch = dezembro), dia (01h ... 1Fh), dia da semana (00h = domingo ... 06h = sábado), hora (00h ... 17h), minuto (00h ... 3Bh) e finalmente segundo (00h ... 3Bh). De notar que estas leituras automáticas, à semelhança da leitura não automática (comando *DCP_62*), podem ou não fazer o *reset* do contador, de acordo com o *bit* 5 do parâmetro geral. Existem comandos para inicializar os valores dos contadores (*WCI_A0* e *WCT_C1*). Ainda relativamente às leituras automáticas, é de referir que o comando *DRN_D4* faz o mesmo efeito, mas sem ter de esperar pela data programada. Por fim, o *bit* 3 do parâmetro geral, indica se as leituras devem ou não ser guardadas no módulo, em vez de serem transmitidas

sempre que se faz a leitura. Se as memorizar localmente, o mestre pode enviar um *DRL_65* para serem feitas as leituras.

Com a entrada a funcionar em modo de alarme, sempre que este acontece é enviada ao *host* uma mensagem informacional *ALM_B0*, com o qualificador a '0', bem como as outras mensagens pré-programadas. Neste caso a mensagem de informação enviada no fim é *STA_B1*.

Com a entrada do tipo estado, as mensagens de informação podem não ser enviadas, ser enviadas na transição para estado '1', para estado '0' ou nas duas transições. Quando são enviadas mensagens trata-se da mensagem informacional *STA_B1* e eventuais mensagens pré-definidas, para actividade reflexa.

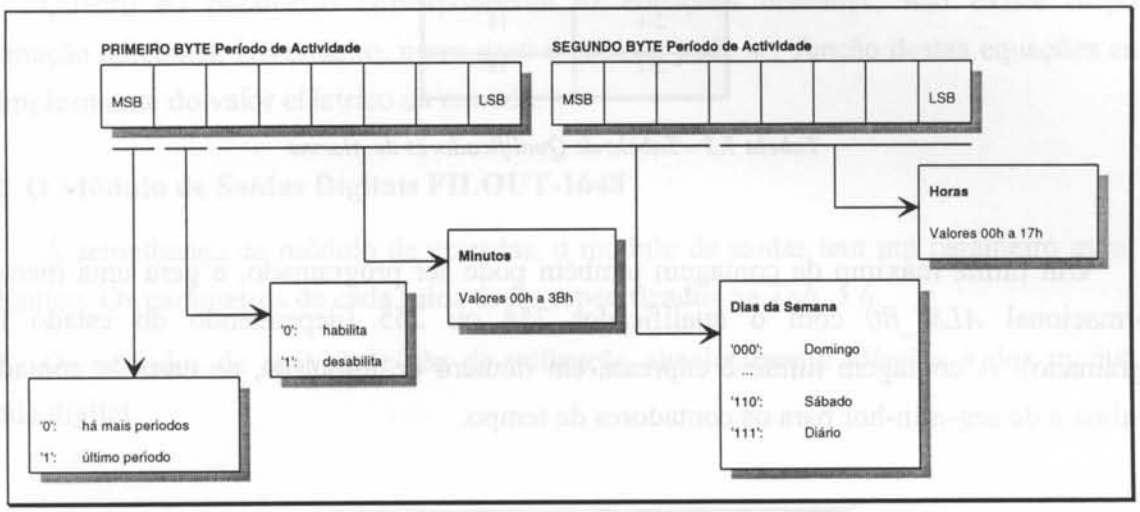


Figura 5.5 *- Codificação dos Períodos de Actividade*

A diferença entre os dois tipos resume-se ao tipo de mensagem informacional que é enviada. Programada como alarme, *ALM_B0* é enviada, enquanto na configuração como estado é enviada *STA_B1*. O estado pode no entanto ser lido por uma mensagem *DIG_B9*, se foi enviado ao módulo um comando *EVO_47*. Este *DIG_B9* só é recebido quando uma entrada a funcionar em modo estado for alterada. As leituras podem ainda ser feitas recorrendo aos comandos *DAL_60 DEA_61* e *DPC_62*.

É indiferente que seja enviado um *DAL_60* ou um *DEA_61*. O escravo, no entanto, responde com um *PAL_C0* ou um *PST_C1* consoante o que está programado. Quer o *DAL_60* quer o *DEA_61* têm a possibilidade de pedir a leitura de todas as entradas. Para cada entrada que funcione como estado ou alarme vem uma resposta correspondente.

Com entradas a funcionar como contador de impulsos, é possível programar 3 níveis de contagem e a respectiva histerése, conforme a informação de *Tab. 5.6*. O primeiro *byte* de cada

valor de contagem é o mais significativo. O processamento de alarmes é feito da mesma forma que para o modo alarme. Nesses casos, se as mensagens de informação estão programadas para serem enviadas ao *host* são enviados *ALM_B0*, em que o qualificador indica que tipo de ocorrência foi. São contadas o número passagens do estado lógico '0' para o estado lógico '1' que ocorrem por segundo.

Nível	Qualifier
0-1	1
1-2	2
2-3	3
1-0	17
2-1	18
3-2	19

Tabela 5.5 *- Tabela de Qualificadores de Alarme*

Um limite máximo de contagem também pode ser programado, e gera uma mensagem informacional *ALM_B0* com o qualificador 254 ou 255 (dependendo do estado lógico programado). A contagem limite é expressa em número de impulsos, no caso de contador de impulsos e de seg-min-hor para os contadores de tempo.

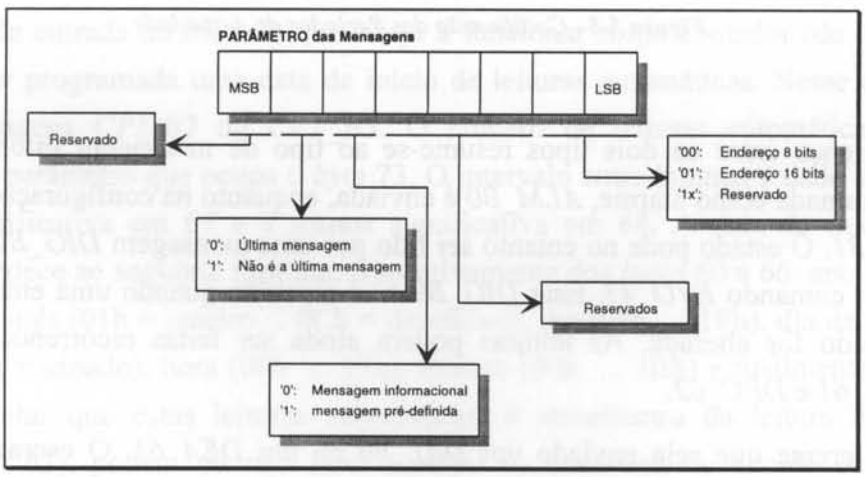


Figura 5.6 *- Codificação do Parâmetro das Mensagens Pré-definidas*

No caso das entradas que funcionam como contadores, e no caso do mestre ter enviado o comando *EVO_47*, as entradas que funcionam como contadores, enviam leituras (*PCI_C2* ou *PCT_C3*) de acordo com o parâmetro 'intervalo entre relatórios de evolução' de *Tab. 5.6*. Os dois

primeiros *bytes* deste parâmetro definem contagens entre leituras (só utilizada se a entrada for do tipo contador de impulsos). Os dois últimos *bytes* correspondem a segundos entre leituras. Se estes valores forem '0' não são feitas leituras, mesmo que um módulo tenha sido programado com *EVO_47*.

Quanto às mensagens reflexas, as mensagens que são função do modo de funcionamento e da situação (mensagens informacionais), não precisam de ser colocadas na tabela. Só é necessário fazer isso para as mensagens informacionais que se pretendem enviar para outros módulos que não o *host*. Abstraindo o endereçamento estendido, as mensagens informacionais têm dois bytes: um parâmetro e o destino. As outras mensagens são idênticas às mensagens FILBUS. A estrutura do parâmetro é a que se apresenta na *Fig. 5.6*.

Quanto ao parâmetro correspondente às equações boleanas, não existe disponível informação adicional. No entanto, nesse caso a entrada pode ser função destas equações em vez de simplesmente do valor eléctrico da entrada.

5.8.2. O Módulo de Saídas Digitais FILOUT-1648

À semelhança do módulo de entradas, o módulo de saídas tem um parâmetro geral (E/S 0) idêntico. Os parâmetros de cada saída estão especificados na *Tab. 5.6*.

Os períodos de actividade são de utilização absolutamente idêntica à dos módulos de entrada digital.

Posição na tabela	Definição
0-3	Reservado
4-59	Periodos de actividade
60-91	Reservado
92	Tempo de T1 MSB
93	Tempo de T1 LSB
94	Tempo de T2 MSB
95	Tempo de T2 LSB
96	Tempo de T3 MSB
97	Tempo de T3 LSB
98	Tempo de T4 MSB
99	Tempo de T4 LSB
100	Tempo de T5 MSB
101	Tempo de T5 LSB
102-255	Reservado

***Tabela 5.6** - Parâmetros de Cada Saída de um Módulo FILOUT-1648*

Todos os valores de tempo são em décimos do segundo. O valor de T1 corresponde ao tempo que medeia entre a chegada de um comando para activar a saída e a execução dessa ordem. O valor de T2 e de T3 correspondem ao tempo respectivamente em que a saída está activa e a saída está inactiva. T2 deve ser colocado a 65535 e T3 a 0, para obter uma saída biestável. O valor de T4 corresponde ao tempo que medeia entre uma ordem para desactivar a saída e a execução dessa ordem. Por fim, o valor de T5, determina o tempo durante a qual a saída está activa. O tempo será ilimitado se T5 vale '0'.

5.9. Referências e Bibliografia

[FRT -] *FILBUS - Le Reseau de Terrain*, GESPAC International SA FILBUS Product Description.

[GFB - 88] *FILBUS -The GESPAC's Field Bus*, GESPAC International SA, Junho 1988.

[CPUM - 92] *FILBUS Configuration Program User's Manual V2.3*, GESPAC International SA, Abril 1992.

[SSMS - 92] *FILBUS System Software for MS-DOS Systems*, V2.3, GESPAC International SA, Abril 1992.

[HFSM - 92] *FILBUS Hardware and Firmware Specification Manual V1.1*, GESPAC International SA, Abril 1992.

[INDS - 94] *FILINP-16xx Data Sheet*, GESPAC International SA, 1994.

[OUDS - 94] *FILOUT-16xx Data Sheet*, GESPAC International SA, 1994.

[CDS - 94] *FILCIO-16xx Data Sheet*, GESPAC International SA, 1994.

Capítulo 6

O MMS Como Elemento Federador na Arquitectura MAP 3.0

O MMS (*Manufacturing Message Specification*) é desde 1990 uma norma internacional que especifica um poderoso elemento de camada de aplicação, especialmente vocacionado para o ambiente industrial. Como o seu próprio nome indica, o MMS constitui-se num protocolo para a troca de mensagens. O MMS destina-se a facilitar a comunicação entre equipamentos de fabrico programáveis através do fornecimento de serviços normalizados. Apesar de à primeira vista o conjunto total de serviços MMS parecer despropositadamente grande, a realidade é que os diferentes modelos previstos no MMS são especialmente adequados aos diferentes ambientes de fabrico.

6.1. Alguns Elementos do Modelo de Referência OSI

A norma ISO 7498 [ISO/A-84], que descreve o modelo de referência OSI, está dividida em duas partes. A primeira parte descreve os elementos constituintes da arquitectura usados na construção do modelo de 7 camadas. Na segunda parte, são descritos os serviços e as funções de cada uma das camadas.

No ambiente OSI, as comunicações têm lugar entre Processos de Aplicação que correm em sistemas distintos.

Uma camada - *(N)-Layer* - é vista como sendo composta por entidades. Entidades na mesma camada são entidades homólogas e são designadas por *(N)-Entities*.

Cada camada fornece serviços à camada superior, os *(N)-Services*. Um serviço é uma capacidade da *(N)-Layer* e é fornecido às *(N+1)-Entities*. Os *(N)-Services* são fornecidos às *(N+1)-Entities* através de pontos de acesso a serviços, os *(N)-SAP*. Os *(N)-SAP* representam interfaces lógicas entre entidades em camadas adjacentes. Um *(N)-SAP* só pode servir uma entidade e ser usado por uma só entidade. No entanto, uma entidade pode servir vários *(N)-SAP* e uma entidade pode usar vários *(N)-SAP*.

6.1.1. Nomeação e Endereçamento em Ambiente OSI

O modelo de referência OSI, define identificadores para entidades, SAPs e conexões, por forma a que sejam unicamente identificados no ambiente.

Cada *(N)-Entity* é identificada por um título global, o *(N)-Title*, que é único e a identifica em qualquer lugar no sistema aberto.

Cada *(N)-SAP* é identificado por um endereço, o *(N)-Address*, que o identifica unicamente na fronteira entre uma *(N)-Layer* e uma *(N+1)-Layer*.

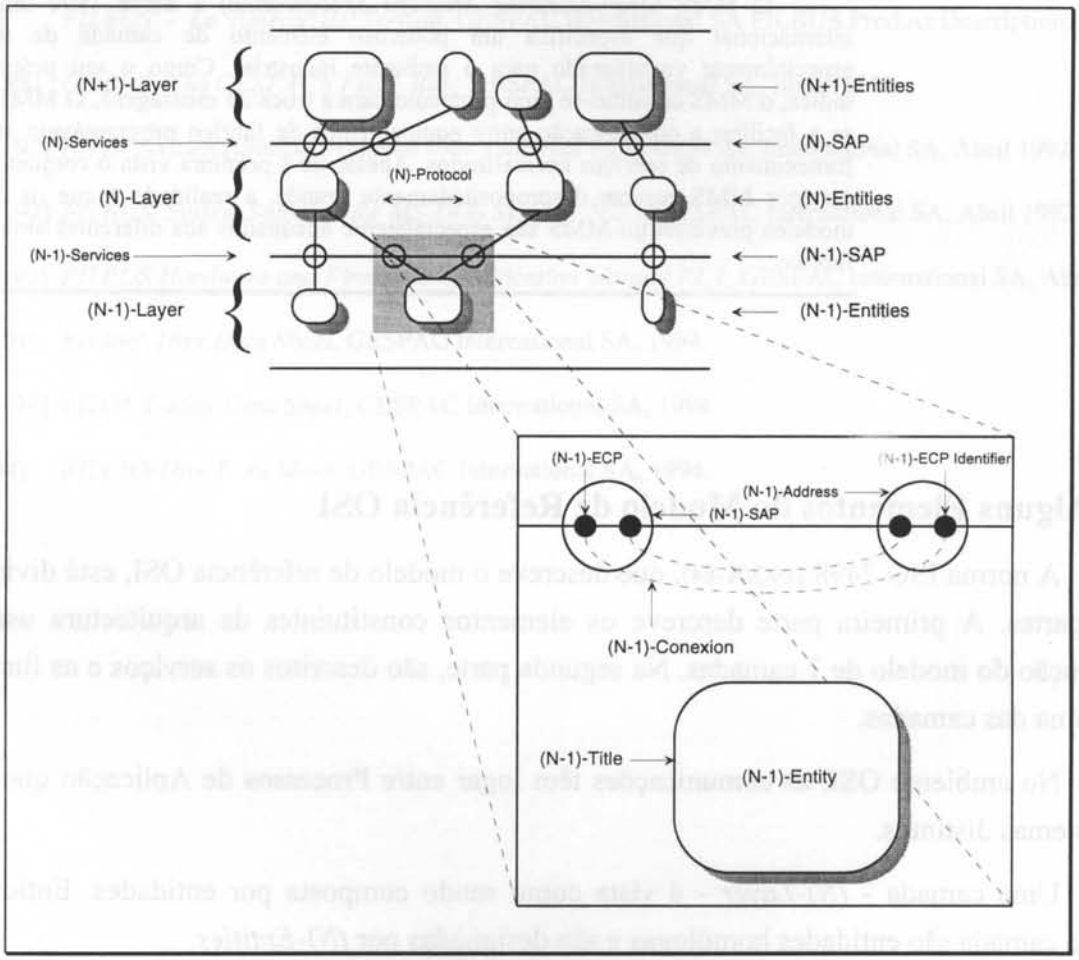


Figura 6.1 *- Elementos Constituintes da Arquitectura OSI*

Os mapeamentos entre as *(N)-Entities* e os *(N-1)-SAP* estão definidos num *(N)-Directory*, que indica as correspondências entre os títulos globais das *(N)-Entities* e os *(N)-Address* através dos quais elas podem ser acedidas.

6.1.2. Conexões

Um serviço comum a todas as camadas consiste em prover uma associação entre *(N)-SAPs* homólogos. O terminus de uma conexão designa-se por ponto de fim de conexão, *(N)-ECP* (*End Connection Point*). Diversas conexões poderão existir entre o mesmo par de *(N)-SAP*. Cada *(N)-ECP* é identificado unicamente num *(N)-SAP* por um *(N)-ECP Identifier*.

6.2. A Estrutura da Camada de Aplicação

Operações cooperativas em sistemas abertos reais são conseguidas através de interacções entre processos de aplicação (*AP - Application Process*). *APs* são modelos abstractos dos elementos que num sistema real executam realmente actividades de processamento de informação. Invocações de um *AP*, *AP-Invocations*, são subconjuntos das capacidades de um processo de aplicação e são responsáveis pela interacção entre diferentes *APs*. Num momento particular, um *AP* pode ser constituido por diversos *AP-Invocations*.

As entidades de aplicação (*AE - Application Entity*) representam conjuntos particulares de capacidades de comunicação OSI de um *AP*. Um *AP* pode ser constituido por um conjunto de *AEs*, sendo no entanto cada um deles de um só tipo. Uma invocação de um AE, *AE-Invocation*, é uma utilização específica de parte ou da totalidade de uma *AE* para o suporte das comunicações.

ASEs (*Application Service Elements*) são conjuntos de funções que fornecem capacidades de comunicação para a interoperação de *AE-Invocations*. Uma *AE*, pode ser constituida por uma ou mais *ASEs* de tipos diferentes.

Uma associação da aplicação (*AA - Application Association*) é uma relação cooperativa entre duas *AE-Invocations*. Quando é preciso uma comunicação entre duas *AEs*, é necessária a existência de um ou mais *AAs* (que podem existir simultaneamente, sequencialmente ou em ambas as formas). Um *AA-Identifier* poderá existir e será único no âmbito das duas *AEs* envolvidas.

As *AAs* devem veicular um conjunto de regras comuns a ambas as *AE-Invocations*. Trata-se do contexto da aplicação (*AC - Aplication Context*). O ACSE, *Application Context Service Element*, contem as capacidades específicas para a gestão das *AAs*. Cada *AA* está associada a um *(P)-SAP*. Cada *AE* pode estar ligada a mais de um *(P)-SAP*.

Quando pretende uma associação, uma *AE-Invocation* especifica ao serviço de apresentação a localização de uma AE homóloga, através do seu endereço de apresentação. Eventualmente poderá usar ainda um ou mais dos seguintes identificadores para a identificação

da *AE-invocation* homóloga: *AE-title* e *AP-titles*. Neste caso, esta informação identifica a *AE* independentemente do endereço de apresentação.

Uma AE está ligada a um ou mais *P-SAP*, para que seja endereçável no modelo OSI. Uma *AA* está ligada a uma só *(P)-Connection* ao contrário do resulta do conceito de *(N)-Association*, não abrangendo no entanto a camada de aplicação.

A norma ISO 7498-3 [ISO/C-89] contem as regras para o mapeamento das diferentes categorias de informação. Estas funções são responsabilidade da *AE-Invocation*. A informação sobre estes mapeamentos pode ser guardada localmente ou estar disponível remotamente, sendo de qualquer das maneiras responsabilidade do sistema local ir buscar essa informação (utilizando protocolos OSI).

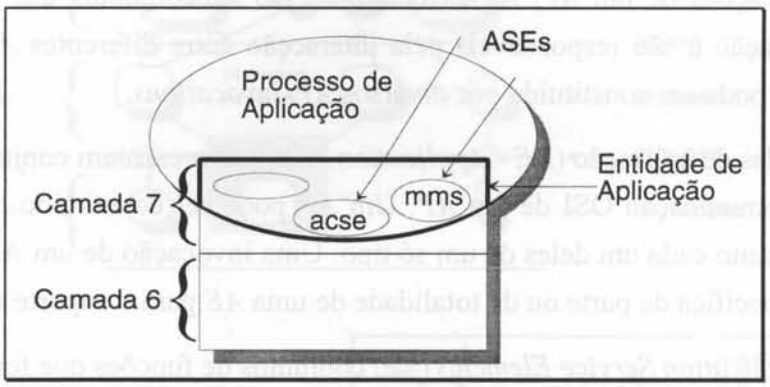


***Figura 6.2** - Mapeamento e Endereçamento OSI*

6.3. O MMS no Ambiente OSI

AEs representam a parte dos *APs* (*user programs*) envolvidos nas comunicações. Os AEs são constituidos por conjuntos de ASE. O MMS encaixa dentro do OSI como sendo um ASE.

O MMS (*Manufacturing Message Specification*) [MMS2 - 90] destina-se a permitir a cooperação funcional entre equipamentos informáticos em ambiente industrial. Isso é possível se forem definidos serviços de aplicação normalizados que permitam a cooperação entre equipamento heterogéneo que vai desde autómatos programáveis até aos computadores industriais, passando por sistemas de controlo numérico, por máquinas ferramenta, etc.

De acordo com a aproximação habitual encetada no âmbito da ISO, a norma que rege o MMS constitui-se em dois documentos fundamentais. Um diz respeito à descrição de serviços suportados pela norma [MMS1 - 90]. Um segundo [MMS2 - 90] diz respeito à especificação do protocolo, definindo o funcionamento interno do MMS para desenvolvimento de produtos MMS.

A cooperação entre equipamentos não é possível sem que eles partilhem o mesmo "Universo de Discurso" explicitado de uma maneira formal num "Esquema Conceptual Partilhado" [NUS - 91]. No MMS, o esquema conceptual correspondente ao mundo industrial, é especificado formalmente por modelos que são definidos no MMS e dos quais os principais são: o modelo cliente-servidor, o modelo de VMD (*Virtual Manufacturing Device*) e o modelo de objectos.

No restante deste capítulo são abordados alguns destes modelos, numa perspectiva de melhor fundamentar algumas das discussões que serão levadas a cabo ao longo da terceira parte desta dissertação.

6.4. Modelo Cliente/Servidor

Se bem que na rede MAP os participantes se relacionem de uma forma simétrica, as trocas de serviços MMS são organizadas de uma maneira assimétrica, seguindo uma relação do tipo cliente-servidor. Esta relação resulta, por um lado, do facto de estas trocas serem frequentemente do tipo pedido-resposta, como por exemplo um pedido de leitura de variável à distância, seguido de uma resposta contendo o valor da variável. Por outro lado, os dispositivos informáticos de comando directo de máquinas de fabrico, poderão ser considerados, sem reservas, como sendo servidores de por exemplo um controlador de oficina ou célula.

Assim, os serviços oferecidos pelo MMS ao utilizador, que é no fim de contas um processo de aplicação, são definidos por recurso a primitivas, por extensão ao método adoptado nas camadas inferiores do modelo OSI. A maior parte dos serviços MMS são do tipo confirmado, com a emissão de uma primitiva de pedido (*Request*) e retorno de uma primitiva de confirmação (*Confirmation*) ao nível de um cliente MMS, e emissão de uma indicação (*Indication*) e retorno de uma resposta (*Response*) ao nível de um servidor de MMS. O utilizador que emite o pedido tem o papel de *Requesting User*, e o que emite a resposta tem o papel de *Responding User*. O utilizador que emite um pedido ou uma resposta é chamado expedidor (*Sending User*) no sentido em que estas primitivas correspondem ao envio de unidades de dados (*Data Units*). Do lado oposto ter-se-ão os utilizadores que recebem indicações ou confirmações, sendo nesse papel designados de utilizadores destinatários (*Receiving Users*). Por fim, o MMS designa de chamador (*Calling User*) o utilizador que pede o estabelecimento de uma relação MMS com um seu correspondente, designado neste caso por chamado (*Called User*).

Uma aplicação pode genericamente enviar vários pedidos sem obter confirmação. Cada pedido tem no entanto um identificador, o *Invoke ID*, que permite relacionar sem qualquer ambiguidade um pedido e uma confirmação ou uma indicação e uma resposta correspondentes.

Um serviço MMS pode ser síncrono se só for permitido um pedido de cada vez ou assíncrono caso contrário. A *Fig. 6.3*, ilustra sucintamente o modelo cliente-servidor do MMS.

6.5. Dispositivo Virtual de Fabrico (VMD)

O VMD aparece como sendo um dispositivo abstracto mapeado nas funcionalidades e recursos de um dispositivo de fabrico real. As propriedades do VMD são manifestadas por via dos objectos MMS que ele contém: variáveis, programas, domínios, condições de acontecimento.

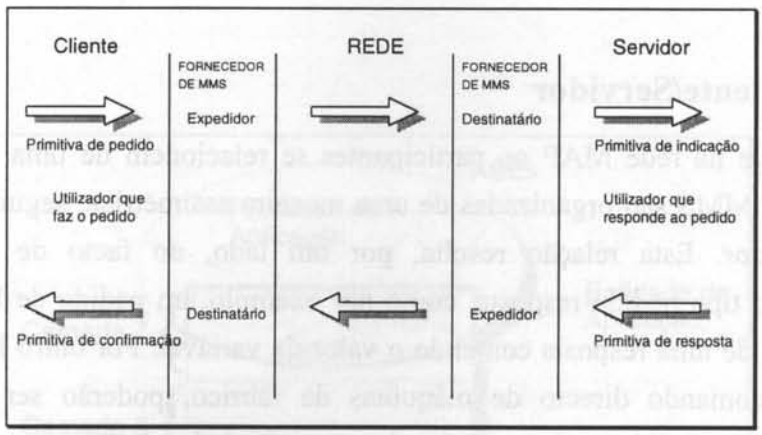


***Figura 6.3** - O Modelo Cliente/Servidor do MMS*

Os serviços assegurados por um VMD são, na sua maior parte, serviços do tipo confirmado. Nesse sentido, o VMD mantém uma lista de objectos de transacção (*Transaction Object*). Cada objecto dessa lista contém informação de estado referente a uma transacção particular, e esses objectos são criados e apagados dinamicamente. São criados após a chegada de uma indicação e apagados após o envio da primitiva de resposta MMS.

```
Objecto: Transacção
        Atributo Chave: Identificador de Lançamento
        Atributo Chave: Identificador da Associação de Aplicação
        Atributo: Lista de Modificadores de Pré-execução
        Atributo: Referência do Modificador Corrente
        Atributo: Pedido de Serviço Confirmado
        Atributo: Lista de Modificadores de Pós-execução
        Atributo: Anulação
```

O identificador de de lançamento (*Invoke ID*) distingue uma transacção particular no âmbito de uma determinada associação de aplicação. Os modificadores de pré-execução consistem em condições que devem ser adicionadas às máquinas de estado e que condicionam, de alguma forma, o atendimento a um pedido. Os modificadores de pós-execução consistem numa lista de acções a encetar após o fim do serviço (envio da resposta). O MMS só define dois

modificadores: o *AttachToSemaphore* e o *AttachToEventCondition*. Por fim, o atributo anulação indica se um pedido de serviço pode ser ou não cancelado, por recurso ao serviço MMS *Cancel*.

A *Fig. 6.4*, adaptada de [EWO - 90], ilustra o papel do VMD no âmbito das transacções MMS. Nesta figura, a sigla MMPM diz respeito a *Manufacturing Message Protocol Machine*. Esta figura reforça também a relação entre VMD e servidor MMS, no sentido de que este último só o é se contiver pelo menos um VMD. No entanto, um utilizador MMS pode ser quer um cliente, quer um servidor.

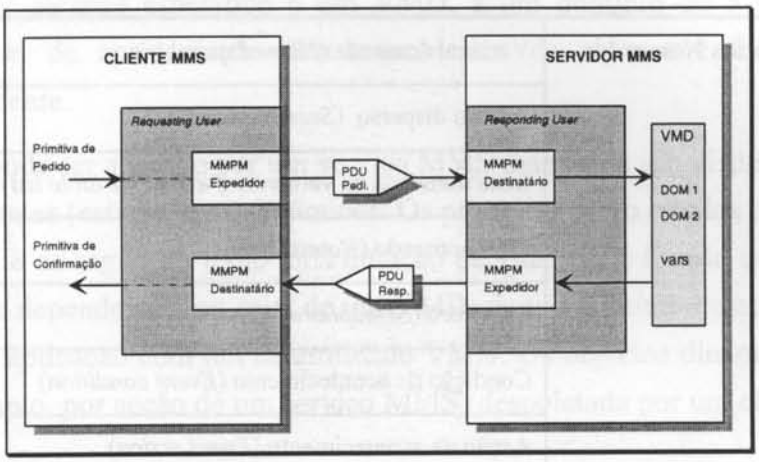


***Figura 6.4** - Clientes e Servidores MMS e o Conceito de VMD*

6.6. Objectos MMS

Cada VMD contém diversos objectos abstractos designados por objectos MMS. Em termos das normas MMS, as funcionalidades associadas a um VMD são descritas quer em termos dos objectos de VMD, quer em termos das operações que podem ser executadas sobre esses objectos.

Resulta deste modelo, orientado ao objecto, que cada objecto é uma instância particular de uma das classes de objectos contempladas na norma MMS.

No MMS, as classes são sintacticamente definidas como sendo conjuntos de objectos. A sua definição é como se segue [MMS1 - 90]:

```
Objecto: (nome da classe)
        Atributo Chave: (nome do tipo de atributo (valores))
        ...
        Atributo Chave: (nome do tipo de atributo (valores))
        Atributo: (nome do tipo de atributo (valores))
        ...
        Atributo: (nome do tipo de atributo (valores))
        Restrições: (expressões de restrição)
                Atributo: (nome do tipo de atributo (valores))
                ...
```

A cada classe é dado um nome, sendo caracterizada por um número de tipo de atributos que servem para descrever algumas das capacidades externamente visíveis de todos os objectos desta classe.

A norma MMS distingue objectos nomeados (*Named Object*), que são acessíveis por um nome, de objectos anónimos (*Unnamed Object*), que não são de todo acessíveis aos utilizadores ou então são acedidos por intermédio de endereço. As dezasseis classes (12 de nomeados e 4 de anónimos) de objectos definidas pelo MMS são as constantes da *Tab. 6.1*.

Objectos Nomeados	Variável nomeada (*Named variable*)
	Acesso disperso (*Scattered access*)
	Lista nomeada de variáveis (*Named variable list*)
	Tipo nomeado (*Named type*)
	Semáforo (*Semaphore*)
	Condição de acontecimento (*Event condition*)
	Acção de acontecimento (*Event action*)
	Implicado na acção de acontecimento (*Event enrollement*)
	Jornal (*Journal*)
	Domínio (*Domain*)
	Instância de programa (*Program invocation*)
	Operador de Estação (*Operator station*)
Objectos Anónimos	Transacção (*Transaction*)
	Autómato de Salvaguarda (*Upload state machine*)
	Variável não nomeada (*Unnamed variable*)
	Rúbrica de semáforo (*Semaphore entry*)

***Tabela 6.1** - As Classes de Objectos MMS*

De notar nessa tabela a presença de um objecto que é o objecto transacção (*transaction*). Este objecto, já anteriormente referido, serve unicamente para regular o funcionamento interno do VMD, por forma a assegurar serviços confirmados. Por outro lado, o objecto VMD não

aparece referenciado. De facto o VMD é um caso particular que não entra na tipologia anterior, uma vez que não sendo ele um objecto nomeado ele é acessível aos utilizadores através do endereço de apresentação. Este assunto particular que diz respeito à correspondência entre o 'mundo' OSI e a estrutura da camada de aplicação especificada pelo MMS, será abordado com maior detalhe na parte final deste capítulo.

O objecto MMS nomeado deverá ser único no seu ambiente. Nesse sentido, o nome de um objecto MMS é constituido por dois parâmetros. Um primeiro diz respeito ao seu alcance ou âmbito (*Scope*). Um segundo diz respeito a um identificador que deverá ser único no ambiente de alcance do objecto: alcance específico a um VMD, a um domínio ou a uma associação de aplicação. A ordem de apresentação destes ambientes de alcance é feita segundo a sua visibilidade decrescente.

Um nome pode ser alterado por um serviço MMS mas não a sua visibilidade. Os objectos podem ser permanentes (estáticos) ou dinâmicos. Os primeiros serão criados durante a criação ou *set-up* do servidor, e os segundos terão uma duração de vida que é função do elemento do qual são estruturalmente dependentes, ou seja, de um VMD, de um domínio desse VMD ou ainda de uma associação de aplicação com um determinado VMD. Os objectos dinâmicos são criados ou retirados, por exemplo, por acção de um serviço MMS (despoletada por um cliente MMS) ou por acção local.

6.7. Serviços MMS

A norma MMS define doze classes de objectos nomeados e 79 serviços para manipular esses objectos. Na prática, a maioría das aplicações reais não necessita de mais do que um pequeno subconjunto desses serviços MMS. Os serviços de MMS podem ser agrupados por famílias de serviços relacionados, conforme se descrevem na *Tab. 6.2*.

O conjunto total dos serviços MMS é no entanto de 88, dos quais 83 confirmados e 5 não confirmados. É feita a seguir uma pequena descrição funcional dos conjuntos de serviços mais relevantes do modelo MMS.

6.7.1. Serviços de Gestão do Ambiente MMS

Os serviços MMS incluídos neste grupo são: *Initiate*, *Conclude*, *Abort*, *Cancel* e por fim, *Reject*. Estes serviços são utilizados para gerir ligações lógicas entre processos de aplicação. Genericamente esta gestão implica a implementação das seguintes funcionalidades: negociar (*Initiate*), manter, libertar (*Conclude*) e abortar (*Abort*) uma ligação lógica. Os serviços MMS *Cancel* e *Reject* aplicam-se a serviços MMS e não à associação de aplicação. Os serviços ligados

com a gestão da associação de aplicação são directamente mapeados nos correspondentes do ACSE (*Application Context Service Element*).

Gestão do ambiente MMS (*Environement and general management*)
Suporte de VMD (*VMD support*)
Gestão de domínios (*Domain management*)
Gestão de instâncias de programa (*Program invocation management*)
Acesso a variáveis (*Variable access*)
Gestão de semáforos (*Semaphore management*)
Comunicações com o operador (*Operator communication*)
Gestão de acontecimentos (*Event management*)
Gestão de jornais (*Journal management*)
Gestão de ficheiros (*File management*)

***Tabela 6.2** - Os Grupos de Serviços MMS*

O serviço *Initiate*, é utilizado para estabelecer uma associação de aplicação com outro utilizador MMS, utilizando um determinado contexto MMS, em termos de serviços suportando, número de indicações pendentes, tipos de variáveis suportadas (*Parameter CBB*), etc. Alguns dos parâmetros relevantes utilizados nesta negociação são: o número de pedidos que podem estar pendentes, o nível de imbricamento suportado nas variáveis complexas, a versão de protocolo, os serviços suportados, etc. Obviamente este serviço deve ser chamado antes de qualquer outro. Se for bem sucedido, o utilizador MMS chamado deverá enviar uma resposta positiva sendo esta designada por contra-proposta. Esta é definitiva e deverá vigorar durante o tempo de vida da associação. As capacidades negociadas deverão respeitar o papel de cliente-servidor nesta associação. Para cada um dos parâmetros negociados este aspecto deverá ser respeitado.

O serviço *Conclude* é utilizado para efectuar uma conclusão, não impulsiva, da associação. Os recursos alocados durante a comunicação em cada um dos lados são libertados. O serviço MMS não confirmado *Abort* também pode ser utilizado para interromper, neste caso abruptamente, uma associação. Este é um serviço mapeado directamente no correspondente do ACSE, e pode ser despoletado por um dos utilizadores ou por um dos MMS-*Provider*. O atributo *Locally Generated* da primitiva de indicação MMS que poderá ser recebida pelos dois

utilizadores, denuncia a origem do *Abort*. A utilização deste serviço implica, quase sempre, a perda de dados bem como de serviços pendentes.

O serviço confirmado *Cancel*, poderá ser enviado por um cliente para cancelar um pedido ainda não confirmado. Obviamente que não pode ser gerado um *Cancel* de um pedido de *Cancel*. Por fim, o serviço MMS *Reject* é um serviço não confirmado gerado pelo MMS-*Provider* quando um erro de protocolo é detectado ao fazer o *parsing* da mensagem. Ambos os utilizadores recebem uma primitiva de indicação.

6.7.2. Serviços de Suporte de VMD

Os serviços de suporte de VMD permitem ao utilizador obter informação àcerca de um determinado VMD. Existem para o efeito seis serviços. São eles o *Status*, o *UnsolicitedStatus*, o *GetNameList*, o *Identify*, o *Rename* e o *GetCapabilityList*.

Os dois primeiros veiculam informação àcerca do estado lógico e físico do VMD (servidor). O primeiro corresponde a um pedido feito por um cliente MMS, enquanto que um segundo se trata de um serviço MMS de iniciativa do próprio servidor. Um dos parâmetros (aliás o único) da primitiva de pedido de *Status*, diz respeito à forma de obter o estado. Geralmente implica executar determinadas acções localmente que permitam obter a informação a colocar na primitiva de resposta.

O serviço *GetNameList*, é utilizado por um cliente para obter uma lista completa ou parcial dos objectos pertencentes a esse VMD. No pedido devem ser especificados a classe, o âmbito, etc. O serviço *Identify*, permite a um cliente MMS obter informação identificativa do VMD. O serviço *Rename* permite a um cliente MMS alterar o nome de um objecto do servidor, mas não o seu âmbito. Por fim, o serviço *GetCapabilityList*, serve para obter a lista de capacidades do VMD. As capacidades são disponíveis por intermédio de cadeias de caracteres e são específicas de acordos de implementação.

6.7.3. Serviços de Gestão de Domínios

Um domínio é formalmente um subespaço subordinado do VMD, onde os objectos deverão ser identificados de forma única. Nesse sentido, um domínio representa um subconjunto do VMD, também em termos de funcionalidades, constituindo-se assim como parte das capacidades do VMD. Os domínios são manipulados por um conjunto de serviços de gestão do domínio, permitindo estes a transferência, incluindo a eventual criação, entre utilizadores MMS. Paralelamente, os domínios estão relacionados com o grupo de serviços de Gestão de Instâncias de Programa, uma vez que um domínio pode conter programas.

Os serviços MMS de Gestão de Domínios são 12, sendo 8 para carga ou descarga de domínios, e 4 para gestão de domínios. Uma operação de descarga de um domínio (*download*) consiste em transferir informação directamente para o domínio de um servidor remoto, a partir de um cliente MMS. A operação de carga (*upload*) permite a um servidor passar o conteúdo de um domínio quer para um cliente MMS, quer para um terceiro 'parceiro'. Na realidade, estas duas operações não são executadas por um único serviço confirmado, mas sim por uma sequência de serviços segmentados (ou compósitos).

Os serviços MMS utilizados pela operação de descarga são: *InitiateDownloadSequence*, *DownloadSegment*, *TerminateDownloadSequence*. O cliente MMS envia um pedido de início de sequência de descarga. No servidor é criado um domínio e é preparado para descarga. O servidor envia a resposta ao pedido. O servidor envia um pedido de descarga, e o cliente responde com uma porção da informação a descarregar. Um campo específico desta resposta (parâmetro *MoreFollows*) indica se ainda há ou não mais dados a transmitir. Finalmente o servidor envia um serviço para terminar a operação de descarga.

De notar que para esta operação, o modelo cliente-servidor torna-se mais fluído, uma vez que um dispositivo pode alternar o papel de servidor e cliente ao longo da duração de uma associação de aplicação. De qualquer forma, um servidor deve ser definido como sendo a entidade de comunicação que se comporta como um VMD para um serviço particular, enquanto que um cliente é sempre uma entidade de comunicação que faz uso de um VMD num serviço particular de MMS. No entanto, como foi visto, um servidor (onde os objectos de um VMD são manipulados) pode ter um papel de *Requester*, quer de serviços confirmados, quer de seviços não confirmados. Neste caso, o servidor executa o pedido (primitiva *Request*) e o cliente MMS recebe a indicação (primitiva *Indication*).

O início da operação de descarga pode ainda ser despoletado pelo próprio servidor, através do serviço MMS *RequestDomainDownload*. A *Fig. 6.5*, adapatada de [EWO - 90], ilustra a sequência de transacções para a operação de descarga de domínio.

A operação de carga de um domínio recorre sempre a pedidos MMS formulados por um cliente. Os serviços utilizados são o *InitiateUploadSequence*, *UploadSegment* e *Terminate-UploadSequence*. Também aqui, o despoletar desta sequência poderá ser feito pelo servidor, com recurso ao serviço *RequestDomainUpload*.

Para além da carga e descarga de domínios, há a possibilidade de guardar ou ir buscar o conteúdo de domínios em terceiras estações. Para o caso de ir buscar o conteúdo de domínios a terceiras estações, o cliente MMS faz um pedido MMS de *LoadDomainContent*. A sequência corresponde à que é apresentada na *Fig. 6.6*.

Em [NUS - 91], aparece uma designação adequada para esta operação, que é 'telecarga'. A operação para telesalvaguarda, recorre ao serviço MMS de *StoreDomainContent*.

Por fim, é de referir dois últimos serviços, que são o *DeleteDomain*, que permite retirar um determinado domínio de um VMD remoto localizado num servidor, e o serviço *GetDomainAttributes*, que permite obter os atributos de um domínio.

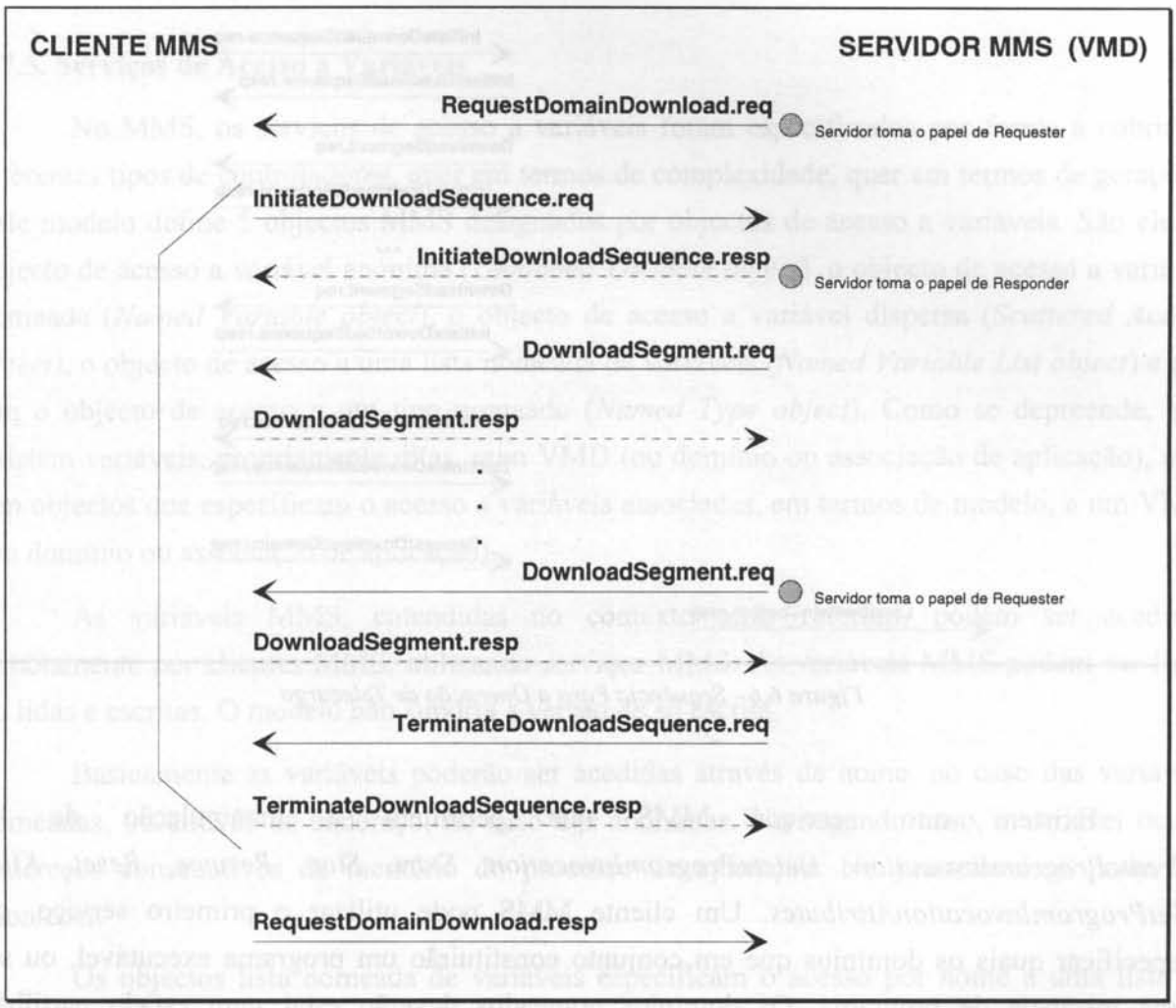


***Figura 6.5** - Sequência Para a Descarga de um Domínio*

6.7.4. Serviços de Gestão de Instâncias de Programa

O conceito de instância de programa (*Program Invocation*), pode ser entendida como sendo uma tarefa de um sistema operativo multitarefa. Por natureza, uma instância de programa é um objecto dinâmico, que pode ser pré-definido e gerido quer localmente, quer pela rede. Uma instância de programa pode estar num dos seguintes oito estados: *idle*, *starting*, *running*,

stopping, *stopped*, *resuming*, *resetting* e *unrunable*. Outros atributos das I.P. são, por exemplo, o poderem-se apagar ou não e os domínios incluídos.

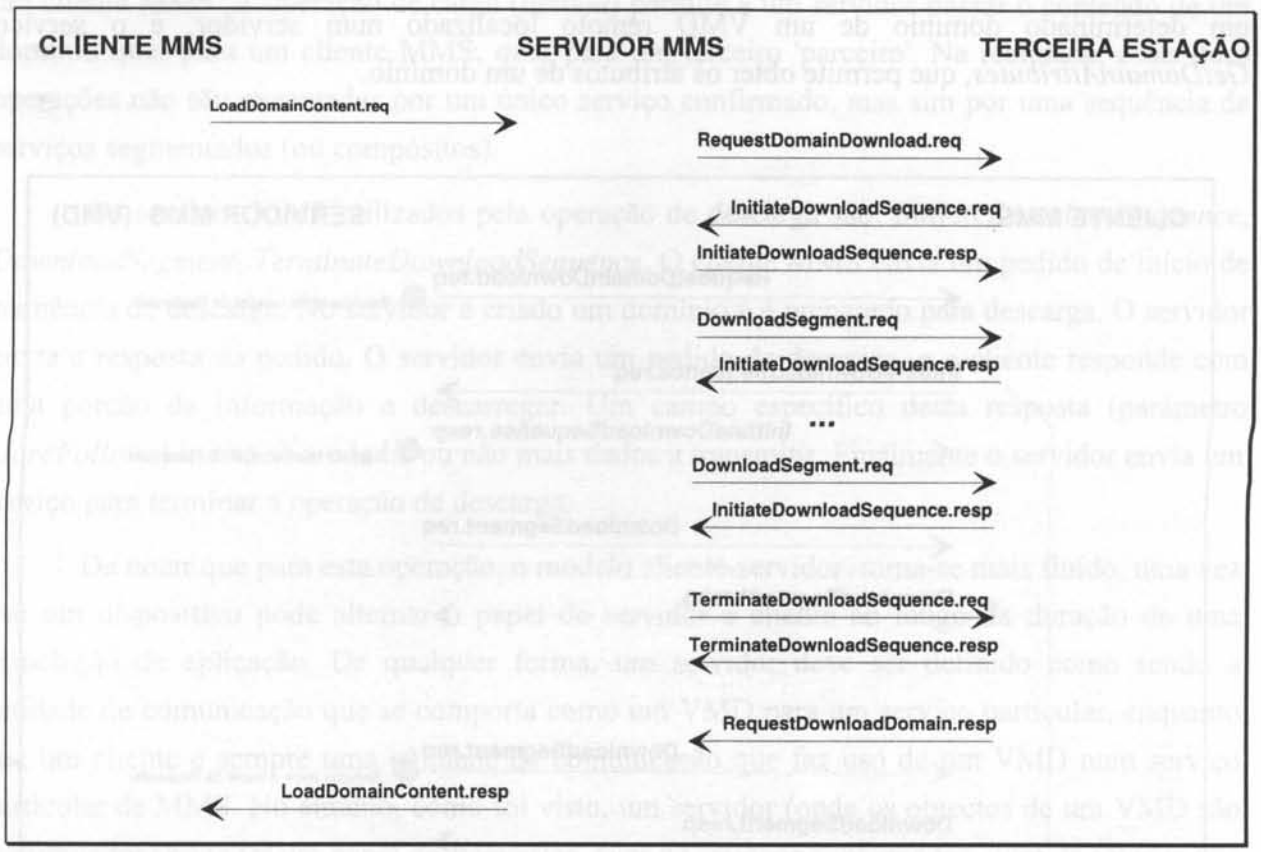


Figura 6.6 *- Sequência Para a Operação de Telecarga*

Existem oito serviços MMS que permitem a manipulação de I.P.: *CreateProgramInvocation*, *DeleteProgramInvocation*, *Start*, *Stop*, *Resume*, *Reset*, *Kill* e *GetProgramInvocationAttributes*. Um cliente MMS pode utilizar o primeiro serviço, para especificar quais os domínios que em conjunto constituirão um programa executável, ou seja, uma instância de programa. Os domínios nomeados deverão estar num estado partilhável (*shareable*) ou num estado de não ser utilizados por uma outra I.P. (*unused*). O serviço de criação de I.P. deve ser chamado antes da utilização de um serviço de gestão de I.P. Se uma I.P. tiver o atributo *monitor* activo, sempre que ela sair do estado *running*, será gerado um acontecimento e o cliente MMS é notificado.

O serviço *Start* é utilizado por um cliente MMS para iniciar a execução de uma instância de programa previamente criada. A I.P. passa do estado *idle* para o estado *running*. O serviço *Stop*, é utilizado por um cliente MMS para passar uma I.P. para o estado *Stopped*. Mais tarde essa I.P. poderá passar para o estado *running*, se recorrendo ao serviço MMS de *Resume* ou

passar para o estado *idle* se for utilizado o serviço MMS *Reset*. Este último serviço também pode ser utilizado para passar a I. P. do estado *running* para o estado *idle*. O serviço *Kill* é utilizado por um cliente MMS para colocar a I.P. no estado *unrunable* (situações de anomalia não resolvidas por *Stop* ou *Reset*). A saída deste estado é normalmente feita por recurso ao serviço *DeleteProgramInvocation*. Por fim, o serviço *GetProgramInvocationAttributes* permite a um cliente MMS determinar os atributos associados a uma determinada I.P. (estado, lista de domínios dependentes, apagável ou não, monitorado, re-utilizável, etc).

6.7.5. Serviços de Acesso a Variáveis

No MMS, os serviços de acesso a variáveis foram especificados por forma a cobrirem diferentes tipos de controladores, quer em termos de complexidade, quer em termos de gerações. Este modelo define 5 objectos MMS designados por objectos de acesso a variáveis. São eles o objecto de acesso a variável anónima (*Unnamed Variable object*), o objecto de acesso a variável nomeada (*Named Variable object*), o objecto de acesso a variável dispersa (*Scattered Access object*), o objecto de acesso a uma lista nomeada de variáveis (*Named Variable List object*) e por fim o objecto de acesso a um tipo nomeado (*Named Type object*). Como se depreende, não existem variáveis, propriamente ditas, num VMD (ou domínio ou associação de aplicação), mas sim objectos que especificam o acesso a variáveis associadas, em termos de modelo, a um VMD (ou domínio ou associação de aplicação).

As variáveis MMS, entendidas no contexto atrás referido, podem ser acedidas remotamente por clientes MMS, utilizando serviços MMS. As variáveis MMS podem ser lidas ou lidas e escritas. O modelo não suporta a versão de só escrita.

Basicamente as variáveis poderão ser acedidas através de nome, no caso das variáveis nomeadas, ou através de endereço, no caso das anónimas. No segundo caso, a variável ocupa endereços consecutivos da memória do processo de aplicação. No primeiro isso pode não acontecer.

Os objectos lista nomeada de variáveis especificam o acesso por nome a uma lista de variáveis do tipo nomeada, anónima ou dispersas. Cada um destes objectos pode ser uma variável estruturada. Um objecto disperso não tem de ser necessariamente estruturado (caso degenerado de um só elemento).

A diferença entre os objectos variável nomeada e variável não nomeada e o objecto de acesso disperso é que este último a continuidade de endereço não é um requisito, enquanto que nos primeiros, se o acesso for público, existe esse requisito.

A principal diferença entre as variáveis do tipo lista de variável e as variáveis do tipo acesso disperso, é que na resposta, o sucesso da operação é especificado para cada variável, no

caso da primeira, enquanto que na segunda não é isso que se passa, o sucesso tem de ser global, bastando falhar um para haver insucesso.

O MMS especifica 14 serviços de manipulação destes 5 objectos: *Read*, *Write*, *InformationReport*, *GetVariableAccessAttributes*, *DefineNamedVariable*, *DefineScatteredAccess*, *GetScatteredAccessAttributes*, *DeleteVariableAccess*, *DefineNamedVariableList*, *GetNamedVariableListAttibutes*, *DeleteNamedVariableList*, *GetNamedTypeAttributes*, *DeleteNamedType*.

Um serviço de acesso a uma variável tem de incluir a especificação do acesso a uma variável:

```
Tipo de acesso (lista de variáveis ou nome de uma lista)
        Lista de variáveis
                Especificação de variável
                Acesso alternativo
        Nome da lista de variáveis


Especificação de variável:
Tipo de variável (Nomeada, anónima, temporária, acesso disperso)
        Nome (excepto a anónima e a temporária)
        Endereço (na anónima)
        Descrição da variável (na temporária)
                Endereço
                Especificação do tipo
        Descrição do acesso disperso
```

A descrição dos tipos de dados é feita por recurso a sintaxes abstractas. As funções *V-Put* e *V-Get* fornecem o acesso ao valor propriamente dito das variáveis reais.

6.7.6. Serviços de Gestão de Semáforos

Os semáforos MMS são utilizados para fazer face a duas situações principais: controlar o acesso a recursos partilhados e, permitir a sincronização de múltiplos processos concorrentes. Para o efeito, o MMS define dois tipos de semáforos: semáforos do tipo *TOKEN* e semáforos do tipo *POOL*.

Os semáforos do tipo *TOKEN* são colecções de testemunhos idênticos que evoluem entre dois estados estáveis possíveis: livres ou possuídos. O número máximo de testemunhos representa o número máximo de potenciais possuidores de um semáforo. Um semáforo composto de um só testemunho corresponde à implementação de um mecanismo de exclusão mútua.

Os semáforos do tipo *POOL* são colecções de testemunhos nomeados. À semelhança do tipo anterior, cada testemunho nomeado evolui entre dois estados estáveis: livre e possuído. A única diferença entre os dois tipos reside na maneira de assumir o controlo do semáforo.

O MMS define oito serviços de manipulação do objecto semáforo: *TakeControl*, *RelinquishControl*, *DefineSemaphore*, *DeleteSemaphore*, *ReportSemaphoreStatus*, *ReportPool-SemaphoreStatus*, *ReportSemaphoreEntryStatus*, *AttachSemaphoreModifier*.

O serviço *TakeControl* permite a um cliente MMS tomar controlo de um semáforo num servidor MMS remoto. O serviço *RelinquishControl* executa a operação inversa. O serviço *DefineSemaphore* é utilizado por um cliente MMS para criar uma colecção de semáforos do ti *TOKEN* num servidor remoto. A operação inversa é executada por *DeleteSemaphore*. O serviço MMS *ReportSemaphoreStatus* é utilizado para obter informação não detalhada sobre um semáforo específico (classe do semáforo, número de testemunhos livres, número de utilizadores, etc). Um outro serviço permite obter o estado específico de um semáforo do tipo *POOL*. O serviço *ReportSemaphoreEntryStatus*, permite obter o estado detalhado. Por fim o último serviço acima listado, permite sincronizar aplicações concorrentes. Este serviço serve para atrasar a execução de qualquer serviço confirmado MMS até que o cliente detenha o semáforo ou até ele ser cancelado.

6.7.7. Serviços de Comunicação com o Operador

Neste conjunto de serviços são especificadas as funcionalidades mínimas no diálogo entre consolas e os dispositivos. Questões mais sofisticadas poderão requerer o suporte de VT (*Virtual Terminal* ISO 9040-9041), que poderá coexistir com o MMS.

Os dois serviços implementados são o *Input* e o *Output*. O primeiro é utilizado por um cliente MMS para obter uma cadeia de caracteres a partir de uma consola associada ao servidor MMS. O segundo é utilizado para por um cliente MMS para visualizar informação de saída numa consola associada a um servidor MMS remoto. Os elementos de saída são cadeias de caracteres, correspondendo cada uma delas a uma linha na consola.

6.7.8. Serviços de Gestão de Acontecimentos

O tratamento de eventos ou acontecimentos é uma função importante de um sistema de exploração distribuído. O MMS supre essa funcionalidade com um modelo extremamente potente e flexível. Um evento pode ser modelizado por uma variável boleana.

O modelo de eventos suportado pela norma MMS, inclui três objectos específicos, dezanove serviços MMS para os manipular, e um serviço modificador. Os objectos são o *Event Condition*, o *Event Action* e o *Event Enrollement*. Os serviços MMS suportados são: *DefineEventCondition*, *DeleteEventCondition*, *GetEventConditionAttributes*, *ReportEvent-ConditonStatus*, *AlterEventConditionMonitoring*, *TriggerEvent*, *DefineEventAction*, *Delete-EventAction*, *GetEventActionAttributes*, *ReportEventActionAttributes*, *ReportEventActionStatus*, *DefineEventEnrollement*, *DeleteEventEnrollement*, *GetEventActionAttributes*, *ReportEvent-*

EnrollementStatus, *AlterEventEnrollement*, *EventNotification*, *AcknowlwdgeEventNotification*, *GetAlarmSummary*, *GetAlarmEnrollementSumary*, *AttachToEventConditionModifier*.

Existem duas sub-classes para objecto *Event Condition*, que para todos os efeitos modelizam duas formas diferentes de despoletar os eventos. Os eventos podem ser despoletados pela rede (*Network Triggered*) ou despoletados pelos próprios dispositivos reais mapeados no servidor MMS (*Monitored*), por exemplo uma determinada variável de processo que execede um determinado valor. Os eventos do tipo monitorados, são detectados pelo próprio VMD, e reportados a um cliente MMS, via serviço MMS *EventNotification*.

O objecto *Event Action* especifica a acção a tomar pelo VMD após a uma transição de estado numa condição de evento. Está associado a um objecto do tipo *Event Condition*. Uma acção pode ser, por exemplo, o envio de um serviço MMS confirmado de leitura de variável ou o arranque de uma Invocação de Programa. De notar que não circulam primitivas referentes a esse serviço. A especificação do pedido de serviço confirmado é um atributo do próprio objecto *Event Action*, e a informação de resposta a esse pedido confirmado é veiculada no próprio serviço não confirmado enviado pelo servidor, ou seja, o serviço MMS *EventNotification*.

É importante, para uma maior clareza, ver a definição das três classes de objectos MMS relacionadas com o modelo de eventos, bem como a estrutura de argumentos do serviço não confirmado *EventNotification*, como definidos em [MMS1 - 90]:

```
Objecto: Event Condition
        Atributo Chave: Nome da Condição de Evento
        Atributo: Apagável via MMS (VERDADEIRO, FALSO)
        Atributo: Subclasse (DESPOLETADO PELA REDE, MONITORADO)
        Atributo: Estado (DESABILITADO, INACTIVO, ACTIVO)
        Atributo: Prioridade
        Atributo: Severidade
        Atributo: Detalhes adicionais
        Atributo: Lista de registos de evento associados
        Restrições: Subclasse = MONITORADO
                Atributo: Abilitado (VERDADEIRO, FALSO)
                Atributo: Relatório de Alarmes (VERDADEIRO, FALSO)
                Atributo: Referência da variável monitorada
                Atributo: Intervalo de tempo entre avaliações
                Atributo: Instante da última transição para ACTIVO
                Atributo: Instante da última transição para INACTIVO

Objecto: Event Action
        Atributo Chave: Nome da Acção de Evento
        Atributo: Apagável via MMS (VERDADEIRO, FALSO)
        Atributo: Pedido de serviço confirmado
        Atributo: Lista de modificadores
        Atributo: Lista de registos de evento associados
        Atributo: Detalhes adicionais

Objecto: Event Enrollement
        Atributo Chave: Nome da Acção de Evento
        Atributo: Apagável via MMS (VERDADEIRO, FALSO)
        Atributo: Subclasse (MODIFICADOR, NOTIFICAÇÃO)
        Atributo: Condição de evento associada
```

```
Atributo: Transições de de condição de evento (DESABILITADO -> ACTIVO,
          DESABILITADO -> INACTIVO, INACTIVO -> ACTIVO,
          INACTIVO -> DESABILITADO, ACTIVO -> INACTIVO,
          ACTIVO -> DESABILITADO, QUALQUER -> APAGADO)
Atributo: Identificador local da associação de aplicação
Restrições: Subclasse = MODIFICADOR
      Atributo: Invoke ID
      Atributo: Atraso aceitável
Restrições: Subclasse = NOTIFICAÇÃO
      Atributo: Notificação perdida (VERDADEIRO, FALSO)
      Atributo: Referência da acção de evento
      Atributo: Referência da condição de evento
      Atributo: Duração (CORRENTE, PERMANENE)
      Atributo: Aplicação cliente
      Atributo: Regra de notificação de alarme (NENHUM, SIMPLES, ACK ACTIVO,
          ACK TODOS)
      Atributo: Instante de recepção de ACK, qundo activo
      Atributo: Instante de recepção de ACK, quando inactivo
      Atributo: Estado (DESABILITADO, INACTIVO, ACTIVO, INACTIVO N-A-I,
          INACTIVO N-A-A, ACTIVO N-A-A, ACTIVO A, INACTIVO A)
```

Nome do parâmetro	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Event Enrollement Name	M	M(=)		
Event Condition Name	M	M(=)		
Severity	M	M(=)		
Current State	C	C(=)		
Transition Time	M	M(=)		
Notification Lost	M	M(=)		
Alarm Acknowledge Rule	C	C(=)		
Action Result	C	C(=)		
Event Action Name	M	M(=)		
Sucess or Failure	M	M(=)		
Comand Service Response	S	S(=)		
Comand Service Error	S	S(=)		

***Tabela 6.3** - Parâmetros do Serviço EventNotification do MMS*

O objecto *Event Enrollement* (registo de eventos), associa clientes e tipos de serviços a eventos. Obviamente que para um cliente ser notificado precisa de que haja uma associação de aplicação activa.

Para cada um dos três objectos, o cliente pode efectuar as seguintes operações: definir, apagar, obter os atributos, e obter o estado do objecto. Para estas quatro operações sobre os três objectos são utilizados 12 dos serviços anteriormente referidos. O serviço *AlterEventCondition-Monitoring* permite a um cliente alterar alguns dos atributos de uma condição monitorada (habilitada/desabilitada, prioridade, etc). A prioridade é relativamente importante, uma vez que é consumido tempo de processamento no servidor para fazer o verificação periódica das diferentes variáveis monitoradas. O serviço MMS *TriggerEvent*, é utilizado por um cliente MMS para despoletar um evento do tipo *NetworkTriggered*. O serviço *AlterEventEnrollement*, permite a um cliente alterar alguns atributos do objecto, como por exemplo, a transição na condição de evento ou regra de notificação de alarme. O serviço *EventNotification* é utilizado para notificar os cliente registados, da ocorrência de uma transição de estado associada a uma determinada condição de evento. Apesar de este serviço não ser confirmado, um servidor MMS pode pedir que o serviço *EventConditon* seja confirmado por um outro enviado pelo cliente. O cliente pode ignorar ou enviar o serviço *AcknowledgeEventNotification*. Este procedimento deve ser utilizado em aplicações críticas.

O serviço *GetAlarmSummary* é utilizado por um cliente MMS para obter um sumário dos alarmes que ocorreram num VMD, referentes a um conjunto específico de condições de evento. Um alarme é um evento particular que o que o cliente deve guardar numa tabela de sumários de alarme. O serviço *GetAlarmEnrollementSummary* permite a um cliente MMS obter o estado corrente de todos os *Event Enrollements* aos quais se aplica a regra de notificação de alarme.

Por fim, o serviço *AttachToEventConditionModifier* é utilizado por um cliente MMS para especificar que a respota a um determinado serviço num VMD fica condicionada à ocorrência de uma transição no estado de uma condição de evento.

6.7.9. Serviços de Gestão de Jornais

Os serviços de gestão de jornais (*Journal Management Services*) são especialmente adequados para os historiais de dados. A informação contida num jornal pode ser, por exemplo, de acontecimentos associados a conteúdos de variáveis ou notas e observações feitas pelo operador. Um jornal MMS não é mais do que uma sequência cronologicamente ordenada de informação diversa. No entanto, a norma MMS não especifica sincronização temporal entre Cliente e Servidor, pelo que algumas entradas poderão não representar uma cronologia muito precisa.

O MMS suporta seis serviços de manipulação do objecto MMS jornal: *ReadJournal*, *WriteJournal*, *InitializeJournal*, *ReportJournalStatus*, *CreateJournal*, *DeleteJournal*. O primeiro, pode ser utilizado pelo cliente MMS para leitura de uma ou mais entradas de um jornal num servidor MMS remoto. O cliente pode especificar, no pedido, filtros por forma a selecionar a

informação apropriada (as normas companheiras também regem esta matéria). O serviço *WriteJournal*, é utilizado por um cliente MMS para introduzir uma ou mais entradas num jornal de um VMD remoto. O serviço *InitializeJournal* é utilizado por um cliente MMS para apagar, total ou parcialmente, entradas de um jornal. O serviço *ReportJournalStatus* é usado por um cliente MMS para obter o número de entradas de um objecto jornal de um VMD remoto. Também neste caso, as normas companheiras especificam informação adicional. Os dois últimos serviços são de funcionalidade óbvia.

6.7.10. Serviços de Gestão de Ficheiros

Existe um só serviço normativo que permite a um cliente instruir um servidor (VMD) para obter um determinado ficheiro: *ObtainFile*. Quando o servidor recebe a indicação deste serviço nela vai especificado o ficheiro a obter por ele. Se esta localização não for especificada é assumido que o ficheiro reside no cliente MMS que emite o pedido.

Em anexo da norma, o MMS descreve 6 serviços de gestão de ficheiros: *FileOpen*, *FileRead*, *FileClose*, *FileRename*, *FileDelete*, *FileDirectory*. Estes serviços do MMS são semelhantes (sub-conjunto) aos utilizados pelo FTAM ISO 8571. O *FileOpen* prepara um ficheiro para um sequência de leitura. O *FileRead* é utilizado para transferir parcial ou totalmente o conteúdo de um ficheiro aberto num servidor MMS, para o cliente MMS. *FileClose* é utilizado para terminar a sequência de leitura de um ficheiro (libertando os recursos entretanto utilizados). Os três últimos serviços são serviços utilizados para a gestão de ficheiros: mudança de um nome de um ficheiro localizado num servidor remoto, apagar um ficheiro (se tiver previlégios para tal) e obter informação sobre ficheiros, ou grupos de ficheiros (utilizando o mecanismo já referido de *MoreFollows* para a transmissão de grandes quantidades de informação).

6.8. Normas Companheiras

Ao especificar o MMS, houve algum cuidado em possibiltar uma abrangência tanto maior quanto o possível de diferentes tipos de equipamentos. Um núcleo de serviços genéricos foi definido como atrás se descreveram. Contudo, diversos grupos de tabalho têm vindo a desenvolver especificações detalhadas orientadas a áreas específicas. O seu objectivo, é harmonizar classes de utilização do MMS, especificando a utilização dos parâmetros que em inúmeros serviços são matéria da implementação e não regulamentados na norma principal do MMS. Entre outros aspectos, esta normas companheiras do MMS (MMS *Companion Standards*) deverão identificar um conjunto de funcionalidades específicas de cada área, a maneira como mapear essas funcionalidades no modelo de objectos do MMS em termos de objectos abstractos

(VMDs, domínios, semáforos, etc), e especificar as extensões semânticas e sintácticas dos serviços abstractos MMS.

Alguns grupos de trabalho foram definidos, tendo-se já chegado a alguns resultados e respectivas normas companheiras como normas internacionais. Destacam-se os grupos de trabalho para arquitecturas de controlo de processos (*Process Control Architectures*) [PCA - 89], para *Robots* [ROB - 89], para controlo numérico [NC - 89], para PLCs (Autómatos Programáveis) [PLC - 89], e para Gestão da Produção [PM - 89].

6.9. Mapeamento do MMS no Modelo OSI

A questão da correspondência entre OSI e MAP/MMS foi objecto de debates em diversos comités, numa altura em que a norma MMS estava já quase estabilizada. A esse propósito, e a título de exemplo, podem ser transcritas [EMUG - 89] algumas interrogações constantes de uma minuta referente a uma reunião de trabalho tida no âmbito do EMUG (*European MAP User Group*): "*How an AEi may have more than one AA?*"; "*Does MMS constrain AEi to only one PSAP per AE-invocation or NOT?*"; "*Does MMS constrain AEi to only one AA (1:1 mapping)?*".

Como foi dito em §6.3, o MMS encaixa no modelo OSI como uma ASE (*Application Service Element*).

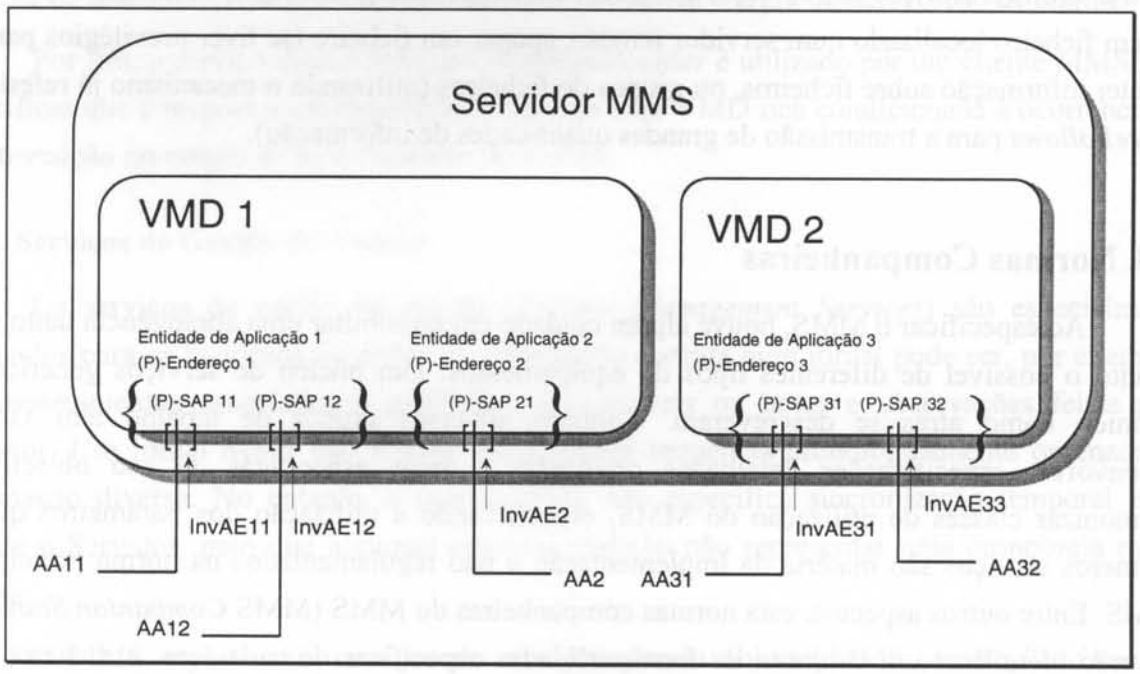


***Figura 6.7** - Modelização MMS*

A *Fig. 6.7* sintetiza as correspondências entre os modelos MMS e o modelo de referência OSI. Nessa figura, o servidor MMS é o Processo da Aplicação (AP - *Application Process*). O servidor poderá ter vários VMDs. O Processo de Aplicação pode ter capacidades de comunicação, chamadas Entidades da Aplicação (AE - *Application Entities*). Cada AE está relacionada com um e um só VMD. Uma AE pode ter 0 ou mais Invocações de AE (InvAE). Cada InvAE modeliza um tipo de utilização de um AE numa Associação da Aplicação (AA - *Application Association*). Vários InvAEs podem ser utilizados para modelizar diferentes AA num VMD que contém essa AE. De uma maneira genérica, cada VMD pode ter vários AEs, cada um com várias InvAE. Cada AE-Título está mapeado num e num só Endereço de Apresentação - (P)-Endereço, que identifica um conjunto de P-SAPs.

6.10. Referências e Bibliografia

[BAR - 83] **Paul D. Bartoli**, *The Application Layer of the Reference Model of the Open Systems Interconnection*, Proceedings of the IEEE, Vol. 71, no. 12, December 1983, pp. 1404-1407.

[CHA - 83] **A. Lyman Chapin**, *Connections and Connectionless Data Transmission*, Proceedings of the IEEE, Vol. 71, no. 12, December 1983, pp. 1365-1371.

[DAY - 83] **John Day**, **Hubert Zimmermann**, *The OSI Reference Model*, Proceedings of the IEEE, Vol. 71, no. 12, pp. 1334-1340, Dezembro de 1983.

[ISO/A-84] ISO 7498, *Information Processing Systems - Open Systems Interconnection - Basic Reference Model*, ISO 1984.

[EMUG - 89] *Minutes of the 8th EWOS/MMS Meeting*, Karlsruhe, 6 de Novembro de 1989.

[TAN - 88] **Andrew S. Tanenbaum**, *Computer Networks*, CAP. 6 - The Transport Layer, Prentice-Hall International, 1988.

[PCA - 89] *Process Control Architecture Companion Standard to MMS*, ISA DS72-02, Junho 1989.

[ROB - 89] *Robot Companion Standard to MMS*, ISO/TC184/SC2/WG6 , Fevereiro 1989.

[NC - 89] *Companion Standard to MMS - Numerical Control*, EIA PN 1924 , Fevereiro 1989.

[PLC - 89] *Companion Standard to MMS - Programble Logical Control*, NEMA - IEC/SC65A/WG6/TF7 , Fevereiro 1989.

[PM - 89] *Production Management Companion Standard*, ISO/TC184/SC5/W2 - N141, Agosto 1988.

[ISO/B-89] ISO 9545, *Information Technology - Open Systems Interconnection - Application Layer Structure*, ISO/IEC 1989.

[ISO/C-89] ISO 7498-3, Information Processing Systems - Open Systems Interconnection - Parte 3: Naming and Addressing, 1989

[LIBA - 89] **Liba Sbodova**, *Implementing OSI Systems*, IEEE Journal on Selected Areas in Communications, Vol. 7, no. 7, pp. 1115-1130, Setembro de 1989.

[MMS1 - 90] ISO/IEC 9506-1, *Manufacturing Part Specification, Part 1 Service Definition*, ISO/IEC 1990.

[MMS2 - 90] ISO/IEC 9506-2, *Manufacturing Part Specification, Part 2 Protocol Specification*, ISO/IEC 1990.

[EWO - 90] *Manufacturing Message Specification Tutorial EWOS Technical Guide*, EWOS Draft 2.50, Bruxelas, Nov 1990.

[PIM - 90] **Juan R. Pimentel**, *Communications Networks for Manufacturing*, Prentice-Hall International Editions, 1990.

[NUS - 91] **Henri Nussbaumer**, *Téléinformatique Vol. IV Messagerie Électronique X.400, Messagerie Industrielle MMS, FTAM*, Presses Polytechniques et Universitaires Romandes, Lausanne, 1991.

[SNI - 92] *MMS Tutorial*, SNI AP 111, Siemens Nixdorf AG, Nürnberg, 1992.

TERCEIRA PARTE

Do Dispositivo de Interligação

Capítulo 7

Análise Funcional e Proposta de Arquitectura

Capítulo 8

Apontamentos de Implementação

Capítulo 9

Considerações Finais e Conclusões

Capítulo 7

Análise Funcional e Proposta de Arquitectura

Neste capítulo é abordada a interligação da rede do tipo *Field Bus*, FILBUS, com a arquitectura de comunicações MAP. Esta interligação com o MAP potencia a integração em ambiente CIM das actividades cobertas pela rede de *Field Bus*. Mais do que a consideração genérica da interligação com o MAP, será de salientar o MMS como entidade federadora. Isto é tanto mais importante uma vez que o MMS é suportado como camada de aplicação por outras redes de valência hierarquica intermédia, como o MiniMAP, o PROFIBUS, o FIP ou o FAIS.

7.1. Introdução

A integração da rede de *Field Bus* (neste caso o FILBUS) no ambiente MAP 3.0 implica, dadas as características genéricas de ambas as arquitecturas, fazer a interligação das duas redes ao nível da camada de aplicação. A tarefa é tanto mais difícil quanto, como ficou claro no capítulo 5, os serviços da camada de aplicação da rede de *Field Bus* considerada não se aproximam de todo aos serviços disponibilizados pela arquitectura MAP, em particular aos serviços MMS. Isto é válido quer no que se refere aos modelos implementados (orientados ao objecto no MMS) quer no que se refere à sua sintaxe e semântica. A filosofia cliente-servidor do MMS, mas com mais preponderância, a confirmação remota da maioria dos serviços (com a consequente existência de quatro primitivas) não tem também paralelo na rede de *Field Bus* em causa. Na realidade, e na medida em que a aproximação é válida, o paradigma de comunicações mais aproximado para esta rede de *Field Bus*, será um outro também muito difundido, o de produtor-consumidor [WIL - 90], presente por exemplo no FIP.

Do ponto de vista do modelo cliente-servidor do MMS, o dispositivo (computador) que deverá suportar as duas arquitecturas de comunicação, doravante designado por dispositivo de interligação, deverá ser um servidor MMS, que mapeará de uma forma virtual a rede ou redes *Field Bus* a ele ligadas. Potenciais clientes MMS poderão comunicar com o *Field Bus*, utilizando

serviços para manipular objectos abstractos MMS, objectos esses que corresponderão de uma forma virtual à rede de *Field Bus*.

Este servidor deverá fazer corresponder, reciprocamente, serviços e modelos MMS, suportados pela norma MAP, a serviços e modelos disponibilizados pela rede particular de *Field Bus*. O MMS é um protocolo e um conjunto de serviços extremamente completo e complexo. O FILBUS é, acima de tudo, extremamente flexível em termos de funcionalidade. A combinação destas duas ilações faz antever as inúmeras opções de modelização que poderão ser consideradas para este servidor de MMS. Em grande medida, estas opções irão ser analizadas à luz de três ou quatro perspectivas, que no seu conjunto formaram as fronteiras desta análise.

A primeira, e talvez a mais importante destas perspectivas, tem a ver com as características técnicas e funcionais, e a tipologia desta rede de *Field Bus*. Uma característica importante reside no facto de suportar horizontalmente actividade reflexa. Isto é, um módulo poderá alterar parâmetros que regem o funcionamento de outro módulo, exigindo eventual reestruturação dinâmica do VMD. Mesmo que esta actividade reflexa só veicule informação relativa a alteração de E/S (entrada/saída), surge uma natural dificuldade, que é a de manter a coerência entre o(s) VMD(s) e o dispositivo real. É também uma característica importante o facto de a maioria dos módulos suportar números relativamente grandes de E/S, podendo cada uma delas ser controlada em termos funcionais por tabelas independentes de parâmetros. Mais ainda, diferentes tipos de módulos correspondem a diferentes parametrizações, e diferentes tipos de serviços de aplicação. Por exemplo, existem diferentes serviços para aceder a uma mesma E/S do módulo, dependendo de como ela está programada.

Sob outra perspectiva, as opções deverão ser condicionadas pelas aplicações para as quais a rede está vocacionada. Exigências de tempo crítico, distribuição dos módulos pela aplicação, necessidade de utilizar diferentes E/S de uma mesmo módulo em diferentes subprocessos de uma aplicação mais vasta, são alguns aspectos importantes. É preciso ter em consideração o tipo de informação que deve ficar ao nível da rede de *Field Bus* e a que deverá fluir verticalmente. Por exemplo, deverão as acções de controlo ser tomadas ao nível hierarquico superior?

Do lado do MAP/MMS também deverão ser consideradas as classes de conformidade, a limitação de canais de comunicação OSI suportados pelo *hardware* MAP, etc. Finalmente, deverão ser referidas as limitações impostas pela plataforma, nomeadamente a utilização de um sistema operativo que não suporta multitarefa.

Como se verá, muitos outros aspectos contribuem directa ou indirectamente para a pouca linearidade que existe na especificação de um dispositivo de interligação que, acima de tudo, se pretende não deva restringir as funcionalidades fornecidas por esta rede de *Field Bus*.

Numa primeira parte, este capítulo abordará as questões relacionadas com aquilo a que se poderá designar por modelo estático do servidor, ou seja, questões relacionadas com a estruturação do servidor em objectos MMS, essencialmente no que se refere a VMDs, domínios, e objectos de acesso a variáveis. Numa segunda parte serão analizadas as questões relacionadas com o modelo dinâmico do servidor, ou seja a manipulação desses objectos por parte das duas arquitecturas de comunicações, no âmbito da sua interligação.

7.2. Mapeamento da Rede de Field Bus em Objectos Abstractos MMS

7.2.1. Do Mapeamento em Termos de VMDs e Endereçamento OSI

No capítulo 6 foi abordada a correspondência entre o modelo OSI, em particular as suas duas camadas superiores, e a arquitectura MAP/MMS. Esta matéria assume um certo relevo, resultante do envolvimento directo no endereçamento OSI dos objectos VMD a suportar pelo dispositivo de interligação. As opções a tomar a esse respeito, condicionarão, directa ou indirectamente, toda a restante arquitectura do dispositivo de interligação.

Da análise feita nesse capítulo, poder-se-ão fazer as seguintes afirmações. Um servidor de MMS é um processo de aplicação, tal como entendido no modelo OSI. Um servidor de MMS poderá conter vários VMDs, no mínimo um. Uma entidade de aplicação está relacionada com um e um só VMD. No entanto, um VMD poderá estar associado a mais do que uma entidade de aplicação. É ainda válido dizer-se que a cada entidade de aplicação corresponde um endereço de apresentação, que a identificará univocamente no ambiente OSI, permitindo assim o estabelecimento de uma associação de aplicação entre dois utilizadores. Uma entidade de aplicação pode suportar simultaneamente várias associações de aplicação.

O conceito de servidor de MMS está, como foi visto, profundamente associado ao conceito de VMD. Um VMD pretende ser um dispositivo abstracto onde são mapeadas as funcionalidades e os recursos do dispositivo de fabrico real. No caso presente, a rede de *Field Bus* corresponderá ao(s) dispositivo(s) a mapear.

A análise do modelo de servidor de MMS, em termos de objectos abstractos, deverá começar por ser feita quanto à sua estruturação em VMDs. Como mapear no servidor, em termos de VMDs, esta rede de *Field Bus*?

Considerando por um lado o conceito de VMD do MMS, e por outro as características dos dispositivos desta rede de *Field Bus*, não se afigura descabida uma resposta no sentido do mapeamento unívoco de cada módulo, num VMD distinto. A *Fig. 7.1* ilustra essa opção, para uma rede de *Field Bus* constituída por três nós, um controlador (neste caso o mestre), um módulo

de saídas (p. ex., o FILOUT-1648) e um módulo de entradas (p. ex., o FILINP-1648), admitindo apenas a possibilidade do estabelecimento de uma só associação de aplicação por VMD.

Sustentar para os módulos FILBUS a classificação de dispositivo não é, de uma forma objectiva, quantificável ou qualificável pela referência a um qualquer conjunto das suas características. No entanto, e pela caracterização feita no capítulo 5, afigura-se como sendo óbvia. Por isso, se o único requisito para justificar tal mapeamento fosse o facto de cada módulo poder ser considerado *per si* um 'verdadeiro' dispositivo, então esta opção não sería sequer questionável. Não é no entanto o único requisito, pelo que a análise desta questão não pode ser tão liminar.

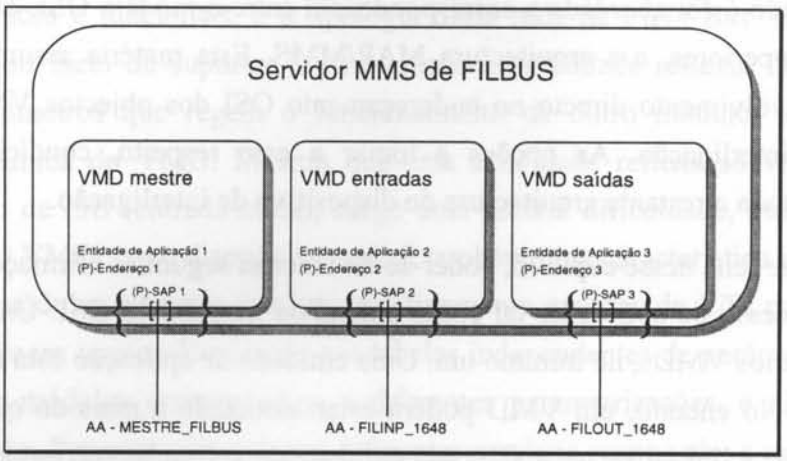


***Figura 7.1** - Modelo de Servidor com um VMD por Módulo*

A análise que se segue, parte da filosofia de mapeamento patente na *Fig. 7.1* para, apontando algumas insuficiências que lhe são inerentes, tentar justificar a sua menor adequação, face a uma alternativa que advoga apenas a utilização de um só VMD para modelizar toda a rede de *Field Bus*.

7.2.2. Da Opção Por um Só VMD

A rede de comunicações do tipo *Field Bus* aqui considerada, poderá suportar até 250 módulos. Na realidade, muitas aplicações para as quais este tipo de redes está vocacionado, têm como característica, o facto de terem associados centenas de dispositivos de instrumentação distribuídos pelo campo. A título de exemplo, em [USER - 92] é referida uma plataforma petrolífera contendo aproximadamente quatro mil pontos de medida.

Para a filosofia de mapeamento OSI retratada na *Fig. 7.1*, isso implica a coexistência de tantos endereços de apresentação, e por isso entidades de aplicação, quantos os módulos da rede

a integrar. Admitindo uma rede de *Field Bus* com ***N*** módulos, sería necessário suportar, ao nível do servidor, ***N*** associações de aplicação simultâneas, com outras tantas entidades de aplicação, através de ***N*** endereços de apresentação. Para este caso concreto, cada entidade de aplicação coincide com um VMD.

O número de associações de aplicação a considerar pode todavia não ficar por aqui.

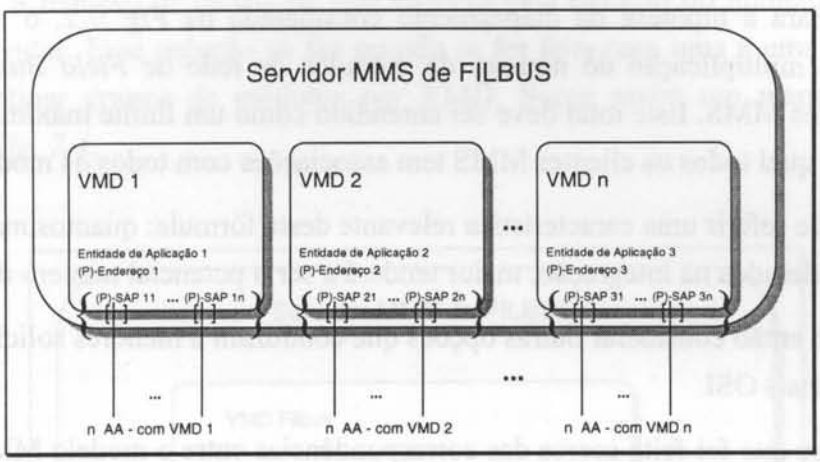


***Figura 7.2** - Modelo de Servidor com um VMD por Módulo e Mais do que uma AA por VMD*

Neste tipo de redes os módulos suportam conjuntos de E/S e não uma só E/S isolada. Na realidade, para esta rede de *Field Bus* em particular, existem módulos que suportam até 32 E/S, como por exemplo, os módulos FILINP 32xx, FILOUT32xx ou os FILCIO32xx. Perante essa disponibilidade por um lado e o preço de cada módulo por outro (pelo menos duas centenas de contos) será de considerar que, num mesmo módulo, diferentes E/S façam parte de diferentes subaplicações ou mesmo aplicações. Isto poderá significar que o mesmo módulo poderá estar associado a diferentes sistemas de supervisão/controlo e, em última análise, a potenciais diferentes clientes MMS (de notar que esta rede tem serviços de aplicação e parametrização orientada às E/S).

Sendo assim, pode-se considerar como não satisfatório um mapeamento OSI que apenas permite uma associação de aplicação com cada uma das entidades de aplicação (e por isso módulo). A alternativa sería então considerar um mapeamento OSI como o representado na *Fig. 7.2.*

Parece ser agora oportuno falar de um facto objectivo: qualquer *hardware* de comunicações OSI suporta um número de canais limitado, por exemplo, 8 ou 16 canais. Um canal de comunicação deve ser entendido como representando uma potencial associação com

outra aplicação. Isto quer dizer que um determinado sistema OSI suporta um número máximo de associações de aplicação simutâneas. A limitação em termos de canais é óbvia.

Cada associação de aplicação tem lugar através de um canal de comunicação. De acordo com as normas OSI de endereçamento, uma entidade de aplicação deverá ser registada num canal ou em vários canais, correspondendo neste último caso, a múltiplas associações num único endereço de apresentação.

Assim, para a hipótese de mapeamento considerada na *Fig. 7.2*, o número de canais corresponderá à multiplicação do número de módulos da rede de *Field Bus* pelo número de potenciais clientes MMS. Este total deve ser entendido como um limite máximo, correspondente a um cenário no qual todos os clientes MMS tem associações com todos os módulos.

É ainda de referir uma característica relevante desta fórmula: quantos mais módulos *Field Bus* forem considerados na integração, maior tenderá a ser o potencial número de clientes MMS.

É preciso então considerar outras opções que conduzam a menores solicitações em termos de número de canais OSI.

Da análise que foi feita àcerca das correspondências entre o modelo MMS e o modelo de referência OSI, ficou claro que uma entidade de aplicação está relacionada com um e um só VMD. Por isso, um endereço de apresentação não permite endereçar mais do que um VMD.

Pode-se concluir, então, que a redução das exigências em número de canais de comunicação OSI passa pela redução do número de VMDs. É também verdade que um número mais reduzido de VMDs no servidor não conduz necessariamente a uma menor quantidade de canais de comunicação OSI. Veja-se por exemplo a *Fig. 7.3*.

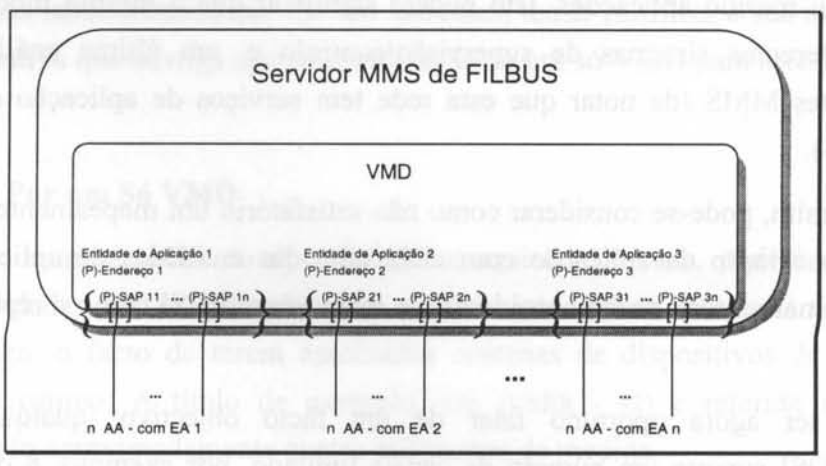


***Figura 7.3** - Modelo de Servidor Com Um VMD e N Entidades de Aplicação*

Nesse caso, o VMD mapea funcionalmente toda a rede de *Field Bus,* passando a conter os objectos MMS abstractos necessários para a modelização global, o que corresponderá, na essência, ao somatório dos considerados originalmente para cada um dos VMDs. Em termos do endereçamento OSI, continuam a existir tantos endereços de apresentação quantos os módulos da rede de *Field Bus*, pelo que, para os clientes MMS é quase a mesma coisa (no papel de *requester*, não é relevante a estruturação em VMDs).

Portanto, a redução de canais só será efectiva pela redução do número de entidades de aplicação no servidor. Essa redução só faz sentido se for feita para uma e uma só. De facto não faz sentido aglutinar grupos de módulos por VMD. Surge assim um mapeamento como o apresentado na *Fig. 7.4*.

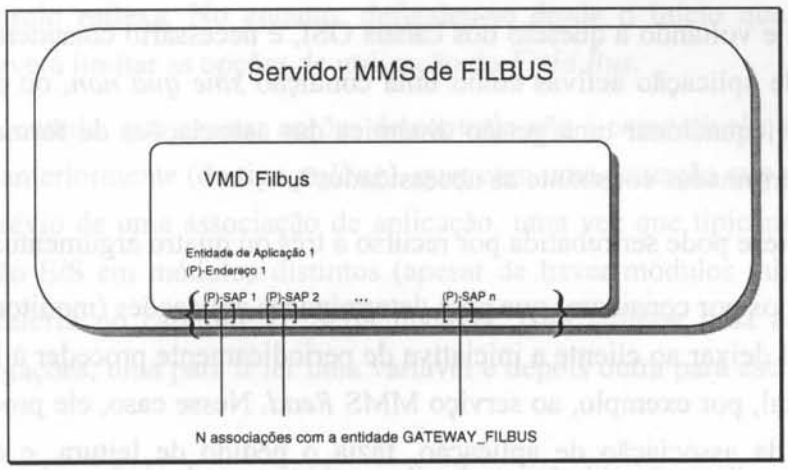


***Figura 7.4** - Modelo de Servidor Com Um VMD e Uma Entidade de Aplicação*

Face ao mapeamento da *Fig. 7.3*, esta última hipótese só apresentará menor elegância no endereçamento. Esta não é contudo uma característica minimamente relevante em termos funcionais. Apresenta contudo uma grande vantagem, que é o muito menor número de canais de comunicação a suportar. Esse número passa a ser igual ao número de potenciais clientes MMS que simultaneamente podem utilizar os 'serviços' do dispositivo de interligação.

Até aqui, contestou-se a hipótese inicial, isto é, a do mapeamento de cada módulo da rede de *Field Bus* num VMD distinto, com base no excessívo número de canais OSI requeridos.

Ora, essa hipótese inicial apresenta algumas vantagens. Apresenta alguma elegância do ponto de vista da implementação de alguns dos modelos ou serviços MMS, assumidos como adequados ao caso presente. Podem ser disso exemplo os serviços de gestão de VMD (*Status*, *Identify*, etc) ou eventualmente alguns modelos MMS mais sofisticados, como o de gestão de programas ou de gestão de semáforos, como se deduz da análise feita no capítulo 6. No serviço

Status, por exemplo, a determinação do estado físico e do estado lógico resultarão, eventualmente, facilitadas, se adoptado esse tipo de mapeamento.

Essa hipótese inicial, e já foi atrás referido, poderá ainda ter alguma elegância do ponto de vista da transparência de endereçamento do dispositivo de interligação: um potencial cliente MMS, para comunicar com os diferentes módulos, estabeleçe associações com diferentes entidades de aplicação e por conseguinte, diferentes endereços OSI.

Mais ainda, este mapeamento poderá de alguma forma facilitar algumas questões relacionadas com a implementação. Por exemplo, nos eventuais mapeamentos de serviços MMS em serviços FILBUS, a recepção de indicações MMS através de um determinado canal OSI (associação de aplicação), corresponde à manipulação de objectos respeitantes a um único módulo, pelo que o endereço do módulo FILBUS a ser endereçado é único para esse canal.

Será que, e voltando à questão dos canais OSI, é necessário considerar a simultaneadade de associações de aplicação activas como uma condição *sine qua non*, ou por outras palavras, não será possível equacionar uma gestão dinâmica das associações de forma a que estas sejam estabelecidas e terminadas consoante as necessidades?

Esta hipótese pode ser rebatida por recurso a três ou quatro argumentos.

Comecemos por considerar que para determinadas aplicações (monitorização, supervisão) pode ser razoável deixar ao cliente a iniciativa de periodicamente proceder à leitura de variáveis, recorrendo para tal, por exemplo, ao serviço MMS *Read*. Nesse caso, ele procedería primeiro ao estabelecimento da associação de aplicação, fazia o pedido de leitura, e depois terminava a associação de aplicação. Atendendo ao número elevado de variáveis que previsivelmente existirão ao nível do *Field Bus*, esta actividade MMS iría impôr uma pesada carga no servidor, que se baseia numa plataforma *MS-DOS* que tem igualmente que sustentar o *Host* da rede de *Field Bus*.

Poder-se-ía tentar ultrapassar esta situação procedendo em sentido inverso. De facto, a maioría dos módulos desta rede de *Field Bus*, em particular os módulos do tipo digital, pode ser programado para enviar automaticamente ao *host* notificações de alteração do estado das suas E/S digitais. Sendo assim, porque não 'prolongar' esta notificação até ao eventual cliente MMS? Para mais, o MMS permite modelizar este tipo de situações através de pedidos de serviço não confirmados, de iniciativa do servidor. O servidor pode enviar a um cliente um pedido não confirmado de *InformationReport*, contendo o valor das variáveis alteradas que modelizam as E/S dos módulos. O modelo de eventos também pode ser utilizado. Neste caso, é a própria máquina MMS que toma a iniciativa de enviar um *EventNotification* veiculando a informação adequada aquando da alteração da variável monitorada.

Isto traduz uma dupla vantagem. Por um lado reduz-se a actividade MMS. Por outro lado melhoram-se as características de supervisão e monitorização do cliente MMS.

Contudo, também esta possível solução se depara com uma dificuldade: acontece que um servidor só pode enviar a um cliente pedidos de serviço MMS como sejam o *InformationReport*, o *EventNotification* ou o *UnsolicitedStatus*, se houver uma associação de aplicação activa com esse cliente MMS. Ora, para uma determinada sessão, a responsabilidade de estabelecimento de uma determinada associação de aplicação cabe ao cliente, pelo que elas teríam de ter um carácter permanente.

Outro argumento para rebater a hipótese de uso de gestão dinâmica de associações pode ser usado para outro tipo de aplicações que envolvem controlo realizado a partir de clientes MMS. Poderá dizer-se que isso não é necessário, uma vez que esta rede de *Field Bus* suporta actividade de controlo reflexa. No entanto, defendeu-se desde o início que o dispositivo de interligação não deverá limitar as opções de utilização do *Field Bus*.

Acontece, contudo, que encetar acções de controlo não é compatível quer com a situação de leitura referida anteriormente (do tipo *polling*), quer com uma actuação que necessitaría de um estabelecimento prévio de uma associação de aplicação, uma vez que tipicamente as acções de controlo envolverão E/S em módulos distintos (apesar de haver módulos mistos de entradas e saídas, como se referiu no capítulo 5). Sería inviável, para realizar uma acção de controlo, estabelecer duas ligações, uma para ir ler uma variável e depois outra para escrever a variável de saída.

Vem a propósito referir alguns estudos realizados no âmbito da análise de performance de redes MAP, suportando MMS [MASM - 88], [STWE - 88], [MOE - 90], [VACI - 90]. O estabelecimento de uma associação (serviço MMS *Initiate*) é precisamente um dos mais penalizadores em termos de tempo consumido, em parte porque no estabelecimento de uma associação de aplicação, existe primeiro o estabelecimento (negociação) de uma associação ao nível das camadas de transporte e só depois ao nível da aplicação.

Alguns desses estudos referem também algumas avaliações para tempo de atraso na execução do serviço de leitura MMS: leitura da primeira variável numa lista, 642ms; leitura da 100ésima variável numa lista, 645ms; leitura da 200ésima, 647ms; leitura de um grupo de 30 variáveis, 690ms.

Pode-se abrir aqui um parentisis para referir a possibilidade de os clientes poderem aceder a conjuntos de variáveis num só pedido MMS, podendo por via disso diminuir a degradação de performance. São exemplos disso o acesso distribuído (*Scatterd Access*) ou a leitura de listas de variáveis, mas estas teríam que fazer parte do mesmo VMD, como é óbvio.

Para além do mais existem algumas restrições, como sendo as de conformidade, ou seja, os clientes (ou mesmo um servidor genérico) poderão não suportar acesso distribuído.

Um último argumento contra esta hipótese da gestão dinâmica das associações, reside no facto de não haver possibilidade de estabelecer critérios para, ao nível do servidor, ir registando e desregistando as diferentes entidades de aplicação (endereços de apresentação) nos diferentes canais disponíveis. Enquanto que, havendo apenas um VMD e um endereço de apresentação, em todos os canais disponíveis se regista o mesmo endereço de apresentação.

Por fim, e para além destas questões do número de canais, e da impossibilidade de gerir dinamicamente as associações de aplicação, pode-se referir um último argumento em favor do mapeamento num só VMD. Esse argumento está relacionado com a estruturação hierarquica dos objectos MMS. Qualquer que seja a utilização de um servidor de MMS, e independentemente do tipo de acesso a variáveis utilizado, é necessário definir os tipos das variáveis. O número de variáveis associado a cada módulo é relativamente grande. Eventualmente, diferentes tipos de módulos suportarão diferentes tipos de variáveis.

No MMS, os tipos das variáveis são definidos em ASN.1 (*Abstract Syntax Notation One*) [ASN1 - 90] [BER - 90], apresentando por isso grande *overhead*. Contudo, módulos do mesmo tipo têm obviamente associados os mesmos tipos de variáveis. Acontece que em MMS, um objecto abstracto nomeado (caso do objecto *named type*) é específico do VMD, ou quando muito, específico de um domínio do VMD ou ainda específico de uma associação de aplicação. As duas primeiras implicam a adição de tipos a cada VMD. A utilização de tipos específicos das associações implicava a sua criação dinâmica por parte do cliente.

Na consideração de um só VMD não existe esta redundância. É importante realçar que a utilização de diferentes VMDs, implica também alocar estruturas de controlo que suportem os vários VMDs. Isto é, ocupam muito menos espaço de memória os objectos MMS relacionados com os diferentes módulos, se estiverem organizados num só VMD do que se estiverem organizados em diferentes VMDs.

Este facto tem algum relevo, se se pensar que qualquer interface MMS irá ocupar muita memória, quer corra ao nível do *host*, como é o caso da implementação considerada no capítulo 8, quer corra ao nível da carta de comunicações, como acontece na carta de comunicações OSI da Computrol LP-42, que com 2 MByte de RAM, permite nela carregar versões específicas do MMS-EASE, libertando o *host* para outras funções. A título de exemplo, poder-se-á dizer que o executável referido no anexo A, ocupa mais de 300Kbytes, e só suporta apenas 5 serviços MMS.

Pelo exposto, surge assim uma solução de compromisso razoável: a estruturação do servidor num só VMD, com uma entidade de aplicação (um só endereço de apresentação), mas

com possibilidade de diferentes invocações dessa AE, ou seja diferentes associações de aplicação. Esta proposta está retratada na *Fig. 7.4.*

7.2.3. Da Estrutura do VMD em Termos de Domínios

Após a opção pelo mapeamento das funcionalidades da rede de *Field Bus* num só VMD, é preciso decidir da constituição e estruturação desse VMD. Continuando numa abordagem hierarquica, poder-se-á avaliar a estruturação do VMD em diferentes domínios.

Em termos genéricos, a estruturação de um VMD em domínios poderá ser útil quando se pretende repartir as suas funcionalidades. No caso presente, esta hipótese pôr-se-ia em termos de mapeamento de cada módulo da rede de *Field Bus* num domínio diferente do VMD, uma vez que, por razões óbvias, não faz sentido pensar num mapeamento de ***N*** módulos da rede num só domínio.

Quais são as vantagens de mapear os diferentes módulos em diferentes domínios?

Por um lado, pode ser referida uma vantagem organizacional dos objectos MMS no VMD, vantagem essa intrínseca a uma estrutura hierarquica: a quantidade de variáveis inerentes a cada módulo pode ser bastante grande. Por exemplo, um módulo que tenha 32 E/S poderá ter, pelo menos, 32 variáveis complexas correspondentes aos parâmetros de cada uma delas e 32 variáveis referentes às E/S propriamente ditas.

Indirectamente, poderá aparecer uma vantagem de implementação que reside no facto de uma variável ser unicamente identificada dentro do domínio a que pertence. Por exemplo, as variáveis correspondentes a uma entrada digital de dois módulos FILBUS distintos, poderão ter rigorosamente o mesmo nome, uma vez que pertencerão a domínios diferentes, sendo por isso objectos da classe variável nomeada de âmbito específico do domínio. Assim, ao receber um pedido de escrita MMS, na estrutura disponível na indicação (objecto de transacção do MMS) obtem-se facilmente o domínio a que a variável pertence e por conseguinte a correspondente informação de mapeamento para o módulo da rede de *Field Bus*. Isto também pode ser conseguido se todas as variáveis pertencentes a um módulo tiverem uma parte do nome comum, e única.

A vantagem mais importante na estruturação em domínios reside na utilização dos serviços de gestão de domínios. Estes, permitindo manipular toda a informação relativa a um módulo através de uma só operação (apesar de que composta por uma sequência de serviços), permitem equacionar uma alternativa válida para a estruturação ou reestruturação dinâmica do servidor face a alterações de topologia e de composição da rede de *Field Bus*.

O modelo MMS de domínios prevê uma máquina de estados. Os domínios podem estar em 5 estados possíveis: *LOADING*, *COMPLETE*, *INCOMPLETE*, *READY*, *IN-USE*. No entanto,

estes diferentes estados resultam da utilização de serviços compósitos para fazer a manipulação do domínio, à excepção do último que corresponde a um estado no qual o domínio está a ser utilizado por uma invocação de programa. Não é por isso linear transpôr para o domínio alguma da modelização possível ao nível do VMD, como por exemplo o estado de um VMD. Essa informação de estado poderá ser associada a variáveis de aplicação, não estando contudo associadas a mecanismos do protocolo MMS, estando por isso exclusivamente dependentes da implementação.

A estruturação do VMD em domínios, sendo cada um deles correspondente a um módulo, à excepção de ocupar mais recursos de memória, não só não é limitativa das funcionalidades da rede de *Field Bus*, como potencia a utilização de serviços MMS que poderão aumentar a funcionalidade do sistema integrado.

Por fim, pode ser referida a inclusão no servidor de MMS de capacidades de execução de programas. A implementação de blocos de controlo ao nível do servidor (PIs, PIDs) pode ser equacionada no âmbito de modelos MMS. Tipicamente, as variáveis dos processos a controlar estarão relacionadas com módulos distintos. Sendo assim, poderão ser considerados domínios específicos, que conterão blocos de programa executáveis, por recurso a serviços MMS de Invocação de Programas.

7.2.4. Da Estrutura do VMD em Termos de Objectos de Acesso a Variáveis

Dos serviços MMS de acesso a variáveis apenas 3 dizem respeito à manipulação dos valores propriamente ditos das referidas variáveis. São eles os serviços *Read*, *Write* e *InformationReport*.

Quando um cliente MMS pretende obter o valor de uma determinada entrada de um módulo de entradas da rede de *Field Bus*, deverá fazer um pedido de leitura MMS *Read* ao dispositivo de interligação. De notar que a mesma informação poderá ser fornecida ao cliente por iniciativa do próprio dispositivo de interligação. Nesse caso, e em situações específicas, o dispositivo de interligação utilizará o serviço MMS *InformationReport*.

O serviço MMS *Read* vai ser utilizado pelos clientes MMS para obetr informação disponibilizada pela rede de *Field Bus*. Da mesma forma, o serviço MMS *Write* irá ser utilizado por clientes MMS para alterar informação ao nível da rede de *Field Bus*.

Concentremo-nos nas operações de consulta (utilização do MMS *Read*).

Vamos admitir que a informação a consultar não está directamente disponível no dispositivo de interligação (servidor de MMS). Por exemplo, vamos supor que um cliente MMS pretende conhecer a tabela de parâmetros de uma determinada E/S de um módulo específico. Nesse caso, o serviço MMS *Read* chegado ao dispositivo de interligação deverá ser mapeado

num serviço *DPA_DF* a enviar à rede de *Field Bus*. A resposta obtida da rede de *Field Bus*, serviço *RPA_CF*, conterá a informação a enviar na resposta ao pedido MMS *Read*.

No caso do cliente MMS pretender obter o estado de uma determinada entrada de um módulo de entradas, então o pedido MMS *Read* chegado ao dispositivo de interligação deverá ser mapeado noutros serviços a enviar à rede de *Field Bus*. Nesse caso, deverá ser enviado um dos seguintes serviços, consoante o modo de funcionamento da entrada: *DEA_60*, *DAL_61* ou *DCP_62*. A resposta a estes serviços conterá informação que será utilizada para responder ao pedido MMS *Read*.

É elementar que o mapeamento de um pedido MMS em procedimentos ao nível do dispositivo de interligação (por exemplo o envio de pedidos remotos à rede de *Field Bus*) deverá ser feito em função da variável MMS especificada no pedido MMS *Read*, uma vez que um mesmo serviço MMS deverá ser mapeado em diferentes serviços da rede de *Field Bus*. O que é dito aqui em relação ao serviço MMS *Read* é obviamente válido também para os serviços MMS *Write* e *InformationReport*.

O mapeamento vertical do serviço MMS *Read* da forma como foi sugerido, pode permitir não ter que implementar, no VMD, dos objectos MMS do tipo *Named Variable* (ver o modelo de acesso a variáveis do MMS - capítulo 6). De facto, se o mapeamento das diferentes respostas ('indicações') provenientes da rede de *Field Bus* for feito de imediato numa resposta ao pedido de MMS *Read*, não será necessário ter a variável MMS definida no VMD, e portanto proceder à actualização do valor dessa variável do VMD.

Ao contrário, se se admitir que de alguma forma os módulos da rede de *Field Bus* dispõem dos mecanismos que possibilitem informar o dispositivo de interligação de todas as alterações que se processem, por exemplo, nas suas entradas, então poderá ser necessário prever no VMD as variáveis MMS respectivas. Quando chegar ao dispositivo de interligação um pedido de MMS *Read* do estado de uma determinada entrada de um módulo, não será necessário mapear esse pedido MMS num serviço remoto de consulta à rede de *Field Bus*, uma vez que a informação está directamente disponível no dispositivo de interligação.

Neste último cenário, também não é estritamente necessário considerar, no VMD, os objectos MMS do tipo variável nomeada. Ao chegar uma 'indicação' da rede de *Field Bus* de uma alteração, a informação nela veiculada pode ser imediatamente mapeada num serviço MMS *InformationReport*. Em qualquer dos casos, os tipos MMS terão obviamente de ser definidos no VMD.

Esta discussão em torno da necessidade de considerar ou não, no VMD, os objectos MMS do tipo variável nomeada, depende, como ficou claro da análise anterior, das opções a tomar quanto ao funcionamento dinâmico do dispositivo de interligação. Por isso, as opções

nesse sentido só serão mais aprofundadamente analisadas nas secções seguintes desta dissertação.

O mesmo acontece em relação à definição de um conjunto de diferentes variáveis para cada tipo de módulo da rede de *Field Bus*. Pelas mesmas razões já apontadas no parágrafo anterior, a definição total das variáveis a considerar para cada tipo de módulo só será feita nas secções seguintes. De notar que independentemente da existência ou não dos objectos MMS do tipo variável nomeada, terão sempre que ser definidos nomes para as variáveis e os respectivos tipos MMS associados, uma vez que os mapeamentos são feitos consoante a variável a ser acedida. É importante salientar ainda que no caso deste dispositivo de interligação não é adequado utilizar o acesso a variáveis não nomeadas, isto é, acesso por intermédio do seu endereço. Este tipo de acesso só será razoável para outro tipo de servidores MMS, por exemplo, PLCs.

Assim, nesta secção serão referidos apenas alguns critérios que deverão ser tomados para a estruturação de cada domínio (correspondente a cada módulo) em variáveis e os seus diferentes tipos MMS.

Assim, para um cliente MMS obter o valor de uma determinada informação disponível na rede de *Field Bus*, ele deverá enviar um pedido MMS *Read* de uma variável MMS cujo nome corresponda univocamente a uma (ou mais) variável *Field Bus*, através de um mapeamento específico. De notar que a opção tomada pelo mapeamento de cada módulo num domínio diferente do VMD facilita parte do mapeamento, uma vez que a variável especificada no pedido MMS *Read* pertence a um domínio particular do VMD, isto é, um módulo particular da rede de *Field Bus*.

Um cliente MMS que pretenda ler entradas de um módulo, deverá fazer uma leitura a uma variável MMS denominada, por exemplo, *IOINDI*. Se pretender ler os valores respeitantes aos parâmetros de um módulo, então nesse caso deverá referir o nome de uma outra variável MMS, por exemplo *PARIOS*.

Tendo em conta que os serviços de manipulação das E/S da rede de *Field Bus* são orientados precisamente à E/S e não ao módulo, o nome das variáveis deverá ainda permitir este grau de mapeamento. Assim, os nomes das variáveis atrás referidas passaría a ser, por exemplo, *IOINDI_xx* e *PARIOS_xx* respectivamente, onde *xx* representaría o número da E/S.

Poder-se-á propor uma alternativa à consideração de tantas variáveis MMS, por exemplo, *PARIOS_xx* quantas as E/S de um determinado módulo, se se considerar uma variável MMS do tipo vector que aglutine todas as previamente consideradas individualmente.

A seguinte variável C representa a tabela de parâmetros de uma entrada de um módulo de entradas.

```
typedef struct _inp_par
{
        UBYTE           gen_inp_para;
        UBYTE           mult_alar_def;
        USHORT  inp_deb_del;
        struct
        {
                UBYTE           minutes;
                UBYTE           hours;
                UBYTE           days_of_week;
                BOOLEAN act_deact;
                BOOLEAN more_follows;
        } activity_period[28];
        struct
        {
                UBYTE   year;
                UBYTE   month;
                UBYTE   day;
                UBYTE   day_of_week;
                UBYTE   hour;
                UBYTE   minute;
                UBYTE   second;
        } auto_start_date;
        USHORT  int_bet_read;
        UBYTE   num_of_read;
        USHORT  min_count_s1;
        USHORT  max_count_s0;
        USHORT  min_count_s2;
        USHORT  max_count_s1;
        USHORT  min_count_s3;
        USHORT  max_count_s2;
        USHORT  lim_count_val_a[2];
        USHORT  lim_count_val_b[2];
        USHORT  step_evoulution [2];
        UBYTE           infor_mess[4];
        struct
        {
                UBYTE   dest;
                UBYTE   filbus_service;
                UBYTE   data[5];
        }predef_mess[8];
} inp_par;
```

Assim, a variável MMS correspondente aos parâmetros das estradas de um módulo de entradas, podería ser um vector de 16 dessas estruturas. Em C podería ser:

```
inp_par parios[16];
```

Esta estruturação em vector tem uma implicação grave em termos de MMS. Se um cliente MMS pretender ler um só parâmtero, terá de ser utilizado o acesso alternado do MMS, *alternate access*. Este tipo de acesso do MMS não é, na maioría das vezes, suportado pelas interfaces MMS. Uma interface de MMS que suporte este modo de acesso diz-se que suporta o parâmetro de conformidade (*parameter Conformance Building Block - CBB*) VALT.

Estes aspectos de conformidade no acesso a variáveis são relevantes em termos de critério geral a seguir na estruturação das variáveis, uma vez que eles serão exigidos a potenciais clientes MMS.

Por essa razão, em termos de conformidade no acesso a variáveis, este dispositivo de interligação deverá apenas exigir as funcionalidades mais básicas, como são os parâmetros de conformidade VNAM, STR1 e STR2. VNAM diz respeito ao acesso a variáveis nomeadas e os dois seguintes dizem respeito à utilização de variáveis MMS complexas respectivamente do tipo vector e estrutura. Em relação a estas últimas deverá ser exigido um baixo nível de imbrincamento (*nesting level*).

A estruturação em vector que foi anteriormente poderá ter vantagens também em termos estáticos, uma vez que mesmo que não estejam a ser utilizadas, serão consideradas todas as E/S de um módulo. Essa desvantagem é no entanto compensada pelo facto de se considerar um muito menor número de variáveis MMS (neste caso 16 contra 1).

Em termos estáticos tem muito maior relevo a complexidade que se dá à definição do tipo de uma variável. Por exemplo, em relação à estruturação apresentada para a variável que podería representar uma tabela de parâmetros de uma E/S, atrás definida como sendo do tipo *inp_par*, podem ser feitos alguns reparos.

Do ponto de vista do cliente MMS, teria algum interesse em definir essa variável dessa maneira. Contudo, uma especificação desse tipo, apresenta inúmeras desvantagens, quando comparada com uma outra mais à imagem da própria tabela de parâmetros de 256 *bytes* suportada ao nível do *Field Bus*, ou seja, um simples *array* de 256 *bytes*. Esta última opção apresenta óbvias vantagens, querem estáticas quer dinâmicas.

As vantagens estáticas resultam de todos os módulos passarem a ter o mesmo o mesmo tipo MMS para a especificação das suas tabelas de parâmetros, e pelo facto de, como já foi referido, uma estrutura complexa como a anterior, apresentar um enorme *overhead* em termos de definição ASN.1. Este *overhead* pode inclusivamente ser crítico na transmissão dos serviços, uma vez que se podem ultrapassar comprimentos limitativos de PDUs a diferentes níveis da *stack* OSI.

Por outro lado, não é menos relevante a muito menor quantidade de código de *software* necessário. A conversão bidireccional dos diferentes tipos para *arrays* de 256 *bytes*, não sendo minimamente elaborada no que se refere a algorítmo, é no entanto extensa e exaustiva, como é fácil de antever. Mais grave ainda é constatar que consoante o tipo de módulo, as funções de conversão dos parâmetros teríam de ser diferentes, reforçando por isso, o problema do espaço ocupado pela aplicação.

As vantagens em termos dinâmicos resultam por isso de uma maior rapidez de mapeamento. Existe um serviço da rede de *Field Bus* que permite a escrita de parâmetros num módulo, o *PAR_81*. Este serviço permite escrever qualquer quantidade de *bytes* desde que consecutivos na tabela de parâmetros, uma vez que o serviço especifica um *offset* e um número de *bytes*.

Ainda no que diz respeito aos parâmetros de conformidade no acesso a variáveis, será de evitar, tanto quanto o possível, níveis de imbrincamento (*nesting level*) superiores a 1 ou 2. A variável exemplificada anteriormente apresenta um nível de imbricamento igual a 3. A justificação reside mais uma vez no facto de algumas implementações de MMS serem limitadas neste aspecto. Refira-se que a codificação dos tipos utilizando as regras básicas de codificação ASN.1 [BER - 90], é complicada pelo seu carácter recursivo.

A maioría dos serviços da rede de *Field Bus* são orientados à E/S. No entanto alguns veiculam informação referente a todas as E/S, como por exemplo o *DIG_B9* ou o *RCL_75*. As opções por definir variáveis globais ou individuais ou ter as duas simultaneamente será também dependente de considerações que serão feitas nas secções seguintes.

Cabe aqui no entanto referir, que o modelo de acesso a variáveis do MMS, permite efectuar acesso distribuído (*Scatterd Access*). Sendo assim, em vez de se definir uma variável global, poder-se-á pensar na definição de um objecto de acesso distribuído a variáveis individuais. Cabería definir local ou remotamente um objecto de acesso distribuido que envolvesse todas as variáveis individuais. Quem fala num objecto distribuido, também poderá falar num objecto lista de variáveis.

Mais uma vez aqui se poderão colocar os problemas de conformidade. Os parâmetros de conformidade exigidos a um cliente iríam ser o VSCAT (acesso a variáveis distribuídas) e o VLIST (acesso a listas de variáveis). Pelas razões já apontadas anteriormente isso deverá ser evitado. Para além do mais, este tipo de acesso tornaría muito complicado o mapeamento vertical, uma vez que tipicamente uma variável implicaría a emissão de um conjunto diverso de comandos à rede de *Field Bus*. A noção dessa dificuldade resultará mais clara com a leitura das restantes secções deste capítulo.

Em relação ao que se analisou até agora, poder-se-ão extrair alguns critérios genéricos para a definição da variáveis MMS.

Por um lado a sua estruturação deverá de alguma forma ser a mais adequada aos serviços disponibilizados pela rede de *Field Bus*, uma vez que diferentes nomes de variáveis servirão para mapear em diferentes serviços da rede de *Field Bus*. Por outro lado a composição das diferentes variáveis deverá facilitar ao máximo as operações de conversão de e para os diferentes formatos suportados pelos serviços da camada de aplicação da rede de *Field Bus*. Um exemplo disso diz

respeito aos parâmetros das E/S. Apesar de tudo, e não obstante não considerar esse como um requisito determinante, eventualmente compromissos deverão ser considerados entre facilidades de mapeamento e redução do número total de variáveis MMS a considerar. Exemplos desta análise serão patenteados nas secções seguintes.

Por fim, e ainda relacionado com as variáveis MMS, poderá ser questionada a necessidade de considerar no VMD os próprios objectos de acesso às variáveis. Poderia bastar ter a definição dos tipos. Por exemplo, chegava ao dispositivo de interligação um pedido de leitura da variável *PARIOS_00*, especificada como sendo do tipo MMS *PARAM*. Ao receber a indicação desse pedido, o dispositivo de interligação enviaría um pedido *DPA_DF* ao módulo e entrada respectivo. A resposta, *RPA_CF*, sería utilizada para preencher a estrutura de resultado de acesso que estaría presente na resposta ao pedido de leitura.

Nas secções que se seguem, demonstrar-se-á que independentemente da opção em relação ao VMD (coerente ou não com o estado da rede de *Field Bus*), permitindo em termos gerais a primeira opção fazer mapeamentos síncronos de acesso a variáveis sem mapeamento em serviços remotos da rede de *Field Bus* e a segunda opção implica, dadas as características de comunicação entre o dispositivo de interligação e a rede, serviços MMS concorrentes, que seja sempre necessário prever no VMD os objectos de acesso as variáveis, uma vez que é preciso manter no VMD a informação do valor das diferentes variáveis.

Estas questões resultarão muito mais claras com a leitura das secções subsequentes.

Por fim, pode ser questionado o âmbito a dar aos objectos MMS de definição do tipo das variáveis. Referiu-se anteriormente que sería uma vantagem considerá-los específicos do VMD, evitando por isso redundâncias. Por outro lado, esta opção poderá retirar alguma da vantagem na estruturação em domínios (*download* e *upload* de toda a informação de um módulo).

7.3. Manipulação dos Objectos MMS do Servidor

7.3.1. Da Coerência entre VMD e Dispositivos Reais

O VMD contém objectos abstractos MMS que serão manipulados quer por potenciais clientes MMS quer por entidades pertencentes à rede de *Field Bus*. Do ponto de vista funcional, os serviços MMS mais importantes serão, sem dúvida, os de acesso a variáveis.

Por exemplo, um cliente MMS pretende efectuar uma leitura remota do estado de uma entrada de um determinado módulo de entradas. Admitindo a estruturação do servidor de acordo com a análise feita nos pontos anteriores, o cliente efectua um pedido de serviço MMS *Read*, a

uma determinada variável do VMD que representa essa entrada do módulo. Caberá ao servidor enviar uma resposta a esse serviço MMS contendo o valor dessa variável.

O processamento ao nível do servidor poderá ser simplificado se se garantir que o VMD reflecte continuamente (na medida em que este termo pode ser aplicado) o estado dos dispositivos reais. Nessa situação, ao ser recebida a primitiva de indicação de *Read*, é obtida a especificação de acesso à variável nomeada, cujo valor está disponível no servidor, e imediatamente enviada a resposta sem necessidade de efectuar uma leitura remota ao próprio módulo. A *Fig. 7.5* ilustra esse cenário.

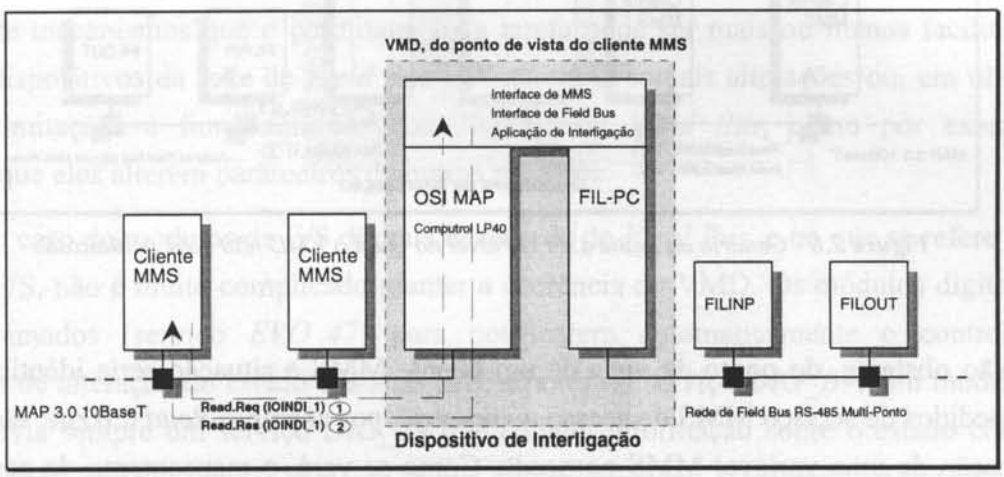


***Figura 7.5** - Cenário de Leitura de Variável no Qual o VMD Está Actualizado*

Se essa variável não reflecte o estado real do recurso que ela modeliza (neste caso uma E/S de um módulo), então a chegada de um pedido de leitura MMS *Read* terá de ser processado, ao nível do dispositivo de interligação, de forma diversa. Neste caso, antes de responder ao pedido, o dispositivo de interligação terá de obter remotamente o valor da variável pretendida. Para o exemplo concreto, a chegada desse pedido de leitura MMS deverá implicar o envío de um comando desta rede de *Field Bus*, comando esse correspondente a um pedido de leitura da entrada em causa. A *Fig. 7.6,* ilustra este último cenário.

A consideração deste último cenário, isto é, o do mapeamento de uma indicação MMS recebida num comando da rede de *Field Bus*, torna oportuno retomar a discussão sobre a estruturação do VMD em termos de objectos MMS de acesso a variáveis. Se fosse possível generalizar a todas as situações de manipulação de variáveis este tipo de mapeamento remoto, e se não houvesse outras 'indicações' da rede de *Field Bus* a chegar ao servidor não relacionadas com esse pedido, então não sería necessária a consideração de objecto de acesso a variáveis no VMD. Como já foi referido, nesse caso bastaría definir no VMD os objectos da classe Tipo

Nomeado (*Named Type*), conseguindo-se assim uma redução importante dos recursos (neste caso memória) utilizados. Esta redução é ainda maior pelo facto de muitas variáveis serem do mesmo tipo.

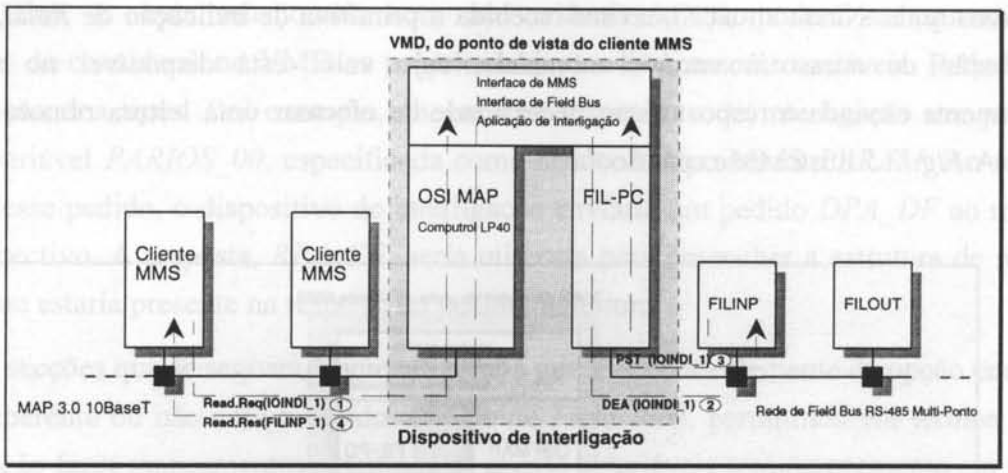


Figura 7.6 *- Cenário de Leitura de Variável no Qual o VMD não Está Actualizado*

Não obstante, do ponto de vista de um cliente MMS a situação seria idêntica, uma vez que nos pedidos de serviço MMS de acesso a variáveis, por exemplo *Read* e *Write*, ele incluiria a especificação de uma variável MMS nomeada. Como se verá, o mapeamento de comunicações vertical aqui proposto será em grande medida baseado no nome das variáveis. Dependendo da variável a ser acedida, um mesmo pedido MMS poderá ser mapeado em diferentes pedidos à rede de *Field Bus*. Assim, do ponto de vista do cliente MMS e no que diz respeito à sintaxe e semântica dos serviços da camada de aplicação, as situações retratadas nas *Fig. 7.5* e *Fig. 7.6* são, usando uma terminologia cara aos sistemas de computação, transparentes. Obviamente que os tempos de espera pela resposta serão tipicamente mais curtos no caso da *Fig. 7.5* do que no caso da *Fig. 7.6*.

O VMD deverá mapear os recursos físicos desta rede de *Field Bus*. Como é que o estado dos recursos físicos são alterados e em que medida se poderá manter a coerência entre eles e o VMD?

Podem ser consideradas duas grandes causas para a alteração do estado dos recursos físicos desta rede de *Field Bus*. Uma, diz respeito a alterações resultantes da iniciativa de clientes MAP/MMS. Um cliente MMS que deseje alterar o estado de um determinado módulo na rede de *Field Bus* (p. ex., alteração de uma saída de um módulo de saídas), faz um pedido de escrita de uma variável MMS do servidor, pedido esse que é recebido pelo dispositivo de interligação. Esta

situação não é problemática, uma vez que o 'pedido de alteração do VMD' passa pelo próprio dispositivo de interligação.

O mesmo não se poderá dizer em relação a alterações feitas por iniciativa de entidades do âmbito da rede de *Field Bus*. Por um lado, o estado das variáveis é função do processo de aplicação com elas relacionado (variáveis de entrada p. ex.), ou de programas de aplicação residentes nos próprios módulos. Por outro lado, os próprios módulos da rede de *Field Bus* podem alterar o estado de outros módulos, desde o estado de E/S até parâmetros relacionados com o seu próprio funcionamento.

Se for um requisito *sine qua non* manter a coerência atrás referida, é preciso dotar o servidor de mecanismos que o permitam. Esta tarefa pode ser mais ou menos facilitada, se os próprios dispositivos da rede de *Field Bus* reportarem eventuais alterações ou, em último caso, fazendo limitações à funcionalidade dos dispositivos *Field Bus*, como por exemplo, não deixando que eles alterem parâmetros de outros módulos.

No caso do módulos de E/S digitais desta rede de *Field Bus*, e no que se refere ao estado das suas E/S, não é muito complicado manter a coerência do VMD. Os módulos digitais podem ser programados (serviço *EVO_47*) para notificarem automaticamente o controlador, da ocorrência de alteração do estado das suas E/S, através do serviço *DIG_B9*. Um módulo de E/S digitais envia sempre um serviço *DIG_B9* que contem informação sobre o estado corrente das suas E/S, bem como a indicação das E/S que foram alteradas. Com esta informação pode ser actualizada uma variável do VMD que representa todas as E/S do módulo. Isto quer dizer que mesmo determinada actividade reflexa pode ser detectada, isto é, por exemplo alterações do estado de saídas como resultado de mensagens reflexas a partir de um módulo de entradas. Numa situação dessas, o módulo de saídas reportaría automaticamente essa alteração.

Em relação aos parâmetros dos módulos, não será tão fácil arranjar mecanismos ao nível do servidor que, perante alteração de parâmetros como resultado de actividade reflexa, lhe permitam manter o VMD actualizado. Uma hipótese pode ser, pura e simplesmente, não permitir este tipo de situações. Considerando o tipo de parâmetros dos módulos (ver capítulo 5), conclui-se logo que sería uma restrição pouco razoável. Um exemplo, de entre muitos outros possíveis, pode ser o de uma saída de um módulo de saídas, onde existem parâmetros de temporização da oscilação da saída.

De qualquer forma, pode ser possível controlar a alteração de parâmetros, uma vez que com as mensagens reflexas é enviada ao controlador uma mensagem *ALM_B0*. Ao receber esta mensagem, poder-se-ía ir verificar se, nos parâmetros do módulo que enviou as mensagens reflexas (em geral um módulo de entradas), haverían alguma mensagem de alteração de parâmetros de um outro módulo, ou seja, o comando *PAR_81*. Por outro lado, não é menos

verdade que uma alteração de um parâmetro de um módulo implica, da parte deste, uma notificação ao controlador com a mensagem *ACK_91*. Trata-se inclusivamente da única situação de 'escrita' num módulo da rede de *Field Bus* que é de alguma forma confirmada.

Existem, no entanto, algumas situações para as quais não é tão óbvia a manutenção da coerência do VMD. São exemplo disso, situações que envolvam variáveis de entrada a funcionar como contador (quer de impulsos quer de tempo) ou variáveis de entrada de grandezas analógicas. Refira-se novamente, que este estudo, apesar de incidir em particular nos módulos de grandezas digitais, pretende ser tão generalista quanto o possível.

Para as entradas a funcionar como contador, é possível programá-las para enviar leituras automáticas. Para tal, é preciso programar o módulo com o serviço *EVO_47*, e ter os parâmetros correspondentes na tabela de parâmetros com valores diferentes de '0' (ver capítulo 5). Por exemplo, pode-se programar o módulo para enviar uma contagem de 1 em 1 segundos (período mínimo), ou de ***n*** em ***n*** incrementos do contador. Basicamente, isto corresponde a ter a mesma funcionalidade que para a alteração do estado de variáveis digitais, ou seja *DIG_B9*.

Nesse caso, serão recebidos as 'indicações' da rede de *Field Bus PCI_C2* e *PCT_C3*, que contêm informação de contagem. Só fará sentido dizer que o VMD está actualizado se o intervalo temporal entre notificações periódicas for adequado à aplicação. Para uma determinada aplicação, uma leitura periódica feita de 30 em 30 segundos pode ser razoável, noutra aplicação pode não o ser de todo. Por outro lado, se a programação de leituras for feita em termos de incrementos da contagem, pode-se optar por um incremento de uma unidade.

Poder-se-á concluir que, em termos genéricos e dentro de algumas limitações, é possível considerar que o VMD que mapeia em termos abstractos a rede de *Field Bus* aqui considerada, pode ser mantido coerente com o estado dos recursos físicos, isto é, o estados das E/S e os valores dos parâmetros dessas E/S, independentemente da existência ou não de actividade reflexa.

Assim sendo, todo e qualquer serviço MMS, podería ser processado em modo síncrono, com particular relevo para os serviços de acesso a variáveis. Obviamente que esta afirmação se adequa melhor a pedidos de consulta do VMD, uma vez que a escrita terá de ser mapeada no envio de comandos específicos à rede de *Field Bus*.

No modo síncrono de utilização de MMS, a recepção de uma indicação MMS, por exemplo um pedido de leitura de variável, implica o envio da resposta a esse pedido antes de ser processada outra indicação MMS. Se o valor da variável pretendida existe disponível no VMD essa situação será perfeitamente plausível. No caso da recepção de um pedido de escrita, implicando o envio de um ou mais comandos à rede de *Field Bus*, o modo síncrono também será possível se não se pretender uma confirmação remota da escrita. Processando as primitivas MMS

em modo síncrono, o servidor de MMS, ou seja, o dispositivo de interligação, só terá no máximo uma indicação MMS pendente.

Neste caso, o VMD conteria variáveis MMS, cujo valor seria 'refrescado' por indicações de notificação automática provenientes da rede de *Field Bus*, ou alterado por escritas MMS feitas por clientes MMS, este último caso, com eventuais mapeamentos em comandos não confirmados da rede de *Field Bus*. As consultas ao dispositivo de interligação seríam caracterizáveis pela *Fig. 7.5*, e as alterações feitas a partir de um cliente MMS seríam caracterizáveis pela *Fig. 7.7*.

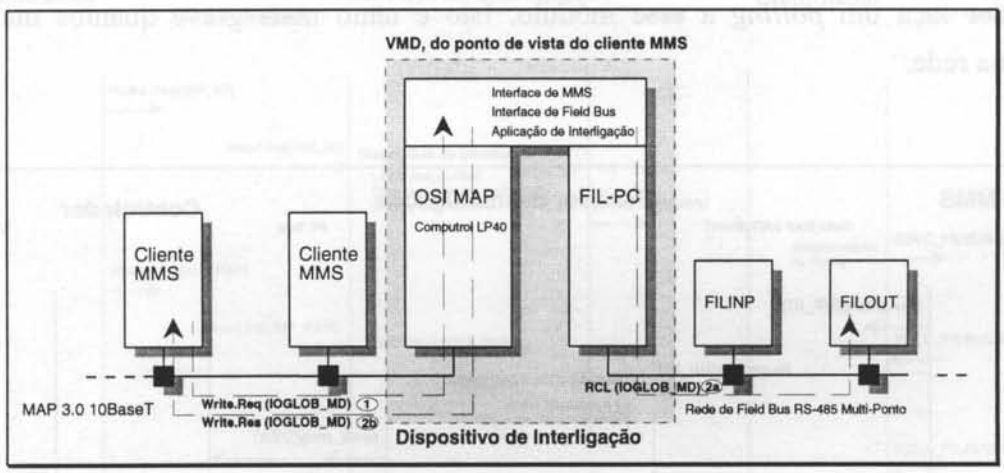


***Figura 7.7** - Cenário de Escrita de Variável sem Confirmação Remota*

De notar que a maioría das funcionalidades mapeáveis no serviço de escrita MMS, por exemplo, a escrita de variáveis de saída dos módulos de saída (*RCL_75*, *RAN_B6*, *ORR_B7* ou *RXO_B8*) ou a escrita de valores de contagem iniciais dos contadores (*WCI_A0* e *WCT_A1*) não são confirmados pela rede de *Field Bus*, pelo que a utilização do serviço MMS *Write* em modo síncrono também sería viável. Após a recepção da indicação de *Write*, actualiza-se o VMD, envia-se o serviço *Field Bus* correspondente, e responde-se ao cliente MMS com a primitiva de resposta do *Write*. Eventualmente, na escrita de parâmetros (comando *PAR_81*), pode-se considerar que existe uma confirmação da parte do módulo da rede de *Field Bus*. Aliás, se os módulos estiverem a funcionar com *EVO_47*, pode-se considerar como confirmação de uma escrita de saídas digitais, a recepção do *DIG_B9* respectivo.

Em qualquer destes últimos dois casos, relacionados com as leituras dos contadores e com a confirmação remota de escritas, o suporte de indicações MMS em modo assíncrono será necessário, uma vez que enquanto se espera o serviço de confirmação da situação pretendida, poderão chegar outras 'indicações' do *Field Bus*, respeitantes a outras situações e que precisam de ter processamento adequado. Não faz sentido efectuar qualquer tipo de processamento,

respeitante a outras transacções, dentro de uma função de indicação MMS. Por outro lado, é preciso ter em mente que só se pode ler uma mensagem *Field Bus* de cada vez.

Outras argumentos poderão ser apontados em favor da consideração, se necessário, do processamento assíncrono de indicações MMS.

Um diz respeito a eventuais sobrecargas de mensagens, como resultado do uso generalizado de mecanismos de notificação automática e de leituras periódicas de mensagens. A dimensão dos *buffers* de mensagens dos diferentes módulos, é limitada. Por exemplo, num módulo de 16 ou 32 entradas, rapidamente se pode atingir o limite de mensagens, até que o controlador faça um *polling* a esse módulo. Isto é tanto mais grave quantos mais módulos existam na rede.

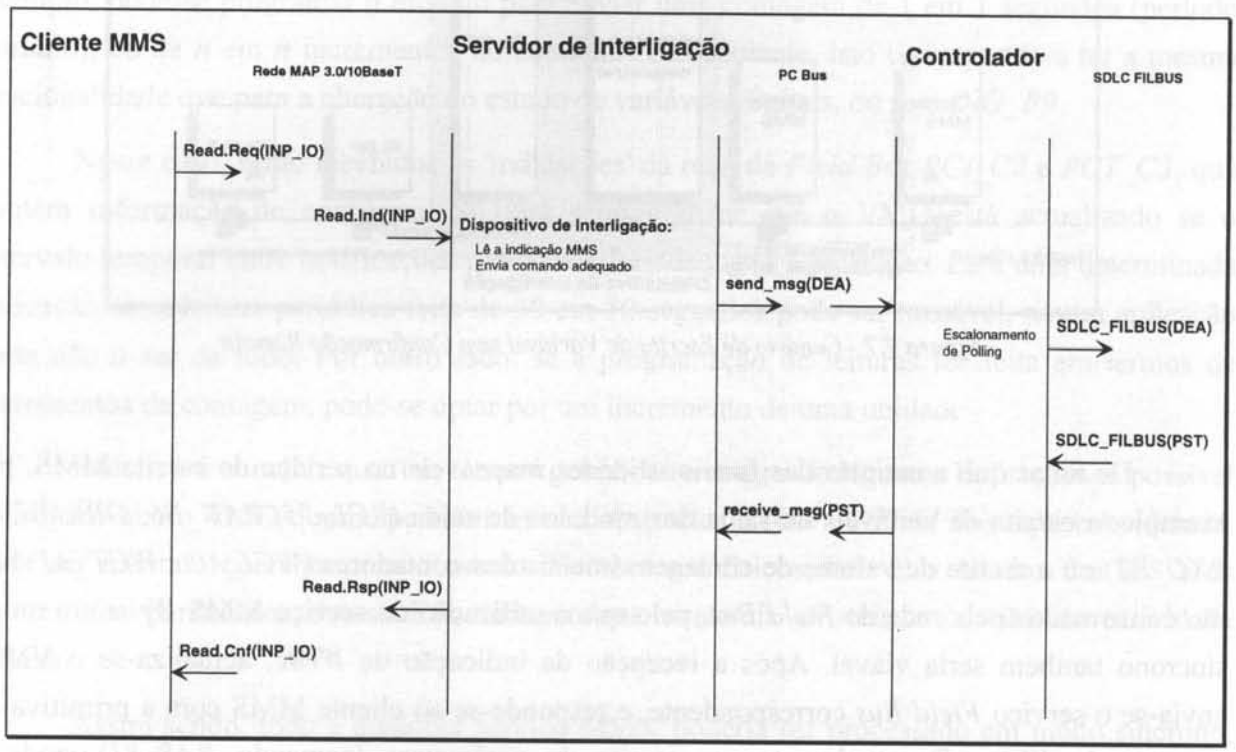


***Figura 7.8** - Sequência de Transacções Referentes a um Pedido de Leitura do Estado de Uma Entrada*

Quanto maior for a frequência de *polling* dos módulos escravos, menor será esse risco, ao nível desses módulos. No entanto, este será tanto maior ao nível do controlador. É preciso ter em mente a necessidade de 'multiplexar' o atendimento de indicações MMS com 'indicações' da rede de *Field Bus*. Para atender a catadupa de informação proveniente da rede de *Field Bus*, terá de se penalizar o atendimento a indicações MMS, e por isso a performance de interligação ao MAP/MMS. Acrescente-se que todas as actividades de notificação automática da rede de *Field Bus* são optativas (existe o serviço *SEV_48* para cancelar o efeito do *EVO_47*, ...), e que desde o

início deste capítulo se tem tentado nortear a especificação deste dispositivo de interligação pelo princípio de não restringir as funcionalidades da rede de *Field Bus*.

Propõe-se então uma arquitectura de mapeamento vertical de comunicações, na qual o atendimento das indicações provenientes de ambas as arquitecturas, se faz em modo assíncrono. A resposta a indicações MMS pendentes será sincronizada por 'indicações' provenientes da rede de *Field Bus* que veiculam informação associada ao pedido MMS.

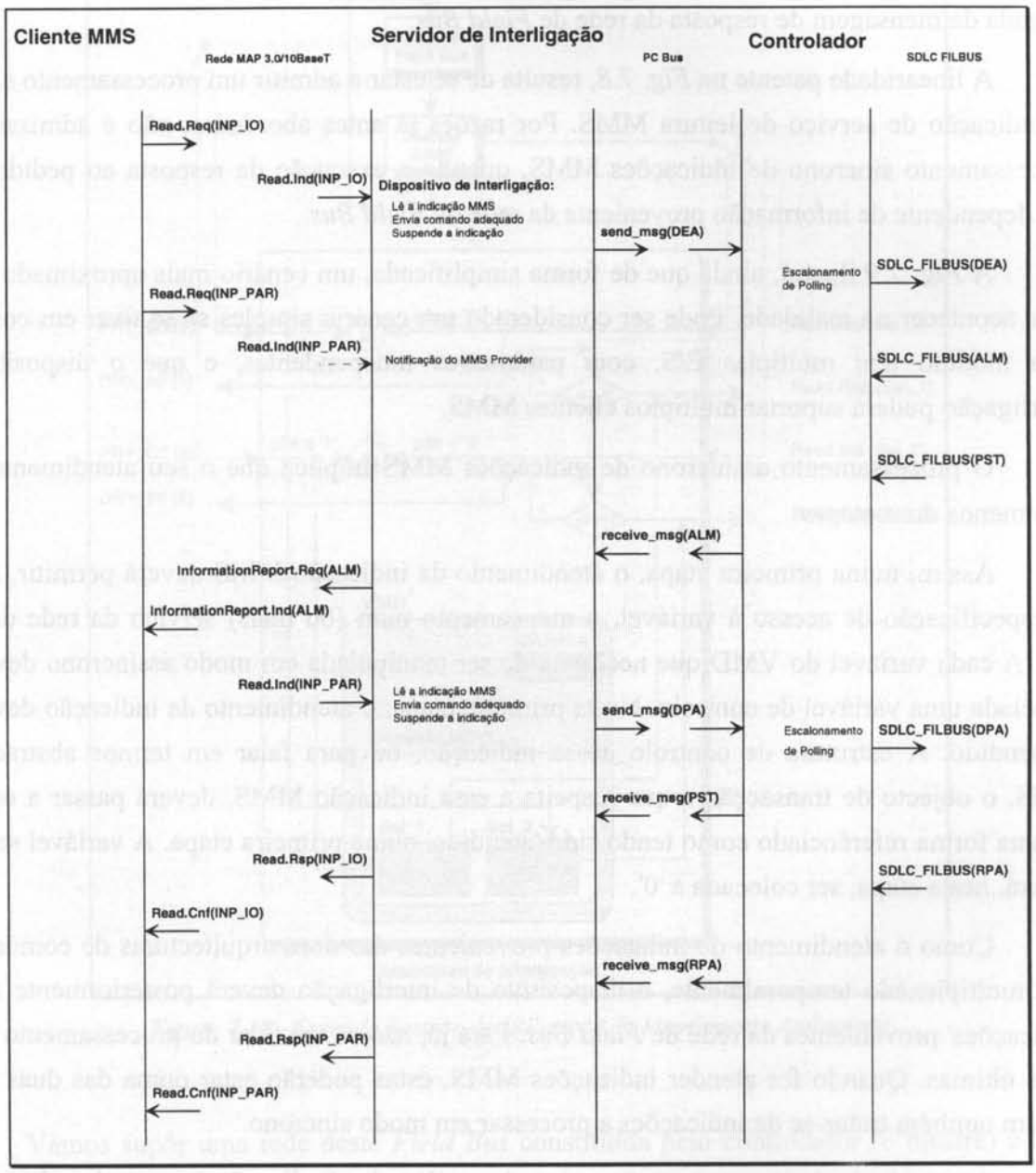


***Figura 7.9** - Ex. de Cenário de MMS em Modo Assíncrono com Entrelaçamento de Mapeamentos Verticais*

7.3.2. Da Utilização do MMS em Modo Assíncrono

A *Fig. 7.8*, ilustra um mapeamento referente a uma situação de pedido de leitura de uma E/S por parte de um cliente, com necessidade de proceder à leitura remota. Ao receber a indicação de MMS *Read*, o dispositivo de interligação irá, a partir da especificação da variável que é veiculada no pedido, construir a mensagem adequada a enviar para a rede de *Field Bus*, isto é, uma mensagem que contem o código de comando, o endereço do módulo e a E/S do módulo. O envio da primitiva de resposta MMS será condicionado, no que se refere à sua execução, pela chegada da mensagem de resposta da rede de *Field Bus*.

A linearidade patente na *Fig. 7.8*, resulta de se estar a admitir um processamento síncrono da indicação de serviço de leitura MMS. Por razões já antes abordadas, não é admissível um processamento síncrono de indicações MMS, quando a execução da resposta ao pedido MMS fica dependente de informação proveniente da rede de *Field Bus*.

A *Fig. 7.9* ilustra, ainda que de forma simplificada, um cenário mais aproximado ao que pode acontecer na realidade. Pode ser considerado um cenário simples se se tiver em conta que cada módulo tem múltiplas E/S, com parâmetros independentes, e que o dispositivo de interligação puderá suportar múltiplos clientes MMS.

O processamento assíncrono de indicações MMS implica que o seu atendimento tenha pelo menos duas etapas.

Assim, numa primeira etapa, o atendimento da indicação MMS deverá permitir, a partir da especificação de acesso à variável, o mapeamento num (ou mais) serviço da rede de *Field Bus*. A cada variável do VMD que necessite de ser manipulada em modo assíncrono deverá ser associada uma variável de controlo. Nesta primeira etapa, o atendimento da indicação deverá ser suspendido. A estrutura de controlo dessa indicação, ou para falar em termos abstractos do MMS, o objecto de transacção a que respeita a essa indicação MMS, deverá passar a estar, de alguma forma referênciado como tendo sido 'atendido' numa primeira etapa. A variável sentinela deverá, nesta etapa, ser colocada a '0'.

Como o atendimento de indicações provenientes das duas arquitecturas de comunicação será multiplexado temporalmente, o dispositivo de interligação deverá posteriormente atender 'indicações' provenientes da rede de *Field Bus*. Para já, não se vai falar do processamento a dar a estas últimas. Quando for atender indicações MMS, estas poderão estar numa das duas etapas. Podem também tratar-se de indicações a processar em modo síncrono.

As variáveis de sentinela serão colocadas a '1' pelas 'indicações', com ela relacionadas, provenientes da rede de *Field Bus*.

A segunda etapa para as indicações MMS a serem processadas em modo assíncrono, consiste, de uma forma simplificada, em verificar se a variável sentinela referida anteriormente já vale '1'. Se sim, processa-se a resposta ao pedido MMS, e a indicação é dada como processada, deixando de estar pendente. Deverá fazer parte do processamento das 'indicações' da rede de *Field Bus*, entre outros aspectos, a colocação da tal variável de sentinela a '1'.

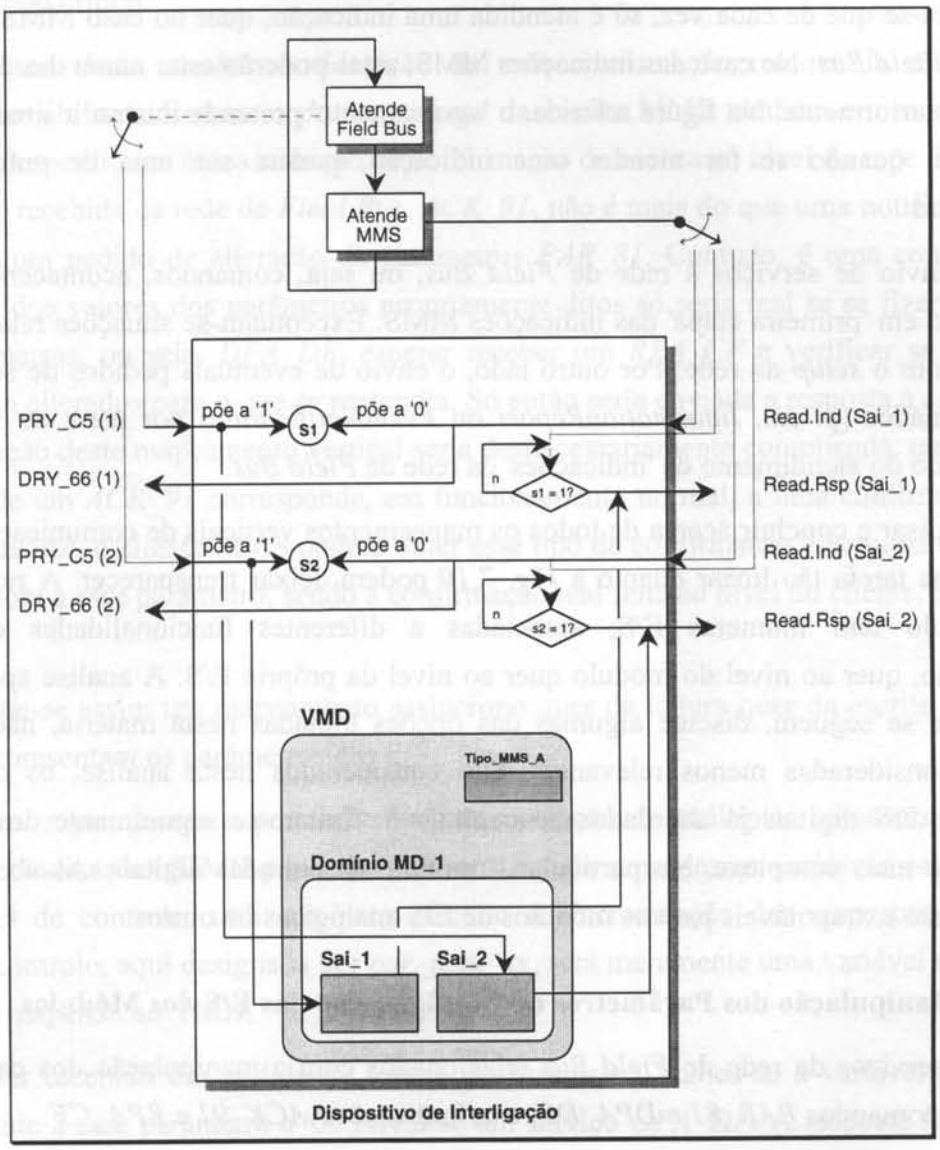


Figura 7.10 *- Exemplo Restrito de Utilização de Mapeamento Assíncrono*

Vamos supôr uma rede deste *Field Bus* constituída pelo controlador (o mestre) e um só módulo de E/S, neste caso de saídas. Quanto aos serviços de aplicação, considerem-se apenas o *DRY_66*, para pedir o valor de uma determinada saída, e o *PRY_C5*, que será a resposta ao

pedido anterior. O módulo de saídas está mapeado num domínio do VMD. Nesse domínio estão definidas duas variáveis MMS, SAI_1 e SAI_2, que modelizam as duas únicas saídas do módulo.

A *Fig. 7.10* ilustra o mapeamento assíncrono de leituras MMS efectuadas a partir de um determinado cliente MMS. As variáveis de sentinela s1 e s2 servirão para sincronizar as respostas aos pedidos MMS. Está patente, nessa mesma figura, a multiplexação temporal do atendimento de indicações MMS e atendimento de 'indicações' provenientes da rede de *Field Bus*. Admita-se que de cada vez, só é atendida uma indicação, quer no caso MMS quer no caso da rede de *Field Bus*. No caso das indicações MMS, estas poderão estar numa das duas 'etapas' já referidas anteriormente. Na figura referida, o 'agulhamento' pretende ilustrar a situação de lógica *OR*, isto é, quando se for atender uma indicação, poderá ser uma de entre várias das representadas.

O envio de serviços à rede de *Field Bus*, ou seja, comandos, acontecerá aquando do atendimento em 'primeira etapa' das indicações MMS. Exceptuam-se situações relacionadas, por exemplo, com o *setup* da rede. Por outro lado, o envio de eventuais pedidos de serviços MMS não confirmados (p. ex., *InformationReport* ou *EventNotification*) por parte do servidor, será feito aquando do atendimento de 'indicações' da rede de *Field Bus*.

Analisar e concluir àcerca de todos os mapeamentos verticais de comunicações não é, no entanto, uma tarefa tão linear quanto a *Fig. 7.10* poderá deixar transparecer. A realidade é que cada módulo tem inúmeras E/S, associadas a diferentes funcionalidades e modos de programação, quer ao nível do módulo quer ao nível da própria E/S. A análise apresentada nas secções que se seguem, discute algumas das opções tomadas nesta matéria, não abrangendo situações consideradas menos relevantes. São considerados nesta análise, os dois tipos de módulos de E/S digitais já abordados no capítulo 5. Tratam-se seguramante dos módulos de modelização mais complexa, em particular o módulo de entradas digitais. As abordagens aqui analizadas são extrapoláveis para os módulos de E/S analógicas ou outros.

7.3.3. Da Manipulação dos Parâmetros de Configuração das E/S dos Módulos

Os serviços da rede de *Field Bus* relacionados com a manipulação dos parâmetros das E/S, são os comandos *PAR_81* e *DPA_DF*, e as 'indicações' *ACK_91* e *RPA_CF*.

Estes serviços são todos orientados às E/S pelo que, seguindo as orientações apontadas para o acesso a variáveis, se deverá considerar uma variável MMS por cada tabela de parâmetros associada a uma determinada E/S. Uma vez que o nome das variáveis deverá ser utilizado para informação de mapeamento, essa variável poderá designar-se por *PARIOS_xx*, onde *xx* corresponde ao número da entrada. Note-se que *PARIOS_00* corresponderá aos parâmetros gerais de cada módulo (ver capítulo 5).

Como já foi referido, sería possível suportar em modo síncrono as indicações MMS de acesso a variáveis que representam os parâmetros das E/S de um módulo da rede de *Field Bus*. No entanto, e para estas varáveis em particular, as situações leitura por parte de um cliente MMS não serão muito frequentes, pelo que a opção por uma leitura MMS em modo assíncrono não será errada face à necessidade de implementar os mecanismos de manutenção de coerência já referidos anteriormente. Mais ainda, essa leituras não serão porventura incluídas no conjunto das acções de tempo crítico.

Por outro lado, a não consideração de uma confirmação remota da escrita de um parâmetro, possibilitando assim o processamento da escrita MMS também em modo síncrono, sería estar a desperdiçar a única situação de confirmação de escrita ao nível da rede de *Field Bus*. A 'indicação' recebida da rede de *Field Bus*, *ACK_91*, não é mais do que uma notificação de que foi recebido um pedido de alteração de parâmetros *PAR_81*. Contudo, é uma confirmação. A confirmação dos valores dos parâmetros propriamente ditos só sería real se se fizer uma leitura desses parâmetros, ou seja, *DPA_DF*, esperar receber um *RPA_CF* e verificar se de facto os valores foram alterados para o que se pretendia. Só então sería enviada a resposta à escrita MMS. A sincronização deste mapeamento vertical sería desnecessariamente complicada, tanto mais que a recepção de um *ACK_91* corresponde, em funcionamento normal, a uma confirmação válida. Em última análise, o cliente MMS poderá obter esse tipo de confirmação efectuando em seguida a leitura remota a esse parâmetro, sendo a confirmação real feita ao nível do cliente, e só se este o desejar.

Propõe-se assim um mapeamento assíncrono quer da leitura quer da escrita de variáveis MMS que representam os parâmetros das E/S.

Assim, e por forma a possibilitar a sincronização de múltiplas respostas referentes a outras tantas indicações MMS pendentes, deverá a cada uma dessas variáveis estar associada uma variável de controlo, utilizada para efectuar a sincronização das respostas MMS. Essa variável de controlo, aqui designada por *par_updt_xx*, será meramente uma variável da aplicação, por isso não mapeada no VMD.

Ao ser recebido um pedido de leitura de parâmetro, coloca-se a variável *par_updt_xx* correspondente a esse parâmetro a '0'. Envia-se um serviço *DPA_DF*. A resposta fica suspensa, até que *par_updt_xx* valha '1'. Quando for recebida a indicação da rede *Field Bus*, *RPA_CF*, respeitante ao módulo e entrada visadas, deverá ser actualizada a variável MMS do VMD que mapeia esse parâmetro e colocada a '1' a variável de controlo de sincronização associada. Quando ser for outra vez processar a indicação MMS pendente, se essa variável de controlo vale '1', envia-se a resposta MMS, na qual é veiculado o resultado de acesso, ou seja, o valor da variável *PARIOS_xx*. Caso contrário, a indicação MMS mantém-se pendente.

A *Fig. 7.11* ilustra a sequência de transacções envolvidas neste mapeamento. De notar o processamento em três fases no servidor. O processamento designado por 'A', diz respeito ao atendimento de uma indicação do serviço MMS *Read* para efectuar a leitura de uma variável que corresponde aos parâmetros de uma E/S de um módulo da rede de *Field Bus*. O processamento desta indicação corresponde a fazer a identificação do módulo destinatário e da E/S visada, e, obviamente, enviar um pedido à rede de *Field Bus*, neste caso, o *DPA_DF*. A indicação MMS ficará pendente. A variável de controlo associada a esta variável particular do VMD, é colocada a '0'.

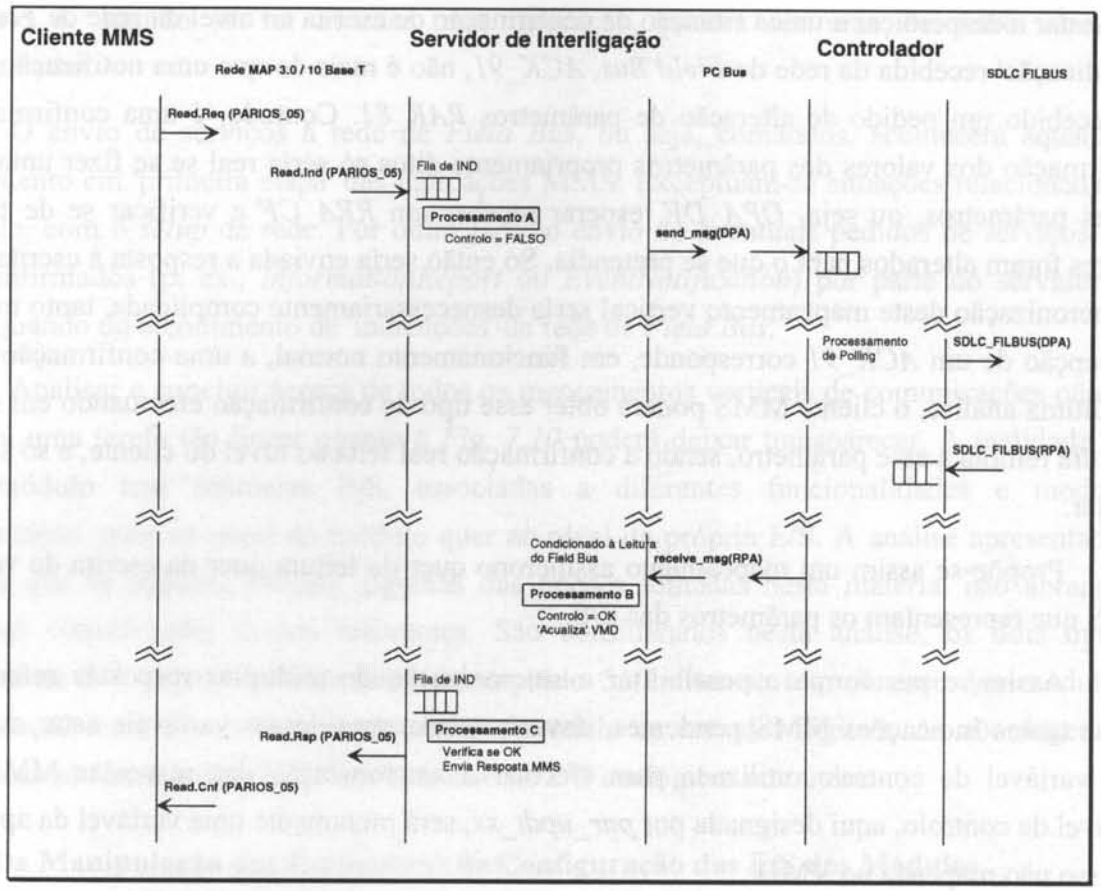


Figura 7.11 *- Cenário de Leitura de Parâmetro de Uma E/S de Um Módulo*

O segundo processamento, na figura designado por 'B', vai ter lugar aquando do atendimento de uma 'indicação' proveniente da rede de *Field Bus* correspondente ao serviço *RPA_CF*. A recepção deste serviço implicará sempre a colocação a '1' da variável de controlo associada ao parâmetro cujo valor é veiculado nessa mensagem. Obviamente, proceder-se-á ao 'refrescamento' do valor da variável do MMS associada a esse serviço. Finalmente, o processamento C, será executado quando, num próximo atendimento de indicações MMS, se

verificar que a variável MMS em causa já está actualizada. Só nessa situação é que será enviada a resposta MMS correspondente à indicação (pedido do cliente) pendente.

Relembre-se que, entre as transacções descritas atrás poderão ocorrer outras, referentes ao processamento de outros pedidos MMS ou à recepção de 'indicações' do *Field Bus*, despoletadas por iniciativa da própria rede, com eventual mapeamento em pedidos MMS não confirmados. As transacções são encaixadas aleatoriamente, uma vez que os processamentos ao nível do *Field Bus* dependem da sequência nas tabelas de *polling*. O mapeamento proposto para a leitura de parâmetros de E/S é o que se apresenta na *Tab. 7.1*.

Cliente MMS	Servidor	Field Bus
Read.Req (PARIOS_xx) →		
	DPA_DF →	
	←	*RPA_CF*
←	*Read.Rsp (PARIOS_xx)*	

***Tabela 7.1** - Mapeamento Para a Leitura de Parâmetros de Uma E/S*

Analise-se agora a actividade relacionada com a alteração de parâmetros de uma E/S de um módulo, por iniciativa de um cliente MMS.

No sentido de aproveitar a 'indicação' *ACK_91* que é subsequente a um comando *PAR_81*, a recepção de uma indicação MMS *Write.Ind (PARIOS_xx)* deverá ser mapeada num comando *PAR_81* da rede de *Field Bus*. A resposta a este pedido MMS será condicionada pela chegada da referida 'indicação' *ACK_91*.

O mapeamento proposto para a escrita de uma variável MMS correspondente aos parâmetros de uma E/S de um módulo da rede de *Field Bus* é que se apresenta na *Tab. 7.2*.

Cliente MMS	Servidor	Field Bus
Write.Req (PARIOS_xx) →		
	PAR_81 →	
	←	*ACK_91*
←	*Write.Rsp (PARIOS_xx)*	

***Tabela 7.2** - Mapeamento Para a Escrita de Parâmetros de Uma E/S*

O mapeamento vertical de comunicações agora proposto, e em particular a filosofia utilizada para o sincronizar remotamente, poderá conter situações críticas, apesar de que pouco plausíveis.

Essas situações críticas podem corresponder, por exemplo, a acessos consecutivos quer de leitura quer de escrita a uma mesma variável, feitos a partir de um mesmo cliente, ou de clientes MMS diferentes. Esses acessos consecutivos serão críticos se implicarem a coexistência de mais de uma indicação MMS pendente para a mesma variável. No que se refere meramente à questão da sincronização dos mapeamentos verticais, é absolutamente indiferente se se trata do mesmo cliente ou de diferentes clientes MMS. Estas situações são pouco plausíveis, uma vez que para um mesmo cliente não será muito normal efectuar pedidos MMS sem ter obtido a resposta a um anterior, por outro lado não é muito normal que uma mesma variável seja utilizada por diferentes clientes MMS.

Analisam-se seguidamente essas situações, para os casos de leitura e escrita de variáveis correspondentes aos parâmetros de E/S. Os resultados desta análise são extrapoláveis a outras situações de mapeamento vertical, em particular, os relacionados com as E/S propriamente ditas.

Tome-se como primeiro exemplo, a chegada de uma indicação MMS de leitura, seguida de uma indicação MMS de escrita de um mesmo parâmetro. A sequência de acontecimentos relacionada com este mapeamento vertical é a que se apresenta a seguir, não sendo consideradas as respostas aos pedidos MMS.

```
1.    Chega um Read.Ind (PARIOS_05)
      É enviada a mensagem DPA_DF e coloca-se par_updt[05] a '0'

2.    Chega Write.Ind (PARIOS_05) do mesmo ou de outro cliente MMS
      É enviado um PAR_81 e coloca-se e coloca-se par_updt[05] a '0'

3.    Chega um RPA_CF
      Actualiza-se o VMD e coloca-se par_updt[05] a '1'

4.    Chega um ACK_91
      Coloca-se par_updt[05] a '1', o VMD só é actualizado quando enviar Write.Rsp
```

A numeração dada aos acontecimentos apenas é relevante pelo facto de os ordenar cronologicamente entre si, uma vez que outras indicações MMS ou 'indicações' da rede de *Field Bus* poderão acontecer entre as que atrás se relatam. Tratar-se-ão, no entanto, de transacções referentes a outros mapeamentos verticais de comunicações. Deve-se admitir também que é mantida a cronologia de mensagens ao nível da rede de *Field Bus*, isto é, respostas a pedidos feitos chegam ao controlador do *Field Bus* na mesma ordem em que foram os respectivos pedidos. Considere-se por fim que, ao contrário do que fôra admitido para a *Fig. 7.10*, todas as indicações MMS pendentes serão verificadas (situação normal em muitas interfaces MMS) no sentido de se aquilatar se se poderá ou não enviar a resposta.

Voltemos então aos acontecimentos atrás referidos. Vamos supôr que é feito o atendimento das indicações MMS pendentes de 1 e 2 antes do instante 3. Essas duas indicações

pendentes não vão ser respondidas, uma vez que a variável de controlo ainda não valída o pedido feito à rede de *Field Bus*.

Se o atendimento das indicações MMS for feito entre os instantes 3 e 4, e considerando, como se disse, que todas as indicações MMS são verificadas, então será enviada a resposta referente quer ao pedido de leitura MMS, quer ao pedido de escrita MMS, este último sem ter chegado o *ACK_91*. Sendo assim, o valor da leitura enviado ao cliente MMS, será o do parâmetro antes de ser feito o pedido de alteração e é enviada a confirmação de escrita sem existir de facto confirmação da parte do *Field Bus*. De notar que o módulo recebe *DPA_DF* antes de *PAR_81*, e por isso coloca no seu *buffer* de saída primeiro o *RPA_CF* e depois o *ACK_81*.

A menor elegância deste comportamento não reside no facto de se falsear a leitura pedida pelo primeiro cliente. De facto o pedido de leitura é feito antes de haver um pedido de escrita. Reside sim, no facto de se ter 'atalhado' o mecanismo de sincronização da resposta MMS.

Com isto não se está a questionar a coexistência simultânea de um pedido de escrita e outro de leitura MMS. Dever-se-á recorrer, no entanto, a variáveis de controlo distintas para as situações de escrita e de leitura. No caso de leitura e escrita de parâmetros, o *ACK_91* coloca a '1' a variável de controlo da escrita, e o *RPA_CF* coloca a '1' a variável de controlo de leitura.

Vamos agora supôr um cenário idêntico ao anterior mas com ordem inversa de chegada de indicações MMS: primeiro *Write* e depois o *Read*. Mantem-se na mesma uma só variável de controlo para as situações de escrita e de leitura.

```
1.    Chega um Write.Ind (PARIOS_05)
      É enviado um PAR_81 e coloca-se par_updt[05] a '0'

2.    Chega um Read.Ind (PARIOS_05) do mesmo ou de outro cliente MMS
      É enviada a mensagem DPA_DF e coloca-se par_updt[05] a '0'

3.    Chega um ACK_91
      Coloca-se par_updt[05] a '1', o VMD só é actualizado quando enviar Write.Rsp

4.    Chega um RPA_CF
      Actualiza-se o VMD e coloca-se par_updt[05] a '1'
```

Se as indicações MMS pendentes em 1 e 2 forem processadas antes de 3, nenhuma delas é respondida. Se forem processadas entre 3 e 4 são ambas respondidas, apesar de para a situação de leitura ainda não ter chegado a resposta.

De qualquer maneira, a resposta MMS ao pedido de leitura já leva o valor pós escrita, uma vez que o módulo recebe o pedido de escrita, altera os parâmetros e só depois recebe o pedido de leitura, colocando no seu *buffer* de saída um *RPA_CF* já com esse novo valor.

Em termos funcionais, o falseamento existente numa das sincronizações volta a não ser relevante. Basicamente, confirma-se a validade funcional de permitir o processamento paralelo de um mapeamento vertical de uma escrita e de uma leitura remota.

Vamos agora supôr uma situação de duas leituras consecutivas. A sequência de acontecimentos será a seguinte:

```
1.    Chega um Read.Ind (PARIOS_05)
      É enviada a mensagem DPA_DF e coloca-se par_updt[05] a '0'

2.    Chega um Read.Ind (PARIOS_05) do mesmo ou de outro cliente MMS
      É enviada a mensagem DPA_DF e coloca-se par_updt[05] a '0'

3.    Chega um RPA_CF
      Actualiza-se o VMD e coloca-se par_updt[05] a '1'

4.    Chega um RPA_CF
      Actualiza-se o VMD e coloca-se par_updt[05] a '1'
```

Se o processamento das indicações MMS pendentes é chamado entre os instantes 3 e 4, vão ser enviadas as duas respostas com o valor que resulta de 3. Se forem chamadas depois de 4, vão ser as duas enviadas com o valor que resulta de 4. Isto só não corresponde à realidade se acontecer entre essas duas leituras uma escrita desses parâmetros, ou por parte de um cliente MMS, ou como resultado de actividade reflexa. Analise-se a primeira das situações, considerando agora duas variáveis de controlo distintas, isto é, *par_updt_rd_xx* para a leitura e *par_updt_wr_xx* para a escrita.

```
1.    Chega um Read.Ind (PARIOS_05)
      É enviada a mensagem DPA_DF e coloca-se par_updt_rd[05] a '0'

2.    Chega um Write.Ind (PARIOS_05)
      É enviado um PAR_81 e coloca-se par_updt_wr[05] a '0'

3.    Chega um Read.Ind (PARIOS_05)
      É enviada a mensagem DPA_DF e coloca-se par_updt_rd[05] a '0' (Já estava!)

4.    Chega um RPA_CF
      Actualiza-se o VMD e coloca-se par_updt_rd[05] a '1'

5.    Chega um ACK_91
      Coloca-se par_updt_wr[05] a '1', o VMD só é actualizado quando enviar Write.Rsp

6.    Chega um RPA_CF
      Actualiza-se o VMD e coloca-se par_updt_rd[05] a '1'
```

Se o processamento de indicações é feito entre 4 e 5, ambas as respostas aos pedidos de leitura vão ser feitas com o valor resultante de 4, isto é antes de ser efectivada a escrita. Se for entre 5 e 6 ou depois de 6 ambos vão ser respondidas com o valor já posterior à escrita. De notar que estamos a falar de períodos de tempo para estes acontecimentos, na ordem dos milisegundos.

Em relação a este problema da escrita, poder-se-á obrigar a que só se efectue a resposta à leitura, após a escrita ser feita, isto é, se também *par_updt_wr_xx* for igual a '1', uma vez que o VMD é alterado quando se processa a resposta MMS ao pedido de escrita.

Por fim, analise-se a situação de escritas concorrentes de uma mesma variável. Veja-se o seguinte exemplo:

```
1.    Chega um Write.Ind (PARIOS_05)
      É enviado um PAR_81 e coloca-se par_updt[05] a '0'

2.    Chega um Write.Ind (PARIOS_05)
      É enviado um PAR_81 e coloca-se par_updt[05] a '0'

3.    Chega um ACK_91
      Coloca-se par_updt[05] a '1', o VMD só é actualizado quando enviar Write.Rsp

4.    Chega um ACK_91
      Coloca-se par_updt[05] a '1', o VMD só é actualizado quando enviar Write.Rsp
```

Uma vez que só na resposta à escrita MMS é que o VMD é alterado, se se garantir um processamento cronológico das duas indicações MMS de escrita pendentes, o VMD será sempre alterado duas vezes, e ficará em vigor a última alteração, que corresponderá à realidade da rede de *Field Bus*. Portanto, do ponto de vista da actualização do VMD será absolutamente indiferente. Contudo a confirmação seria simplesmente uma figura de estilo. Vamos supor que por alguma razão o segundo pedido de escrita não chegava ao destinatário. Nunca chegaría o *ACK_91* associado ao *PAR_81*. Isto quería dizer que os parâmetros não seríam remotamente alterados, mas o VMD reflectiría essa alteração. Por isso, não deverão ser toleradas escritas remotamente confirmadas simultâneas à mesma variável.

Ainda a propósito do processamento cronológico das indicações. Vamos supôr que foi pedida uma leitura MMS de *PARIOS_05*. Esse pedido fica pendente. Entretanto chega a resposta do *Field Bus*. Actualiza-se o VMD, e coloca-se *par_updt_rd_05* a '1'. Quando se for fazer o atendimento de indicações, pode acontecer que tenha chegado uma nova indicação para leitura da mesma variável. É imperativo atender as indicações MMS pela sua ordem cronológica, caso contrário ir-se-ia colocar *par_updt_rd_05* a '0', e ao verificar a indicação que já estava pendente, esta não sería, indevidamente, respondida.

Por fim, será de referir que em situações de leitura remota, deverá ser enviado o pedido remoto de leitura, mesmo que esteja a decorrer um (*par_updt_rd_xx* igual a '0'). O valor entre leituras poderá ser diferente, o que, em determinadas situações, poderá provocar respostas não actualizadas ao cliente MMS. Contudo o VMD será actualizado de acordo com as respostas recebidas da rede de *Field Bus*. Por outro lado, este tipo de leitura consecutiva não será muito plausível a partir do mesmo cliente, mas sim a partir de clientes diferentes, por hipótese interessados na mesma informação. Por outro lado aínda, as variáveis cuja leitura será quase sempre remota serão precisamente os parâmetros de E/S e E/S analógicas, cuja dinâmica de alteração é compatível com estas situações.

Em conclusão, para o acesso às variáveis MMS correspondentes aos parâmetros das E/S dos módulos da rede de *Field Bus*, deverá ser possível a leitura MMS assíncrona concorrente, isto é, haver mais de um pedido pendente de leitura para uma mesma variável. No que diz respeito à escrita, só deverá existir um pedido pendente para uma mesma variável. Deverá ser

possível a coexistência de pedidos assíncronos de leitura com um e um só pedido assíncrono de escrita, estando neste caso a resposta à leitura condicionada à confirmação da escrita. Para as variáveis que possam ser escritas e lidas por um cliente MMS, deverão ser consideradas duas variáveis de controlo de sincronização independentes para a escrita e leitura.

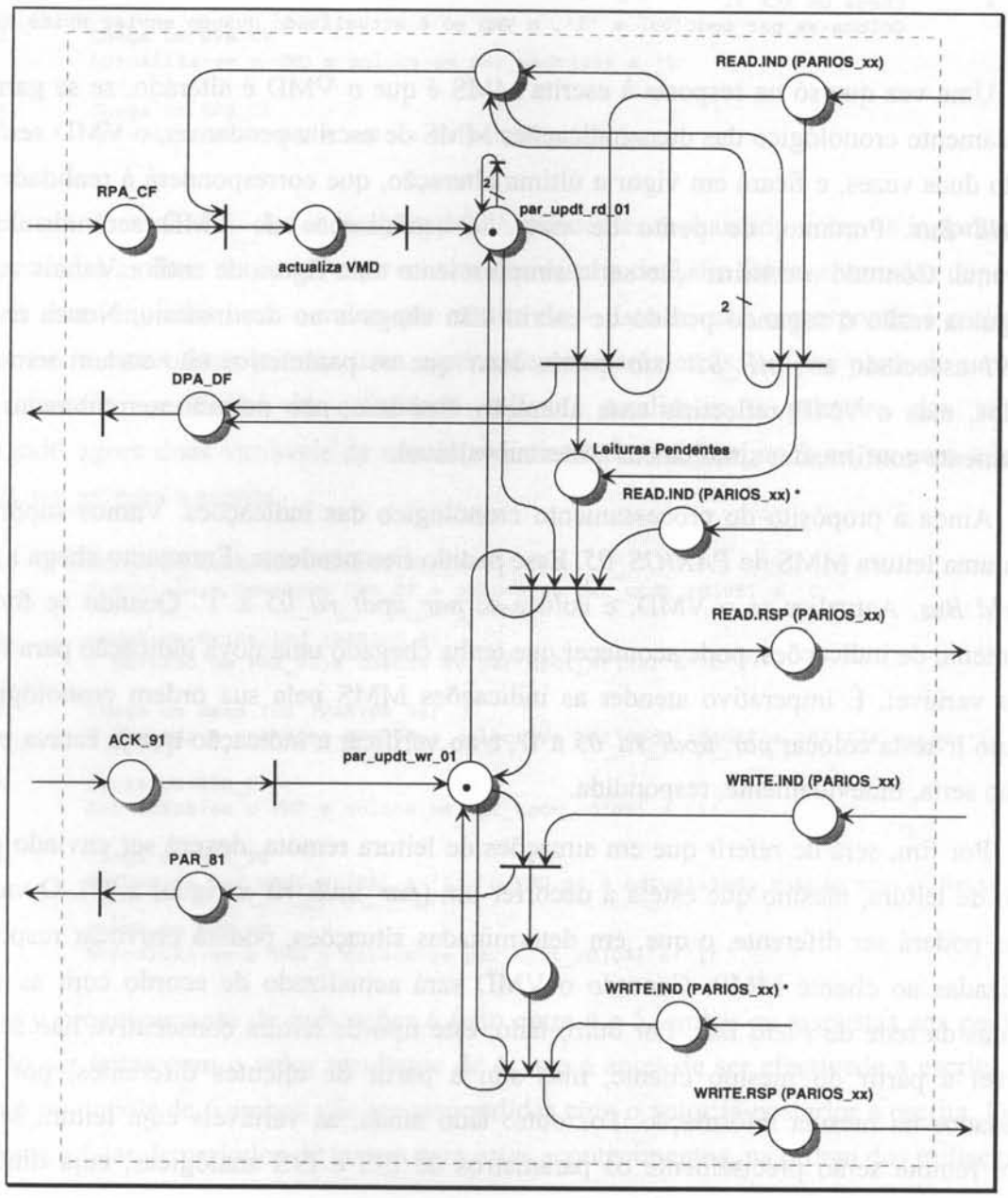


***Figura 7.12** - Rede de Petri Para a Manipulação de Variáveis MMS PARIOS_xx*

A *Fig. 7.12* ilustra a arquitectura de mapeamento proposto, considerando, por simplificação da Rede de Petri, apenas uma variável de controlo e por isso apenas um parâmetro.

A variável de controlo associada ao módulo, deverá ter, por via da leitura e escrita de parâmetros, os seguinte elementos (todos elas deverão ser colocadas a '1' no *setup* do VMD):

```
struct module_ctrl
{
        par_updt_rd [17];
        par_updt_wr [17];
};
```

Cada módulo deverá ter uma variável MMS, designada por *PARIOS_xx*, por cada E/S. A estrutura C desta variável deverá ser:

```
unsigned char par[256];
```

7.3.4. Da Manipulação de Variáveis de Entradas dos Módulos da Rede de Field Bus

Os serviços da rede de *Field Bus* relacionados com a manipulação de E/S dos módulos de entrada digitais, são os comandos *DAL_60*, *DEA_61*, *DCP_62*, *WCI_A0*, *WCT_A1* e as 'indicações' *ALM_B0*, *STA_B1*, *CPI_B2*, *CPT_B3*, *DIG_B9*, *PAL_C0*, *PST_C1*, *PCI_C2* e *PCT_C3*.

A estruturação em variáveis é de alguma forma condicionada pela possibilidade de se poder programar o módulo com *EVO_47*, enviando este notificações automáticas *DIG_B9*, que contêm informação relativa a todas as entradas. É ainda preciso ter em atenção que podendo as entradas estar programadas de forma diferente, os serviços da rede de *Field Bus* que manipulam essas entradas veiculam diferentes tipos de informação.

O mapeamento de cada entrada do módulo de entradas digitais numa variável MMS distinta poderá, de alguma forma, simplificar as comunicações verticais.

Assim, e designando cada uma dessas variáveis por *IOINDI_xx*, um cliente MMS que desejasse efectuar a leitura dessa variável, enviaría ao dispositivo de interligação um pedido MMS *Read.Req (IOINDI_xx)*. Ao receber a indicação dever-se-á verificar qual o tipo da entrada, por exemplo, por consulta a um campo que se designará *inp_types_xx* da estrutura de controlo do módulo, já referida anteriormente. Se a entrada for do tipo alarme ou do tipo estado, então dever-se-á verificar se o módulo está a funcionar com notificação automática de mudanças de estado das entradas. Um campo da mesma estrutura de controlo, por exemplo a designar por *evo_status*, poderá conter essa informação. Se *evo_status* for '1', então a variável em causa poderá ser obtida de imediato no VMD sem necessidade de enviar um pedido de leitura à rede de *Field Bus*, sendo enviada sincronamente a resposta ao pedido MMS.

Se a variável for do tipo contador, então a decisão por uma resposta síncrona ou assíncrona deverá passar também pela verificação dos parâmetros de relatório automático dos contadores, informação essa disponível na variável MMS *PARIOS_xx*. Só no caso de esses

parâmetros serem diferentes de '0', é que a leitura poderá ser executada em modo síncrono. Admitindo que se suportam leituras assíncronas concorrentes, então a *Fig. 7.13* representa a Rede de Petri correspondente à manipulação de só uma dessas variáveis, não considerando para já a influência de 'indicações' do tipo evento ao nível da rede de *Field Bus*.

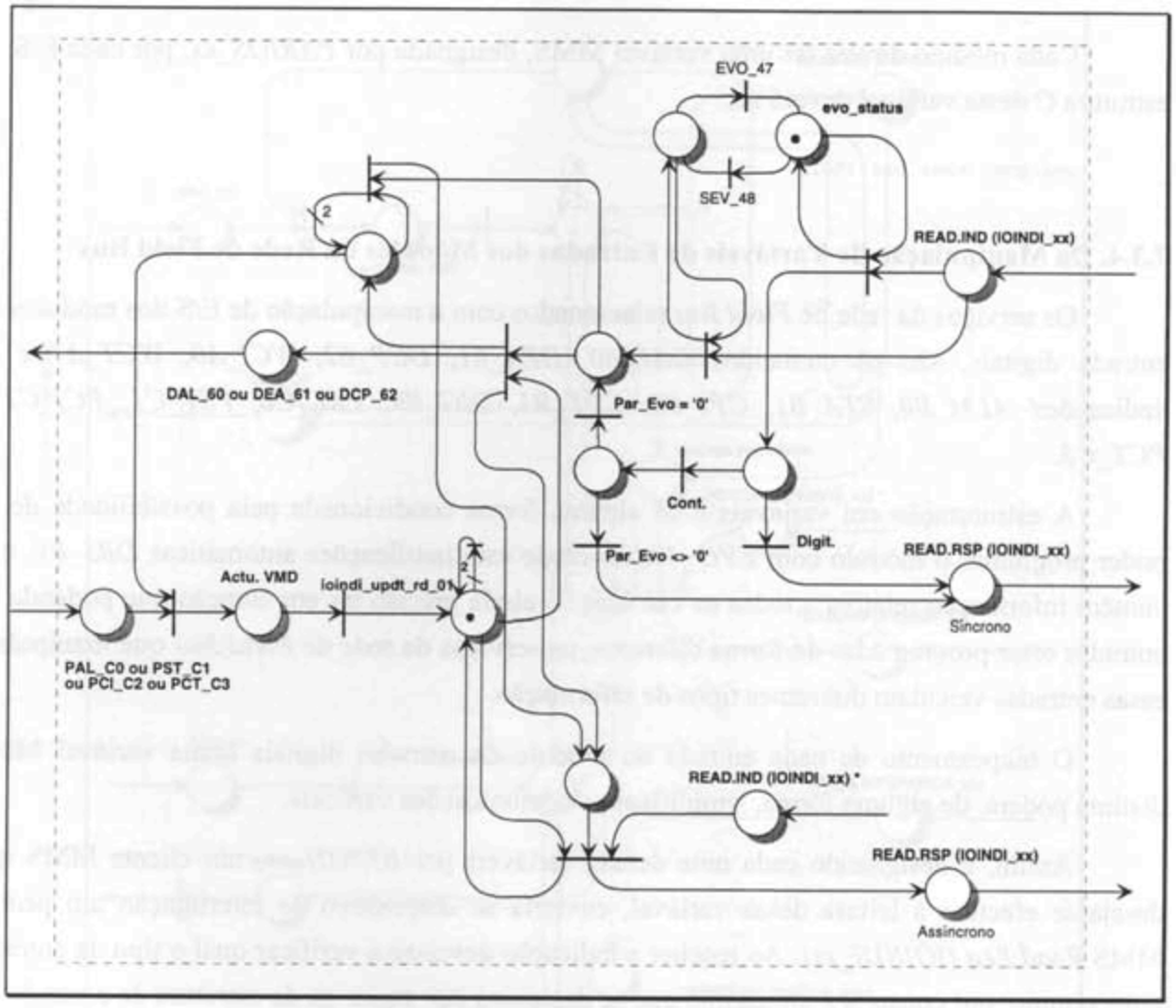


Figura 7.13 *- RdP da Leitura de Variáveis de Entrada, Síncrona ou Assíncrona Para Qualquer Modo*

O problema é que se admitiu anteriormente que os valores dos parâmetros das E/S dos módulos teríam de ser obtidos remotamente. Em particular não é possivel garantir que os parâmetros de relatório automático não tenham sido alterados como resultado de actividade reflexa. Assim, para uma entrada a funcionar em modo contador, um pedido de leitura MMS deverá ser mapeado assincronamente num pedido de leitura remoto *DCP_62*. Isto será tanto mais acertado, quanto se um módulo está programado com relatórios automáticos, não será muito natural efectuar leituras por iniciativa de um cliente MMS, uma vez que estas podem ser

facultadas ao cliente, se este o desejar, por intermédio de serviços MMS não confirmados de iniciativa do servidor, como por exemplo o *InformationReport* ou o *EventNotification*.

Para qualquer tipo de entrada, se o seu valor tiver de ser obtido remotamente, deverá ser enviado o pedido da rede *Field Bus* adequado ao tipo de entrada, isto é, *DAL_60*, *DEA_61* ou *DCP_62*. A variável de controlo de sincronização *ioindi_updt_rd_xx* deverá ser colocada a '0'. Esta variável de controlo deverá ser colocada a '1', à chegada das respectivas respostas do *Field Bus*. Pelas razões já apontadas anteriormente aquando da discussão dos mapeamentos para a leitura de parâmetros, poderão ser suportadas leituras concorrentes.

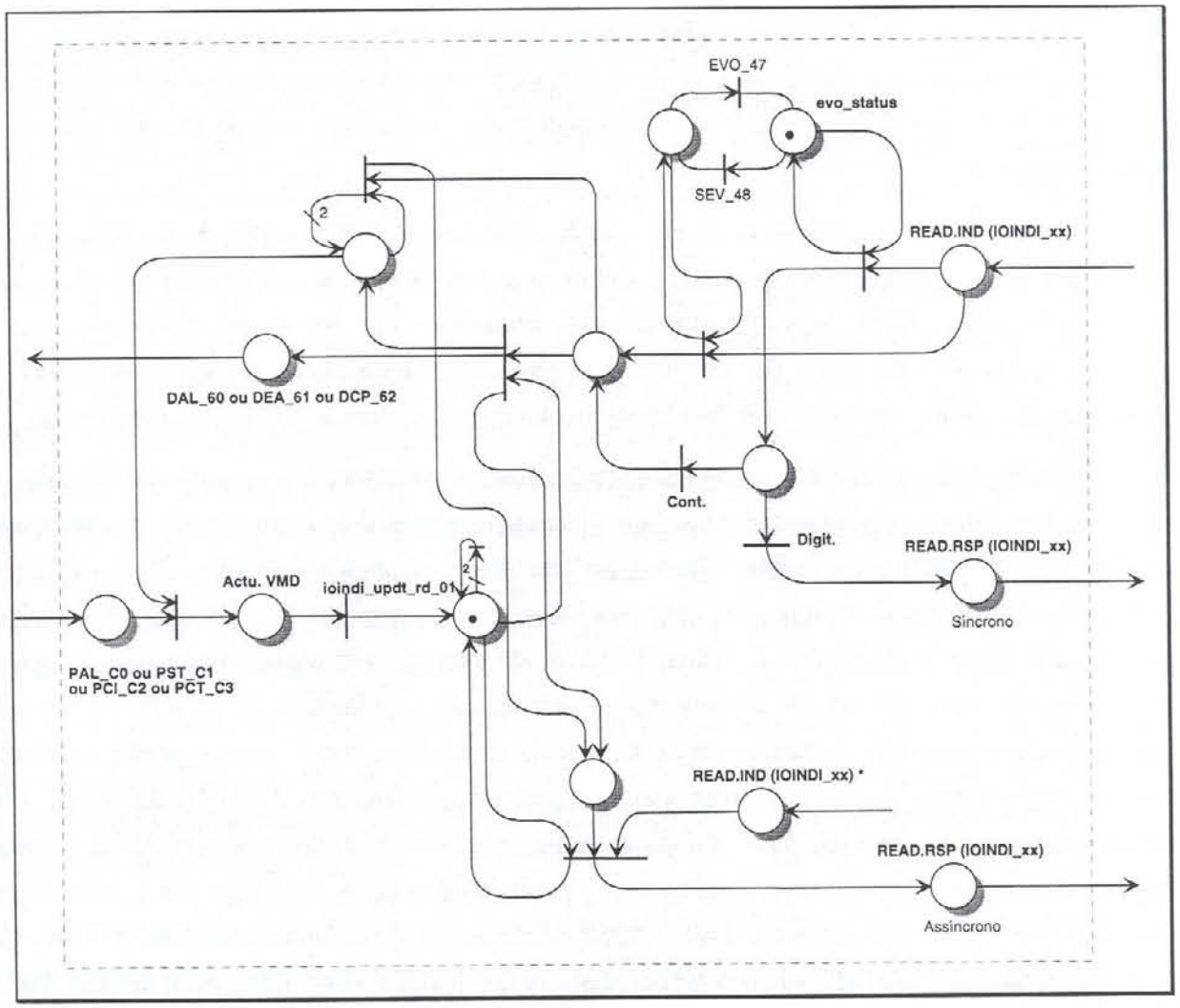


***Figura 7.14** - RdP da Leitura de Variáveis de Entrada, Assíncrona no Caso de Contadores*

Assim, a *Fig. 7.14* representa esta nova situação, na qual leituras a variáveis de entrada do tipo contador, serão sempre mapeadas em modo assíncrono. A chegada de um pedido de

leitura numa situação em que está a decorrer um mapeamento remoto implica na mesma o envio do respectivo pedido.

Contudo, deverá ser prevista uma variável que permita o acesso a todas as entradas a funcionar em modo digital, tirando assim proveito do serviço *DIG_B9*. Sendo que grande parte das variáveis serão do tipo puramente digital, ter-se-á a possibilidade de aceder simultaneamente a todas elas. Em alternativa sería possivel numa só leitura MMS obter todos os diferentes valores efectuando um pedido a uma lista de variáveis.

Essa variável poderá ter o nome *IOGLOB_MD*, e só terá significado para as entradas que não estejam a funcionar como contador.

Vamos supor que um cliente MMS deseja ler o estado de todas as variáveis digitais. Para tal, esse cliente enviará ao dispositivo de interligação um pedido de serviço MMS *Read.Req (IOGLOB_MD)*. A indicação MMS recebida poderá ser processada em modo síncrono ou em modo assíncrono.

No caso do processamento em modo assíncrono, uma variável de controlo de sincronização da resposta, por exemplo, *ioglob_updt_rd*, deverá ser colocada a '0'. Para cada entrada em modo digital, essa informação está disponível em *inp_types_xx* da estrutura de controlo, envia-se um *DAL_60* ou um *DEA_61*, colocando as variáveis respectivas de sincronização, *ioindi_updt_rd_xx* a '0'. O atendimento dessa indicação deverá ser suspendido.

Esse pedido MMS, só será respondido, quando a variável de controlo *ioglob_updt_rd* valer '1'. Essa variável poderá passar a valer '1' quando a informação pretendida passar a estar disponível no VMD. Isso poderá acontecer em duas situações distintas. Uma primeira, corresponde à recepção de um *DIG_B9*. Para que tal aconteça, o módulo teria de ter sido previamente programado com o comando *EVO_47*. Isto é absolutamente possível com a arquitectura de mapeamento de comunicações verticais aqui defendida. Uma segunda situação corresponderá à recepção de tantas respostas quantas as pedidas. Isto é, para suportar a primeira das opções, sempre que se atender uma 'indicação' proveniente da rede de *Field Bus*, correspondente a um serviço *DIG_B9*, dever-se-ão colocar a '1' todas as variáveis de controlo individual de actualização (*ioindi_updt_rd_xx*) para as entradas a funcionar em modo digital, bem como a variável de controlo *ioglob_updt_rd*. Dever-se-á, também actualizar o valor das variáveis mapeadas no VMD (quer a global, quer as individuais). Por outro lado, sempre que se receber uma das duas 'indicações' da rede *Field Bus*, *PAL_C0* ou *PST_C1*, dever-se-ão actualizar as variáveis mapeadas no VMD com a informação veiculada e colocar a '1' o valor da variável de controlo associada. Para além disso, e se a variável *ioglob_updt_rd* está a '0', deverá verificar-se se com este novo '1', todas as variáveis do tipo digital já estão actualizadas. Se sim, deverá também colocar-se a '1' a variável de controlo *ioglob_updt_rd*.

Para que este procedimento funcione, se qualquer cliente MMS pedir o valor individual de uma variável de entrada, ou seja, *Read.Req (IOINDI_xx)*, que esteja programada sem ser do tipo contador, então, se a variável de controlo *ioglob_updt_rd* estiver a '0' e o módulo não estiver a funcionar com relatório automático, não devem ser pedidos serviços à rede de *Field Bus* e a resposta será dada quando a respectiva variável de controlo valer '1', o que pode ser nesse instante, ou de uma próxima vez que for processado o pedido.

Portanto, perante um pedido de leitura de uma variável digital, ou seja, *Read.Req (IOINDI_xx)*, sem ser contador, dever-se-á verificar primeiro a variável de controlo *evo_status*. Se esta vale '1', processa-se logo a resposta. Caso contrário, se a variável de controlo *ioglob_updt_rd* vale '0', então verifica se a pretendida já foi obtida, caso contrário suspende-se a resposta até que isso seja verdade. Se a variável de controlo *ioglob_updt_rd* vale '1', então dever-se-á colocar a variável de controlo respeitante a essa variável (*ioindi_updt__rd_xx*) a '0' e enviar para a rede de *Field Bus* um comando *DAL_60* ou *DEA_61*. A resposta será suspensa, até que essa variável de controlo valha '1', o que só acontecerá aquando da recepção de uma 'indicação' do *Field Bus PAL_C0* ou *PST_C1*.

Note-se o peso do processamento do *DIG_B9*. De facto, aquando da recepção desta indicação, tirando por isso partido de uma funcionalidade que é possibilitada de notificação automática de alterações de estado das E/S digitais, é preciso actualizar diferentes variáveis do VMD. Tudo sería simplificado se só houvesse uma variável que correspondesse a todas as E/S digitais. Um cliente MMS, para ler uma variável digital, teria de fazer uma leitura global, ou seja, *Read.Req (IOGLOB_MD)*. Isto só sería válido se o módulo estivesse a funcionar com *EVO_47*. Caso contrário, um cliente para ler uma só entrada precisava de uma actualização de todas. Poder-se-ía pensar num outro tipo de mapeamento de nomes, e também se pode impôr que os módulos funcionem sempre com *EVO_47*.

As variáveis do tipo contador serão tratadas sempre individualmente, conforme se ilustra na *Fig. 7.14*. Pelas razões já expostas anteriormente, serão sempre processadas em modo assíncrono. A contagem pode ser pedida com o serviço *DCP_62*. A este pedido, a rede de *Field Bus* responde com *PCI_C2* ou *PCT_C3*.

Por outro lado, o módulo pode ser programado para leituras automáticas, a partir de uma determinada data, ou com o comando *DRN_D4*. Nesse caso ele vai passar a enviar serviços *CPI_B2* ou *CPT_B3*. Enquanto que *PCI_C2* e *PCT_C3* poderão ser usados à semelhança de *DIG_B9*, como uma espécie de notificação de evolução, estes últimos deverão simplesmente ser mapeados para os clientes directamente. No entanto poderão também actualizar a variável de controlo de actualização, se entretanto foi enviado à entrada um *DRN_D4*. De notar que o *EVO_47* é dirigido ao módulo, enquanto que *DRN_D4* é orientado à E/S. É pouco provável que

aconteça uma situação em que um pedido de leitura de contador está a decorrer, e entretanto envia um comando para começar leituras automáticas. Antes, terá de programar os parâmetros adequados.

Os mapeamentos propostos para a leitura do estado de entradas são os que se apresentam nas tabelas seguintes.

Cliente MMS		Servidor	Field Bus
Read.Req (IOGLOB_MD)	⟶		
	⟵	*Read.Rsp (IOGLOB_MD)*	

***Tabela 7.3** - Mapeamentos Para a Leitura Global das Entradas, Módulo com Evolução Automática*

Cliente MMS		Servidor		Field Bus
Read.Req (IOGLOB_MD)	⟶			
		DAL_60, All ou		
		DEA_61, All	⟶	
			⟵	*PAL_C0* ou *PST_C1*
			⟵	*PAL_C0* ou *PST_C1*
			...	
		ou	⟵	*DIG_B9*
	⟵	*Read.Rsp (IOGLOB_MD)*		

***Tabela 7.4** - Mapeamentos Para a Leitura Global das Entradas, Módulo Sem Evolução Automática*

Cliente MMS		Servidor	Field Bus
Read.Req (IOINDI_xx)	⟶		
	⟵	*Read.Rsp (IOINDI_xx)*	

***Tabela 7.5** - Mapeamentos Para a Leitura de Uma Só Entrada, Módulo Com Evolução Automática*

Cliente MMS			Servidor	Field Bus
Read.Req (IOINDI_xx)	⟶			
			DAL_60, xx ou	
			DEA_61, xx ⟶	
			ou nada se já estiver controlo a '0'	
			⟵ *PAL_C0* ou *PST_C1*	
		ou	⟵ *DIG_B9*	
	⟵		*Read.Rsp (IOINDI_xx)*	

***Tabela 7.6** - Mapeamentos Para a Leitura de Uma Só Entrada, Módulo Sem Evolução Automática*

Cliente MMS		Servidor	Field Bus
Read.Req (IOINDI_xx)	⟶		
		DCP_62, xx ⟶	
		ou nada se já estiver controlo da variável globala '0'	
		⟵	*PCI_C2* ou *PCT_C3*
	⟵	*Read.Rsp (IOINDI_xx)*	

***Tabela 7.7** - Mapeamentos Para a Leitura de Uma Entrada Contador, Módulo Sem Evolução Automática*

Não é necessário para o módulo de entradas, mas uma vez que no caso do módulo de saídas é preciso suportar a leitura e a escrita de variáveis, dever-se-á considerar, à semelhança do que acontecia com os parâmetros das E/S, variáveis de controlo distintas para a situação de leitura e para a situação de escrita. A variável de controlo associada a cada módulo passará então a ter a seguinte estrutura:

```
struct module_ctrl
{
        unsigned char  evo_status;
        unsigned char  input_type [16];
        unsigned char  ioglob_updt_rd;
        unsigned char  ioglob_updt_wr;
        unsigned char  ioindi_updt_rd [16];
        unsigned char  ioindi_updt_wr [16];
        unsigned char  par_updt_rd [17];
        unsigned char  par_updt_wr [17];
};
```

Cada módulo de entradas digitais, deverá ainda ter associada uma variável MMS, designada por *IOGLOB_MD*. A estrutura C desta variável deverá ser:

```
typedef struct _ioglob_md_c_type
{
        unsigned char dig_val[16];
        long int      stamp_secs;
        long int      stamp_mics;
} ioglob_md_c_type;
```

A cada variável E/S deverá corresponder uma variável *IOINDI_xx*, cuja composição estrutural é a seguinte:

```
typedef struct _ioindi_c_type
{
        unsigned char input_type;
        long          input_val;
        long          stamp_secs;
        long          stamp_mics;
        long          lev_0_secs;
        long          lev_0_mics;
        long          lev_1_secs;
        long          lev_1_mics;
} ioindi_xx_c_type;
```

De notar que existe o serviço *ALM_B0*, que indica uma série distinta de situações de alarme. Este serviço veiculará informação do número da entrada, um qualificador do alarme e a data e a hora em que o alarme ocorreu. Em termos de estrutura C, o seu tipo poderá ser o que se apresenta a seguir:

```
struct MD_Alarm_C_Type
{
        unsigned char  mod_address;
        unsigned_char  md_type;
        unsigned char  io_type;
        unsigned char  alm_quali;
        unsigend char  value;
        long           stamp_secs;
        long           stamp_mics;
};
```

É evidente que não faz sentido ter uma variável destas para cada E/S, uma vez que só é processada cada 'indicação' *ALM_B0* da rede de *Field Bus*, uma de cada vez, e este tipo de informação será de mapear, obviamente, para clientes MMS interessados. Poderá haver uma varável destas por módulo. Pode nem sequer ser considerado o respectivo objecto de acesso a variável no VMD, uma vez que o pedido de serviço MMS a ser enviado pelo dispositivo de interligação não é de todo confirmado. Esta questão do envio de serviços MMS não confirmados *InformationReport*, *EventNotification* ou *UnsolicitedStatus* será abordada mais à frente.

7.3.5. Da Manipulação das Variáveis de Saída dos Módulos da Rede de Field Bus

Os serviços da rede de *Field Bus* relacionados com a manipulação do estado das E/S dos módulos de saída são os comandos *DRY_66*, *RCL_75*, *LAN_B6*, *ORR_B7*, *RXO_B8*, e as 'indicações' *ALM_B0*, *DIG_B9* e *PRY_C5*.

Ao contrário do módulo de entradas, os serviços da rede de *Field Bus* referentes a estas E/S veiculam informação meramente digital. Por essa razão poder-se-ía pensar na existência de uma só variável do tipo global, ou seja *IOGLOB_MD*.

Se *evo_status* do módulo vale '1', então é enviada sincronamente a resposta. Se *evo_status* do módulo vale '0' então é enviado o pedido *DRY_66* à rede de *Field Bus*, especificando leitura para todas as saídas. São colocadas a '0' todas as variáveis de controlo de sincronização individuais ou sejam *ioindi_updt_rd_xx* e a variável de controlo *ioglob_updt_rd*. A resposta a este pedido só será processada quando a variável *ioglob_updt_rd* valer '1'. Isto poderá acontecer quer aquando da recepção de um *DIG_B9*, quer aquando da recepção de todos os *PRY_C5*. De notar que de facto só existe a variável *IOGLOB_MD*. Será feita uma actualização global da variável na recepção de *DIG_B9* e colocado a '1' *ioglob_updt_rd*. Quando se recebem indicações *PRY_C5*, essa informação deverá ser utilizada para actualizar a parte respectiva da variável *IOGLOB_MD*, colocar a '1' *ioindi_updt_rd_xx* e verificar se com esse se completam

todos os pedidos. Se sim, colocar também *ioglob_updt_rd* a '1'. No caso de ser recebido um MMS *Read.Ind(IOGLOB_MD)* enquanto outro se encontra em processamento, não deve ser enviado o pedido *DRY_66*, e essa indicação deverá ser suspendida.

A *Fig. 7.15*, ilustra este cenário.

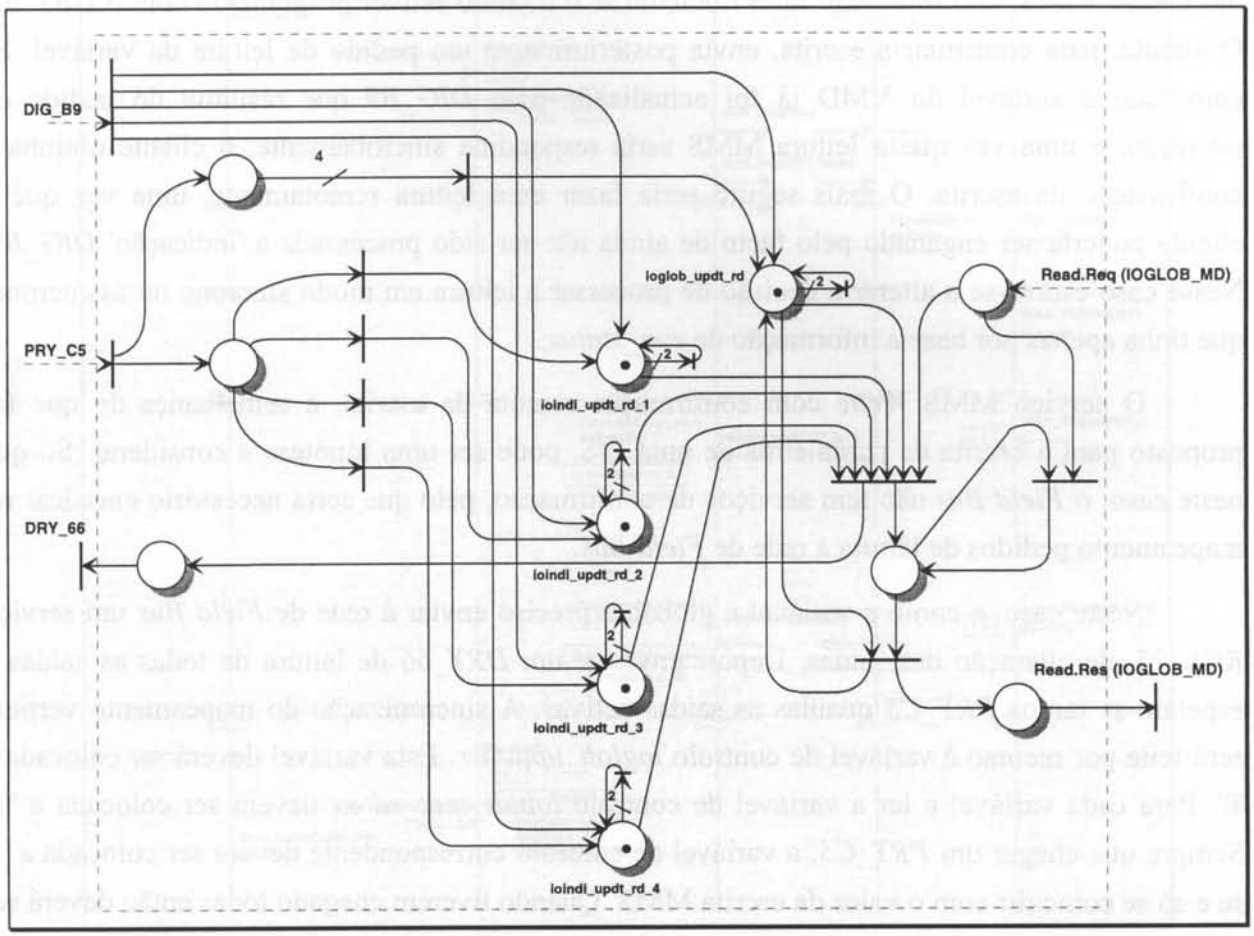


Figura 7.15 *- Rede de Petri Para a Leitura Remota da Variável Global*

Analise-se agora a manipulação de escrita da variável global *IOGLOB_MD*.

Um cliente MMS deseja alterar uma determinada saída. O servidor recebe uma indicação de escrita da variável *IOGLOB_MD*, ou seja *Write.Ind (IOGLOB_MD)*. Existem quatro serviços da rede de *Field Bus* que permitem alterar o estado lógico das saídas de um módulo de saídas digitais. São eles o *RCL_75*, que permite fazer uma mudança simultânea de todas as saídas de um determinado módulo, o *RAN_B6* que coloca a '1' uma determinada saída (não a altera se ela já está a '1'), o *ORR_B7* que tem acção semelhante à anterior, mas para o estado lógico '0', e por fim o *RXO_B8* que complementa o estado lógico de uma determinada saída. Em princípio, e

mantendo o mapeamento numa variável global, estes três últimos serão reservados para mensagens reflexas.

Assim, um pedido de escrita formulado por um cliente MMS será mapeado no serviço *RCL_75*. Como já se disse, a escrita MMS poderá não ser remotamente confirmada, e nesse caso a resposta será processada em modo síncrono. A confirmação poderá ser feita posteriormente por um cliente MMS. Isto será tanto mais eficiente se o módulo estiver programado com o *DIG_B9*. O cliente, para confirmar a escrita, envia posteriormente um pedido de leitura da variável. Se entretanto a variável do VMD já foi actualizada pelo *DIG_B9* que resultou do pedido de alteração, e uma vez que a leitura MMS sería respondida sincronamente, o cliente obtinha a confirmação da escrita. O mais seguro sería fazer essa leitura remotamente, uma vez que o cliente podería ser enganado pelo facto de ainda não ter sido processada a 'indicação' *DIG_B9*. Nesse caso estava-se a alterar a decisão de processar a leitura em modo síncrono ou assíncrono, que tinha apenas por base a informação de *evo_status*.

O serviço MMS *Write* com confirmação remota da escrita, à semelhança do que foi proposto para a escrita de parâmetros de uma E/S, pode ser uma hipótese a considerar. Só que neste caso, o *Field Bus* não tem serviços de confirmação, pelo que sería necessário encadear no mapeamento pedidos de leitura à rede de *Field Bus*.

Neste caso, e como a variável é global, é preciso enviar à rede de *Field Bus* um serviço *RCL_75*, de alteração das saídas. Depois envia-se um *DRY_66* de leitura de todas as saídas e esperam-se tantos *PRY_C5* quantas as saídas activas. A sincronização do mapeamento vertical será feita por recurso à variável de controlo *ioglob_updt_wr*. Esta variável deverá ser colocada a '0'. Para cada variável a ler a variável de controlo *ioindi_ipdt_rd_xx* deverá ser colocada a '0'. Sempre que chegar um *PRY_C5*, a variável de controlo correspondente deverá ser colocada a '1' se e só se coincidir com o valor da escrita MMS. Quando tiverem chegado todas então deverá ser colocada a '1' a variável *ioglob_updt_rd*. Se alguma das escritas não for confirmada, que corresponderá a uma situação de erro, então *ioglob_updt_rd* poderá ser colocada por exemplo a '2', para indicar uma situação de erro remoto. Torna-se claro que para este caso particular não será possível haver leituras remotas concorrentes com a escritas remotas, ao contrário do que foi aludido para a escrita e leitura de parâmetros das E/S. Naquele caso era perfeitamente possível, uma vez que são 'indicações' diferentes da rede de *Field Bus* a sincronizar a resposta MMS.

Deve neste momento ser referido que a situação de escrita remota confirmada só será válida se durante o tempo de mapeamento vertical não tiver havido uma alteração reflexa. Por outro lado, é de referir que das duas uma, ou se trata de uma situação de emergência, e nesse caso o cliente MMS terá de, previamente, escrever os parâmetros dos módulos que têm actividade

reflexa para impedir isso, ou se trata de uma situação de controlo normal feita por um cliente, e nesse caso, em princípio não haverá acção reflexa.

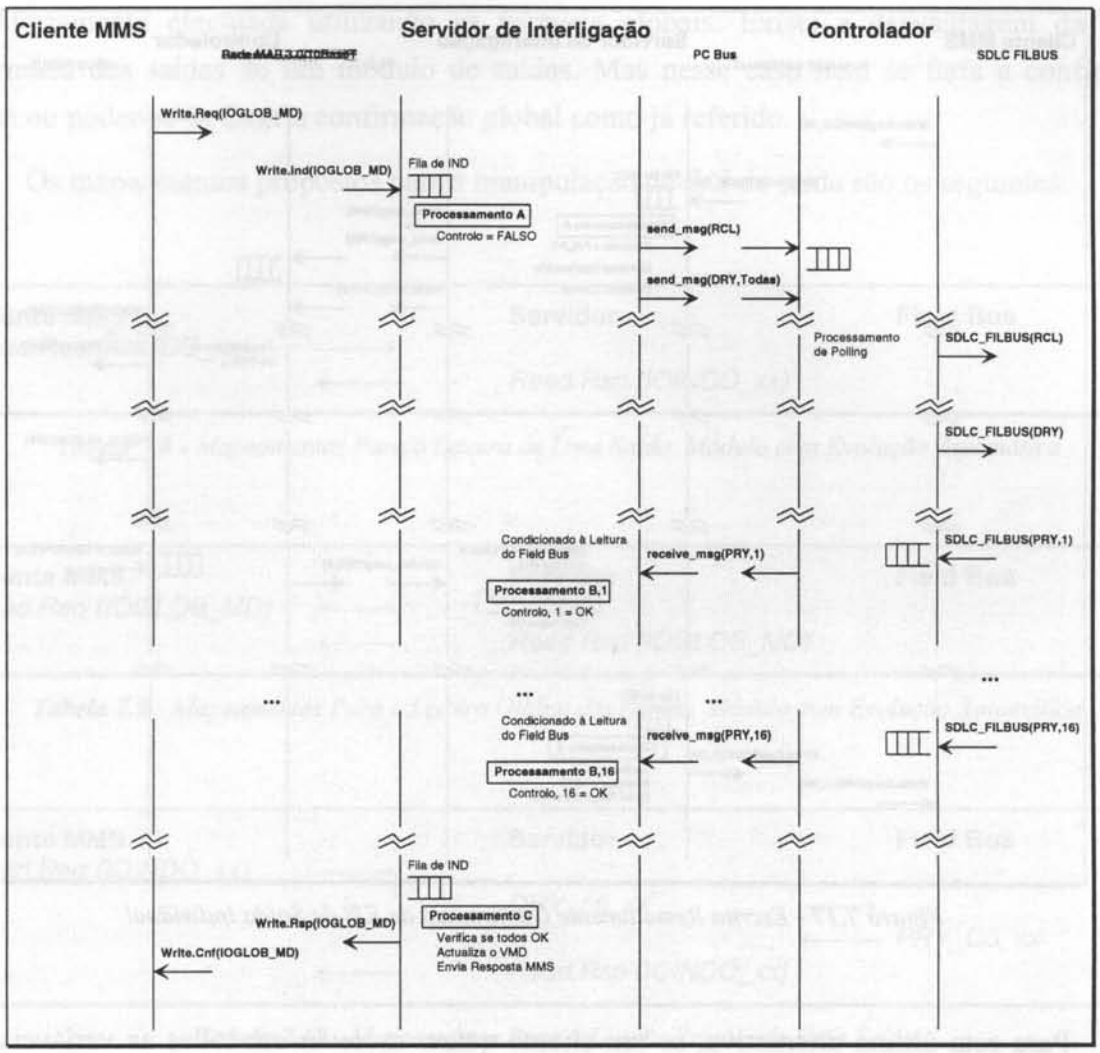


***Figura 7.16** - Escrita Remotamente Confirmada de E/S de Saída Global*

A *Fig. 7.16* ilustra esta proposta para a confirmação remota de escrita de uma variável global que representa todas as saídas de um módulo de saídas digitais da rede de *Field Bus* em análise.

Esta situação, de escrita remotamente confirmada das saídas, convida a que se proceda também à organização das E/S por variáveis individuais MMS *IOINDO_xx*. Nesse caso, na escrita, não seria enviado o serviço *RCL_75* (valores para todas as saídas) mas os serviços da rede de *Field Bus RAN_B6* ou *ORR_B7*, consoante se pretende-se escrever um '1' ou um '0' lógicos. A confirmação seria então mais facilmente obtida pelo envio de um *DRY_66*, agora só para uma saída, e pela resposta apenas com um *PRY_C5*. Se se fizesse um acesso à variável

global com *alternate access* (parâmetro de conformidade VALT) seria sensivelmente a mesma coisa. A *Fig. 7.17* ilustra esta alternativa.

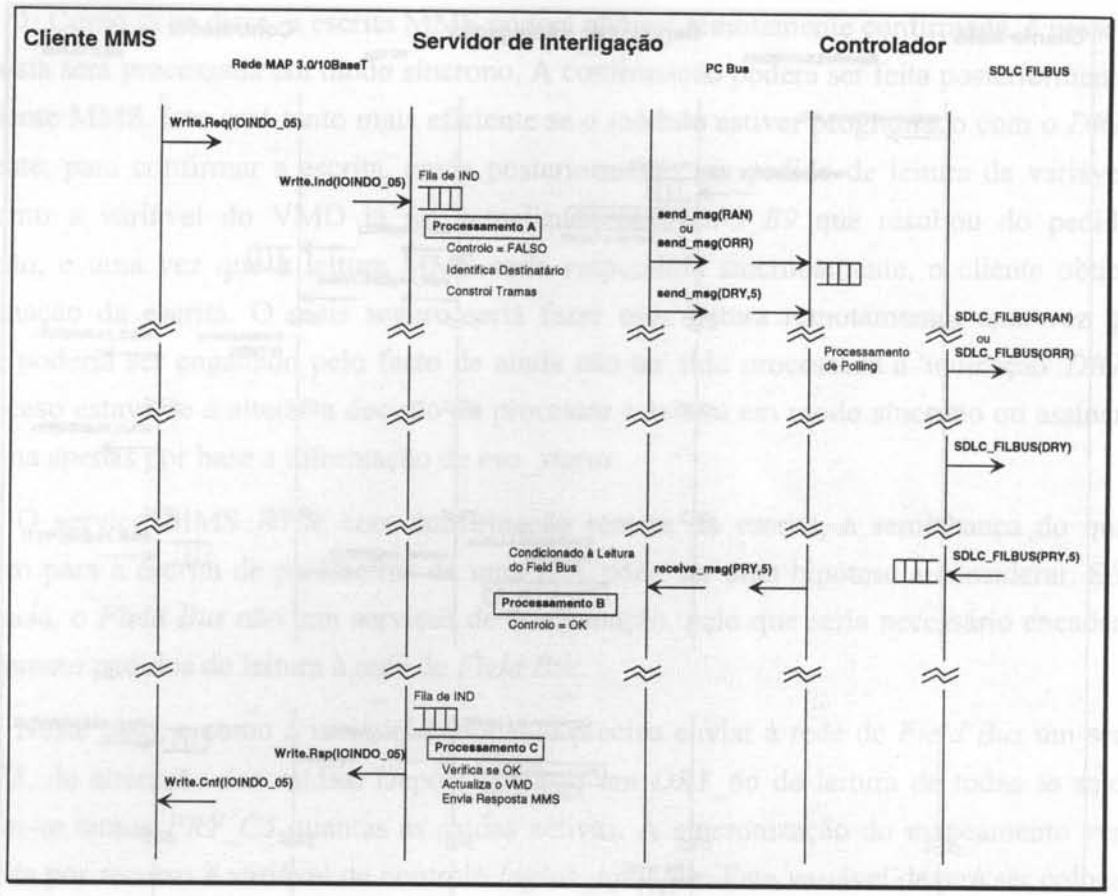


***Figura 7.17** - Escrita Remotamente Confirmada de E/S de Saída Individual*

Para esta última alternativa, se um cliente quizer escrever em todas as variáveis de um mesmo módulo, tem de enviar 16 pedidos MMS, o que se torna indubitavelmente mais pesado. Assim, e à semelhança do módulo de entradas poderão ser suportadas quer a manipulação global quer a manipulação individual das saídas.

Já foi dito que não deverão ser suportadas escritas e leituras simultâneas para a mesma variável. No caso de manipulação de variáveis globais é como se estivesse a manipular todas. No caso das leituras, poderão ser suportadas leituras concorrentes, mas sem o envio de pedidos de leitura remota no caso de estar a decorrer um pedido de leitura remota da variável global. Nesse caso, isto é, se *ioglob_updt_rd* estiver a '0', e se chegar um pedido de leitura individual ou global, pelas mesmas razões do módulo de entradas, não é enviado o pedido *DRY_66* à rede de *Field Bus*, simplesmente suspende-se a indicação MMS. Caso *ioglob_updt_rd* esteja a '1', procede-se

normalmente. O processamento do *DIG_B9* torna-se pesado, uma vez que é preciso actualizar cada uma das variáveis, à semelhança do que acontecia para o módulo de entradas.

Isto seria evitado se a manipulação das variáveis puramente digitais fosse obrigatoriamente efectuada utilizando as varáveis globais. Existe a desvantagem da escrita confirmada das saídas de um módulo de saídas. Mas nesse caso nem se faria a confirmação remota ou poder-se-ia fazer a confirmação global como já referido.

Os mapeamentos propostos para a manipulação de E/S de saída são os seguintes:

Cliente MMS		Servidor	Field Bus
Read.Req (IOINDO_xx)	⟶		
	⟵	*Read.Rsp (IOINDO_xx)*	

***Tabela 7.8** - Mapeamentos Para a Leitura de Uma Saída, Módulo com Evolução Automática*

Cliente MMS		Servidor	Field Bus
Read.Req (IOGLOB_MD)	⟶		
	⟵	*Read.Rsp (IOGLOB_MD)*	

***Tabela 7.9** - Mapeamentos Para a Leitura Global das Saídas, Módulo com Evolução Automática*

Cliente MMS		Servidor		Field Bus
Read.Req (IOINDO_xx)	⟶			
		DRY_66, xx	⟶	
			⟵	*PRY_C5, xx*
	⟵	*Read.Rsp (IOINDO_xx)*		

***Tabela 7.10** - Mapeamentos Para a Leitura Remota de Uma Saída*

Cliente MMS		Servidor		Field Bus
Read.Req (IOGLOB_MD)	⟶			
		DRY_66, all	⟶	
			⟵	*PRY_C5, 01*
			⟵	*PRY_C5, 02*
			⟵	...
			⟵	*PRY_C5, 16*
	⟵	*Read.Rsp (IOGLOB_MD)*		

***Tabela 7.11** - Mapeamentos Para a Leitura Remota Global das Saídas*

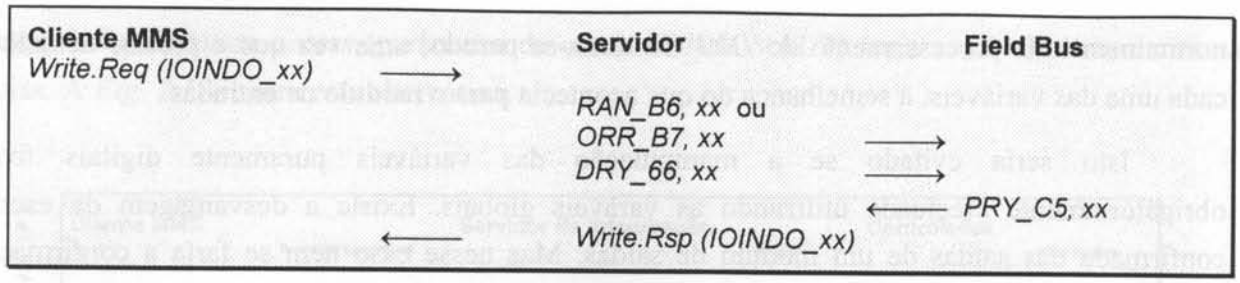


Tabela 7.12 - Mapeamentos Para a Escrita de Uma Saída

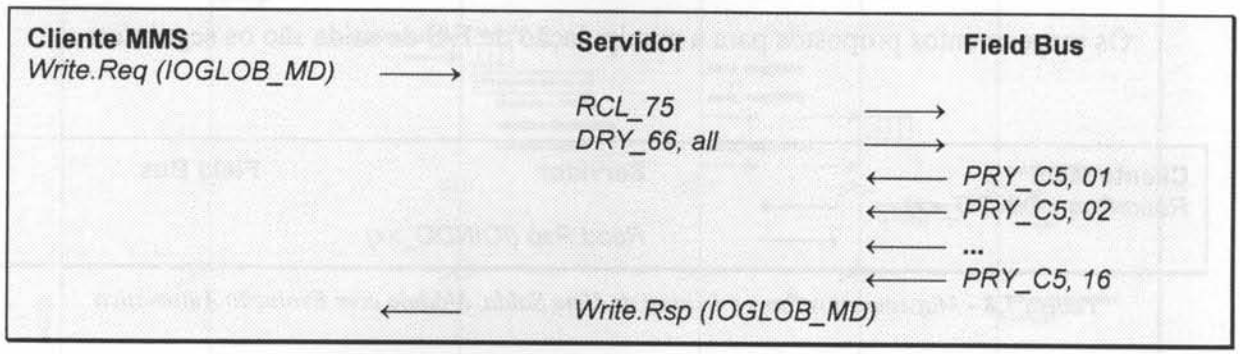


Tabela 7.13 - Mapeamentos Para a Escrita Global das Saídas

7.3.6. Mapeamento Vertical de Eventos Gerados Pela Rede de Field Bus

Existe um conjunto de situações de eventos gerados pela rede de *Field Bus*, que terão de ser mapeados em pedidos de serviço MMS não confirmados, como o *InformationReport*, o *EventNotification* ou o *UnsolicitedStatus*.

No que diz respeito a serviços despoletados pela rede de *Field Bus* relativos às E/S, estes deverão ser mapeados ou no *InformationReport* ou no *EventNotification*, sendo eles o *DIG_B9*, que é orientado ao módulo e só existe para os módulos de E/S do tipo digital, e os *ALM_B0*, *STA_B1*, *CPI_B2*, *CPT_B3*, *PCI_C2* e *PCT_C3*, todos relativos ao módulo de entradas, com a excepção do *ALM_B0* que é utilizado também pelo módulo de saídas, neste caso para notificar a existência de uma situação de curto circuito numa das suas saídas.

O problema do mapeamento deste serviços num pedido de serviço MMS não confirmado não é simples, uma vez assumido o mapeamento OSI referido anteriormente. Se a cada módulo houvesse associado um só canal OSI, esses pedidos MMS seríam enviados através do canal OSI associado ao módulo que gerou os serviços da rede de *Field Bus* referidos. A 'subscrição' para os serviços orientados às E/S era, obviamente, conseguida por manipulação dos parâmetros das E/S, neste caso de entrada. Este modelo pressuporía um só cliente MMS associado a cada canal MMS.

O modelo de servidor proposto anteriormente permite no entanto que a informação associada a cada módulo possa veiculada por serviços MMS através de diferentes canais OSI, e portanto potencialmente para diferentes clientes MMS. Mais ainda, o modelo proposto permite

que uma mesma variável possa ser manipulada por mais do que um cliente MMS. Se adicionarmos o facto de grande parte dos serviços do tipo evento despoletados pela rede de *Field Bus* serem orientados às E/S e não ao módulo, o número de combinações de subscrição, isto é, combinações de clientes MMS interessados nesses serviços, é enorme.

A ideia é possibilitar que, por exemplo, um *DIG_B9* seja 'prolongado' via MMS *InformationReport* para um determinado cliente MMS, ou que um alarme (*ALM_B0*) numa entrada x de um módulo seja enviada via MMS *InformationReport* para um clinte MMS A e um alarme referente a uma entrada y seja reportada aos cliente MMS B e C.

A opção 'cega' sería enviar um *InformationReport* através de todos os canais OSI activos.

Este é o tipo de questões que não se põe quanto às indicações MMS recebidas pelo servidor, uma vez que cada indicação está associada a um só canal MMS.

Uma hipótese para implementar estes mecanismos de subscrição de eventos gerados pela rede de *Field Bus*, passa obviamente pela utilização do modelo de eventos do MMS. Assim, associada a cada uma das variáveis MMS na qual seríam mapeadas as 'indicações' *Field Bus* recebidas, podería haver uma variável boleana que mudaría de estado sempre que a variável MMS fosse 'refrescada' pelas ditas 'indicações'. Essas variáveis boleanas seríam monitorizadas, estando a cada uma delas associada um objecto MMS da classe *Event Condition*. A acção, isto é, o objecto MMS da classe *Event Action*, podería consistir em enviar a resposta MMS *Read* da variável refrescada. Esta resposta a um serviço confirmado *Read* sería, como vimos no capítulo 6, veiculada no serviço não confirmado MMS *EventNotification*. De reparar que o *EventNotification* veiculando uma resposta a um MMS *Read* correponde a um *InformationReport*. O problema das subscrições a clientes é resolvido pelo MMS com os objectos da classe *Event Enrollement*.

A utilização deste modelo do MMS implica, como é fácil de ver, grandes quantidades de recursos, bem como a implementação de bastantes mais serviços MMS, quer ao nível dos clientes quer ao nível do dispositivo de interligação. Assim, e reportando ao capítulo 6, pode-se referir o suporte de serviços MMS como o *EventNotification*, eventualmente o *AcknowledgeEvent-Notification*, bem como uma série de serviços de gestão dos objectos relacionados com o modelo de eventos (criar, apagar, alterar, adquirir atributos). De notar ainda que não é trivial a modificação (pelos clientes) da estrutura de um modelo de eventos ao nível do servidor quando este em exploração.

Propõe-se, por isso, uma arquitectura que implementa o equivalente ao sistema fornecido pelo modelo de eventos do MMS, implementado de uma forma menos elegante, mas sem dúvida mais eficiente e menos exigente em termos de implementação, quer ao nível do servidor, quer ao nível dos clientes MMS.

Propõe-se então a existência no servidor de uma variável MMS, associada a cada módulo de E/S, sendo o seu nome *SUBSCR_MD*. Essa variável poderá ser escrita por qualquer cliente MMS. Uma escrita nessa variável MMS será mapeada em funções de manipulação de uma tabela de controlo de subscrição existente na aplicação do servidor.

A variável *SUBSCR_MD* será estruturada e terá três campos.

```
typedef struct
{
        unsigned char  event_type;
        unsigned char  io_num;
        unsigned char  status;
}subscr_C_Type;
```

O campo *event_type* poderá ter os valores *DIGIT*, para subcrever a 'indicação' da rede de *Field Bus DIG_B9*, *ALARM*, para subscrever as 'indicações' *ALM_B0* e *STA_B1*, e por fim *COUNTER* que servirá para subscrever as indicações relacionadas com os contadores, isto é, *CPI_B2*, *CPT_B3*, *PCI_C2* e *PCT_C3*.

O campo *io_num* indicará a entrada à qual se deve aplicar e só terá significado se event_type for diferente de *DIGIT*. O campo *status* poderá valer *ON* ou *OFF* para iniciar ou acabar uma determinada subscrição. À estrutura de controlo dos diferentes módulos deverão ser adicionados os seguintes campos:

```
unsigned char        digit_subscr [8];
unsigned char        alarms_subscr [32][8];
unsigned char        counter_subscr [32][8];
```

Esta estrutura de informação permite ter informação relativa a potenciais 8 canais OSI. Se valer '1', quererá dizer que é preciso enviar MMS *InformationReport* a esse canal.

Sempre que o servidor receber uma indicação MMS *Conclude*, deverá fazer-se o *reset* desse canal em todos os campos de subscrição dos módulos. A indicação MMS *Write.Ind(SUBSCR_MD)* será, obviamente, respondida sincronamente.

7.3.7. De Outros Mapeamentos

1 - Comandos de Programação a Enviar aos Módulos de Field Bus

Esta questão diz respeito à necessidade de mapear, no servidor, a possibilidade de um determinado cliente MMS poder enviar comandos de funcionamento e outros para a rede de *Field Bus*. Por exemplo, *EVO_47*, *SEV_48*, *DRN_D4*, *WCI_C0*, *WCT_C1*, etc. Estes serviços não corresponderão ao mapeamento normal numa variável, pelo que se sugere um outro tipo de mapeamento.

Deverá existir para cada módulo uma variável que deverá ser escrita por MMS em modo síncrono e sem confirmação remota, e que se propõe tenha o nome *COMAND_MD*. A estrutura C dessa variável poderá ser a seguinte:

```
typedef struct _comand_md_c_type
{
        unsigned char  command;
        unsigned char  io_num;
        long           value;
        long           lev_0_secs;
        long           lev_0_mics;
        long           lev_1_secs;
        long           lev_1_mics;
}comand_md_c_type;
```

Dependerá do número de comando o tipo de mapeamento que é feito. Por exemplo, no caso de *command* valer 47h, será enviado um *EVO_47* a esse módulo, e será colocada a variável da estrutura *module_ctrl evo_mode* a valer '1'. Se o valor de *command* for 48h será enviado um *SEV_48* e será colocada a zero a variável referida anteriormente. Os campos de *value* e *lev_x_xxxx* serão utilizados para veicular a informação referente aos comando *WCI_C0* e *WCT_C1*, respectivamente.

2 - Informação de Estado dos Módulos da Rede de Field Bus

Algumas das mensagens gerais da rede de *Field Bus*, mensagens cujos códigos estão entre 00h e 3Fh, poderão ser utilizadas no âmbito do modelo MMS de Gestão de VMD (ver capítulo 6).

Essas mensagens, cuja funcionalidade está descrita no capítulo 5, são as 'indicações' *DEF_ 04*, *NER_05*, *AER_16*, *MAS_17*, *RES_18*, *RFX_19* e *FLG_46* e os pedidos *DER_06*, *DMA_07*, *DRE_08*, *DFX_09* e *AFX_49*. *FLG_46* e *AFX_49* são duas mensagens que fazem parte do grupo de mensagens particulares dos módulos, podem no entanto estar relacionadas com o modelo de gestão de VMD.

Os serviços de suporte de VMD do MMS correspondem ao *Status*, *UnsolicitedStatus*, *GetNameList*, *Identify*, *Rename*, *GetCapabilityList*. No caso da arquitectura aqui proposta têm algum interesse os quatro primeiros, mas, em particular, os dois primeiros.

O serviço MMS *Status* e o serviço MMS *UnsolicitedStatus* apenas diferem um do outro na medida em que o primeiro se trata de um pedido feito por um cliente MMS no sentido de aquilatar do estado do VMD enquanto que o segundo é utilizado por iniciativa do próprio servidor de MMS e mediante determinadas condições da aplicação. A resposta do MMS *Status* e o pedido do MMS *UnsolicitedStatus* têm três parâmetros. São eles o *VMDLogicalStatus*, o *VMDPhysicalStatus* e o *LocalDetail*. A norma MMS [MMS1 - 90] especifica que quer o primeiro

quer o segundo dos parâmetros correspondem a atributos do objecto VMD. O terceiro parâmetro corresponde a informação adicional do estado e corresponde a uma cadeia de comprimento até 128 *bits* com um significado definido em particular para o servidor. É ainda de notar que no pedido de MMS *Status*, existe um único parâmetro: *ExtendedDerivation*. Este parâmetro, do tipo booleano, permite a indicação de dois tipos de métodos para a determinação do estado do VMD.

A norma MMS de especificação de protocolo [MMS2 - 90] define, em ASN.1, a resposta ao pedido de MMS *Status* da maneira que se indica a seguir. Esta definição, que inclui os *bindings*, é igual à do pedido MMS *UnsolicitedStatus*:

```
Status-Response ::= SEQUENCE {
        vmdLogicalStatus        [0] IMPLICIT INTEGER  {
                state-changes-allowed           (0),
                no-state-changes-allowed        (1),
                limited-services-permited       (2),
                support-services-allowed        (3)
                },
        vmdPhysicalStatus       [1] IMPLICIT INTEGER  {
                operational                     (0),
                partionally-operational         (1),
                inoperable                      (2),
                needs-commissioning             (3)
                },
        localDetail             [2]     IMPLICIT BIT STRING (SIZE(0..128)) OPTIONAL
        }
```

O estado lógico do VMD reflecte restrições no conjunto de serviços capazes de operar sobre o VMD. O estado lógico '0', indica que todos os serviços suportados pelo VMD podem ser executados. O estado lógico '1' indica que não deverão ser suportandos serviços que alterem o estado do VMD (ex. MMS *Write*, MMS *Define****, MMS *Delete****,...). O estado lógico '2' só permite que sejam executados os seguintes serviços MMS: *Abort*, *Conclude*, *Status*, *Identify*. O estado lógico '3' permite todos os serviços suportados pelo VMD, à excepção do *Start*, *Stop*, *Reset*, *Resume* e *Kill*.

Quando for recebido um pedido de MMS *Status* é preciso obter o estado do dispositivo real. No caso presento do dispositivo de integração esse mapeamento não é linear, uma vez que o único VMD do servidor mapeia ***N*** módulos da rede de *Field Bus*.

Assim, por exemplo, não utilizando *ExtendedDerivation*, poderá bastar informar se cada módulo está ou não desligado. Isto é possível se se mantiver na aplicação uma estrutura de dados que será actualizada sempre que chegar o serviço da rede de *Field Bus DEF_04*. Basta construir a trama de *localDetail* com essa informação. Neste caso o pedido MMS *Status* é respondido em modo síncrono.

No modo derivado podem-se utilizar outros serviços da rede de *Field Bus* para obter informação adicional. Ao receber um pedido MMS de *Status*, o dispositivo poderá enviar os seguintes pedidos à rede de *Field Bus*: *DER_06*, *DMA_07*, *DRE_08*, *DFX_09*. Neste caso, o

serviço MMS *Status* será respondido assincronamente, até que sejam obtidas as respostas aos referidos pedidos. A sincronização das respostas MMS, será idêntica à utilizada para o acesso aos valores das E/S dos módulos. A chegada das 'indicações' *AER_16*, *MAS_17*, *RES_18*, *RFX_19*, que são respectivamente as respostas aos pedidos anteriormente referidos, irá colocar a '1' as variáveis de controlo de sincronização associadas a cada pedido. Enquanto estiver a decorrer um pedido de MMS *Status* em modo derivado, não deverá ser aceite um outro também em modo derivado.

A 'indicação' *AER_16* fornece, em dois passos, os erros ocorridos em cada módulo (até 250 módulos). Envia ainda o número de erros totais e o número total de transmissões ocorridas na rede.

A 'indicação' *MAS_17* retorna o endereço do módulo, o endereço da rede e o estado da rede. Relativamente ao estado da rede, ela pode estar inactiva (sem mestre), o controlador pode corresponder ao mestre (situação mais normal), o controlador pode corresponder a um escravo ou o módulo conectado não tem um parâmetro válido.

A 'indicação' *RES_18* retorna o estado de todos os módulos localizados na rede. O estado pode ser: 0 - não localizado na rede, 1 - módulo na tabela de *polling* síncrono, 2 - módulo na tabela de *polling* assíncrono, 3 - o módulo não está na tabela de *polling*, 4 - o módulo está em modo *download*.

Por fim, a 'indicação' *RFX_19* retorna, em dois passos, o estado dos *buffers* de todos os módulos na rede, permitindo assim algum controlo de fluxo. Para cada módulo, retorna '0' (*buffers* cheios) ou '1' (módulo aceita mensagens).

Estes serviços da rede de *Field Bus* são todos orientados à rede, pelo que basta considerar uma estrutura de dados para controlar o envio da resposta ao pedido MMS *Status* em modo assíncrono. Essa estrutura poderá ser a seguinte:

```
struct _status_updt_ctrl
{
        BYTE fil_net_errors[2];
        BYTE master_info;
        BYTE md_status[2];
        BYTE flow[2];
}status_updt_ctrl;
```

A estrutura de dados que guarda a informação veiculada nas 'indicações' *AER_16*, *MAS_17*, *RES_18*, *RFX_19* poderá ser a seguinte:

```
struct _Fil_Status_Info
{
        BYTE    master_addr;
        BYTE    net_addr;
        BYTE    net_state;
        struct _md_info
```

```
        {
                BYTE    num_errors;
                BYTE    md_state;
                BYTE    md_flow_ctrl;
        }md_info[8];
        USHORT  net_total_errors;
        ULONG   net_total_transmissions;
}Fil_Status_Info;
```

A informação presente nesta estrutura deverá ser codificada nos 128 *bits* disponíveis para o *LocalDetail*. O problema reside no número reduzido de *bits* disponíveis, sabendo-se que uma rede de *Field Bus* poderá ter um número elevado de módulos. Dever-se-á compactar essa informação.

As mensagens *DEF_04*, que assinala a conexão ou desconexão de um módulo, *NER_05*, que assinala a ocorrência de mais do que 256 erros em pelo menos um módulo, *MSS_14*, que assinala que um determinado mestre se vai desligar, e por fim *FLG_46*, que assinala o estado dos *buffers* do módulo, deverão ser mapeadas no serviço MMS de *UnsolicetedStatus*. A mensagem *AFX_49* pode ser enviada a um módulo (escrita da variável MMS *COMAND_MD*) se não se quiser que ele envie o serviço *FLG_46*. O comando *AFX_49* não afecta o controlador.

Obviamente, o MMS *UnsolicitedStatus* deverá ser enviado através de todos os canais OSI activos, informando assim todos os clientes MMS que estão a utilizar informação da rede de *Field Bus*.

Field Bus		**Servidor**	**Cliente(s) MMS**
DEF_04 ou NER_05 ou FLG_46	→		
		UnsolicitedStatus.Req	→
		...	
		UnsolicitedStatus.Req	→

Tabela 7.14 *- Utilização do MMS UnsolicitedStatus*

Cliente MMS		**Servidor**		**Field Bus**
Status.Req	→			
		DER_06	→	
		DMA_07	→	
		DRE_08	→	
		DFX_09	→	
			←	2 x *AER_16*
			←	*MAS_17*
			←	RES_18
			←	2 x *RFX_19*
	←	*Status.Rsp*		

Tabela 7.15 *- Utilização do MMS Status (modo Derivado)*

Cliente MMS		Servidor	Field Bus
Status.Req	⟶		
	⟵	*Status.Rsp*	

Tabela 7.16 *- Utilização do MMS Status (modo não Derivado)*

3 - Outros Serviços da Rede de Field Bus

A maioría dos serviços de módulos já foi anteriormente referida. Podem ser feitas considerações ácerca de alguns outros não muito relevantes para a análise da arquitectura proposta.

Por exemplo, o serviço *DRL_65* pede o envio de relatórios automáticos (*PCI_C1* e *PCT_C2*) que tivessem sido armazenados localmente. Esta funcionalidade não necessita de ser considerada. Os serviços *RUN_42*, *STP_43*, *CLR_45*, *DYP_4A*, *MAH_5 e TYP_5A* não terão mapeamento para o MAP/MMS. São comandos utilizados no *setup* ou restruturação da rede, ou ainda para fazer face a situações anómalas. Isto pode não ser válido para o *MAH_52*, que permite a programação da hora e da data. Não deverá ser o cliente a enviar o valor da hora. Esse valor deverá ser obtido localmente no servidor e enviada para a rede de *Field Bus*.

Em relação aos serviços de E/S é de referir que *DAT_80* é um serviço específico dos módulos de saídas por interface série.

7.4. Caracterização Geral da Arquitectura Proposta

7.4.1. Das Variáveis MMS

	Variável	Serviços MMS de Acesso	Tipos de Módulos
1.	PARIOS_xx	Read e Write	FIL-PC, FILINP, FILOUT
2.	IOGLOI_MD	Read, InformationReport	FILINP
3.	IOGLOO_MD	Read, Write, InformationReport	FILOUT
4.	IOINDI_xx	Read, InformationReport	FILINP
5.	IOINDO_xx	Read, Write	FILOUT
6.	COMAND_MD	Write	FIL-PC, FILINP, FILOUT
7.	ALARMS_MD	InformationReport	FILINP, FILOUT
8.	SUBSCR_MD	Write	FILINP, FILOUT

Tabela 7.14 *- Variáveis MMS Consideradas e Tipo de Acesso*

7.4.2. Atendimento de Indicações de MMS Read em 1ª Etapa

A *Fig. 7.18* corresponde ao atendimento de uma indicação MMS *Read* em primeira etapa.

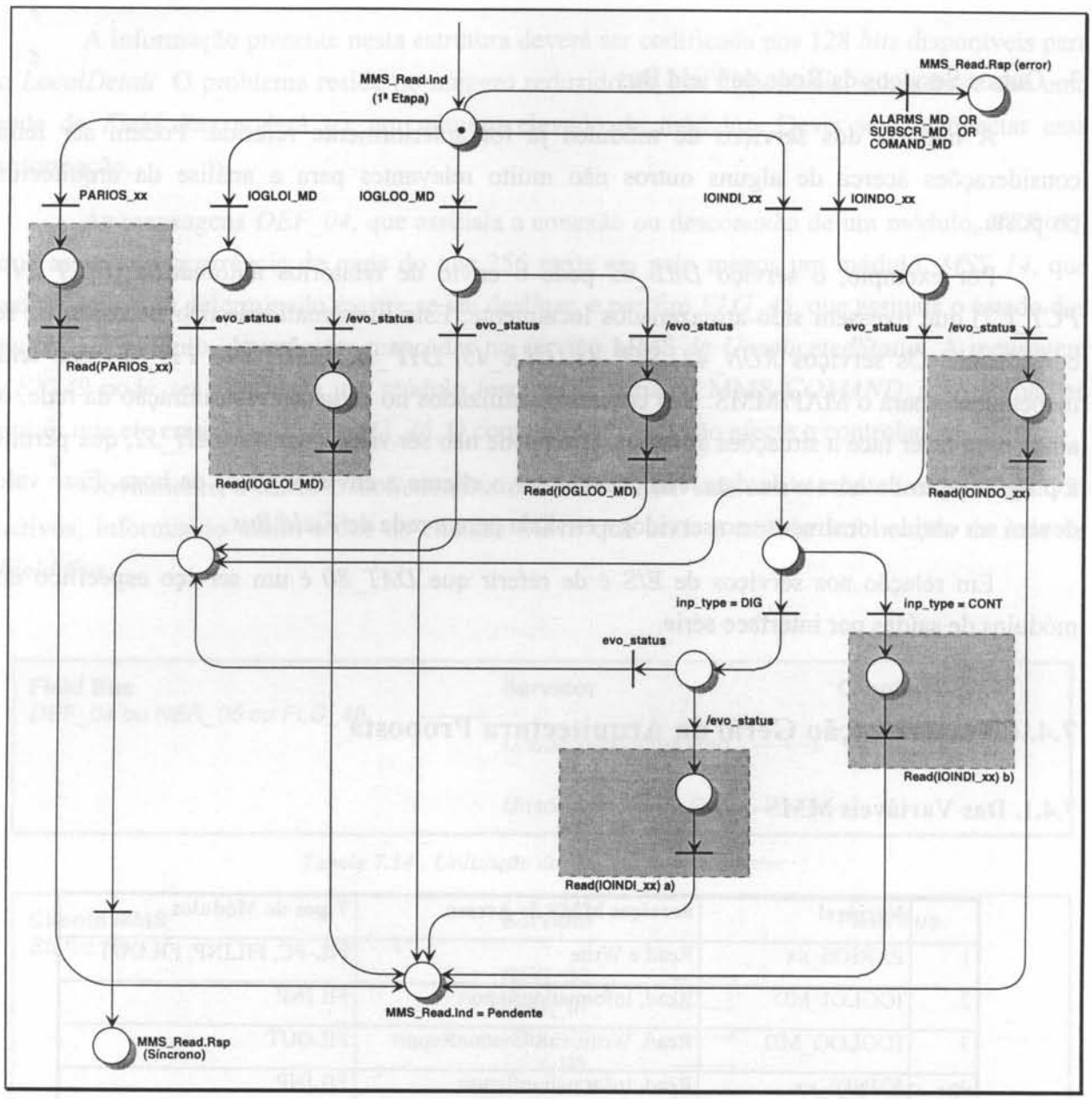


Figura 7.18 *- Rede de Petri Para o Atendimento de Indicações MMS Read em 1ª Etapa*

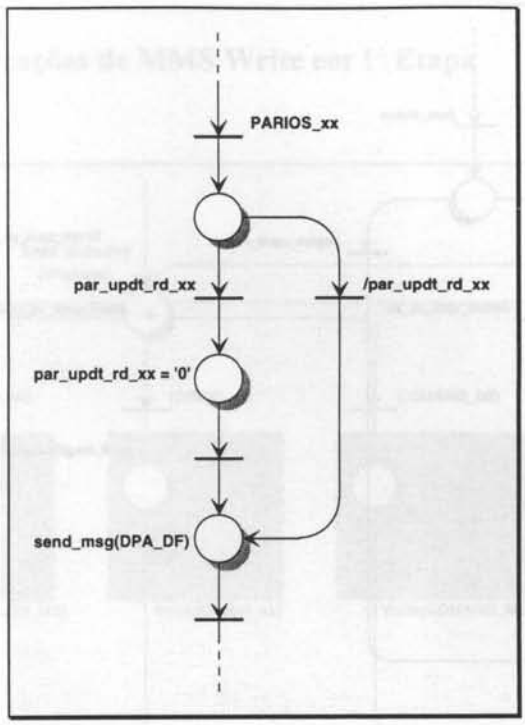


Figura 7.19 *- RdP de Processamento de Uma Indicação de MMS Read (PARIOS_xx)*

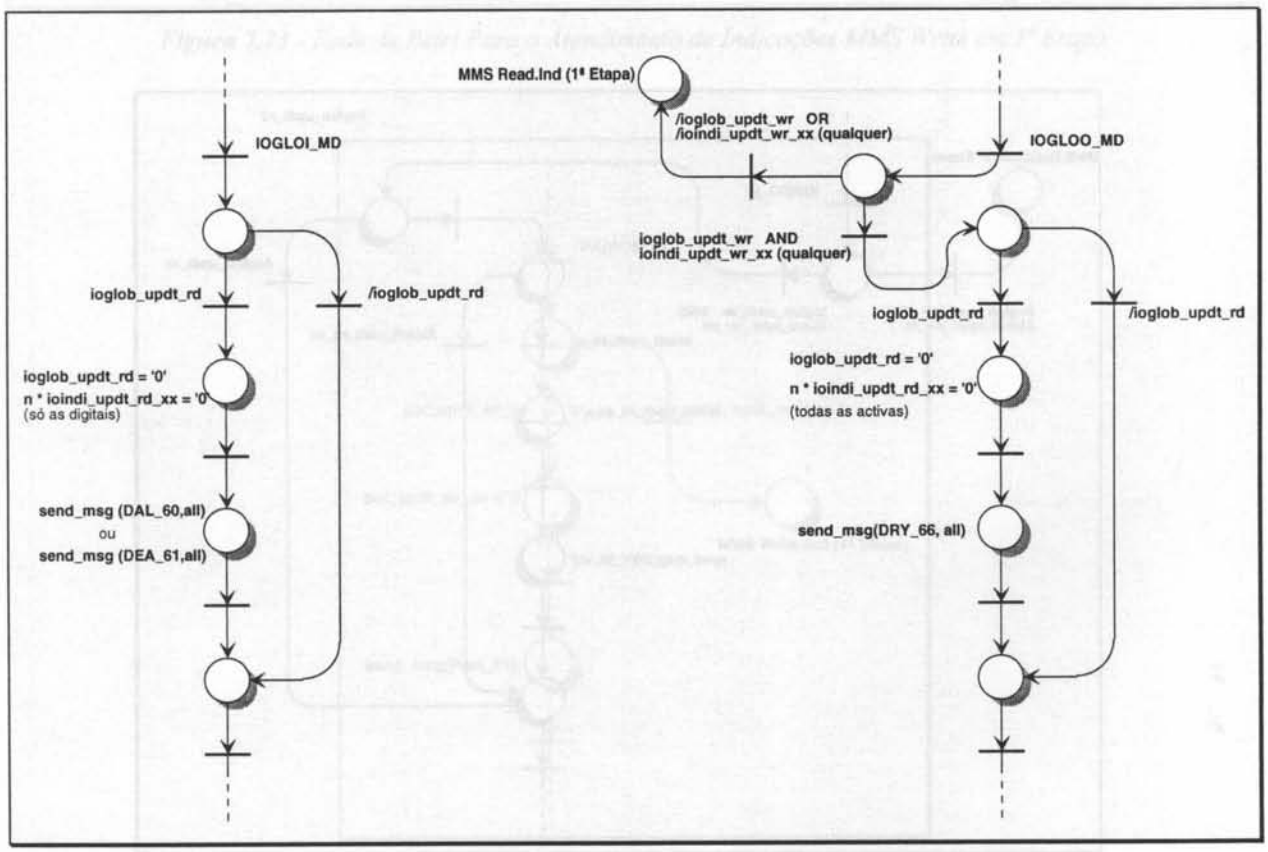


Figura 7.20 *- RdPs de Processamento de Indicações MMS Read (IOGLOI_MD) ou MMS Read (IOGLOO_MD)*

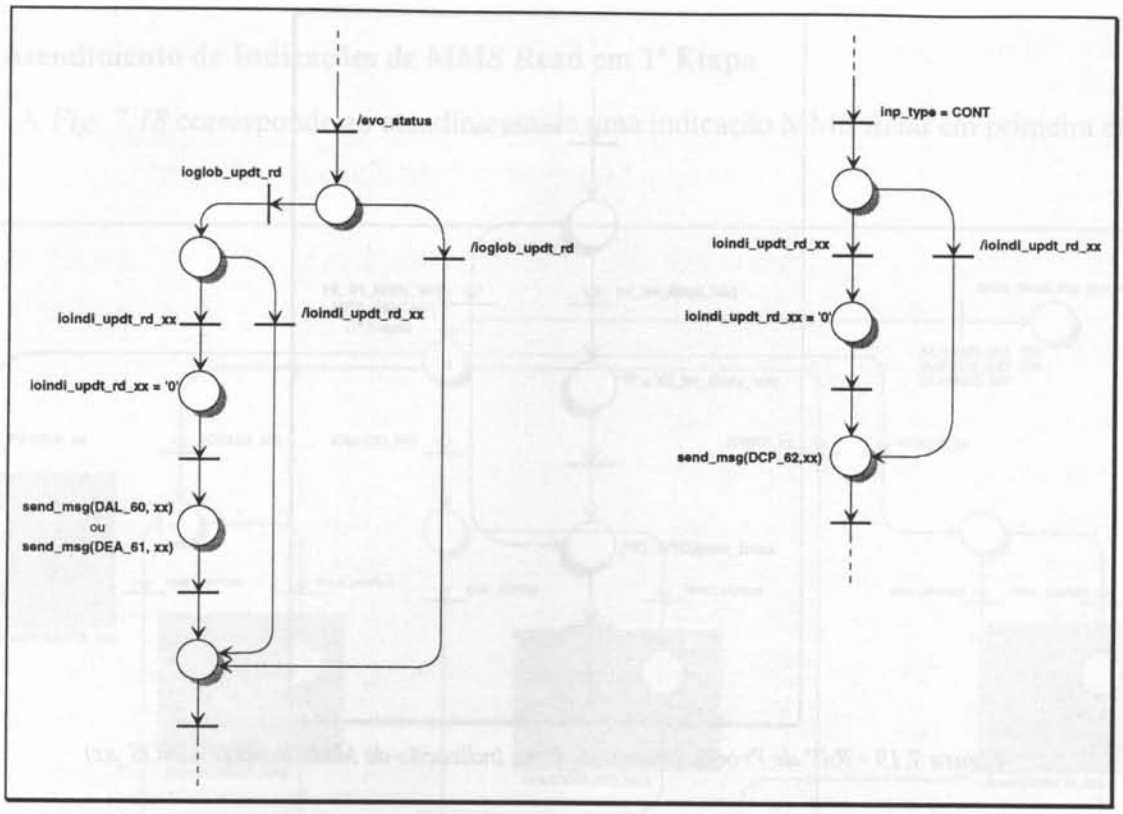


Figura 7.21 *- RdPs de Processamento de Indicações MMS Read (IOINDI_xx)*

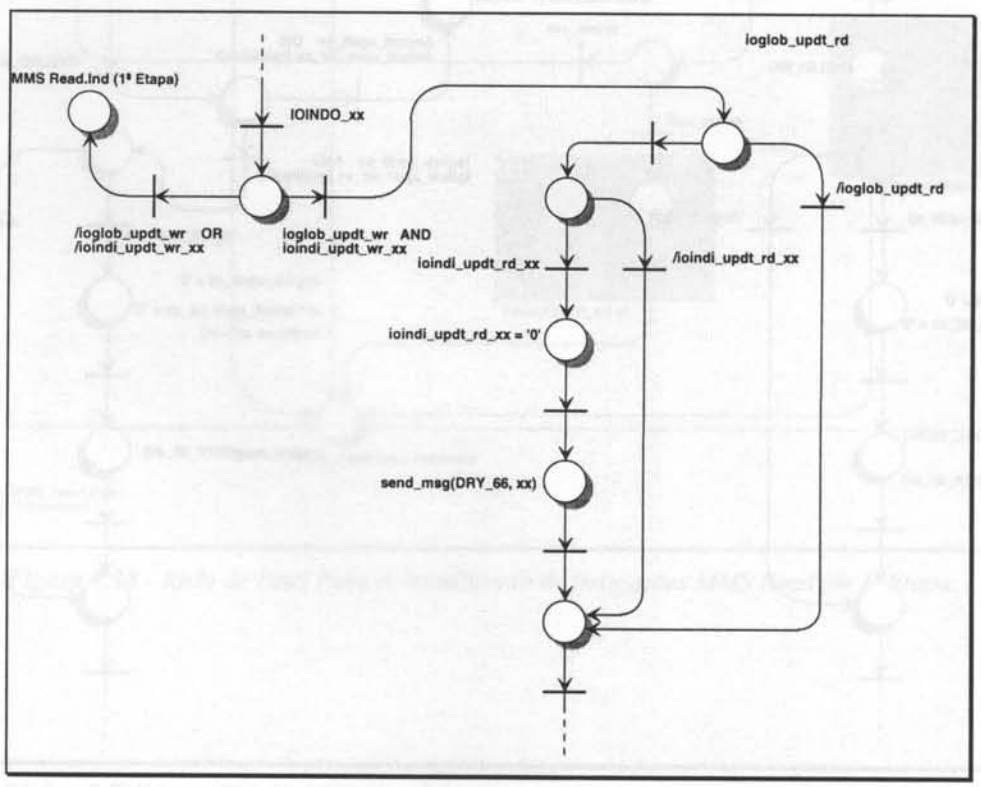


Figura 7.22 *- RdP Para o Processamento de Uma Indicação de Leitura MMS Read (IOINDO_xx)*

7.4.3. Atendimento de Indicações de MMS Write em 1ª Etapa

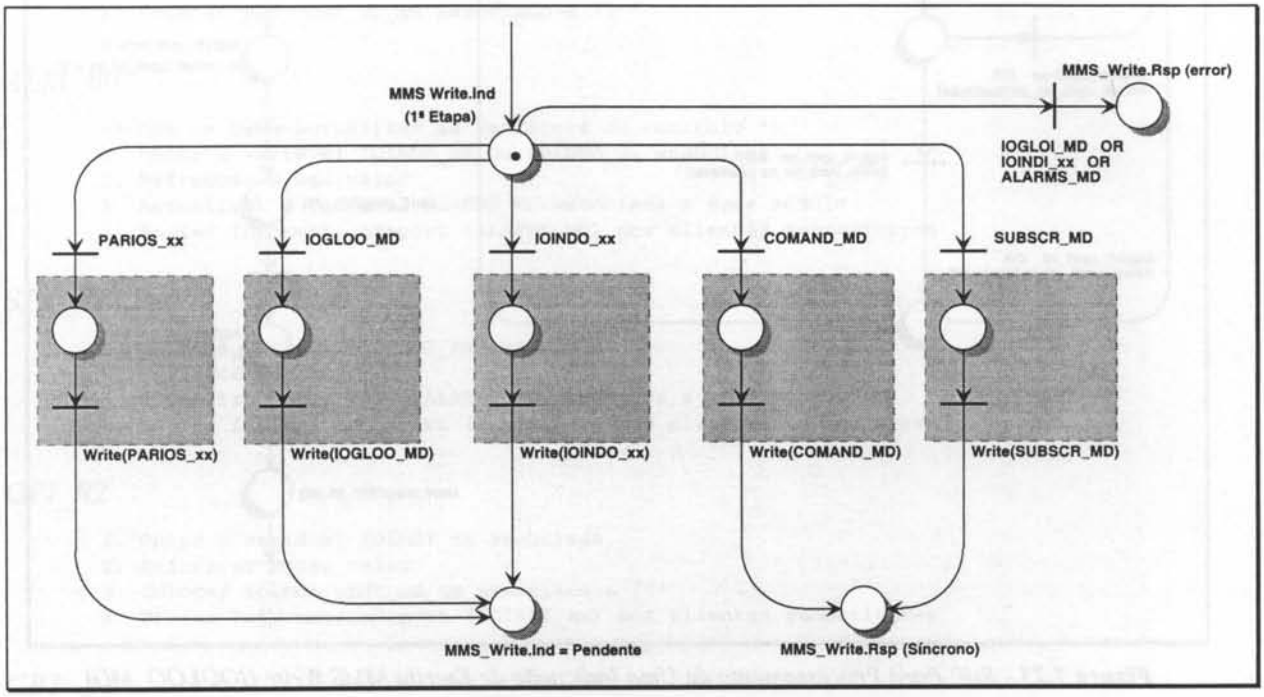


Figura 7.23 *- Rede de Petri Para o Atendimento de Indicações MMS Write em 1ª Etapa*

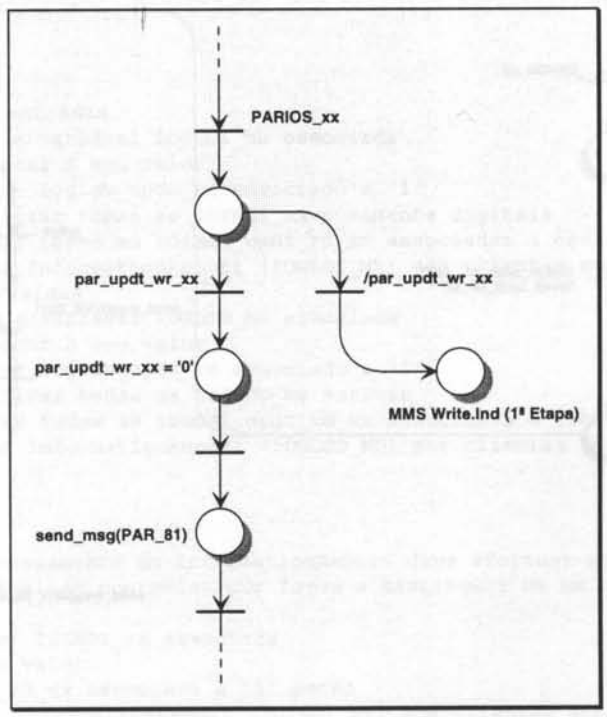


Figura 7.24 *- RdP Para Processamento de Uma Indicação de Escrita MMS Write (PARIOS_xx)*

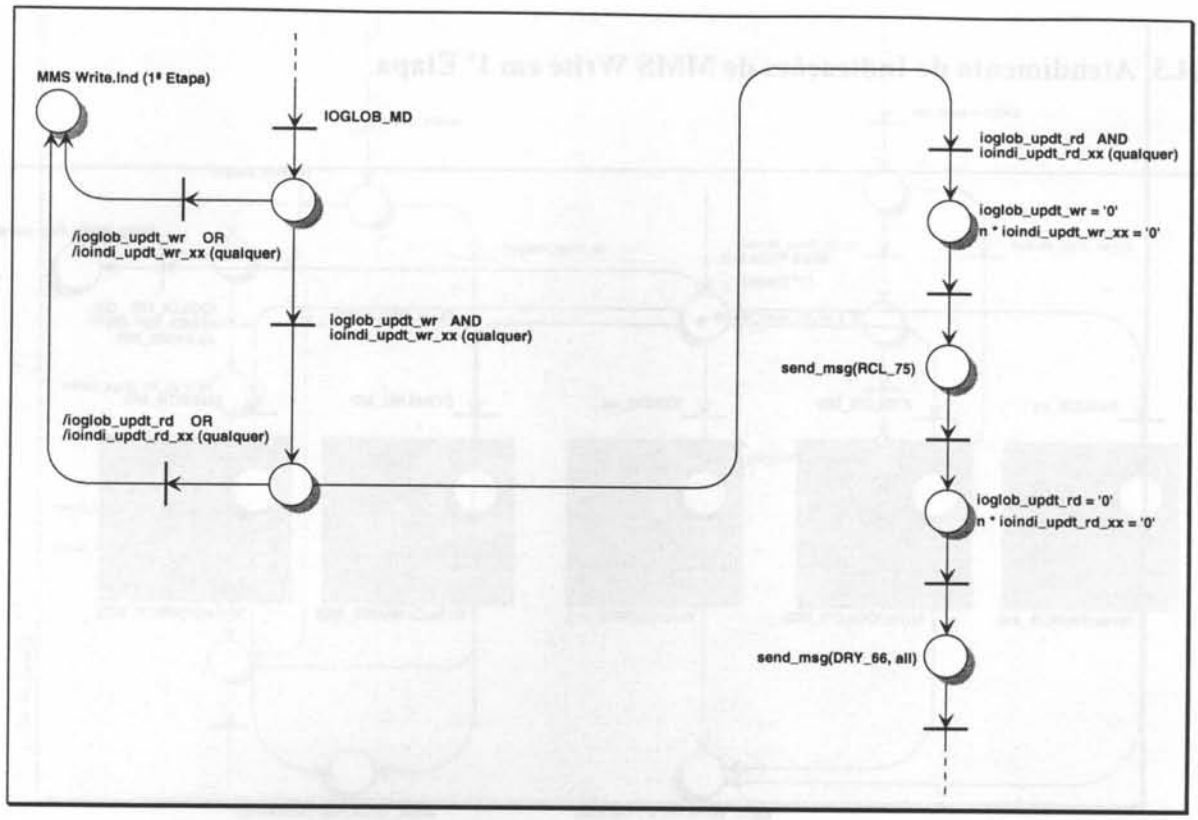


Figura 7.25 *- RdP Para Processamento de Uma Indicação de Escrita MMS Write (IOGLOO_MD)*

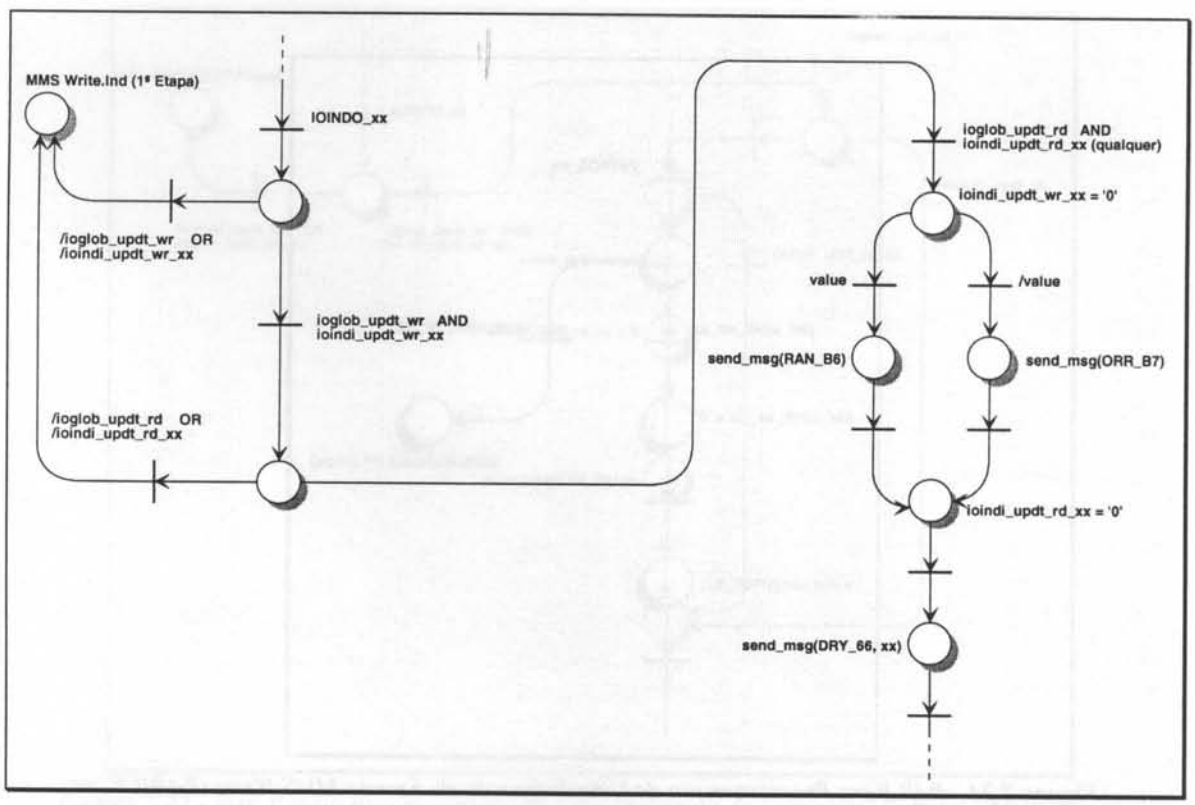


Figura 7.26 *- RdP Para Processamento de Uma Indicação de Escrita MMS Write (IOINDO_xx)*

7.4.4. Processamento das Indicações Recebidas do Field Bus

ACK_91

```
1. Colocar par_updt_wr_xx associado a '1'
```

ALM_B0

```
(* não se deve actualizar as variáveis de controlo *)
1. Obter a variável IOINDI_xx ou IOINDO_xx associada
2. Refrescar o seu valor
3. Actualizar a variável ALARMS_MD associada a esse módulo
4. Enviar InformationReport (ALARMS_MD) aos clientes subscritores
```

STA_B1

```
1. Obter a variável IOINDI_xx associada
2. Refrescar o seu valor
3. Actualizar a variável ALARMS_MD associada a esse módulo
4. Enviar InformationReport (ALARMS_MD) aos clientes subscritores
```

CPI_B2

```
1. Obter a variável IOINDI_xx associada
2. Refrescar o seu valor
3. Colocar ioindi_updt_rd_xx associado a '1'
4. Enviar InformationReport (IOINDI_xx) aos clientes subscritores
```

CPT_B3

```
1. Obter a variável IOINDI_xx associada
2. Refrescar o seu valor
3. Colocar ioindi_updt_rd_xx associado a '1'
4. Enviar InformationReport (IOINDI_xx) aos clientes subscritores
```

DIG_B9

```
1. Se é módulo de entradas
        1.1. Obter a variável IOGLOI_MD associada
        1.2. Refrescar o seu valor
        1.3. Colocar ioglob_updt_rd associado a '1'
        1.4. Actualizar todas as IOINDI_xx puramente digitais
        1.5. Colocar todas as ioindi_updt_rd_xx associadas a essas variáveis a '1'
        1.6. Enviar InformationReport (IOGLOI_MD) aos clientes subscritores
2. Se é módulo de saídas
        2.1. Obter a variável IOGLOO_MD associada
        2.2. Refrescar o seu valor
        2.3. Colocar ioglob_updt_rd associado a '1'
        2.4. Actualizar todas as IOINDO_xx activas
        2.5. Colocar todas as ioindi_updt_rd_xx associadas a essas variáveis a '1'
        2.6. Enviar InformationReport (IOGLOO_MD) aos clientes subscritores
```

PAL_C0

```
(* O eventual processamento do InformationReport deve efectuar-se antes de *)
(* alterar a variável de controlo, por forma a distinguir de um pedido    *)

1. Obter a variável IOINDI_xx associada
2. Refrescar o seu valor
3. Se ioindi_updt_rd_xx associado a '1' então
        3.1 Enviar InformationReport (IOINDI_xx) aos clientes subscritores
4. Colocar ioindi_updt_rd_xx associado a '1'
5. Se ioglob_updt_rd == 0 então
        5.1. Se todos os ioindi_updt_rd_xx(dig.) == 1 então ioglob_updt_rd = 1
```

PST_C1

```
(* O eventual processamento do InformationReport deve efectuar-se antes de *)
(* alterar a variável de controlo, por forma a distinguir de um pedido      *)

1. Obter a variável IOINDI_xx associada
2. Refrescar o seu valor
3. Se ioindi_updt_rd_xx associado a '1' então
       3.1 Enviar InformationReport (IOINDI_xx) aos clientes subscritores
4. Colocar ioindi_updt_rd_xx associado a '1'
5. Se ioglob_updt_rd == 0 então
       5.1. Se todos os ioindi_updt_rd_xx(dig.) == 1 então ioglob_updt_rd = 1
```

PCI_C2

```
(* O eventual processamento do InformationReport deve efectuar-se antes de *)
(* alterar a variável de controlo, por forma a distinguir de um pedido      *)

1. Obter a variável IOINDI_xx associada
2. Refrescar o seu valor
3. Se ioindi_updt_rd_xx associado a '1' então
       3.1 Enviar InformationReport (IOINDI_xx) aos clientes subscritores
4. Colocar ioindi_updt_rd_xx associado a '1'
```

PCT_C3

```
(* O eventual processamento do InformationReport deve efectuar-se antes de *)
(* alterar a variável de controlo, por forma a distinguir de um pedido      *)

1. Obter a variável IOINDI_xx associada
2. Refrescar o seu valor
3. Se ioindi_updt_rd_xx associado a '1' então
       3.1 Enviar InformationReport (IOINDI_xx) aos clientes subscritores
4. Colocar ioindi_updt_rd_xx associado a '1'
```

PRY_C5

```
(* Em termos teóricos até poderá haver escritas em simultâneo com leituras *)

1. Obter a variável IOINDO_xx associada
2. Refrescar o seu valor
3. Colocar ioindi_updt_rd_xx associado a '1'
4. Se ioglob_updt_rd == 0 então
       4.1. Se todos os ioindi_updt_rd_xx == 1 então ioglob_updt_rd = 1
5. Se ioindi_updt_wr_xx == 0 então
       5.1. ioindi_updt_wr_xx = 1
       5.2. Se ioglob_updt_wr == 0 então
              5.2.1. Se todos os ioindi_updt_wr_xx == 1 então ioglob_updt_wr = 1
```

7.4.5. Aspectos de Conformidade do Servidor de MMS

1 - Serviços Suportados Como *User Responder*

Initiate, *Conclude*

Status, *GetNameList*, *Identify*

Read, *Write*

2 - Serviços Suportados Como *User Requester*

UnsolicitedStatus

InformationReport

3 - Parâmetros CBB Suportados Por Este Servidor

STR1, STR2, NEST ≤ 2, *VNAM*

7.5. Referências e Bibliografia

[ACDJ - 85] **J. M. Ayache, J. P. Courtiat, M. Diaz, G. Juanole**, *Utilization des Réseaux de Petri pour la Modelization et la Validation des Protocoles*, Techenique et Science Informatiques, vol. 4, nº 1, 1985.

[MASM - 88] **Marathe, Smith**, *Performances of a MAP Network Adapter*, IEEE Network, Vol. 2, No. 3, Maio 1988.

[STWE - 88] **W. Strayer, A. Weaver**, *Performance Mesurements of Data Transfer Services in MAP*, IEEE Network, Vol. 2, No. 3, pp. 75-81, Maio 1988.

[DAAL - 89] **René David, Hassane Alla**, *Du Grafecet aux Resaux de Petri*, Editions Hermés, PARIS 1989.

[JURO - 89] **G. Juanole, J. L. Roux**, *On the Pertinence of the Extended Time Petri Net Model for Analysing Communications Activities*, Editions Hermés, IEEE 1989.

[WGR - 89] **Fei-Yue Wang, Kevin Gildea, Alan Rubenstein**, *A colored Petri Net model for Connection Management Services in MMS*, Computer Communication Review, nº3 Jul. 1989.

[ASN1 - 90] ISO/IEC 8824, *Specification of Abstract Syntax Notation One (ASN.1)*, ISO/IEC 1990.

[BER - 90] ISO/IEC 8825, *Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*, ISO/IEC 1990.

[FJV - 90] **Cecile Faure, Guy Juanole, Francois Veradat**, *LAPD Protocol at the ISDN Interface: Formal Modeling with Petri nets based Models and Verification by Abstraction*, International Conference on Computer Communication, 1990 New Delhi, India.

[MMS1 - 90] ISO/IEC 9506-1, *Manufacturing Part Specification, Part 1 Service Definition*, ISO/IEC 1990.

[MMS2 - 90] ISO/IEC 9506-2, *Manufacturing Part Specification, Part 2 Protocol Specification*, ISO/IEC 1990.

[MOE - 90] **Moertazavi**, *Performance of MAP in Remote Operation of a CNC*, IEEE Transactions on Software Engeneering, Vol. 16, No. 2, Fevereiro 1990.

[VACI - 90] **Valenzano, Ciminiera**, *Performance Evaluation of MiniMAP Networks*, IEEE Transactions on Industrial Electronics, Vol. 37, No. 3, Junho 1990.

[WIL - 90] **Christopher Williams**, *Client Server and Producer Consumer Models*, IEC 65C WG6 / ISA SP 50 Field Bus Editing Committee, Setembro 1990.

[CMPA - 91] *Computrol's LP-40 & LP-42 ISOcomm 802.3 Controller Boards - Specifications*, AEG-Modicon, Junho 1991.

[CMPB - 91] *Computrol LP-40/LP-42 Controller Installation and Operation Manual*, AEG-Modicon, Junho 1991.

[LLO - 91] **Jean Christophe Lloret**, *Compositional Design and Verification of Communication Protocols, using Labelled Petri*, DIMACS Series in Discrete Mathematics and Theoretica. Computer Science Vol. 3 1991.

[CIS - 92] *Computrol ISOcomm Software for PC-XT and PC-AT Operating Systems User Guide*, AEG-Modicon, Abril 1992.

[HFSM - 92] *FILBUS Hardware and Firmware Specification Manual V1.1*, GESPAC International SA, Abril 1992.

[SSMS - 92] *FILBUS System Software for MS-DOS Systems, V2.3*, GESPAC International SA, Abril 1992.

[USER - 92] *FICIM User Requirements Study*, Fraunhofer IITB, Doc. 92.015/D05/RD, Novembro 1992.

[INDS - 94] *FILINP-16xx Data Sheet*, GESPAC International SA, 1994.

[OUDS - 94] *FILOUT-16xx Data Sheet*, GESPAC International SA, 1994.

[CDS - 94] *FILCIO-16xx Data Sheet*, GESPAC International SA, 1994.

Capítulo 8
Apontamentos de Implementação

Este capítulo aborda questões relacionadas com a implementação do dispositivo de interligação, de acordo com a arquitectura proposta no capítulo anterior. Faz-se primeiro uma breve introdução à interface MMS considerada, o MMS-EASE da SISCO, versão para DOS.

8.1. Introdução

Na *Fig. 8.1*, é retratada a infraestrutura de comunicações e de desenvolvimento utilizada em parte deste estudo. Realça-se a entidade designada por PC-DOS, correspondente ao dispositivo de interligação. É este computador que suporta a ligação a uma rede FILBUS, a uma rede MAP, e é nele que correrá a aplicação responsável pela interligação.

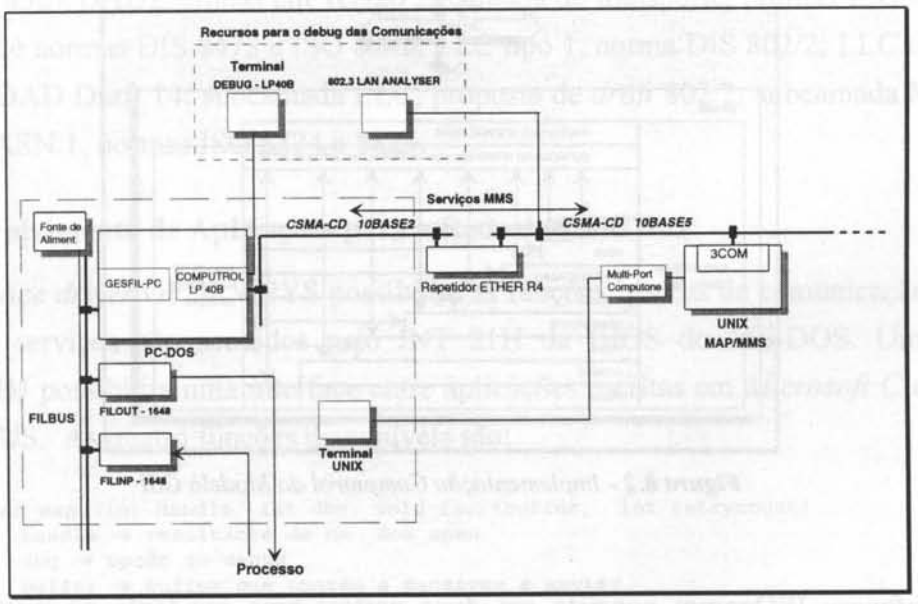


***Figura 8.1** - Descrição Genérica do Infraestrutura de Comunicações*

A aplicação desenvolvida para esse PC-DOS é em *Microsoft C* [MSCR - 90] [MSCI - 90] [MSCA - 90], uma vez que ambas as arquitecturas de comunicações, isto é, o MAP/MMS-EASE por um lado e a rede de comunicações FILBUS por outro, assim o exigem.

8.1.1. Considerações Genéricas Sobre o Hardware e Software MAP 3.0

A carta de comunicações MAP considerada é uma Computrol LP40B [CMPA - 91] [CMPB - 91]. Esta carta de comunicações, juntamente com o *software* ISOcomm [CIS - 92], permite a implementação das sete camadas do modelo OSI, tal como definidas na especificação *MAP 3.0 /TOP 1991 Specification*, libertando o computador de todas as tarefas de comunicação, à excepção do MMS obviamente. De notar que uma outra versão desta carta de comunicações, a LP42B, tem 2Mbytes de memória RAM, ao contrário do 1Mbytes da carta aqui considerada. O objectivo é tornar possível a inclusão de versões específicas do MMS ao nível da própria carta, libertando por isso os recursos do PC, para outras tarefas. Esta possibilidade ainda é mais relevante quando em presença de plataformas MS-DOS utilizadas para a interligação com redes de comunicações de tempo crítico, como é o caso da rede de *Field Bus* considerada.

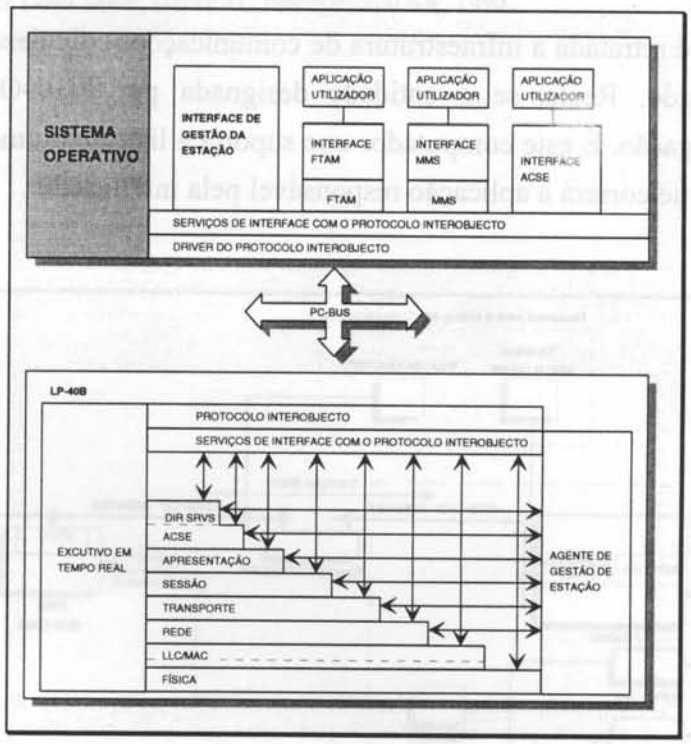


***Figura 8.2** - Implementação Computrol do Modelo OSI*

O *software* ISOcomm consiste em duas partes: uma residente ao nível da carta de comunicações e outra residente ao nível do *host*. A primeira contém a implementação dos sete

níveis do modelo de referência OSI. Cada um dos níveis está implementado numa tarefa separada debaixo do controlo de um executivo em tempo real, designado pelo fabricante de MAPRTX. O *software* residente no *host*, inclui um *device driver* que implementa um protocolo interobjecto da Computrol (IOP - *Interobject Protocol*), que permite aos programas de aplicação enviar e receber mensagens para e do controlador.

O *software* ISOcomm inclui ainda três utilitários. O MACSTAT permite visualizar estatísticas do MAC (*Medium Access Control*). O MAPGO faz o *reset*, carrega e inicializa o controlador MAP, e todo o *software* ISOcomm. Por fim, o MAPSMC possibilita a gestão do nó ISOcomm, em termos de configuração, diagnóstico e reinicialização. A *Fig. 8.2*, mostra a relação entre o *software* ISOcomm e o modelo OSI.

A carta de comunicações LP-40 inclui um microprocessador de 32 *bits*, o Motorola 68020 um controlador de rede AMD IEEE 802.3 LANCE, com interface física 802.3 AUI *Transceiver* (carta LP-40A) ou cabo fino (carta LP-40B), uma DRAM de 1Mbyte para onde se faz o *download* do *software* OSI, 1Kbyte de NVRAM (*Non-Volatile* RAM), onde são colocados os parâmetros de configuração de rede, um MC68801 *Multi-Function Peripheral* com saída série, que permite fazer o *debug on-line* (ver *Fig. 8.1*) e obviamente *interfaces* DMA e linhas de interrupção configuráveis.

O *software* ISOcomm obedece às seguintes normas: serviços de directório, implementação proprietária baseada no *draft* de norma internacional DIS 9594; MMS, norma DIS 9506; FTAM, norma ISO 8571/1-4; ACSE, norma ISO 8649/2 e 8650/2; camada de apresentação, norma ISO 8822 e 8823, camada de sessão, normas ISO 8326 e 8327, DIS 8326 DAD2 e DIS 8327 DAD2, ambas em versão 2; camada de transporte, normas ISO 8072 e 8073; camada de rede normas DIS 8473 e ISO 8348; LLC tipo 1, norma DIS 802/2; LLC tipo 3, norma DIS 802/2, PDAD Draft 14; subcamada LLC, proposta de *draft* 802.2; subcamada MAC, norma IEEE 802.3; ASN.1, normas ISO 8824 e 8825.

8.1.2. Desenvolvimento de Aplicações para a Rede de Field Bus

O *device driver* GFDRV.SYS possibilita as funções básicas de comunicação com a rede FILBUS. Os serviços são acedidos pelo INT 21H da BIOS do MS-DOS. Uma biblioteca FMSCLIB.OBJ possibilita uma interface entre aplicações escritas em *Microsoft C* e o *driver* da rede de FILBUS. As quatro funções disponíveis são:

```
int send_msg (int handle, int dbg, void far *buffer,  int retrycount)
        handle → resultante de um _dos_open
        dbg → opção de debug
        buffer → buffer que contém a mensagem a enviar
        retrycount → valor máximo de tentativas
```

```
int receive_msg (int handle, int dbg, void far *buffer)
        handle → resultante de um _dos_open
        dbg → opção de debug
        buffer → buffer para colocar a mensagem recebida

int get_status (int handle, unsigned char *GESFILstatus)
        handle → resultante de um _dos_open
        GESFILstatus → GESFIL Status value
int set_cmd (int handle, unsigned char *GESFILcmd)
        handle → resultante de um _dos_open
        GESFILcmd → GESFIL Command value
```

Existem ainda mais duas funções disponíveis em FILMON.OBJ. São elas:

```
status = start_network (handle, dbg, netname, GESFILname, netaddr)
        netname → nome fich. que contém a descrição da rede (sem extensão .net)
        GESFILname → nome do módulo  master  (sem a extensão .md)
        netaddr → endereço da rede

status = param_network (handle, dbg, netaddr, msgbuffer)
        netaddr → endereço da rede
        msgbuffer → apontador para a mensagem FILBUS
```

8.1.3. Utilização da Interface de Desenvolvimento do MMS-EASE

O MMS-EASE (*Manufacturing Message Specification - Embedded Application Element*) [EASE - 92] é uma livraria de funções C que fornece uma interface de alto nível e em tempo real para o MMS ASE (*Application Service Element*) do MAP. O MMS-EASE permite a utilização de dois tipos diferentes de interface.

Com um dos tipos, o PPI (*Paired Primitive Interface*), o utilizador tem total controlo sobre a máquina de estados do protocolo dos diferentes serviços MMS. No outro tipo, o VMI (*Virtual Machine Interface*), para além de executar as funções de comunicação, executa acções de processamento e de interpretação dos serviços de comunicações.

Por exemplo, se um nó "deseja" ler uma variável remota, o pedido virtual de leitura emitido (*Virtual Read Request*) apenas necessita de parâmetros relativamente ao número do canal OSI a utilizar, o nome e o tipo da variável remota e um apontador local para a colocação dos dados obtidos. A utilização de pedido de leitura do tipo PPI, ao contrário, obriga a um certo processamento local da primitiva de confirmação.

A interface de aplicação do MMS-EASE consiste em 13 classes distintas de funções.

A classe de funções 1, *Primitive Request Functions*, corresponde a um conjunto de funções relacionadas com as primitivas de pedido de serviço MMS utilizando a interface PPI. A sua forma genérica é *mp_xxxx (chan, info_ptr)*.

A classe de funções 2, *Primitive Response Functions*, está relacionada com o envio de primitivas de resposta MMS, utilizando a interface PPI. A forma genérica dessas funções é *mp_xxxx_resp (req_ptr, info_ptr)*.

A classe de funções 3, *Primitive User Confirmation Functions*, está relacionada com o atendimento de primitivas de confirmação MMS, utilizando a interface PPI. Tratam-se de funções definíveis pelo utilizador. A forma genérica dessas funções é *u_mp_xxxx_conf (req_ptr)*.

A classe de funções 4, *Error Response Functions*.

As classes de funções 5, 6 e 7, equivalentes a anteriores mas para interface VMI, como sejam as *Virtual Request Functions*, as *Virtual Response Functions* e a *Virtual User Confirmation Functions*, cujas formas genéricas são respectivamente, *mv_xxxx (chan, parm1, parm2, ...)*, *mv_xxxx_resp (req_ptr)* e *u_mv_xxxx_conf (req_ptr)*.

A classe de funções 8, *User Indication Functions Class*, que consiste num conjunto de funções definíveis pelo utilizador, chamadas automaticamente de acordo com o serviço MMS, e cuja forma genérica é *u_xxxx_ind (req_ptr)*.

A classe de funções 9, *LLP Request Functions Class*, cuja forma genérica é *mllp_xxxx (parm1, parm2, ...)* e fornecem a interface para o fornecedor da camada inferior (*Lower Layer Provider*).

A classe de funções 10, *User LLP Functions*, cuja forma genérica é *u_mllp_y_xxxx (parm1, parm2, ...)*.

A classe de funções 11, 12 e 13, as *Support Functions*, cuja forma genérica é, *ms_xxxx (parm1, parm2, ...)*, as *Address Resolution Functions* e as *Interface Functions*.

No MMS-EASE existem quatro classes de *Data Structures* que são passadas bidireccionalmente entre o MMS-EASE e o programa do utilizador. São elas:

- do sistema MMS-EASE: contêm o contexto da apresentação, informação de erro e variáveis de configuração do sistema;
- da máquina virtual: suportam a interface com a máquina virtual, contêm informação sobre domínios, gestão de contextos, variáveis e tipos;
- de filas (*queues*): suportam a informação genérica sobre as diferentes primitivas dos serviços (*requests*, *responses*, *indications* e *confirmations*);
- específicas de operações: são específicas de serviços particulares MMS.

8.1.4. Exemplo de Aplicação

O exemplo que se segue, é extraído de uma aplicação desenvolvida para teste do funcionamento do FILBUS com o MAP 3.0 da COMPUTROL. O programa detalhado consta como ANEXO A desta dissertação.

```
main()
{
        BYTE erro;
        int i;
        int stop_polling;
        unsigned int AX;
        unsigned char car;
        char mess[20];
        SHORT fil_timer;
        wn_dmode (SLOW);
        wn_dmode (FLASH);
        init_graph ();
        init_MENU_window ();
        main_menu ();
        start_filbus ();
        if(init_mms()==ERROR) exit (0);
        if(start_assoc()==ERROR)
        {
                scroll (w2,"Reboot the System",text_talker);
                exit (0);
        }
        list_chan_info (CHAN);
        stop_polling = 0;
        while (!stop_polling)
        {
                sleep (5000);
                scroll (w2,"Please, Send Read Request",text_talker);
                if(do_read("filbus_freq","c_filbus_simple_type",&filbus_freq
                                                                )== ERROR)
                {
                        scroll (w2,"MMS READ Error",text_talker);
                        stop_polling = 1;
                }
                else
                {
                        ultoa (filbus_freq, mess, 10);
                /* strcpy (mess, strcat ("Freq. Value: ", mess)); */
                        scroll (w2, mess, text_talker);
                        sleep (40);
                        actualize (filbus_freq);
                }

                /***  READ KEY FROM KEYBOARD   ***/
                AX = _bios_keybrd(_KEYBRD_READY);
                if  (AX  !=  0)          /*  Character present ?? */
                {
                        AX = _bios_keybrd(_KEYBRD_READ); /* Read char from FIFO */
                        car = (unsigned char)AX;
                        if (car == ESC) stop_polling = 1;
                }
        }

        scroll (w2,"Please, Send Conclude Request", text_talker);
        getch();
        if(end_assoc()==ERROR)
        {
                wn_close(w2);
                wn_close(w1);
                exit(0);
```

```
        }
        list_chan_info (CHAN);
        endmms();
        exit (0);
}       /* end main */

actualize (unsigned long timer)
{
        int rc;
        message[0] = 0;
        message[1] = 3;
        message[2] = 1;
        message[3] = 0x81;
        message[4] = 7;
        message[5] = 1;
        message[6] = 0x00;     /* offset para timer 2 00FE */
        message[7] = 0x5e;
        message[8] = 0;
        message[9] = (char) timer;     /* valor timer 2 */
        message[10] = 0;
        message[11] = (char) timer;     /* valor timer 3 */
        message[12] = "/0";

        if ((rc = send_msg (GF_handle, 0, message, 2)) != 0)
        {
                scroll (w2, "erro mensagem parame", text_talker);
        }
        else scroll (w2, "parametros enviados", text_talker);
}
```

Make File:

```
MAP_FIL.exe : MAP_FIL.obj mmsop_en.obj u_stubs.obj object\mmsvar.obj
        link @MAP_FIL.lnk /SE:500 /NOE $(LDFLAGS)

MAP_FIL.obj : SOURCES\MAP_FIL.c HEADERS\GT_DEF.H
        cl -c -AL -Os -Zp1 -G2 -DMAP30_ACSE -DPC sources\MAP_FIL.c

MMSOP_EN.obj : SOURCES\MMSOP_EN.c
        cl -c -AL -Os -Zp1 -G2 -DMAP30_ACSE -DPC sources\MMSOP_EN.c
U_STUBS.obj : SOURCES\U_STUBS.c
        cl -c -AL -Os -Zp1 -G2 -DMAP30_ACSE -DPC sources\U_STUBS.c
```

Link File:

```
MAP_FIL.obj+
\fcs86\filmon.obj+
\fcs86\fmsclib.obj+
mmsop_en.obj+
u_stubs.obj+
object\mmsvar.obj
MAP_FIL.exe
MAP_FIL.map
c:\USERS\TOVAR\WORK\BOSS_WIN\lmsc5.lib+
c:\mms\asn1.lib+
c:\mms\mmsacse.lib+
c:\mms\mms.lib+
c:\mms\suicacse.lib+
c:\map\ci.lib
```

8.2. Da Configuração do Servidor MMS

8.2.1. Instalação das Cartas de Comunicação

As declarações no ficheiro *config.sys* do MS-DOS foram as seguintes:

```
Device = C:\MAP\ICPDOSDR.EXE  0x180/P  5/I  0/C
Device = C:\FCS86\GFDRV.SYS  /R:300  /I:11  /H:3  /N:GF
```

No caso do MAP corresponde a um endereço 180H, *interrupt* 5 e canal de DMA 0. No FILBUS, H corresponde ao tipo de plataforma, e N é o nome do *device driver*. A configuração de *jumpers* tem de ser coerente com estas declarações.

8.2.2. Criação e Utilização da Base de Dados da Rede de Field Bus

O módulo de inicialização automática do servidor é responsável pela configuração do VMD, de acordo com a base de dados obtida a partir do *software* de configuração da rede FILBUS.

A aplicação CFG.EXE, já referida no capítulo 5, vai criar uma base de dados hierarquizada. No caso concreto, e não sendo relevante a definição de diferentes sistemas, poderá haver algum interesse em suportar diferentes configurações pré-programadas, constituindo assim diferentes redes, com diferentes módulos e cada módulo com diferentes utilizações.

Um dos blocos principais deste módulo, consiste então na identificação da rede a ser utilizada, pela leitura do único ficheiro *.SYS e identificar qual o ficheiro *.NET que foi passado como parâmetro.

O ficheiro *.NET contém todos os módulos que são usados na rede, respectivos tipos e endereços na rede. A cada nome de módulo da rede corresponde um ficheiro, do tipo *.MD, onde são indicados os nomes das E/S utilizadas. Cada E/S corresponde a um ficheiro *.IO.

A primeira parte deste módulo consiste na obtenção de uma estrutura de informação baseada nas seguintes unidades de dados:

```
typedef struct io_node
{
        UBYTE io_number;
        UBYTE io_parm[256];
        struct io_node *next_io;
};

typedef struct md_node
{
        UBYTE md_name[8];
        UBYTE md_addr;
        UBYTE md_type;
        struct io_node *ptr;
        struct md_node *next_md;
};
```

```
typedef struct net_node
{
        UBYTE net_file[8];
        UBYTE net_addr;
        struct md_node *md_ptr;
};
```

A partir da criação desta estrutura, o VMD terá de ser configurado, quer em termos de criação dos diferentes domínios (um por cada módulo), quer em termos de criação das variáveis e actualização respectiva.

Do anexo B consta a implementação, em linguagem C, do *software* que permite a interpretação da base de dados criada pelo utilitário CFG.EXE, bem como a criação da estrutura de dados que resulta da sua leitura.

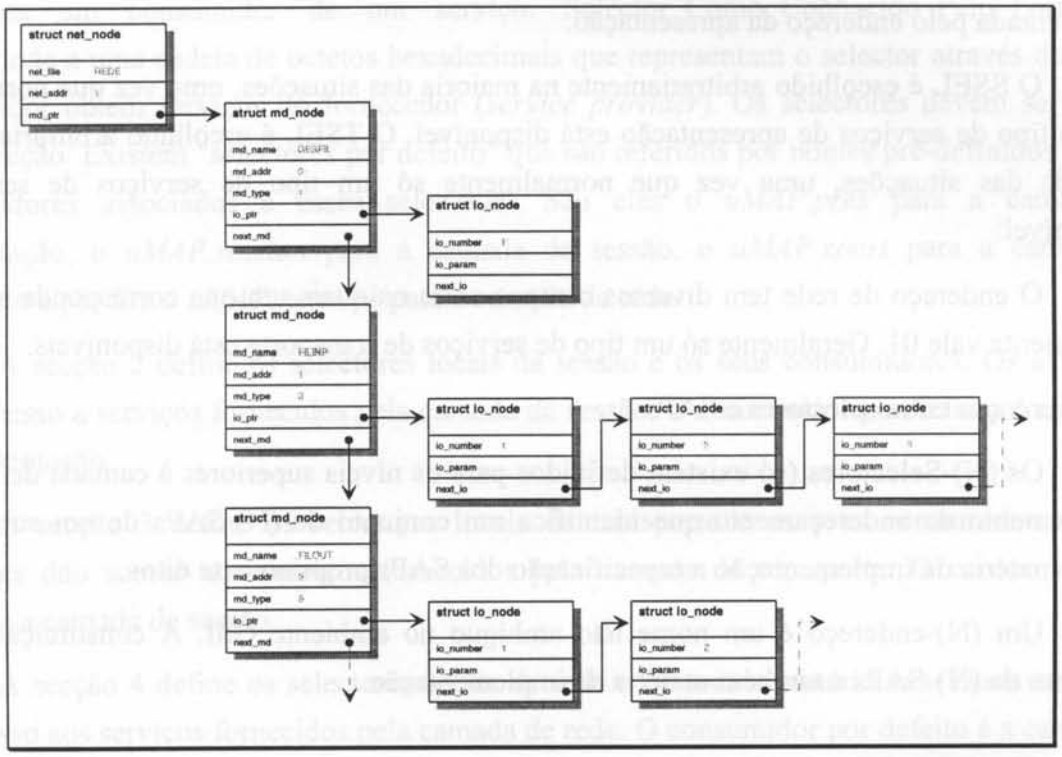


***Figura 8.3** - Estrutra de Informação a Utilizar para a Estruturação do VMD*

8.2.3. Informação de Directório da Stack OSI

Em teoria, a informação de directório OSI, que contém os mapeamentos entre os nomes das entidades e os endereços OSI, poderia ser obtida remotamente ou localmente. Esta versão OSI suporta só a obtenção local dessa informação, que deverá residir numa DIRFILE, mais propriamente num ficheiro designado por *umap_2.dir*.

```
typedef struct net_node
{
        UBYTE net_file[8];
        UBYTE net_addr;
        struct md_node *md_ptr;
};
```

A partir da criação desta estrutura, o VMD terá de ser configurado, quer em termos de criação dos diferentes domínios (um por cada módulo), quer em termos de criação das variáveis e actualização respectiva.

Do anexo B consta a implementação, em linguagem C, do *software* que permite a interpretação da base de dados criada pelo utilitário CFG.EXE, bem como a criação da estrutura de dados que resulta da sua leitura.

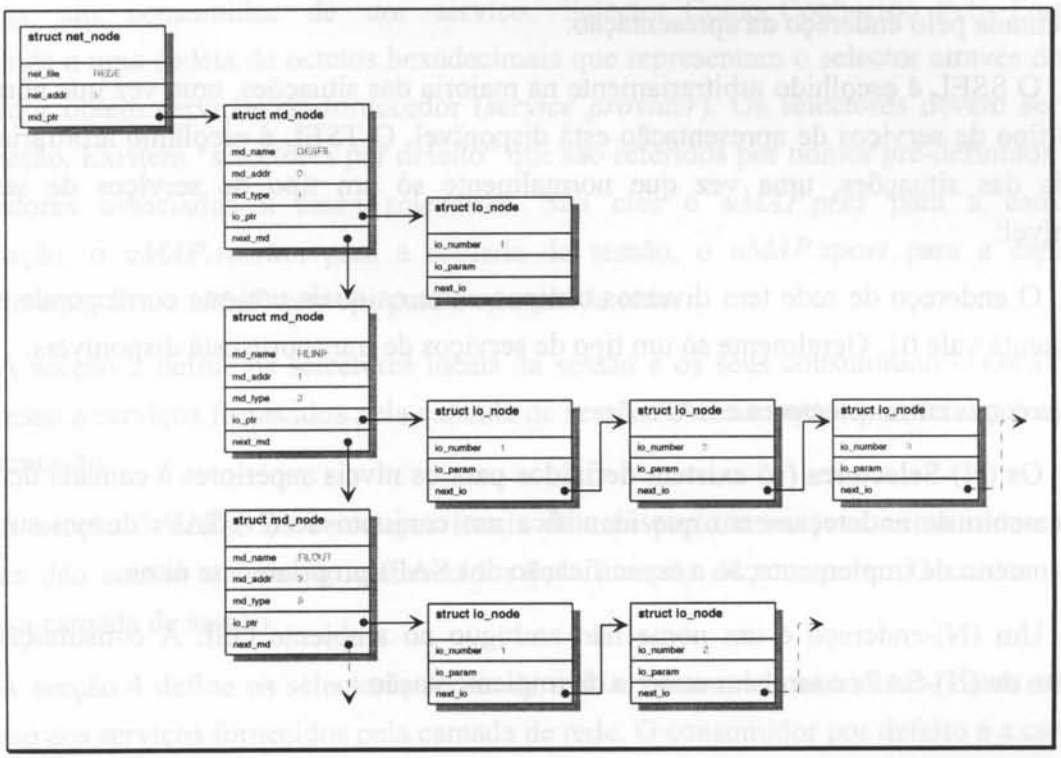


***Figura 8.3** - Estrutra de Informação a Utilizar para a Estruturação do VMD*

8.2.3. Informação de Directório da Stack OSI

Em teoria, a informação de directório OSI, que contém os mapeamentos entre os nomes das entidades e os endereços OSI, podería ser obtida remotamente ou localmente. Esta versão OSI suporta só a obtenção local dessa informação, que deverá residir numa DIRFILE, mais propriamente num ficheiro designado por *umap_2.dir*.

Este ficheiro contém a informação de directório residente no *host* que é carregado para o DUA (*Directory User Agent*) da carta, durante a inicialização do sistema. O DUA é executado como uma tarefa na carta de comunicações que suporta o *software* ISOcomm.

Um consumidor é uma aplicação ou camada OSI que utiliza os serviços de um *provider*. O consumidor de um *provider* está mais alto na *stack* OSI. As conexões ou associações entre consumidores e *providers* é feita através de selectores, que não são mais do que conjuntos de SAPs. Um selector identifica univocamente um conjunto de SAPs.

Os endereços OSI numa qualquer camada são constituídos pelos selectores dos níveis inferiores. No MAP 3.0, o endereço final, o endereço da apresentação, é constituido pelos selectores de apresentação, de sessão e de transporte, e pelo endereço de rede. Numa rede MAP 3.0 meramente local (não ligada a outras redes) basta a informação de endereçamento representada pelo endereço da apresentação.

O SSEL é escolhido arbitrariamente na maioria das situações, uma vez que normalmente só um tipo de serviços de apresentação está disponível. O TSEL é escolhido arbitrariamente na maioria das situações, uma vez que normalmente só um tipo de serviços de sessão está disponível.

O endereço de rede tem diversos campos entre os quais um que corresponde ao NSEL. Geralmente vale 01. Geralmente só um tipo de serviços de transporte está disponíveis.

1 - Diferenças entre selectores e SAPs

Os (N)-Selectores (só existem definidos para os níveis superiores à camada de rede) são um elemento de endereçamento que identifica um conjunto de (N)-SAPs de um sub-sistema, sendo matéria de implementação a especificação dos SAPs propriamente ditos.

Um (N)-endereço é um nome não ambiguo no ambiente OSI. A constituição real do conjunto de (N)-SAPs é também matéria da implementação.

2 - O ficheiro umap_2.dir

As DIRFILE obdecem a algumas regras sintácticas. São constituidas por seis secções que fornecem informação requerida pelo DUA. O símbolo '%' delimita o início de uma secção. O símbolo '#' corresponde a um comentário.

A secção 1 da DIRFILE contém a definição do nome do nó local e a directiva de mapeamento desse nó. Esta secção tem o seguinte formato:

```
Nome_do_Nó.\Directiva_de_Mapeamento
```

O Nome_do_Nó é geralmente uma cadeia de caracteres alfanumérica, usada para referênciar um nó particular. Por convenção, o nome de um nó é terminado por um ponto final '.'. Esse ponto final não tem qualquer significado, apenas permite a distinção entre um nome de um nó e o nome de uma aplicação. A Directiva_de_Mapeamento aparece logo a seguir ao Nome_do_Nó, precedida por um caracter *backslash*. Pelas razões já expostas anteriormente, nesta implementação OSI a única directiva suportada é a *local*. Esta directiva especifica que o ficheiro local DIRFILE é a única fonte de informação de directório OSI.

As secções 2 a 5, inclusivé, apresentam o seguinte formato:

```
Nome_do_Consumidor_do_Serviço\Selector_Como_Conhecido_Pelo_Fornecedor
```

O Nome_do_Consumidor_do_Serviço é uma cadeia de caracteres alfanumérica que representa um consumidor de um serviço. Selector_Como_Conhecido_Pelo_Fornecedor corresponde a uma cadeia de octetos hexadecimais que representam o selector através do qual o consumidor obtem serviços do fornecedor (*service provider*). Os selectores devem ser únicos numa secção. Existem "selectores por defeito" que são referidos por nomes pré-definidos para os consumidores associados a esses selectores. São eles o *uMAP.pres* para a camada de apresentação, o *uMAP.session* para a camada de sessão, o *uMAP.xport* para a camada de transporte e por fim o *uMAP.network* para a camada de rede.

A secção 2 define os selectores locais da sessão e os seus consumidores. Os selectores dão a acesso a serviços fornecidos pela camada de sessão. O consumidor por defeito é a camada de apresentação.

A secção 3 define os selectores locais de transporte e os seus consumidores. Esses selectores dão acesso aos serviços fornecidos pela camada de transporte. O consumidor por defeito é a camada de sessão.

A secção 4 define os selectores locais de rede e os seus consumidores. Esses selectores dão acesso aos serviços fornecidos pela camada de rede. O consumidor por defeito é a camada de transporte. O N-Selector corresponde neste caso à globalidade dos SAPs de rede (NSAP). O NSAP consiste em duas partes: o *Initial Domain Part* (IDP) e o *Domain Specific Part* (DSP).

O IDP corresponde a dois campos: o *Authority and Format Identifier* (AFI) e o *Initial Domain Identifier* (IDI). O AFI define o formato do IDI e a sintaxe do DSP, segundo o estipulado na norma ISO (IS 8348 addendum 2). Aparecem por isso NSAPs de estrutura diversa. Normalmente o AFI utilizado é o 39, correspondendo então à seguinte estrutura em octetos:

AFI	IDI	ORGID	AID	SUBID	STAID	NSEL
39	xxxx	xxxxxx	xxxxxxxx	xxxx	xxxx	xx

Nesta estrutura ORGID corresponde à identificação da organização, AID corresponde à identificação da área, SUBID designa uma sub-rede onde o nó está localizado e STAID designa a estação. O NSEL por defeito é 01, uma vez que a maioría dos nós só tem um conjunto de serviços da camada de transporte.

A secção 5 define os selectores da camada de ligação de dados e os seus consumidores. Esses selectores dão acesso aos serviços fornecidos pela camada de ligação de dados. O consumidor por defeito é a camada de rede. Tal como o N-Selector, o L-Selector corresponde à totalidade dos LSAPs. No caso desta implementação particular OSI, o selector de defeito deverá ser sempre FEh.

Por fim a secção 6 contém a informação de endereçamento para todas as aplicações locais e remotas. Se no formato das entradas desta secção for omitido o selector, é tomado o selector de defeito respectivo, tal como definidos nas secções 2 a 5. O formato das entradas desta secção é o seguinte:

```
Nome_da_Applicação\P-Sel\S-Sel\T-Sel\N-Sel\AE-Qualifier\AP-Title
```

O Nome_da_Aplicação é geralmente uma cadeia de caracteres alfanumérica, usada para referênciar uma aplicação. As aplicações locais não contém o nome do nó, ao contrário das aplicações remotas que deverão ser identificadas juntamente com o nome do nó a que estão associadas.

Uma aplicação remota deve ser definida como uma aplicação local num nó remoto e vice-versa. Uma aplicação local deverá ter apenas um selector especificado, os outros selectores corresponderão aos selectores definidos por defeito.

É importante notar que uma aplicação pode ser ligada directamente a um fornecedor de serviços especificando um selector desse fornecedor. Por exemplo, uma aplicação poderá ser ligada a um selector da camada de transporte:

```
aplicacao\\\0002\\\
```

Ao contrário, todos os selectores de uma aplicação remota deverão estar presentes e deverão corresponder exactamente à DIRFILE do nó remoto, onde essa aplicação está definida como aplicação local.

O *AE-Qualifier* (*Application Entity Qualifier*) e o *AP-Title* (*Application Process Title*) constituem no seu conjunto o *AE-Title* (*Application Entity Title*), tal como referido no capítulo 6. Ambas essas entradas são elementos de endereçamento optativos, não constando usualmente das DIRFILEs.

8.2.4. Ficheiro de Informação de Directório OSI do Dispositivo de Interligação

No capítulo 7 foi proposta uma arquitectura para o dispositivo de interligação, onde se previa apenas uma entidade de aplicação, e por isso um só endereço de apresentação, mas com potenciais conexões concorrentes múltiplas a/com entidades de aplicação remotas.

No que diz respeito ao ficheiro de informação de directório OSI local, na secção 1 é definido o nome do nó correspondente ao dispositivo de interligação, neste caso *map_filbus*. Este deverá ser o nome a utilizar num nó remoto para definir esta aplicação.

Os únicos selectores locais definidos são os selectores a utilizar por defeito. O NSAP utiliza a sintaxe AFI 39, sendo arbitrárias as especificações para IDI, ORGID, AID e SUBID, uma vez que não são relevantes tratando-se de uma rede meramente local. Essas especificações têm, obviamente, de ser iguais nos diferentes nós OSI dessa rede local. Obviamente são diferentes os identificadores de estação, isto é, STAID. O L-Selector por defeito especificado é o FEh, sendo esta uma restrição particular desta implementação da *stack* OSI, como já foi referido.

O nome escolhido para a aplicação local é *filbus_control*, e deverá ser este o nome referido como aplicação remota nos ficheiros DIRFILE de nós remotos. A especificação total desta aplicação, a ser utilizada nos DIRFILEs de nós remotos, é a seguinte:

```
map_filbus.filbus_control\00000001\0001\0001\39840f454e45000000010001000101
```

Apresenta-se a seguir a composição do ficheiro de informação de directório OSI para o nó correspondente ao dispositivo de interligação. De notar que são incluídas, meramente a título de exemplo, três aplicações remotas associadas a dois nós da rede local. São eles o nó *node2* e o nó *node3*. São definidas duas aplicações remotas no *node2*, *cliente_a* e *cliente_b*, correspondendo obviamente a endereços de apresentação diferentes.

```
% Section 1:  NODE NAME and MAPPING DIRECTIVE
map_filbus.\local
% Section 2:  LOCAL S-Selector Consumers
uMAP.pres\0001
% Section 3:  LOCAL T-Selector Consumers
uMAP.session\0001
% Section 4:  LOCAL N-Selector Consumers
uMAP.xport\39840f454e45000000010001000101
% Section 5:  LOCAL L-Selector Consumers
uMAP.network\fe
% Section 6:  LOCAL & REMOTE Presentation Address and Application Entity Titles
# local titles
filbus_control\00000001\\\\\
# remote titles
node2.cliente_a\00000001\0001\0001\39840f454e45000000010001000201
node2.cliente_b\00000002\0001\0001\39840f454e45000000010001000201
node3.cliente_c\00000001\0001\0001\39840f454e45000000010001000301
% END of FILE
```

8.2.5. Registo da Aplicação Local Utilizando a Interface MMS-EASE

O MME-EASE, por conveniências de implementação, permite a utilização de *Application Reference Names* (*AR Names*) em representação dos *AE-Titles*.

Como já foi referido anteriormente, cada canal de comunicação OSI representa uma potencial associação com uma outra aplicação. Cada associação de aplicação activa utiliza um canal de comunicações.

Cada aplicação que utilize o MMS-EASE deverá registar o seu *AR Name* num canal, antes de poder iniciar ou aceitar associações nesse canal. A cada canal é atribuido um número entre zero e o número máximo de canais suportado menos um. Se alguns canais não estão atribuidos a nenhuma aplicação local, então esses canais não estão disponíveis, mesmo que a carta de comunicações seja capaz de manter mais associações de aplicação simultâneas.

Um *AR Name* pode ser registado num canal se e só se ela consta da informação de directório como sendo um *AR Name* local. Caso contrário, a carta de comunicações MAP não poderá resolver os endereços de apresentação associados às PDUs que chegam (*Incoming PDUs*).

Existem várias funções da interface MMS-EASE associadas a esta matéria. De entre elas, pela sua importância, podem destacar-se, e seguindo a classificação já feita no início deste capítulo, cinco da classe de funções 9 (*LLP Request Functions*) e uma da classe de funções 11 (*Support Functions*).

São elas, respectivamente:

```
mllp_act_ar_name (ar_name, llp_type)
mllp_reg_ar_name (chan, ar_name)
mllp_ass_listen (chan)
mllp_ass_stop_ass_listen (chan)
mllp_deact_ar_name (ar_name, llp_type)

ms_create_vmd (num_chan, channels)
```

A função *mllp_act_ar_name (,)* serve para activar os *AR Names* locais definidos no ficheiro *umap_2.dir* e atribuir-lhes um fornecedor de serviços, obviamente de uma camada inferior (*lower_layer_provider*). No caso do MMS-EASE, existem dois tipos de fornecedor possíveis para uma aplicação: o MAP versão 3.0 ACSE (*Application Context Service Element*) e o MAP versão 3.0 LLC ou seja uma interface directa à camada de ligação de dados. Esta última corresponde à arquitectura de três camadas do MiniMAP. No primeiro caso, correspondente ao caso presente, a função é chamada da seguinte forma: *mllp_act_ar_name (filbus_control , ACSE30_LLP)*. A função retorna '0' se não houve erro de activação.

A função *mllp_reg_ar_name (,)* permite registar um *AR Name* local num determinado canal. Um pedido de MMS *Initiate* feito a partir de um cliente, só será recebido quando o canal for posto em modo de *listen.*

A função *mllp_ass_listen ( )* é utilizada quando se pretende receber pedidos de associação de aplicação (serviço MMS *Initiate*) através desse canal. Se esta função não for chamada para um determinado canal, através desse canal só poderão ser enviados pedidos de MMS *Initiate* e não podem chegar indicações de MMS *Initiate*. No caso do dispositivo de interligação, terá de ser chamada esta função para todos os canais activos.

A função *mllp_stop_ass_listen (,)* é utilizada quando se pretende inibir o modo *listen* de um canal. Se por exemplo se pretendesse estabelecer uma associação como cliente, utilizando um canal que estava em modo *listen*, era preciso chamar a função *mllp_stop_ass_listen (,)* para esse canal, e só depois enviar um pedido de MMS *Initiate*.

A função *mllp_deact_ar_name ( )* é usada para desactivar um *AR Name* local activado anteriormente. Qualquer canal registado que use este *AR Name* deverá ficar inactivo e não poderá receber ou enviar um pedido de associação. Se o canal estava em modo de *listen* (caso do servidor MMS), deverá primeiro chamar-se a função *mllp_stop_ass_listen ( )*. Se nalgum canal que utilize esse *AR Name* existe uma associação activa, primeiro deverá proceder-se à conclusão dessa associação.

A função *ms_create_vmd (,)* permite criar um VMD no servidor, e atribui ao VMD diferentes canais. A arquitectura proposta para o servidor de MMS, que constitui o dispositivo de interligação, é composta por apenas um VMD, pelo que, na realidade, não é preciso utilizar essa função para criar no servidor esse VMD.

Existe uma função da interface MMS-EASE da classe 13 (*Interface Functions*) que é responsável por inicializar o sistema MMS-EASE e que precisa de ser chamada antes de qualquer outra função do MMS-EASE o seja. Essa função é a *strt_MMS( )*. Esta função, que inicializa o MMS, cria por defeito um VMD. Quando se registam os diferentes *AR Names* nos diferentes canais, todos eles ficam associados a esse VMD. Essa situação é alterada ao criar outros VMDs.

Relativamente à utilização da função *ms_create_vmd (,)*, é de referir que o parâmetro *num_chan* corresponde ao número de elementos do vector *channels*, que é, por sua vez, o segundo parâmetro dessa função. Cada um dos elementos do vector *channels* corresponde a um canal a atribuir ao VMD. Vamos supor um VMD qualquer ao qual quereríamos atribuir os canais 3, 5 e 6. Em *Microsoft* C sería:

```
...
char canais[8];
canais[0] = 3;
```

```
canais[1] = 5;
canais[2] = 6;
ms_create_vmd (3, canais);
...
```

Vamos supor a arquitectura discutida como primeira hipótese no capítulo 7, arquitectura essa constituída por um VMD por cada módulo da rede de *Field Bus*, com um endereço de apresentação e um SAP por VMD. Este sería o extracto de um potencial ficheiro de directório OSI, para a definição das entidades locais:

```
...
# local titles
mestre_filbus\00000001\\\\\
filinp_1648\00000002\\\\\
filout_1648\00000003\\\\\
...
```

O conjunto de instruções em *Microsoft* C correspondería ao seguinte:

```
...
strt_MMS();

mllp_act_ar_name ("mestre_filbus", ACSE30_LLP);
mllp_act_ar_name ("filinp_1648", ACSE30_LLP);
mllp_act_ar_name ("filout_1648", ACSE30_LLP);

mllp_reg_ar_name (0, "mestre_filbus");
mllp_reg_ar_name (1, "filinp_1648");
mllp_reg_ar_name (2, "filout_1648");

canais[0] = 1;
ms_create_vmd (1, canais);

canais[0] = 2;
ms_create_vmd (1, canais);
...
```

No caso do dispositivo de interligação proposto, e admitindo um limite de 8 canais disponíveis, o conjunto de instruções a utilizar deverá ser o seguinte:

```
...
strt_MMS();

mllp_act_ar_name ("filbus_control", ACSE30_LLP);

for (i = 0; i < 8; i++)
{
        rc = mllp_reg_ar_name (i, "mestre_filbus");
        rc = mllp_ass_listen (i);
}
...
```

8.2.6. Configuração das Estruturas de Dados de Sistema do MMS-EASE

1 - Selecção do Contexto de Apresentação

Uma das estruturas a usar é a estrutura que define o contexto de apresentação a ser utilizado quando se envia um pedido MMS através de um canal. O contexto de apresentação determinará quais as normas companheiras do MMS, se alguma, se vão utilizar. Quando se enviar um MMS *Initiate.Req* através de um canal y, pretendendo-se assim estabelecer uma associação de aplicação, *P_context_sel[y]* deverá ter a máscara que indicará quais os contextos de apresentação suportados para essa associação. No entanto, os outros pedidos MMS enviados através desse canal, só deverão dizer respeito a um contexto (um *bit* a '1'). Só o contexto MMS é suportado por esta versão do MMS-EASE, que corresponde ao *bit* mais significativo dos 16 possíveis para cada elemento do vector. A estrutura é especificada como se segue:

```
USHORT P_context_sel[max_mms_chan]
```

O dispositivo de interligação será um servidor MMS que só receberá indicações MMS *Initiate*, através dos diferentes canais. Esta estrutura não é por isso muito relevante para um servidor, apesar dele poder enviar pedidos MMS, no entanto não confirmados. No caso presente, esses pedidos são, como se viu no capítulo 7, o MMS *InformationReport* e o MMS *UnsolicitedStatus*.

2 - Variáveis de Configuração da Placa de Comunicações

Estas variáveis permitem ao utilizador alterar certas variáveis relacionadas com o ACSE. A alteração, se necessária, deve ser feita antes de chamar a função *strt_MMS ( )*.

```
SHORT s_max_chan
```

Contém o número de canais que o SUIC deverá alocar. Este valor deverá ser inferior ou igual a MAX_SUIC_CHAN, que é o valor de defeito (definido em suicacse.h) e deverá ser superior ou igual a *max_mms_chan*.

```
SHORT s_msgsize
```

Diz respeito ao número máximo de *bytes* que pode ser enviado numa só transferência do ACSE.

O ficheiro suicacse.h contém a seguinte informação:

```
/*****************************************************************************/
/* Select the board type here                                                */
/* These defines are used in the sample source code to make use of           */
/* board specific operations, such as status or debug displays.              */
```

```
/** #define CONCORD **/
/** #define MP_400 **/

#define MVME_372

/* **AO** 06-22-89 */
#define COMPUTROL

                        /* Now select the board specific include file    */
#ifdef CONCORD
#define MAX_SUIC_CHAN 16    /* number of channels                        */
#include "cci12_a.h"        /* CCI 1200                                  */
#endif

#ifdef MP_400
#define MAX_SUIC_CHAN 32    /* number of channels                        */
#include "mp400_a.h"        /* MP 400                                    */
#endif

/* **AO** 06-22-89 */
#ifdef COMPUTROL
#define MAX_SUIC_CHAN 64
#include "cmptrl_a.h"                       /* *ASM*  7-11-89  */
#else
#ifdef MVME_372
#define MAX_SUIC_CHAN 64
#include "moto_a.h"
#endif
#endif
```

O ficheiro cmptrl_a.h corresponde ao seguinte:

```
/*@(#)cmptrl_a.h      4.1           11/22/88*/
/*************************************************************************/
/* SISCO SOFTWARE MODULE HEADER ******************************************/
/*************************************************************************/
/*   (c) Copyright Systems Integration Specialists Company, Inc.,        */
/*       1988 All Rights Reserved                                        */
/*                                                                       */
/* MODULE NAME : cmptrl_a.h                                              */
/* PRODUCT(S)  : ASCE-LLP for Motorola                                   */
/*                                                                       */
/* MODULE DESCRIPTION :                                                  */
/*     This module contains the computrol specific definitions           */
/*     required for the ASCE-LLP                                         */
/*                                                                       */
/* GLOBAL FUNCTIONS DEFINED IN THIS MODULE :                             */
/*                                                                       */
/* MODIFICATION LOG :                                                    */
/*  Date     Who   Rev      Comments                                     */
/* --------  ---  ------   -------------------------------------------  */
/* 06/22/89  AO    1.0       Duplicated moto_a.h to computrol_a.h to     */
/*                           reflect environment, i.e. using QMAC board.*/
/* 07/11/89  ASM   1.1       Duplicated computrol_a.h to cmptrl_a.h to   */
/*                           comply with the MS-DOS filename convention  */
/*                                                                       */
/*************************************************************************/
extern DEFAULT s_num_cpu;              /*maximum number of cpus         */
                                       /*range is from 0-5 not 2-7      */
extern DEFAULT s_base_cpu_num;                /*starting CPU number     */

extern DEFAULT s_aepb_buf_size;               /* maximum ACSE message size  */
extern DEFAULT s_pepb_buf_size;               /* maximum Presentation       */
extern DEFAULT s_init_num_ctrl_buf;

extern FILE *acse_log_dump ;
extern DEFAULT s_dereg_cpu_num;
```

```
extern DEFAULT s_num_entries_on_queue;
extern DEFAULT s_max_num_entries_on_queue;

#ifdef LINT_ARGS
extern DEFAULT s_moto_chan_vs_cpu_set(SHORT, SHORT, DEFAULT);
#else
extern DEFAULT s_moto_chan_vs_cpu_set();
#endif

#ifdef DEBUG
extern SHORT debug_dump_en;
extern FILE* debug_file;
#endif

/*****************************************************************      */
/*                    additional exception codes                        */

#define SX_CONFIG_ERR 40
#define NUM_ACSE_CHAN_MAX 64
```

3 - Variáveis de Configuração do Ambiente MMS

Todas as variáveis deverão ser alteradas antes de ser chamada a função *strt_MMS ( )*. Quanto maiores forem os valores de cada uma das variáveis mais memória o MMS-EASE aloca.

```
SHORT max_mms_chan
```

Número máximo de canais a ser alocados para o MMS-EASE. O valor por defeito é 8. Pode ser alterado dependendo da carta de rede e da memória disponível. O valor por defeito para *max_mms_chan* é adequado ao caso presente, pelo que não necessita de ser alterado.

```
SHORT max_req_pend
```

Esta variável representa o número máximo de pedidos MMS pendentes suportáveis simultaneamente no total de canais. A soma de *mms_chan_info[chan].maxpend_req* (existe uma estrutura *mms_chan_info* associada a cada canal) para todos os canais activos deverá ser inferior a *max_req_pend*. Esta variável é utilizada para controlar a alocação de estruturas de fila MMSREQ_PEND. O valor que existe por defeito é 20. Esta variável não é relevante para a implementação em causa, uma vez que se trata de um servidor de MMS. O seu valor pode ser inclusivamente '0', uma vez que os únicos pedidos enviados são de serviços MMS não confirmados, pelo que não ficam pendentes.

```
SHORT max_ind_pend
```

A variável *max_ind_pend* representa o número máximo de indicações MMS pendentes suportáveis simultaneamente no total de canais. A soma de *mms_chan_info[chan].maxpend_ind* para todos os canais activos deverá ser infereior a *max_ind_pend*. Esta variável é utilizada para controlar a alocação de estruturas de fila MMSREQ_IND. Ao contrário da anterior, esta variável

é obviamente relevante para a implementação em causa. O valor que existe por defeito é 20, que pode não ser considerado suficiente para as solicitações previstas no capítulo 7. O valor desta variável será alterado para 36 uma vez que sendo suportados 8 canais se está a admitir um situação limite, mas pouco plausível de 4 indicações pendentes por canal (valor médio).

As duas variáveis seguintes dizem respeito a números máximos de ficheiros que, respectivamente, podem ser abertos por nós remotos no sistema local e que podem ser abertos pelo sistema local em nós remotos.

```
SHORT max_loc_open
SHORT max_rem_open
```

A segunda hipótese de funcionalidade (está-se a falar da utilização de serviços MMS de acesso a ficheiros) não é possível para um servidor de MMS. A abertura no sistema local pode ser utilizada, por exemplo no que diz respeito aos ficheiros de configuração da base de dados do *field bus*. A avaliação dessa hipótese não é de todo relevante para o estudo aqui efectuado.

```
SHORT max_mmsease_types
```

A variável *max_mmsease_types* corresponde ao numero máximo de tipos MMS nomeados que a máquina virtual deixará definir em todo o ambiente MMS-EASE. Limita assim também o número de tipos que podem ser definidos localmente por um nó remoto, evitando problemas de memória. O valor de defeito desta variável é 50. A versão do MMS-EASE considerada não suporta mesmo mais do que este número máximo. Para o dispositivo de interligação proposto no capítulo anterior foram identificados 7 tipos nomeados, uma vez que as variáveis MMS nomeadas *IOGLOI_MD* e *IOGLOO_MD* são do mesmo tipo. Por isso o valor desta variável pode ser alterado para 10.

```
SHORT max_mmsease_vars
```

A variável max_mmsease_vars diz respeito ao número máximo de variáveis MMS nomeadas que a máquina virtual deixará definir em todo o ambiente MMS-EASE. Limita assim o número de variáveis que podem ser definidas localmente por um nó remoto, evitando problemas de memória. O valor de defeito é também 50, e também corresponde ao limite máximo suportado por esta versão do MMS-EASE.

Esta é de facto uma limitação particular desta interface de MMS, mas que no entanto compromete de alguma forma a proposta de arquitectura feita no capítulo anterior. Assim, cada módulo de 16 E/S tem 17 variáveis MMS correspondentes aos parâmetros das E/S (variáveis *PARIOS_xx*), 16 variáveis MMS individuais correspondentes cada uma delas a cada E/S (variáveis *IOINDI_xx* ou *IOINDO_xx*), uma variável global correspondente ao conjunto de todas as E/S digitais do módulo (variáveis *IOGLOI_MD* ou *IOGLOO_MD*) e ainda três variáveis de

uso diverso (*ALARMS_MD*, *COMAND_MD* e *SUBSCR_MD*). Ora somando estas variáveis verifica-se que conduz pelo menos a 37 variáveis MMS, isto só para um módulo de E/S digitais.

Qualquer que fosse a opção tomada na manipulação da informação do servidor de MMS, isto é, manter o VMD coerente com o estado dos dispositivos reais que mapeia ou a opção de possibilitar mapeamentos verticais de comunicações em modo assíncrono, implica a existência dos objectos da classe MMS variável nomeada.

Poderá efectuar-se uma redução drástica se se optar, como se aventou, por utilizar para modelizar as E/S dos módulos só variáveis globais, obviamente à excepção das variáveis do tipo contador dos módulos de entradas. Isto traría algumas implicações como sejam escritas nas E/S de saída com confirmação remota extremamente "pesadas". A solução sería não ter a confirmação remota em mapeamento assíncrono do serviço MMS *Write*.

As 17 variáveis MMS correspondentes aos parâmetros de um módulo poderíam ser aglutinadas numa só, numa estrutura ou num *array* das consideradas individualmente. Só que então teria de ser suportado o modo de acesso a variáveis MMS do tipo *Alternate Access* (parâmetro de conformidade VALT). Pelas razões apresentadas no capítulo anterior desta dissertação, esta opção não será a mais indicada.

Esta limitação não pode ser considerada de muito relevo, em termos de arquitectura proposta, uma vez que se trata de uma limitação específica desta versão de MMS-EASE. É no entanto um aspecto crítico da arquitectura proposta a quantidade de variáveis MMS consideradas.

```
SHORT max_mmsease_doms
```

A variável *max_mmsease_doms* especifica o número máximo de dominios MMS nomeados que a máquina virtual deixará definir em todo o ambiente MMS-EASE. Limita assim também o número de domínios que podem ser definidos localmente por um nó remoto, evitando problemas de memória. Não existe informação sobre o limite para esta variável. De qualquer maneira será uma limitação específica desta implementação.

```
SHORT max_mods
SHORT mms_max_msgsize
```

A variável *max_mods* especifica o número de modificadores presente na estrutura *modifier_list*. Se o número for superior ao desta variável é gerado um MMS *Reject* do pedido. Esta variável não é relevante para a aplicação, uma vez que não são usados modificadores MMS na arquitectura proposta para o dispositivo de interligação.

Por fim, a variável *mms_max_msgsize* especifica o tamanho máximo da mensagem. Valor de defeito 1000. Este valor pode ser alterado, mas terá sempre de ser inferior a *s_msgsize*.

Para o dispositivo de interligação, as variáveis de sistema do MMS-EASE terão os seguintes valores:

```
s_max_chan = 8;
s_msgsize = 1020;
max_mms_chan = 8;
max_req_pend = 8;
max_ind_pend = 36;
max_loc_open = 0;
max_rem_open = 0;
max_mmsease_types = 12;
max_mmsease_vars = 50;
max_mmsease_doms = 20;
max_mods = 0;
mms_max_msgsize = 1020;
```

8.2.7. Configuração das Estruturas de Dados da Máquina Virtual do MMS-EASE

1 - Variáveis de Gestão do Contexto

```
mms_chan_info[ ]
```

A variável *mms_chan_info[ ]* consiste num vector de estruturas com o contexto negociado para cada canal activo bem como os parâmetros de preferência a utilizar no serviço MMS *Initiate* (*Preferred Initiate Parameters*) para canais livres.

Cada elemento desse vector corresponde à seguinte estrutura:

```
struct mchaninfo
    {
    struct
            {
            SHORT llp_type;             /* type of Lower Layer Provider     */
            ULONG chan_state;           /* current state                    */
            SHORT xmit_pend;            /* pending transmits                */
            USHORT contexts;            /* Presentation Contexts            */
            BYTE locl_ar[MAX_AR_LEN+1]; /* local Application Reference      */
            BYTE rem_ar[MAX_AR_LEN+1];  /* local Application Reference      */
            } ctxt;
    struct
            {
            struct  vmd_ctrl        *vmd;
            struct  domain_objs     *aa_objs;
            } objs;
    SHORT   version;                    /* version to use                   */
    LONG    segsize;                    /* max message size                 */
    USHORT  maxpend_req;                /* max requests outstanding         */
    USHORT  maxpend_ind;                /* max indications outstanding      */
    UBYTE   max_nest;                   /* max data nesting                 */
    UBYTE   param_supp[2];              /* parameter support                */
    UBYTE   service_req[11];            /* services supported as requester  */
    UBYTE   service_resp[11];           /* services supported as responder  */
    BOOLEAN cs_detail_pres;             /* CS detail active                 */
    SHORT   cs_detail_len;
    UBYTE    *cs_detail;
```

```
    USHORT file_blk_size;                    /* file block size                */
    USHORT download_blk_size;                /* download block size            */
    BYTE    *user_info;                      /* reserved for user              */
    };
                              /* ptr to table of chan ctrl structs  */
                              /* [max_mms_chan], allocated at init  */
struct mchaninfo *mms_chan_info;
```

2 - Os Parâmetros Preferências de MMS Initiate

Os *Preferred Initiate Parameters* são variáveis que conterão os valores de defeito, que serão colocados na estrutura *mms_chan_info*, quando a função *strt_MMS ( )* for chamada. Estes valores deverão ser mudados, se necessário, antes da chamada à função *strt_MMS ( )*. No entanto, se necessário, os valores da estrutura *mms_chan_info* deverão ser alterados antes de efectuar um pedido MMS *Initiate*, utilizando para tal a função *mv_init (chan, partner)* (função MMS-EASE da classe 5 - *Virtual Request*) ou para responder a um pedido de MMS *Initiate*, utilizando para tal a função *mv_init_resp (req_ptr)* (função MMS-EASE da classe 6 - *Virtual Response*). Os valores de defeito e os parâmetros em causa são os seguintes:

```
SHORT m_version              = 0
LONG m_segsize               = 1020
USHORT m_maxpend_req         = 5
USHORT m_maxpend_ind         = 5
UBYTE m_max_nest             = 4
UBYTE m_param                = 0xE8
UBYTE m_service_resp[11]     = o que estiver definido no ficheiro MMSOP_EN.H
USHORT m_file_blk_size       = 960
```

Os valores a estabelecer para os *Preferred Initiate Parameters* serão utilizados para negociar com potenciais clientes MMS.

Para o dispositivo de interligação, os *Preferred Initiate Parameters* deverão ter os valores acima apresentados, à excepção de *m_param* que deverá ser 0xE0, uma vez que o servidor não deverá suportar o parametro de conformidade VADR, que diz respeito a acesso a objectos não nomeados.

No caso dos serviços suportados, (à semelhança do parâmetro também se trata de um conjunto de *bitstring*) fica:

UBYTE m_service_resp[11] = {0xEC, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x81}

m_service_resp[0] corresponde à utilização dos serviços MMS *Status*, *GetNameList*, *Identify*, *Read* e *Write*, *m_service_resp[9]* corresponde à utilização dos serviços MMS *UnsolicitedStatus* e *InformationReport*, *m_service_resp[10]* corresponde à utilização dos serviços MMS *Conclude* e *Cancel*.

Existem muitas mais estruturas de dados da máquina virtual do MMS-EASE relacionadas, por exemplo, com a gestão de tipos MMS, das variáveis MMS, da gestão de domínios, de VMD, etc. Essas estruturas serão abordadas separadamente nas subsecções seguintes.

8.2.8. Geração dos Objectos Abstractos MMS no Servidor

Existem funções que permitem a criação local dos objectos MMS das classes domínio nomeado, tipo nomeado, variável nomeada, etc. Essas funções implicam geralmente a alocação de estruturas de controlo para cada um dos objectos, estruturas essas inseridas num conjunto interdependente mais vasto de outras estruturas de controlo de outros objectos MMS. Na interface MMS-EASE, essas funções estão incluídas na classe de funções 11, ou seja *Support Functions*.

1 - Geração de Objectos MMS da Classe Domínio Nomeado

De entre as funções do MMS-EASE da classe 11, destacam-se as funções *ms_add_named_dom (dom_info, pro)* e *ms_del_named_dom (dname)*.

Na função de criação de um domínio é passado um apontador (*dom_info*) para uma estrutura do tipo *struct initdown_req_info*, o segundo parâmetro (*pro*) corresponde ao valor que é colocado no membro *protection* da estrutura *struct named_dom_ctrl* que passa a ser a estrutura de controlo do domínio criado e que retorna da chamada da função referida.

Esta estrutura de controlo de domínio é inserida numa lista alfabetizada de domínios de um determinado VMD. O primeiro elemento da lista é apontado pelo elemento *dom_list* da estrutura do tipo *struct vmd_ctrl* respeitante ao VMD que contém esses domínios.

No caso do dispositivo de interligação, só existirá um VMD. O MMS-EASE tem uma variável global, *vmd_ctrl_list*, que é um apontador para uma estrutura de controlo de VMD, ou seja, do tipo *struct vmd_ctrl*. Como também já foi referido, ao arrancar a máquina MMS-EASE, é criado por defeito um VMD, e por isso é alocada uma estrutura do tipo *struct vmd_ctrl*. A variável global apontará para essa estrutura. A composição destas estruturas relacionadas com o VMD é a seguinte:

```
struct vmd_ctrl
{
        struct dbl_lnk link;
        struct domain_objs vmd_wide;          /* VMD wide scope objects          */
        struct named_dom_ctrl *dom_list;      /* named domains in this VMD       */
        struct prog_inv *prog_inv_list;       /* program invocations in this VMD */
        struct oper_stat *op_stat_list;       /* operater stations on this VMD   */
        struct ae_ctrl *ae_list;              /* for MMSI only                    */
        BYTE *user_info;                      /* reserved for user          */
};
```

```
extern struct vmd_ctrl *vmd_ctrl_list;        /* list of VMD's          */
```

No MMS-EASE existem quatro estruturas de dados relacionadas com a gestão de domínios. São elas as estruturas do tipo *struct domain_objs*, do tipo *struct named_dom_ctrl*, e duas variáveis, a *max_mmsease_doms*, já referida, e a *mms_dom_count*, que contém o número de domínios criados.

A estrutura *struct domain_objs* contém informação sobre todos os objectos MMS definidos no âmbito de um determinado domínio. De notar que a estrutura de controlo de um VMD, *struct vmd_ctrl*, tem um campo, *vmd_wide*, que é do tipo *struct domain_objs*. Neste caso não conterá informação sobre os objectos MMS específicos de um determinado domínio mas sim específicos de um VMD. Para cada domínio, a estrutura de controlo com ele relacionada, *struct named_dom_ctrl*, contém um campo, *objs*, esse sim do tipo *struct domain_objs* e que diz respeito aos objectos específicos desse domínio.

```
struct domain_objs
{
        struct named_var        *var_list;          /* named variable list */
        struct named_var_list   *var_list_list;     /* variable list list  */
        struct named_type       *type_list;         /* named type list     */
        struct scat_acc         *scat_acc_list;     /* scattered access list*/
        struct sem              *sem_list;          /* semaphore list      */
        struct event_cond       *ev_cond_list;      /* event condition list */
        struct event_action     *ev_act_list;       /* event action list   */
        struct journal          *jour_list;         /* journal list        */
        BYTE rsrvd[8];                              /* MMSEASE reserved    */
};


struct named_dom_ctrl
{
        struct dbl_lnk link;                    /* generic list links  */
        BYTE dom_name[MAX_IDENT_LEN+1];         /* name of domain      */
        BYTE protection;                        /* protection attribute */
        BOOLEAN deletable;                      /* MMS delete attribute */
        BOOLEAN sharable;                       /* shareable attribute */
        BYTE state;                             /* domain state attrib. */
        BOOLEAN upl_in_prog;                    /* upload in progress  */
        LONG octet_pos;                         /* position of download */
        SHORT npi;                              /* # of prog inv's containing this dom  */
        BYTE reserved[4];                       /* MMS-EASE reserved   */
        BYTE user_rsrvd[8];                     /* user reserved       */
        BYTE *dom_content;                      /* some ptr to dom cntnt*/
        struct domain_objs objs;                /* dom-spec object ctrl */
        BOOLEAN dl_detail_pres;                 /* download detail present    */
        SHORT dl_detail_len;                    /* download detail ptr & len  */
        UBYTE *dl_detail;                       /* ptr to additional detail   */
        SHORT   num_of_capab;                   /* number of capabilities     */
        /*                                         list of capabilities       */
        /*  BYTE *capab_list [num_of_capab];                                  */
};

extern SHORT   max_mmsease_doms;        /* max number allowed      */

extern SHORT   mms_dom_count;           /* number currently defined        */
```

Como foi dito, o primeiro parâmetro da função *ms_add_named_dom (dom_info, pro)* é do tipo *struct initdown_req_info*. Essa estrutura tem a seguinte composição:

```
struct initdown_req_info
{
        BYTE    dname [MAX_IDENT_LEN +1];       /* domain name                 */
        BOOLEAN sharable;                       /* boolean, no default         */
        BOOLEAN dl_detail_pres;                 /* download detail present     */
        SHORT dl_detail_len;                    /* download detail ptr & len   */
        UBYTE *dl_detail;
        SHORT   num_of_capab;                   /* number of capabilities      */
        /*                                        list of capabilities         */
        /*  BYTE * capab_list [num_of_capab];                                  */
};
```

O campo boleano *sharable*, significa que o domínio poderá ser usado por mais de um invocação de programa MMS. No caso do dispositivo de interligação foi abordada a possibilidade de considerar domínios específicos para a implementação de funções de controlo. Na implementação aqui abordada essa eventualidade não será analisada, pelo que este campo não é relevante.

O nome do domínio será constituído a partir do nome de cada módulo tal como definídos na base de dados da rede de *Field Bus*.

O parâmetro *pro* da chamada à função *ms_add_named_dom (dom_info, pro)* será colocado na estrutura *struct named_dom_ctrl* respeitante a esse domínio. Este código de acesso podería ser comparado com o previlégio de uma determinada entidade de comunicações remota, por forma a que o acesso ao domínio pudesse ser controlado. Esta versão do MMS-EASE não suporta este tipo de funcionalidades.

Assim, reportando à estrutura de informação obtida a partir da base de dados da rede de *Field Bus*, para definir os domínios, será percorrida a lista de *struct md_node* apontada por *md_ptr* da estrutura *struct net_node*. Primeiro terá de ser criada uma estrutura do tipo *struct initdown_req_info*. Para cada domínio a criar, o único campo da estrutura *struct initdown_req_info* a alterar será o correspondente ao nome do domínio. Por outro lado, e reportando ao capítulo 7, deverá ser alocada e criada uma estrutura da aplicação do tipo *struct module_control* com informação relacionada com o módulo. Exemplificação:

```
struct initdown_req_info *dom_info ;
BYTE   pro ;

struct fil_node_tree_type *ap;

dom_info->sharable = 0;
dom_info->dl_detail_pres = 0;
dom_info->num_of_capab = 0;
pro = 0;

ap = head_database.md_ptr;
```

```
dom_info = (struct initdown_req_info *) malloc (sizeof (struct initdown_req_info));

while (ap != (fil_node_tree_type *)NULL)
{
        strcpy (dom_info->dname, ap->md_name);
        if (ms_add_named_domain (dom_info, pro) == 0)
                {
        ...
                }
        add_module_ctrl(ap);
        ...
}
...
free (dom_info) ;
```

2 - Geração de Objectos MMS de Acesso a Variáveis

Existe um conjunto de funções do MMS-EASE, também da classe de funções de suporte, que permitem a criação e manipulação local dos objectos MMS de acesso a variáveis.

Em relação aos objectos MMS de acesso a variáveis, assumem particular relevo os objectos MMS da classe *Named Type*. Com o MMS-EASE podem criar-se esses objectos MMS sem haver necessidade de os especificar em termos de ASN.1. O MMS-EASE suporta uma meta-linguagem para descrição de tipos designada por TDL (*Type Description Language*). A função de suporte *ms_mk_asn1_type (asn1len, asn1, type)* permite a definição ASN.1 de um determinado tipo, a partir da sua especificação TDL, que é passada como parâmetro de entrada (*type*). O parâmetro *asn1* apontará para o início de um *buffer* onde será colocada a definição ASN.1. O parâmetro *asn1len* é um parâmetro simultaneamente de entrada e de saída. Conterá o comprimento real (em *bytes*) da definição ASN.1 resultante.

A função MMS-EASE de suporte *ms_add_named_type (dom, tname, asn1ptr, asn1len)* permitirá adicionar um objecto MMS da classe *Named Type*. Esta definição será adicionada ao membro *type_list* da estrutura *struct domain_objs* relacionada. O parâmetro *dom* da função é um apontador para uma estrutura do tipo *struct domain_objs*. Se se pretender adicionar um tipo ao servidor como sendo específico do VMD, esse parâmetro deverá apontar para o elemento *vmd_wide* de uma estrutura do tipo *struct vmd_ctrl*. Caso se pretenda adicionar tipo MMS a um domínio específico, esse parâmetro deverá apontar para um elemento *objs* de uma estrutura de controlo de um domínio, ou seja uma estrutura do tipo *struct named_dom_ctrl*.

A função *ms_add_named_type (dom, tname, asn1ptr, asn1len)* retorna um apontador para uma estrutura do tipo *struct named_type*. A constituição dessa estrutura é a seguinte:

```
struct named_type               /* type definition table structure   */
{
        struct dbl_lnk link;                    /* generic list links */
        BYTE typename[MAX_IDENT_LEN+1];         /* name of data type                */
        BOOLEAN deletable;                      /* set != 0 if deletable            */
        BYTE protection;                        /* protection afforded this object  */
```

```
        BOOLEAN erased;                          /* flags if deleted, but nref != 0   */
        SHORT nref;                              /* number of var def'ns referencing this*/
        struct runtime_type *rt_head;            /* ptr to runtime type def head      */
        SHORT rt_num;                            /* # of rt_blocks in runtime type def */
        SHORT asn1len;                           /* # bytes in ASN.1 definition       */
        UBYTE *asn1ptr;                          /* pointer to ASN.1 definition       */
        SHORT blocked_len;                       /* length of blocked (packed) data type */
        BYTE *(*read_ind_fun)(BYTE *,SHORT);          /* std read ind function ptr */
        SHORT (*write_ind_fun)(BYTE *,BYTE *,SHORT);       /* std write ind       */
};
```

Existem ainda duas variáveis, à semelhança do que acontece para os domínios:

```
extern SHORT   max_mmsease_types;        /* max number allowed       */
extern SHORT   mms_type_count;           /* number currently defined */
```

Os tipos MMS a criar poderão ser, por uma questão de poupança de recursos, específicos do único VMD do dispositivo de interligação, uma vez que uma determinada variável MMS nomeada, mesmo sendo específica de um domínio, poderá ser de um tipo MMS específico do VMD do qual esse domínio faz parte. A opção por definir os tipos como sendo sempre específicos de domínios, consoante as variáveis consideradas em cada um deles, apenas teria algumas vantagens na consideração de transferências globais de informação dos domínios (*uploads* e *downloads*).

Como já foi dito, a definição ASN.1 pode ser obtida através da função *ms_mk_asn1_type (asn1len, asn1, type)*, na qual o parâmetro *type* corresponde a uma descrição TDL do tipo a acescentar. A TDL descreve tipos de variáveis de uma maneira mais simples do que o ASN.1. A TDL consiste em dois tipos de elementos. Um primeiro tipo de elementos corresponde aos nomes pré-definidos de tipos simples, que combinados ,constituem os tipos mais complexos. Um segundo tipo diz respeito a marcas de controlo, utilizadas para especificar o início e o fim de estruturas, ou a indicação de comprimentos.

O MMS-EASE tem uma série de tipos pré-definidos que estão definidos como tipos globais MMS-EASE tal como se segue (tipo ASN.1 ↔ tipo global MMS-EASE): Integer8 ↔ BYTE, Integer16 ↔ SHORT, Integer32 ↔ LONG, Unsigned8 ↔ UBYTE, Unsigned16 ↔ USHORT, Unsigned32 ↔ ULONG, Boolean ↔ BOOLEAN, Float ↔ FLOAT, Double ↔ DOUBLE, String8 ↔ BYTE[9], String16 ↔ BYTE[17] ... , String256 ↔ BYTE[257].

Os tipos simples utilizados no TDL são o BCD, Bool, Byte, Short, Long, Ubyte, Ushort, Ulong, Float, Double, Gtime, Btime4, Btime6, VstringXX e FstringXX. As suas correspondências com o C (utilizando as redefinições para tipos globais MMS-EASE) está referida em [EASE - 92].

As marcas de controlo utilizadas pelo TDL são as que se indicam a seguir.

As marcas "{" e "}" são utilizadas para delimitar o início "{" e o fim "}" da definição de uma estrutura complexa.

As marcas "[" e "]" são utilizadas para delimitar o início "[" e o fim "]" da definição de vectores (*arrays*). Logo a seguir ao início de um *array* deverá existir ou um "p" ou um número que indica o número de elementos do *array*.

A marca "p" logo a seguir ao início de um *array* ou de uma estrutura complexa indica que os seus elementos deverão ser empacotados (*packed*). O MMS-EASE toma por defeito o não empacotamento das variáveis.

A marca ":" é utilizada para separar diferentes campos numa especificação de tipo, tal como separar o número de elementos de um *array* do nome do tipo dos elementos.

As marcas "(" e ")" são utilizadas para assinalar o início "(" e o fim ")" do nome de um elemento individual de uma estrutura complexa.

As marcas "<" e ">" são utilizadas para assinalar o início "<" e o fim ">" de referências a tipos já colocados na base de dados do MMS-EASE.

A marca "@" é utilizada para referênciar tipos que já existam e que sejam específicos do VMD (@VMD) ou específicos da associação (@AA).

Exemplifique-se, para a adição do tipo MMS *IOGLOB_MMS_TYPE_NAME*, correspondente a variáveis globais dos módulos de E/S digitais.

Como se viu no capítulo 7, a definição para essa variável complexa era a seguinte (definição C):

```
typedef struct _ioglob_md_c_type
{
        unsigned char  dig_val[16];
        struct
        {
                unsigned long  secs;
                unsigned long  msecs;
        } time_stamp;
}ioglob_md_c_type;
```

Utilizando a redefinição de tipos globais MMS-EASE já aludida, a declaração deste tipo C deverá passar a ser:

```
typedef struct _ioglob_md_c_type
{
        UBYTE  dig_val[16];
        struct
        {
                ULONG   secs;
                ULONG   msecs;
        } time_stamp;
}ioglob_md_c_type;
```

A definição TDL para este tipo seria a seguinte:

```
#define IOGLOB_TDL_DESCR        ( "{ (dig_val) [16:Ubyte], \
                                  (time_stamp) {(secs) Ulong, (msecs) Ulong }}" )
```

Admitam-se ainda as seguintes definições:

```
#define IOGLOB_TYPE_NAME        ("IOGLOB_MMS_TYPE")

#define STRING_LENGTH           (500)
```

Assim, pode-se ter um função que permita a adição de qualquer tipo MMS:

```
...
void add_mms_types (void)
{
                /* colocar aqui todos os tipos que se pretendam definir */

        def_specific_type (IOGLOB_TYPE_NAME, IOGLOB_TDL_DESCR, STRING_LENGTH);
        ...

}

...
void def_specific_type (BYTE *type_name, BYTE *tdl_descr, SHORT mms_len)
{
                BYTE *ret;
        BYTE t_name[60];

        BYTE *tdl_buffer = (BYTE *) malloc (sizeof(BYTE)*STRING_LENGTH);
        BYTE *buf_ptr = (BYTE *) calloc (mms_len, sizeof(BYTE));
        struct named_type *nt_ptr;

        strcpy (tdl_buffer, tdl_descr);
        ret = ms_mk_asn1_type (&mms_len, buf_ptr, tdl_buffer);

        strcpy (t_name, type_name);
        nt_ptr = ms_add_named_type (&m_vmd_select->vmd_wide, t_name, buf_ptr,
mms_len);
}
```

Na arquitectura proposta são considerados 7 tipos MMS nomeados. As variáveis designadas por *IOGLOI_MD* e *IOGLOO_MD* são do tipo C *ioglob_md_c_type*. As variáveis designadas por *IOINDI_xx* são do tipo C *ioinpi_xx_c_type*. As variáveis do tipo *IOINDO_xx* são do tipo *iooutI_xx_c_type*. As variáveis do tipo *PARIOS_xx*, são do tipo C *parios_xx_c_type*. As variáveis do tipo *COMAND_MD*, *ALARMS_MD* e *SUBSCR_MD* são dos tipo C, respectivamente *comand_md_c_type*, *alarms_md_c_type* e *subscr_md_c_type*. Os tipos C para esses diferentes tipos MMS são os que se apresentam a seguir.

```
typedef struct _ioglob_md_c_type
{
        UBYTE   dig_val[16];
```

```
        struct
        {
                ULONG   secs;
                ULONG   msecs;
        } time_stamp;
}ioglob_md_c_type;


typedef struct _ioinpi_xx_c_type
{
        UBYTE   input_type;
        ULONG   input_val;
        struct
        {
                ULONG   secs;
                ULONG   msecs;
        } time_stamp;
        struct
        {
                ULONG   secs;
                ULONG   msecs;
        } time_level_0;
        struct
        {
                ULONG   secs;
                ULONG   msecs;
        } time_level_1;
}ioinpi_xx_c_type;


typedef struct _ioouti_xx_c_type
{
        ULONG   output_val;
        struct
        {
                ULONG   secs;
                ULONG   msecs;
        } time_stamp;
}ioouti_xx_c_type;


typedef struct _parios_xx_c_type
{
        UBYTE   par[256];
}parios_xx_c_type;


typedef struct _alarms_md_c_type
{
        UBYTE   mod_address;
        UBYTE   mod_type;
        UBYTE   io_type;
        UBYTE   alarm_qual;
        UBYTE   value;
        struct
        {
                ULONG   secs;
                ULONG   msecs;
        } time_stamp;
}alarms_md_c_type;


typedef struct _comand_md_c_type
{
        UBYTE   command;
        UBYTE   io_num;
        UBYTE   value;
        struct
        {
```

```
                ULONG   secs;
                ULONG   msecs;
        } time_level_0;
        struct
        {
                ULONG   secs;
                ULONG   msecs;
        } time_level_1;
}comand_md_c_type;


typedef struct _subscr_md_c_type
{
        UBYTE   event_type;
        UBYTE   io_num;
        BOOLEAN         status;
}subscr_md_c_type;
```

As descrições TDL e os nomes MMS desses tipos serão, por isso, os seguintes:

```
#define IOGLOB_TDL_DESCR ( "{ (dig_val) [16:Ubyte], \
                              (time_stamp) {(secs) Ulong, (msecs) Ulong }}" )

#define IOINPI_TDL_DESCR ( "{ (input_type) Ubyte,    \
                              (input_val) Ulong,     \
                              (time_stamp) {(secs) Ulong, (msecs) Ulong }, \
                              (time_level_0) {(secs) Ulong, (msecs) Ulong },\
                              (time_level_1) {(secs) Ulong, (msecs) Ulong }}" )

#define IOOUTI_TDL_DESCR ( "{ (output_val) Ulong,    \
                              (time_stamp) {(secs) Ulong, (msecs) Ulong }}" )

#define PARIOS_TDL_DESCR ( "{ (par) [256:Ubyte] }" )

#define ALARMS_TDL_DESCR ( "{ (mod_address) Ubyte,   \
                              (mod_type) Ubyte,      \
                              (io_type) Ubyte,       \
                              (alarm_qual) Ubyte,    \
                              (value) Ubyte, \
                              (time_stamp) {(secs) Ulong, (msecs) Ulong }}" )

#define COMAND_TDL_DESCR ( "{ (command) Ubyte,       \
                              (io_num) Ubyte,        \
                              (value) Ubyte,         \
                              (time_level_0) {(secs) Ulong, (msecs) Ulong },\
                              (time_level_1) {(secs) Ulong, (msecs) Ulong }}" )

#define SUBSCR_TDL_DESCR ( "{ (event_type) Ubyte,    \
                              (io_num) Ubyte,        \
                              (status) Boolean, }            ")
```

```
#define IOGLOB_TYPE_NAME        ("IOGLOB_MMS_TYPE")
#define IOINPI_TYPE_NAME        ("IOINPI_MMS_TYPE")
#define IOOUTI_TYPE_NAME        ("IOOUTI_MMS_TYPE")
#define PARIOS_TYPE_NAME        ("PARIOS_MMS_TYPE")
#define ALARMS_TYPE_NAME        ("ALARMS_MMS_TYPE")
#define COMAND_TYPE_NAME        ("COMAND_MMS_TYPE")
#define SUBSCR_TYPE_NAME        ("SUBSCR_MMS_TYPE")
```

A função apresentada em cima *add_mms_types ( )* passa a ter o seguinte aspecto:

```
...
void add_mms_types (void)
{
        /* colocar aqui todos os tipos que se pretendam definir */

    def_specific_type (IOGLOB_TYPE_NAME, IOGLOB_TDL_DESCR, STRING_LENGTH);
    def_specific_type (IOINPI_TYPE_NAME, IOINPI_TDL_DESCR, STRING_LENGTH);
    def_specific_type (IOOUTI_TYPE_NAME, IOOUTI_TDL_DESCR, STRING_LENGTH);
    def_specific_type (PARIOS_TYPE_NAME, PARIOS_TDL_DESCR, STRING_LENGTH);
    def_specific_type (ALARMS_TYPE_NAME, ALARMS_TDL_DESCR, STRING_LENGTH);
    def_specific_type (COMAND_TYPE_NAME, COMAND_TDL_DESCR, STRING_LENGTH);
    def_specific_type (SUBSCR_TYPE_NAME, SUBSCR_TDL_DESCR, STRING_LENGTH);

}
...
```

Pelas razões já expostas atrás, no servidor terão de ser considerados os objectos MMS da classe *Named Variable*. A função de suporte *ms_add_named_var (dom, vname, type, addr, chan)*, permite definir, no servidor, um objecto MMS do tipo *Named Variable*. A função retorna um apontador para uma estrutura do tipo *struct named_var*.

A função *ms_add_named_var (dom, vname, type, addr, chan)* vai inserir uma variável nomeada numa lista alfabetizada de variáveis, no membro *var_list* da estrutura *struct domains_objs* específica dum domínio ou de um VMD. A função verifica se uma outra variável nomeada, com nome idêntico, já existe ou não definida (não deverão ser consentidos nomes idênticos no mesmo domínio ou, no caso de a variável ser específica de um VMD, no mesmo VMD). À função são passados os parâmetros *dom*, que é um apontador para uma estrutura do tipo *struct domain_objs*, *vname*, que é o nome da variável, *type*, que é um apontador para uma estrutura do tipo *struct object_name*, nome esse que é o do tipo nomeado, o parâmetro *addr*, que é um apontador para uma estrutura do tipo *struct var_acc_addr*, e especifica o endereçamento físico da variável, e por fim o parâmetro *chan*, que só terá significado se a variável a criar for específica de uma associação de aplicação.

As variáveis MMS a criar no âmbito desta implementação particular serão específicas de domínios.

As estruturas do MMS-EASE relacionadas com os objectos MMS da classe *Named Variable* são as seguintes:

```
struct  named_var               /* name table structure          */
{
        struct dbl_lnk link;                            /* generic list links */
        BYTE varname[MAX_IDENT_LEN+1];                  /* variable name      */
        BOOLEAN invalid;                                /* variable invalidated due */
                                                        /* to type or addr deleted  */
```

```
        struct named_type *type;                    /* type name/domain          */
        BOOLEAN deletable;                          /* != 0 if deletable  */
        BYTE rd_pro;                                /* read protection           */
        BYTE wr_pro;                                /* write protection          */
        struct var_acc_addr addr;                   /* address                   */
        BYTE *(*read_ind_fun)(BYTE *,SHORT);        /* read ind fun for this var */
        SHORT (*write_ind_fun)(BYTE *,BYTE *,SHORT);/* write ind fun for var     */
};

extern SHORT    max_mmsease_vars;       /* max number allowed      */

extern SHORT    mms_var_count;          /* number currently defined      */
```

Os campos de protecção de escrita ou de leitura não são relevantes nesta versão do MMS-EASE uma vez que não é suportada essa funcionalidade. Uma vez que esta funcionalidade é necessária para a implementação do dispositivo de interligação tal como proposto no capítulo 7 (por exemplo, as variáveis que representam E/S de entrada não podem ser escritas), terá de ser previsto código adicional nas funções de indicação dos serviços MMS de escrita e leitura (ver redes de petri da especificação das funções de indicação MMS *Read* e *Write*).

A estrutura *struct var_acc_addr* apresenta o seguinte conteúdo:

```
struct var_acc_addr
{
        SHORT   addr_tag;
        union
        {
                ULONG   num_addr;
                BYTE    *sym_addr;
                struct unconst_addr   unc_addr;
        } addr;
};
```

O MMS-EASE tem uma função da classe 12 (*Address Resolution functions*) que permite traduzir a informação presente numa estrutura do tipo *struct named_var* para o endereço físico da variável. Essa função, *u_get_named_addr (vardef)*, é chamada pela máquina virtual quando são chamadas as funções da classe da classe 5 (*Virtual Response*) *mv_read_resp (req_ptr)* e *mv_write_resp (req_ptr)*, respectivamente para enviar respostas a pedidos MMS *Read* e *Write*.

A função *u_get_named_addr (vardef)* é definida pelo utilizador. Tipicamente ela deverá retornar o conteúdo do membro apropriado da estrutura *struct var_acc_addr* apontada por *addr* na estrutura *struct named_var*. Isto permite resolver endereços que poderão ser artificiais. Esta função retorna um apontador para BYTE. No caso do dispositivo de interligação, poderá só ser suportado o endereço numérico.

```
BYTE *u_get_named_addr (vardef)
struct named_var *vardef;
{
        SHORT offset;

        if (vardef->addr.addr_tag == 0 )            /* endereço numérico   */
                return ((BYTE *) vardef->addr.addr.num_addr);
```

```
        else
                return (NULL);
}
```

Assim, para adicionar uma variável MMS nomeada, por exemplo correspondente a uma variavel global digital, dever-se-á alocar memória para essa variável, e actualizar o campo *vardef.addr.num_addr* com o resultado dessa alocação, antes de chamar a função *ms_add_named_var (dom, vname, type, addr, chan).*

```
struct initdown_req_info *dom_info = NULL;
struct named_dom_ctrl *ndom;
struct domain_objs *dom;
struct var_acc_addr adr;
struct object_name type;

/* Criação de um domínio chamado "filinp"                    */

dom_info = (struct initdown_req_info *) malloc (sizeof(struct initdown_req_info));
strcpy (dom_info->dname, "filinp");
dom_info->sharable = 0;
dom_info->dl_detail_pres = 0;
dom_info->num_of_capab = 0;

if (!ms_add_named_dom(dom_info, 0))
        report_error();                 /* reportar situação de erro */


/* Obter um apontador para o estrutura de controlo do domínio criado */
/* e acrescentar uma variável MMS      */

if (!(ndom = ms_find_named_dom("filinp")))
        report_error();
else
{
        dom = &(ndom->objs);
        adr.addr_tag = 0;               /* enderço numérico */
        adr.num_addr = (ULONG *) malloc (sizeof (ioglob_md_c_type));

        type.obj_tag = VMD_SPEC;
        strcpy (type.obj_name.vmd_spec, "IOGLOB_MMS_TYPE_NAME");
        if (!ms_add_named_var(dom, "IOGLOI_MD", &type. &adr, 0))
                report_error();
}
```

8.2.9. Função de Inicialização da Máquina MMS

A seguir apresenta-se um estrutura simplificada da função de inicialização do MMS-EASE e configuração automática do servidor (criação de domínios e variáveis MMS), a partir da informação disponível na base de dados da rede de *Field Bus*. A função *mmsvar ( )* precisa de ser chamada antes de de ser chamada a função *strt_MMS ( )*. A função *mmsvar ( )* põe as variáveis do ambiente MMS-EASE com os seus valores de defeito.

```
void initiate_mms (void)
{
        ...
        char canais[8];
        mmsvar();
        setup_board_param ();
        setup_mms_envirn ();
        setup_preferr_param ();
        strt_MMS();

        mllp_act_ar_name ("filbus_control", ACSE30_LLP);
        for (i = 0; i < 8; i++)
        {
                rc = mllp_reg_ar_name (i, "mestre_filbus");
                rc = mllp_ass_listen (i);
        }
        add_mms_types ();
        read_filbus_database ();
        create_mms_objects ();
        ...

}       /* fim da função */
```

8.2.10. Gestão de Associações de Aplicação

O dispositivo de interligação poderá receber indicações MMS *Initiate*, *Conclude*, *Abort*, *Reject* e *Cancel*.

Conforme foi visto no capítulo 6, a recepção de uma indicação de MMS *Conclude*, corresponde a um pedido do cliente MMS para acabar com a associação de aplicação. Ao receber essa indicação, o dispositivo de interligação deverá apagar esse canal das tabelas de subscrição, conforme foram definidas no capítulo 6.

As funções MMS-EASE da classe 8, *User Indication Functions*, são funções definíveis pelo utilizador chamadas automaticamente de acordo com o serviço MMS. Estas funções têm a forma *u_xxxx_ind (req_ptr)*.

O parâmetro que é passado a este tipo de funções, *req_ptr*, corresponde a um apontador para um dos três tipos de estruturas designadas por estruturas de fila (*queue*) do MMS-EASE. No caso das indicações, a estrutura é do tipo MMSREQ_IND. Esta estrutura corresponde sensivelmente ao objecto de transacção do MMS, existindo uma estrutura deste tipo para cada indicação MMS pendente. O seu conteúdo é o que se apresenta em baixo.

```
typedef struct rspdat
{
        struct  dbl_lnk link;           /* generic double linker list struct  */
        SHORT pri;                      /* indication priority                */
        SHORT  chan;                    /* channel req was sent on            */
        SHORT  llp;                     /* lower layer service select         */
        USHORT context;                 /* P-context select                   */
        USHORT op;                      /* mms opcode                         */
```

```
        ULONG  id;                      /* invoke ID                          */
        BOOLEAN req_info_pres;          /* information present                */
        BYTE    *req_info_ptr;          /* info pointer                       */
        struct list_of_mods   mods;     /* tracking for modifiers             */
        SHORT  cancl_state;             /* indicates if being cancelled & state */
        LONG    ind_time;               /* time recieved                      */
        BOOLEAN rxbuf_pres;             /* flag set when rxbuf must be released */
        UBYTE  *rxbuf;                  /* pointer to buffer rx msg was in    */
        struct cmd_service s;           /* outstanding command service info   */
        UBYTE  add_addr_info_pres;      /* LLC specific addressing info       */
        struct llp_addr_info *add_addr_info;
} MMSREQ_IND;
```

Na estrutura MMSREQ_IND existem, entre outros elementos, dois que devem ser analisados. Um é o *req_info_pres*, que indica se existe ou não informação presente na indicação. Caso *req_info_pres* seja diferente de '0', *req_info_ptr* aponta para uma estrutura específica da operação (código de operação do serviço MMS em causa) que contém informação pertencente a essa indicação. Estruturas específicas de operações correspondem à quarta classe de estruturas de dados de MMS-EASE, como foi visto no início deste capítulo.

As indicações MMS de *Abort* e *Reject* constituem excepções à forma geral da função *u_xxxx_ind (req_ptr)*. Assim, essas funções têm a seguinte forma *u_abort_ind (chan, reason, au_flag)* e *u_reject_ind (chan, info_ptr)*. No caso das indicações MMS de *Cancel* e *Conclude*, as correspondentes funções da classe MMS-EASE são *u_cancel_ind (req_ptr)*, *u_conclude_ind (req_ptr)*.

A função *u_abort_ind (chan, reason, au_flag)* é chamada sempre que é recebida uma indicação de *Abort* através de um determinado canal. Quando esta função é chamada, a associação que existia nesse canal deixa de estar activa. O parâmetro *reason* vale sempre '0', no caso de uma ligação ao ACSE, como é o caso. O parâmetro *au_flag* indica se a origem do *Abort* foi local e da responsabilidade do *provider*, nesse caso vale '0' ou se a origem do *Abort* é remota, e nesse caso terá um valor diferente de '0'.

No caso do dispositivo de interligação, ao ser recebido um *Abort*, deverá ser apagado das tabelas de subscrição (variáveis da aplicação *module_ctrl*) esse canal, e, no caso deste dispositivo de interligação, o canal deverá ser reposto no modo *listen*, isto é chamar-se a função *mllp_ass_listen (chan)*. A função de atendimento da indicação MMS de *Abort* poderá por isso ter o seguinte aspecto:

```
VOID u_abort_ind (chan,reason,au_flag)
SHORT chan;
SHORT reason;
SHORT au_flag;
{
        clear_subscribe (chan);         /* função do utilizador */
        ms_clr_ind_que (chan,1);
        mllp_ass_listen (chan);
}
```

A função de suporte *ms_clr_ind_que (chan, clr_all)* pode ser utilizada para apagar filas de indicações que ainda não tivessem sido respondidas. A *flag clr_all* permite só apagar indicações que ainda não tivessem sido passadas ao utilizador (pela chamada da função *u_xxxx_ind ( )* apropriada), e nesse caso vale '0', ou apagar todas as que não tivessem sido respondidas e nesse caso sería diferente de '0'.

Existem outras funções de suporte do MMS-EASE que poderão potencialmente ser utilizadas dentro da função *u_abort_ind (chan, reason, au_flag)*. Por exemplo as funções *ms_clr_locl_fctrl (chan)* e *ms_clr_rem_fctrl (chan)* poderão ser utilizadas para retirar informação de controlo respeitante a ficheiros remotos, e no caso de ficheiros locais podê-los fechar. Os serviços de gestão de ficheiros MMS não serão considerados nesta implementação, pelo que é desnecessária a inclusão da chamada à função *ms_clr_locl_fctrl (chan)*.

A função *u_conclude_ind (req_ptr)* é chamada sempre que é recebida uma indicação MMS de *Conclude* através de um determinado canal. O pedido MMS de *Conclude*, feito por um cliente MMS, ao contrário do MMS *Abort*, visa terminar uma associação de aplicação de forma não intempestíva. O pedido MMS *Conclude* corresponde a um serviço MMS confirmado, pelo que o servidor deverá responder a esse pedido com uma primitiva de resposta MMS. No entanto, se existirem pedidos que ainda não tenham sido respondidos, poderá ser protelada a resposta a esse pedido (utilização de MMS em modo assíncrono para este serviço) ou simplesmente enviar-se uma resposta negativa.

A resposta MMS a esse pedido deverá ser efectuada por chamada à função MMS-EASE da classe 2, *Primitive Response Functions*, *mp_conclude_resp (req_ptr)*. O parâmetro *req_ptr* é obviamente um apontador para uma estrutura do tipo MMSREQ_IND, e que é passado à função *u_conclude_ind (req_ptr)*. No caso de haver indicações pendentes, deverá ser chamada a função *mp_conclude_err (req_ptr, class, code)*, que é uma função MMS-EASE da classe 4, *Error Response Functions*.

Nesta última função, o parâmetro *class* indica a classe particular de erro tal como especificado na norma MMS ISO 9506. Neste caso deverá ser 9. O parâmetro *code* deverá especificar a razão particular para esse erro no pedido de *Conclude*. O número de indicações pendentes poderá ser determinado por recurso à função MMS-EASE de suporte *ms_count_ind_pend (chan)*.

Portanto, a função de atendimento da indicação MMS de *Conclude* poderá ser implementada da seguinte maneira:

```
VOID u_conclude_ind (req_info)
MMSREQ_IND *req_info;
{
        if (!ms_count_ind_pend (req_info->chan))
                mp_conclude_resp (req_info);
        else
                mp_conclude_err (req_info, 9, 0);
}
```

Na realidade, o canal cuja associação foi concluída ainda não pode ser considerado livre. Isso só acontecerá quando for recebida uma indicação que chamará a função MMS-EASE da classe 8, *u_release_ind (chan)*. Só nessa altura o canal poderá ser posto em modo *listen*.

```
VOID u_release_ind (chan)
SHORT chan;
{
        SHORT ret;
        if ((ret = mllp_ass_listen (chan)))
                report_error();
}
```

A função *u_cancel_ind (req_ptr)* é chamada sempre que é recebida uma indicação MMS de *Cancel* através de um determinado canal. O pedido MMS de *Cancel*, feito por um cliente MMS, permite cancelar um outro pedido previamente efectuado. Na interface MMS-EASE, este pedido tem prioridade uma vez que a função *u_cancel_ind ( )* é chamada aquando da chamada à função *mms_llp_serve ( )*, enquanto que as outras funções de atendimento de indicações são chamadas quando é chamada a função *mms_ind_serve ( )*. Quer *mms_llp_serve ( )* quer *mms_ind_serve ( )* são funções MMS-EASE 13, *Interface Functions*. A sua utilização será discutida um pouco mais à frente, pelo que a ilação feita atrás àcerca da prioridade dada ao atendimento da indicação MMS *Cancel* resultará mais clara.

A função *mv_cancel_resp (req_ptr, sucess)* ou a função *mp_cancel_resp (req_ptr)* poderão ser utilizadas para responder ao pedido MMS *Cancel*. A primeira, uma função MMS-EASE da classe 6, *Virtual Response Functions*, permite efectuar uma resposta sem haver a preocupação de cancelar a indicação directamente. Esta função, quando chamada enviará uma resposta quer ao pedido de serviço original quer à indicação de *Cancel*. Se o parâmetro *success* for igual a '0', é enviada uma resposta positiva ao *Cancel* e uma resposta de erro ao serviço

original. Se *success* for diferente de '0' é enviada uma resposta negativa ao *Cancel* e nenhuma ao serviço original. A função *mp_cancel_resp (req_ptr)* é uma função da classe 2 do MMS-EASE, e deverá ser usada para responder ao pedido MMS *Cancel*, depois de ter sido determinado se o pedido poderá ou não ser cancelado. Se for determinado que o pedido pode ser cancelado, deverá ser enviada a resposta positiva ao *Cancel* e deverá haver uma resposta negativa à indicação original, utilizando para tal a função da classe 4 do MMS-EASE, *mp_error_resp (req_ptr, class, code)*. A utilização da interface PPI do MMS-EASE torna mais flexível a resposta a um pedido de *Cancel*. Essa flexibilidade não é no entanto necessária para a aplicação presente, pelo que se opta pela utilização da interface VMI do MMS-EASE, e por isso a função de atendimento da indicação MMS de *Cancel* terá o seguinte aspecto:

```
VOID u_cancel_ind (req_info)
MMSREQ_IND *req_info;
{
        mv_cancel_resp (req_info,0);
}
```

A função *u_reject_ind (chan,info_ptr)* é chamada sempre que é recebida uma indicação de *Reject* através de um determinado canal, gerada localmente ou remotamente, devido a erros de protocolo que são detectados. O parâmetro *info_ptr* é um apontador para uma estrutura do tipo *struct reject_resp_info*. Esta estrutura pode ser consultada para obter diversa informação sobre o erro ocorrido. A estrutura *struct reject_resp_info* tem a seguinte composição:

```
struct reject_resp_info
{
        UBYTE detected_here;        /* flag indicating where error occurred */
        UBYTE invoke_known;         /* flag indicating if invoke ID known */
        ULONG invoke;               /* invoke ID of service rejected      */
        USHORT pdu_type;            /* pdu type of service rejected       */
        SHORT rej_class;            /* reject class                       */
        SHORT rej_code;             /* reject code                        */
};
```

```
VOID u_reject_ind (chan,rej_ptr)
struct reject_resp_info *rej_ptr;
SHORT   chan;
{
}
```

Por fim, o serviço MMS *Initiate*. Sempre que é recebido uma indicação MMS *Initiate* através de um canal, é chamada a função *u_init_ind (req_ptr)*. Ao receber um pedido de estabelecimento de associação, ele deverá ser aceite sempre que o canal estiver em modo *listen*.

Esse estado pode ser verificado na estrutura *mms_chan_info[chan]* do canal através do qual se recebeu o pedido MMS de *Initiate*. Caso se queira estabelecer a associação, deve ser chamada a função MMS-EASE *mv_init_resp (req_ptr)*, caso contrário deverá ser chamada a função MMS-EASE da classe 4, *mp_init_err(req_ptr,class,code)*.

A estrutura básica da função *u_init_ind (req_ptr)* deverá ser a seguinte:

```
VOID u_init_ind (req_info)
MMSREQ_IND *req_info;
{
        ...
        if (mms_chan_info[req_info->chan].chan_state != (LONG) M_LISTEN)
                mp_init_err(req_info, 8, 0);
        else
                mv_init_resp (req_info);
        ...
}
```

De notar que, em princípio, o canal só poderá receber a indicação se estiver em modo de *listen*, pelo que o teste efectuado é redundante.

8.2.11. Serviços de Suporte de VMD

A função *u_status_ind (req_ptr)* será chamada de cada vez que for recebida uma indicação MMS de *Status*. O membro da estrutura *req_ptr* importante é req_info_ptr. No caso do MMS *Status*, esta estrutura é do seguinte tipo:

```
struct status_req_info
{
        BOOLEAN extended;
};
```

A função de resposta primitiva a utilizar do MMS-EASE (não existe resposta virtual para este caso) recorre à chamada da função *mp_status_resp(req_ptr, info_ptr)*, na qual o parâmetro *info_ptr* é um apontador para uma estrutura do seguinte tipo:

```
struct status_resp_info
{
        SHORT   logical_stat;
        SHORT   physical_stat;
        BOOLEAN local_detail_pres;
        SHORT   local_detail_len;
        UBYTE   local_detail[16];
        BOOLEAN         addl_detail_pres;
        SHORT   addl_detail_len;
        BYTE    *addl_detail;
};
```

De notar que a norma MMS define o parâmetro *LocalDetail* como sendo opcional, e daí o facto de se indicar nesta estrutura se será de incluir ou não. O parâmetro *addl_detail* diz respeito às normas MMS companheiras (MMS *Companion Standards*).

O serviço MMS *Identify* permite a um cliente identificar um determinado servidor MMS. O MMS-EASE suporta a resposta a um pedido de identificação quer no modo virtual, quer no modo primitivo. No caso do modo primitivo, é preciso preencher a estrutura do tipo aquando da resposta:

```
struct ident_resp_info
{
        BYTE    vend[65];
        BYTE    model[17];
        BYTE    rev[17];
};
```

Para utilizar o modo virtual dever-se-á preenceher a estrutura do tipo *struct ident_resp_info* apontada por **mmse_ident_info* logo após chamar a função *strt_MMS( )*, que inicia o MMS.

No caso do dispositivo de interligação, ir-se-á utilizar o modo virtual. Assim, numa fase de inicialização dever-se-á ter o seguinte conjunto de instruções, e que define a identificação para o servidor agora em causa:

```
strcpy( mmse_ident_info->vend, "EduardoTovar" );
strcpy( mmse_ident_info->model, "FILBUS2MAP_GTW" );
strcpy(mmse_ident_info->rev, "1.0" );
```

A função de recepção de indicações do serviço MMS *Identify*, será simplesmente o que abaixo se indica:

```
VOID u_ident_ind (req_info)
MMSREQ_IND *req_info;
{
        mv_ident_resp (req_info);
}
```

8.2.12. Paragem da Máquina MMS

Em qualquer situação, e antes de sair para o sistema operativo, deve ser chamada a função *end_MMS ( )*. Esta é uma função MMS-EASE da classe 13. Antes de chamar a função *end_MMS ( )* deve-se desactivar a *AR Name*, chamando a função já referida *mllp_deact_ar_name ("filbus_control", ACSE30_LLP)*, uma vez que esta informação fica residente na carta de comunicações, o que conduziría a erro quando se pretendesse activá-la novamente.

8.2.13. Inicialização da Rede de Field Bus

A inicialização da rede de *Field Bus* deverá essencialmente proceder à abertura do *device driver* do *Field Bus*, e fazer a chamada à função *start_network ( )* da interface da rede de *Field Bus*. Esta função deverá ser chamada antes de arrancar a máquina MMS.

A função *start_network ( )* procede à inicialização da rede de *Field Bus*, incluíndo as funções de endereçamento do tipo *daisy chain*, *download* de parâmetros, programação da hora, etc. Para tal ela faz a leitura da base de dados da rede de *Field Bus*.

A estrutura básica da função *start_filbus ( )* deverá ser a seguinte:

```
start_filbus ()
{
        int rc;
        /* Open GESFIL driver */
        if (rc = _dos_open (FILBUS_DD, O_RDWR, &GF_handle) != 0)
        {
                ...
        }

        if (chdir (FILBUS_DIRECTORY) != 0)
        {
                        ...
        }

        /* init network name and gesfil master module name  */
        if (rc = start_network (GF_handle, 0, NETWORK, MASTER, 1) != 0)
                {
                switch (rc) {
                        case 0x81:      /* Error trying open network file */
                                        break;
                        case 0x82:      /* Master name not find in .NET file */
                                        break;
                        case 0x83:      /* Error during master initialization */
                                        break;
                        default:
                                                ;
                } /* endswitch */
        }
} /* end start_filbus function */
```

8.3. Suporte de Comunicações Verticais

8.3.1. Estruturas de Dados Auxiliares

A cada módulo estará associada uma estrutura deste tipo e que servirá simultaneamente para a tabela de controlo de actualizações e será preenchida aquando da criação dos domínios (cada módulo).

```
struct module_ctrl
{
        struct named_dom_ctrl *md_domain_name;
        UBYTE   address;
        UBYTE   md_type;
        UBYTE   evo_status;
        UBYTE   io_type [16];
        UBYTE   ioglob_updt_rd;
        UBYTE   ioglob_updt_wr;
        UBYTE   ioindi_updt_rd [16];
        UBYTE   ioindi_updt_wr [16];
        UBYTE   par_updt_rd [17];
        UBYTE   par_updt_wr [17];
        UBYTE   digit_subscr [8];
        UBYTE   alarms_subscr [32][8];
        UBYTE   counter_subscr [32][8];
        struct module_ctrl *Next;
};
```

A seguinte variável global consistirá no primeiro elemento da lista de estruturas do tipo *struct module_ctrl*:

```
struct module_ctrl Head_MD_List;
```

A estrutura de dados que se segue é utilizada aquando da recepção de 'indicações' provenientes da rede de *Field Bus* ou indicações MMS de acesso a variáveis. Esta estrutura irá conter informação útil às operações a efectuar. O conteúdo dessa estrutura é o que se segue.

```
struct GTW_VarDom
{
        MMSREQ_IND *Req_Ptr;
        struct module_ctrl *Mod_Ptr;
        struct named_var *Var_Ptr;
};
```

8.3.2. Atendimento de Indicações MMS

As funções MMS-EASE da classe 13, *Interface Functions*, fornecem os mecanismos necessários para indicar ao programa de aplicação acontecimentos provenientes da rede MAP/MMS (primitivas de indicação e primitivas de confirmação). Essa funções permitem passar as indicações e confirmações recebidas ao programa de aplicação, habilitam ou desabilitam os mecanismos de indicação de acontecimentos e fazem a gestão de filas de espera do LLP (*Low Layer Provider*).

Existem funções relativamente importantes e que têm a haver com o atendimento de indicações MMS ou confirmações MMS. No caso do dispositivo de interligação, só a primeira situação ocorrerá.

Existem, no MMS-EASE, 12 funções da classe 13.

Existe um apontador para função, *(*u_mms_event_notify)(num_events)*, que apontará para uma função definida pelo utilizador. Essa função será chamada sempre que chegar uma mensagem ao nó e se previamente tiver sido abilitado o mecanismo de notificação de eventos do MMS-EASE pela chamada da função MMS-EASE da classe 13, *ms_event_enable ( )*. A função *ms_event_disable ( )* implica que a função apontada por *(*u_mms_event_notify)(num_events)* deixe de ser chamada, impedidndo assim interrupções a meio de, por exemplo, processos de tempo crítico. De notar que a função de notificação de evento é chamada como um prolongamento do processamento de interrupção que tem lugar ao nível da rede de comunicações.

A chamada da função de notificação não permite, *per si*, a identificação do acontecimento. Uma outra função de interface terá de ser utilizada para o efeito, por forma a "passar" esse evento para o programa de aplicação. A função *mms_llp_serve ( )*, tem essa responsabilidade. Se a mensagem que vem do LLP é uma indicação, é feito o *parsing* da PDU e preenchida uma estrutura do tipo MMSREQ_IND, colocando-a numa fila interna para processamento posterior por uma outra função de interface do MMS-EASE, a função *mms_ind_serve ( )*. No caso de se tratar de uma confirmação (o que não será o caso deste dispositivo de interligação) será preenchida uma estrutura do tipo MMSREQ_PEND e chamada a função *mms_conf_serve ( )*.

A função *mms_ind_serve ( )* lê as mensagens da fila de indicações, e chamará a função apropriada do MMS-EASE da classe 8, *User Indication Functions*. Esta função chamará todas as funções do tipo *u_xxxx_ind ( )* para todas as indicações activas que ainda não tenham sido atendidas, seguindo um processamento cronológico (correspondente obviamente à estrutura da lista ligada).

Existe uma outra função de interface, a função *comm_serve ( )*, que por sua vez chama as funções *mms_llp_serve ( )*, *mms_ind_serve ( )* e/ou a função *mms_conf_serve ( )*. Esta função pode ser utilizada em alternativa à sequência *mms_llp_serve ( ), mms_ind_serve ( )*. Aliás, se não estiver abilitado o mecanismo de notificações, basta ir chamando periodicamente a função *comm_serve ( )* para manter o MMS a funcionar.

De notar que ao chamar a função *comm_serve ( )* ou simplesmente *mms_llp_serve ( )* só é passada uma mensagem MMS para a aplicação. Assim, se por exemplo se estiver a utilizar a função de notificação de eventos, o parâmetro *num_events* corresponderá ao número de vezes

que deverá ser chamada a função *comm_serve ( )*. No entanto, é preciso ter em conta que a notificação do evento ocorre durante o processamento de uma interrupção. Se for chamada a função *comm_serve ( )* dentro da função de notificação apontada por *(*u_mms_event_notify) (num_events)*, uma enorme quantidade de *stack* poderá ser necessária para lidar com possíveis acções recursivas sobre a função de notificação, como resultado da chegada de outras mensagens MMS. Poder-se-ia desabilitar e abilitar o mecanismo de notificação dentro da própria função de notificação. Por outro lado, o MS-DOS não é reentrante, pelo que não é de todo recomendável chamar, dentro da função de notificação, funções como *malloc ( )*, *printf ( )*, etc. Em vez de chamar a função *comm_serve ( )* directamente dever-se-á apenas, por exemplo, actualizar uma variável global que indicará quantas mensagens estão à espera de ser processadas por *mms_llp_serve ( )*.

A utilização do mecanismo de notificação de eventos não é relevante para a implementação do dispositivo de interligação. Por um lado, a leitura de indicações provenientes da rede de *Field Bus* funciona em puro *polling*: mesmo que não haja mensagens no *buffer* do controlador da rede, é preciso chamar a função de interface da rede de *Field Bus receive_msg ( )*. Por outro lado, como muitas da indicações MMS serão processadas em modo assíncrono é preciso chamar a função *ms_ind_serve ( )* para que sejam verificadas as indicações pendentes. Uma solução, passa por chamar alternadamente a função *receive_msg ( )*, para atendimento de 'indicações' provenientes da rede de *Field Bus* e a função *comm_serve ( )* para o atendimento de uma nova mensagem MMS e das indicações MMS pendentes, eventualmente incluindo uma nova.

Dadas as características da rede de *Field Bus* e da arquitectura proposta para o dispositivo de interligação, deverá haver uma cadência maior na chamada à função do *receive_msg ( )* do que à função *comm_serve ( )*, por exemplo três chamadas da primeira para uma chamada da segunda. Esta razão pode, inclusivamente, ser variável em função da cadência de recepção de indicações MMS e do fluxo de mensagens na rede de *Field Bus*. Nesse sentido, a utilização da função de notificação automática do MMS-EASE simplesmente para a actualização de uma variável global com o parâmetro *num_events*, poderá ser útil. A *Fig. 8.4* ilustra esta solução. Nessa figura é utilizada de uma forma pouco ortodoxa (em termos de modelização de Redes de Petri) uma variável *N*, que é recalculada de cada vez que é chamada a função *comm_serve ( )*.

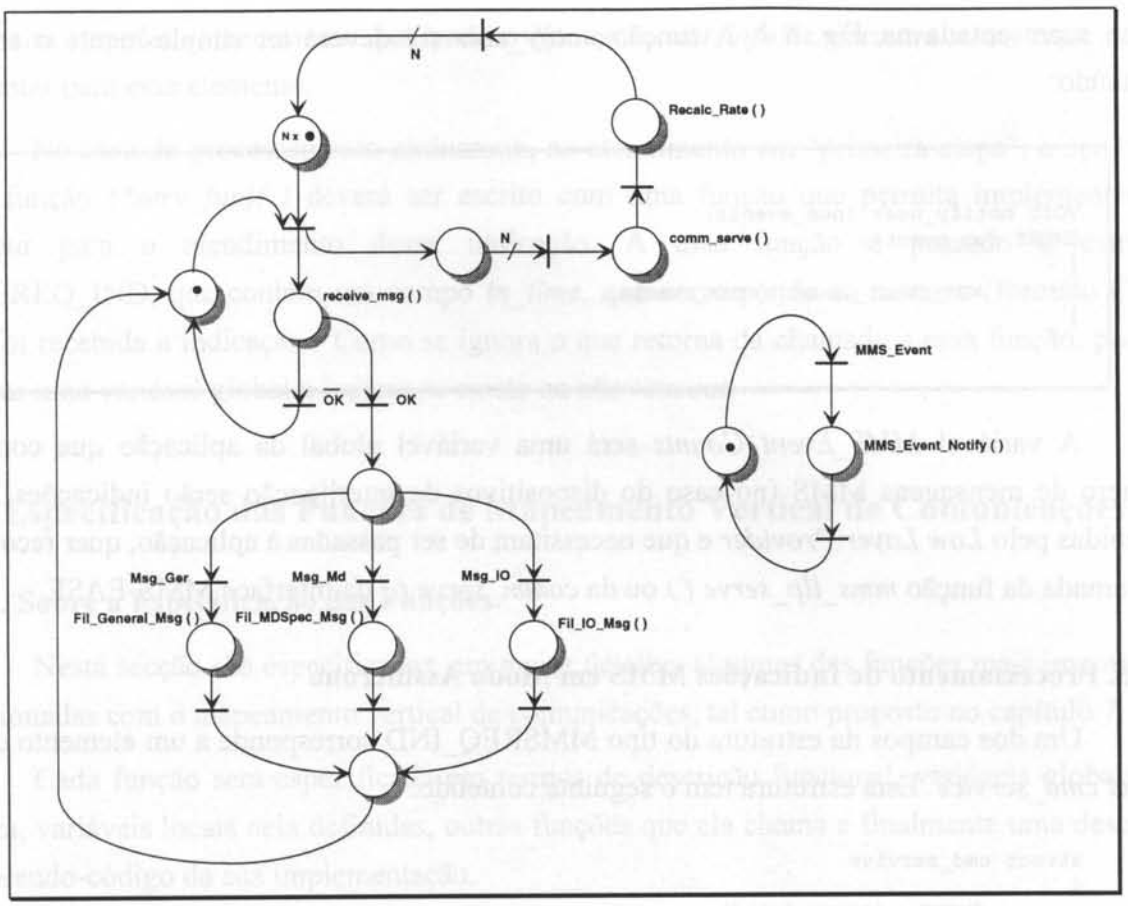


Figura 8.4 *- RdP Referente à Multiplexação Temporal no Atendimento de Indicações*

Depois de fazer a inicialização do MMS e a inicialização da rede de *Field Bus*, dever-se-ão então executar as seguintes instruções:

```
VOID notify_user (SHORT);      /* protótipo função de notificação do utilizador */

...

(*u_mms_event_notify) = notify_user;          /* escrever o apontador para função */

ms_event_enable ();                           /* habilitar o mecanismo de notif.*/

Integration_Core ();                          /* chamada à função nuclear do */
                                              /* dispositivo de interligação */
```

A função *Integration_Core ( )* será uma função que implementará o atendimento multiplexado de indicações provenientes das duas arquitecturas de comunicações a interligar, tal

como representada na *Fig. 8.4.* A função *notify_user ( )* deverá ter simplesmente o seguinte conteúdo:

```
VOID notify_user (num_events)
SHORT num_events;
{
        MMS_Event_Count = num_events;
}
```

A variável *MMS_Event_Counts* será uma variável global da aplicação que conterá o número de mensagens MMS (no caso do dispositivos de interligação serão indicações MMS) recebidas pelo *Low Layer Provider* e que necessitam de ser passadas à aplicação, quer recorrendo à chamada da função *mms_llp_serve ( )* ou da *comm_serve ( )* da interface MMS-EASE.

8.3.3. Processamento de Indicações MMS em Modo Assíncrono

Um dos campos da estrutura do tipo MMSREQ_IND corresponde a um elemento do tipo *struct cmd_service*. Esta estrutura tem o seguinte conteúdo:

```
struct cmd_service
{
        SHORT   (*serv_fun)();
        BYTE    *cmd_info;
        LONG    cmd_stst;
        SHORT   aux_flags[6];
};
```

Quando uma indicação é colocada pela primeira vez, os valores desta estrutura terão valores por defeito. Em particular, o apontador para função *(*serv_fun)( )* apontará para NULL. Se for escrito este apontador, essa função será chamada sempre que se for processar essa indicação.

Esta estrutura pode ser utilizada para implementar uma máquina de estados simples que permita a utilização em modo assíncrono. Por exemplo, *aux_flags[0]* poderá servir para saber se é ou não a primeira vez que é chamada a função de atendimento desta indicação. Se *aux_flags[0]* = '0', então essa será a primeira vez que é chamada a função de atendimento para essa indicação. Caso *aux_flags[0]* = '1' então o atendimento dessa indicação já se encontra numa "segunda etapa", conforma referido no capítulo anterior.

Na "primeira etapa" de atendimento da indicação, *aux_flags[0]* deverá ser colocada a '1'. Deverá também, no caso de haver necessidade de processar essa indicação em modo assíncrono, obter-se o *byte* da tabela de controlo (ver capítulo 7) que deverá ser utilizado para sincronizar a resposta MMS. Por forma a melhorar a performance, a obtenção desse *byte* só será feita na

"primeira etapa" do atendimento, passando o elemento *cmd_info* da estrutura *struct cmd_service* a apontar para esse elemento.

No caso do processamento assíncrono, no atendimento em "primeira etapa", o apontador para função *(*serv_fun)( )* deverá ser escrito com uma função que permita implementar um *timeout* para o atendimento dessa indicação. A essa função é passado a estrutura MMSREQ_IND, que contém um campo *in_time*, que corresponde ao instante (formato C) em que foi recebida a indicação. Como se ignora o que retorna da chamada a essa função, pode-se utilizar uma variável global a indicar se existe ou não *timeout*.

8.4. Especificação das Funções de Mapeamento Vertical de Comunicações

8.4.1. Sobre a Especificação das Funções

Nesta secção são especificadas, em maior detalhe, algumas das funções mais importantes relacionadas com o mapeamento vertical de comunicações, tal como proposto no capítulo 7.

Cada função será especificada em termos de descrição funcional, variáveis globais que utiliza, variáveis locais nela definidas, outras funções que ela chama e finalmente uma descrição em pseudo-código da sua implementação.

A especificação em pseudo-código recorre diversas vezes à directiva *Exit* a meio de uma função. Isto visa simplesmente evitar um nível de imbricamento elevado (resultante em indentações) necessário se se pretendesse dar um cariz mais estruturado ao pseudo-código.

Relativamente às funções que ela chama, junto dos seus nomes poderá aparecer uma anotação indicando tratarem-se de funções disponíveis em bibliotecas. Essas anotações são **FIL**, para as funções disponíveis nas bibliotecas da interface da rede de *Field Bus*, **MMS** para as funções disponibilizadas pela interface MMS-EASE e por fim **MCS** para as funções das bibliotecas do *Microsoft* C.

As funções cuja referência não tenha associada nenhuma das anotações são funções desenvolvidas para a aplicação. Na sua maioría, e dependendo da sua relevância, elas aparecerão espcificadas ao longo desta secção.

Para um melhor entendimento de algumas das funções, é importante o conhecimento da composição das mensagens de aplicação da rede de *Field Bus* considerada. A descrição dessa composição é indicada em anexo D desta dissertação.

8.4.2. Função de Atendimento Multiplexado de Indicações

Função: *VOID Integration_Core (VOID)*

Descrição Funcional:

Esta função implementa o atendimento multiplexado de 'indicações' provenientes da rede de *Field Bus* e de indicações MMS provenientes dos clientes MMS. Basicamente ela vai servindo as mensagens que chegam ao controlador da rede de *Field Bus* e, em determinada situação, vai processando as primitivas de indicação MMS. A variável *Rate*, é uma variável global, do tipo inteiro, e que corresponde ao número de vezes que é processada uma 'indicação' *Field Bus* por cada chamada da função da interface MMS-EASE *comm_serve ( )*. Essa variável vai sendo recalculada dinamicamente, sendo passada à função *Recalc_Rate ( )* a variável global *MMS_Event_Count*.

A leitura do *buffer* de recepção do controlador da rede de *Field Bus* é feita recorrendo a uma das quatro funções disponíveis na biblioteca de funções de interface FMSLIB.OBJ. Se esta função retornar o valor '0', quer dizer que foi lida uma mensagem sem erro. A função de leitura tem um parâmetro que corresponde a um manipulador de ficheiro, que é obtido na fase de inicialização da rede de *Field Bus*, recorrendo à função *_dos_open ( )* do *Microsoft* C.

O código da mensagem FILBUS recebida corresponde ao quarto elemento da mensagem. Três funções são chamadas, consoante o grupo a que a mensagem pertence: mensagens gerais, mensagens particulares dos módulos ou mensagens de E/S.

A função *comm_serve ( )*, como se viu, vai chamar a função *mms_ind_serv ( )*, que por sua vez chamará as funções de atendimento de indicação de cada um dos tipos de serviços MMS.

Variáveis Globais:

```
SHORT MMS_Event_Count
BOOLEAN App_End
UBYTE FIL_In_Buffer[256]
USHORT GF_Handle
```

Variáveis Locais:

```
SHORT Cnt_FIL_Rcv
SHORT Rate
UBYTE RetCode
```

Funções Chamadas:

```
SHORT reiceive_msg (USHORT, UBYTE, UBYTE*)          [FIL]
VOID comm_serv (VOID)                               [MMS]
UBYTE Recalc_Count (SHORT)
VOID FILBUS_Gen_Msg (VOID)
VOID FILBUS_Mdspec_Msg (VOID)
VOID FILBUS_IO_Msg (VOID)
```

Descrição em Pseudo-código:

1. [Inicialização]

```
2. [Ciclo principal no qual se processam indicações provenientes do FILBUS e do MMS Provider]
      Repeat thru step 8 While App_End ≠ TRUE
3. [Inicialização da variável que vai controlar as cadências de processamento]
      Cnt_FIL_Rcv ← 0
4. [Vão ser feitas Rate leituras ao FILBUS]
      Repeat thru step 6 While Cnt_FIL_Rcv < Rate
5. [Leitura de uma Mensagem FILBUS]
      RetCode ← Call receive_msg(GF_Handle, 0, &FIL_In_Buffer)
      Cnt_FIL_Rcv ← Cnt_FIL_Rcv + 1
6. [Processa ou não indicações FILBUS]
      If RetCode = 0 then
            Select case (FILBUS_In_Buffer[3])
                  Case 0 thru 63:
                        Call FIL_General_Msg ( )
                  Case 64 thru 127:
                        Call FIL_MDSpec_Msg ( )
                  Case 128 thru 255:
                        Call FIL_IO_Msg ( )
      else
            ( Situação de erro na leitura )
            Select case (RetCode)
                  Case 1:
                        ...
                  Case 80:
7. [Se existem indicações MMS a processar]
      Call comm_serve( )
8. [Vai recalcular as cadências de leitura]
      Rate ← Call Recalc_Rate (MMS_Event_Count)
9. [Acabou]
      Exit
```

8.4.3. Atendimento de Indicações MMS Read

Função: *VOID u_read_ind (MMSREQ_IND *req_ptr)*

Descrição Funcional:

Esta função será chamada automaticamente pela função *comm_serve ( )* para o processamento de uma indicação de MMS *Read*. Como se viu, a função *comm_serve ( )* é chamada na função *Integration_Core ( )*.

No caso de uma indicação de MMS *Read*, o elemento *req_info_ptr* da estrutura MMSREQ_IND aponta para uma estrutura do tipo *struct read_req_info*. O primeiro passo da função *u_read_ind (req_ptr)* consiste em obter um apontador para essa estrutura. Essa estrutura contem informação àcerca do acesso à variável MMS e, normalmente, não necessita de ser

consultada se se utilizar uma resposta virtual ao pedido de leitura MMS. No caso da implementação deste dispositivo de interligação, a consulta a essa estrutura é no entanto fundamental, mesmo que se utilize a interface virtual do MMS-EASE. Na realidade, é necessário saber qual é a variável que pretende ser acedida, uma vez que essa informação permitirá, caso seja necessário, fazer o adequado mapeamento vertical em serviços da rede de *Field Bus*.

A estrutura *struct read_req_info* tem o seguinte conteúdo:

```
struct read_req_info
{
        BOOLEAN         spec_in_result;
        struct var_acc_spec     va_spec;
        struct  variable_list  var_list[va_spec.num_of_variables];
};
```

Por sua vez a estrutura *struct var_acc_spec* corresponde a:

```
struct var_acc_spec
{
        SHORT   var_acc_tag;
        struct  object_name     vl_name;
        SHORT   num_of_variables;
        struct variable_list    var_list[num_of_variables];
};
```

A estrutura *struct variable_list* tem o seguinte conteúdo:

```
struct variable_list
{
        struct variable_spec   var_spec;
        UBYTE   alt_access_pres;
        struct alternate_access alt_acess;
};
```

A estrutura *struct object_name* tem o seguinte conteúdo:

```
struct object_name
{
        SHORT   object_tag:
        union
        {
                BYTE    vmd_spec[MAX_IDENT_LEN + 1];
                BYTE    item_id[MAX_IDENT_LEN + 1];
                BYTE    aa_spec[MAX_IDENT_LEN + 1];
        } obj_name;
        BYTE    domanin_id[MAX_IDENT_LEN + 1];
};
```

Por fim, a estrutura *struct variable_spec* tem o seguinte conteúdo:

```
struct variable_spec
{
        SHORT   var_spec_tag;
        union
        {
                struct object_name              name;
                struct var_acc_addr             address;
                struct variable_descr           var_descr;
                struct scattered_access         sa_descr;
        } vs;
};
```

Utilizando a especificação de acesso presente na estrutura *struct read_req_info*, é possível obter toda a informação necessária para o acesso à variável. O MMS-EASE fornece no entanto um conjunto de funções de suporte (classe 11) que permitem resultados semelhantes.

Assim, a função *ms_extract_varname (info_ptr, opcode, index)* retorna um apontador para uma estrutura do tipo *struct object_name* que corresponde ao nome da variável nomeada. Nos parâmetros passados a essa função, *info_ptr* é um apontador para uma estrutura apontada pelo elemento *req_info_ptr* da estrutura MMSREQ_IND, *opcode* corresponde ao código de uma das operações *Write*, *Read* ou *InformationReport* e por fim, *index* corresponde ao índice da variável, no caso de ser especificado o acesso a uma lista de variáveis. Esta última situação não é suportada pelo dispositivo de interligação, a não ser em situções de resposta síncrona ao pedido de leitura, utilizando a interface virtual.

A função MMS-EASE de suporte *ms_find_named_var_obj(vname, chan)* permite obter a estrutura do tipo *struct named_var* correspondente à variável especificada no acesso. O parâmetro *vname* corresponde a um apontador para uma estrutura do tipo *struct object_name*. Existe uma função semelhante, a *ms_find_named_var (dom, vname)*, na qual *dom* é um apontador para uma estrutura do tipo *struct domain_objs* e *vname* é um apontador para BYTE correspondente ao nome da variável a encontrar.

No caso da variável não ser encontrada, a função *ms_find_named_var_obj(vname, chan)* retorna um apontador para NULL. Nesse caso pode ser efectuada de imediato a resposta a esse pedido de leitura MMS, utilizando para tal uma função da classe 6 do MMS-EASE: *mv_read_resp (req_ptr)*. Esta função encarrega-se de responder negativamente ao cliente MMS, como já foi referido quando se abordaram as características da interface VMI do MMS-EASE.

Caso a variável exista, é ainda conveniente verificar o tipo de acesso que deverá ser efectuado sobre essa variável. São disso exemplo as variáveis *SUBSCR_MD* ou *COMAND_MD*. A aceitação da leitura destas variáveis não é crítica em termos de funcionamento do dispositivo de interligação. Simplesmente o valor obtido não terá significado nenhum para o cliente MMS.

Sendo assim, nesse caso será chamada a função, já referida, da classe 4 do MMS_EASE *mp_err_resp (req_ptr, class, code)*. Neste caso, o código a utilizar para *class* deverá ser o código 7, especificando problemas de acesso a variáveis, e o parâmetro *code* deverá ter o valor 3, especificando negação de acesso a um objecto.

Após esta primeira verificação obtém-se um apontador para uma estrutura do tipo *struct named_dom_ctrl*, utilizando para tal a função de suporte do MMS-EASE *ms_find_named_dom (dname)*. O parâmetro *dname* deverá ser um apontador para BYTE, início do nome do domínio em causa.

O MMS-EASE fornece uma série de funções de suporte que permitem ao utilizador obter as estruturas de controlo dos diferentes objectos MMS-EASE havendo por isso bastantes opções quanto a esta matéria. Neste caso específico, o nome do domínio está disponível na estrutura do tipo *struct object_name* que foi obtida pela utilização prévia da função *ms_extract_varname (, ,)*.

Se se tratar de uma variável que possa ser lida sincronamente, isto é *IOGLOI_MD*, *IOGLOO_MD* ou *IOINDO_xx* e obviamente o módulo esteja a funcionar com evoluções automáticas das variáveis do tipo digital (campo *evo_status* da estrutura de controlo do módulo *struct module_ctrl*) então poderá se enviado uma resposta virtual, utilizando a função *mv_read_resp (req_ptr)*.

No caso de se tratar de uma variável *IOINDI_xx*, o que foi dito no parágrafo anterior também será válido se se tratar de uma variável de entrada puramente digital. Para tal é preciso verificar se o elemento do vector que constitui o elemento *io_type* da estrutura *struct module_ctrl* corresponde ou não a uma entrada do tipo digital.

Finalmente se não se tratar de nenhuma das situações anteriores, então deverá ser verificado se essa indicação está ou não a ser pela primeira vez processada. Esse estado é indicado pelo elemento *aux_flags[0]* do elemento *s* da estrutura MMSREQ_IND. Caso seja a primeira vez, é mudado o estado de *aux_flags[0]*, é escrito o apontador para função *serv_fun* de *s* com a função *TimeOut_Verify*, e é chamada a função *Updt_Before_Read_Resp (struct GTW_VarDom)*. Esta função, que será descrita posteriormente nesta secção, deverá proceder aos mapeamentos verticais necessários para o acesso às diferentes variáveis. É responsável pela construção das mensagens de pedido a enviar à rede de *Field Bus* e pelo envio dessas mensagens, e é responsável pela determinação dos elementos de controlo de sincronização das respostas MMS que deverão ser colocados a zero.

A função *Updt_Before_Read_Resp (struct GTW_VarDom)* retorna um apontador para BYTE que aponta para o elemento da tabela de sincronização que deverá ser testado em segunda etapa de atendimento da indicação. Este procedimento evitará processamentos posteriores no sentido de o obter outra vez, uma vez que passa a estar disponível na própria estrutura MMSREQ_IND, no elemento *cmd_info* da estrutura *s*.

Se a indicação estiver em segunda etapa de atendimento, então deverá ser verificado se existe ou não erro de *timeout*. Como já foi referido, em primeira etapa de atendimento é escrito o apontador para função *(* serv_fun)( )* da estrutura do tipo *struct cmd_service* que corresponde ao elemento *s* da estrutura MMSREQ_IND. Sempre que for chamada a função *u_read_ind (req_ptr)* para uma determinada indicação que já foi previamente atendida, será chamada a função *TimeOut_Verify*. A esta função é passada a estrutura MMSREQ_IND. Como foi visto, esta

estrutura contém a informação temporal referente à altura em que foi recebida a indicação. A função *TimeOut_Verify* irá fazer uma comparação entre esse valor e o valor actual de tempo. Uma variável global do tipo boleano será alterada em conformidade.

Caso exista uma situação de *timeout* é chamada a função do MMS-EASE *mp_err_resp (, ,)* indicando classe de serviço 5 (*service preempt problems*) e código 1 (*Timeout*). Caso não exista *timeout*, é verificado se a variável de controlo vale ou não '1'. Se sim é chamada a função *mv_read_resp( )*. Obviamente que em situações de erro de *timeout*, as variáveis de controlo deverão associadas ao pedido deverão ser colocadas a '1'.

Variáveis Globais:

```
struct module_ctrl Head_MD_List
BOOLEAN TimeOut
```

Variáveis Locais:

```
struct module_ctrl *Ap_Aux
struct read_req_info *Info_ptr
MMSREQ_IND * Req_ptr
struct named_var *Var_Name
struct named_dom_ctrl *D_Ctrl_ptr
BYTE *Var
UBYTE io_num
struct object_name Objectname
struct GTW_VarDom *VarDom_Ptr
```

Funções Chamadas:

```
struct object_name * ms_extract_varname (MMSREQ_IND *,BYTE, BYTE)        [MMS]
struct named_var *ms_find_named_var_obj (struct object_name *, BYTE)     [MMS]
VOID mp_err_resp (MMSREQ_IND *, SHORT, SHORT)                            [MMS]
VOID mv_read_resp (MMSREQ_IND *)                                         [MMS]
struct named_dom_ctrl *ms_find_named_dom (BYTE *)                        [MMS]
VOID *memcpy (VOID *, const VOID*, size_t)                               [MSC]
BYTE Calc_Ionum (BYTE *)
BYTE *Updt_VMD_Before_Read_Resp (struct GTW_VarDom *)
```

Descrição em Pseudo-código:

1. [Inicialização]

2. [Verificar se se trata de atendimento em segunda etapa]
   If Req_ptr↑.s.aux_flags[0] = CALLED_BEFORE **then**
   goto 13

3. [Obter um apontador para uma estrutura de informação de read]
   Info_ptr ← CTOP (Req_ptr↑.req_info_ptr, struct_read_req_info ↑)
   (Notação utilizada para o *cast* de uma variável *Convert Type OPeration*)
   (No segundo parâmetro é o novo tipo)

4. [Verificar se a variável que é pretendida ser lida existe ou não no VMD]
   Objectname ← **Call** ms_extract_varname (Info_ptr, READ, 0)
   (o último parâmetro indica o indice da variável a lêr numa lista de var.)
   Var_Name ← **Call** ms_find_named_var_obj(Objectname, Req_ptr↑.chan)

5. [Se não existir processa logo a resposta virtual e termina]
   If Var_Name = NULL **then**
   Call mv_read_resp(Req_ptr)
   Exit

```
6. [Se se tratarem de variáveis que não deverão ser lidas]
        If (Var_Name↑.varname = ALARMS_MD or Var_Name↑.varname = SUBSCR_MD or
                                Var_Name↑.varname = COMAND_MD) then
                Call mp_err_resp (Req_ptr, 7, 3)
                Exit
7. [Obter um apontador para uma estrutura do tipo controlo do domínio]
        D_Ctrl_ptr ← Call ms_find_named_dom(Objectname↑.domain_id)
8. [Obtem a estrutura de controlo do módulo]
        Ap_Aux ← Head_MD_List
        Repeat While Ap_Aux↑.named_dom_ctrl_ptr ≠ D_Ctrl_ptr
                Ap_Aux ← Ap_Aux↑.Next
9. [Preenche a estrutura do tipo struct MMS_Rcv_Dom_Info]
        VarDom_Ptr↑.Request_Ptr ← Req_ptr
        VarDom_Ptr↑.Var_Ptr ← Var_Name
        VarDom_Ptr↑.Mod_Ptr ← Ap_Aux
10. [Obter os 6 primeiros caracteres do nome da variável]
        Call memcpy (Var, Var_Name, 6)
11. [Se se tratar de uma variável que possa ser lida sincronamente]
        If ((Var = IOGLOI or Var = IOGLOO or Var = IOINDO)
                                and Ap_Aux↑.evo_status) ) then
                Call mv_read_resp(Req_ptr)
                Exit
12. [Se se tratar de um entrada é preciso verificar se ela é puramente digital]
        If (Var = IOINDI and Ap_Aux↑.evo_status) then
                io_num ← Call Calc_Ionum (Var_Name↑.varname)
                If Ap_Aux↑.io_type[io_num] = DIGIT then
                        Call mv_read_resp(Req_ptr)
                        Exit
13. [Verifica se está em primeira etapa ou não]
        If Req_ptr↑.s.aux_flags[0] ≠ CALLED_BEFORE then
                Req_ptr↑.s.serv_fun ← TimeOut_Verify
                Req_ptr↑.s.aux_flags[0] ← CALLED_BEFORE
                Req_ptr↑.s.cmd_info ← Call Updt_VMD_Before_Read_Resp(VarDom_Ptr)
        else
                (Indicação em Segunda Etapa)
                If TimeOut = TRUE then
                        (coloca byte da tabela de controlo a '1')
                        (e os outros de situações múltiplas ?)
                        ↑(Req_ptr↑.s.cmd_info) ← UPDATED
                        (erro class = 5 service preempt problems, code = 1 Timeout)
                        Call mp_err_resp (Req_ptr, 5, 1)
                else
                        If ↑(Req_ptr↑.s.cmd_info) = TRUE then
                                Call mv_read_resp(Req_ptr)
14. [Finaliza]
        Return (VOID)
```

Função: *BYTE *Updt_VMD_Before_Read_Resp(struct GTW_VarDom *VarDom_Ptr)*

Descrição Funcional:

Esta função vai proceder ao envio dos pedidos de serviço à rede de *Field Bus*, para o caso dos mapeamentos assíncronos das indicações MMS *Read*. Primeiro, vai ser obtido o endereço do controlador. O endereço do controlador vai ser obtido a partir da lista de estruturas do tipo *struct module_ctrl* apontadas pela variável global *Head_MD_List*. Relembre-se que esta lista de estruturas é criada aquando do arranque do dispositivo de interligação.

Para cada um dos casos de variável, é construída a mensagem a enviar, eventualmente alterados os elementos da tabela de controlo e determinado o elemento que servirá para controlar o envio da resposta MMS ao pedido de leitura.

Cada um dos casos é implementado de acordo com as redes de Petri estipuladas no final do capítulo 7.

De notar, por exemplo, que no caso das variáveis que podem ser escritas e lidas, como são o caso das variáveis cujos nomes são *IOGLOO_MD* e *IOINDO_xx*, poderá haver passagem ao estado de indicação em primeira etapa, se hover situações de escrita concorrente. Nesse caso é alterada *aux_flags[0]* da estrutura *struct cmd_service*.

Variáveis Globais:

```
struct module_ctrl Head_MD_List
UBYTE FIL_Out_Buffer[256]
USHORT GF_Handle
```

Variáveis Locais:

```
struct module_ctrl *Ap_Aux
struct GTW_VarDom *VarDom_Ptr
BYTE *Var
UBYTE io_num
BYTE *Tab_Ctrl
BOOLEAN sent
UBYTE idx
```

Funções Chamadas:

```
VOID *memcpy (VOID *, const VOID*, size_t)                    [MSC]
BYTE Calc_Ionum (BYTE *)
SHORT send_msg (USHORT, UBYTE, UBYTE*, USHORT)                [FIL]
```

Descrição em Pseudo-código:

1. [Inicialização]

2. [Obter apontador para a variável de controlo do módulo do controlador]
   Ap_Aux ← Head_MD_List
   Repeat While Ap_Aux↑.module_type ≠ GESFIL_PC
      Ap_Aux ← Ap_Aux↑.Next

3. [Constroi parte Inicial da Mensagem a Enviar]
   FIL_Out_Buffer[0] ← 0

```
        (endereço do destinatário)
        FIL_Out_Buffer[1] ← VarDom_Ptr↑.Mod_Ptr↑.address
        (endereço do controlador)
        FIL_Out_Buffer[2] ← Ap_Aux↑.address
4. [Obter os 6 primeiros caracteres do nome da variável]
        Call memcpy (Var, VarDom_ptr↑.Var_Ptr↑.varname, 6)
5. [Processar para cada um dos casos]
        Select Case (↑Var)
          Case PARIOS:
                io_num ← Call Calc_Ionum (VarDom_ptr↑.Var_Ptr↑.varname)
                Tab_Ctrl ← &(VarDom_Ptr↑.Mod_Ptr↑.par_updt_rd[io_num])
                (caso esteja um pedido em decurso não é enviada a mensagem)
                FIL_Out_Buffer[3] ← DPA_DF
                FIL_Out_Buffer[4] ← 1
                FIL_Out_Buffer[5] ← io_num
                FIL_Out_Buffer[6] ← '/0'
                If ↑Tab_Ctrl = TRUE then
                        VarDom_Ptr↑.Mod_Ptr↑.par_updt_rd[io_num] ← FALSE
                        Call send_msg (GF_Handle, 0, FIL_Out_Buffer, 2)
          Case IOGLOI:
                Tab_Ctrl ← &(VarDom_Ptr↑.Mod_Ptr↑.ioglob_updt_rd)
                (caso esteja um pedido em decurso não é enviada a mensagem)
                If VarDom_Ptr↑.Mod_Ptr↑.ioglob_updt_rd = TRUE then
                        FIL_Out_Buffer[3] ← DAL_60
                        FIL_Out_Buffer[4] ← 0
                        FIL_Out_Buffer[5] ← '/0'
                        VarDom_Ptr↑.Mod_Ptr↑.ioglob_updt_rd ← FALSE
                        Repeat for Idx = 1 to 16
                                If VarDom_Ptr↑.Mod_Ptr↑.io_type[Idx] = DIGIT then
                                        VarDom_Ptr↑.Mod_Ptr↑.ioindi_updt_rd [Idx] ← FALSE
                        Call send_msg (GF_Handle, 0, FIL_Out_Buffer, 2)
          Case IOGLOO:
                sent ← FALSE
                Repeat for Idx = 1 to 16
                        If VarDom_Ptr↑.Mod_Ptr↑.ioindi_updt_wr[Idx] = FALSE then
                                sent ← TRUE
                        If VarDom_Ptr↑.Mod_Ptr↑.ioglob_updt_wr = FALSE then
                                sent ← TRUE
                (verificar se está a haver uma escrita em simultâneo passa a máquina de )
                (estados para primeira etapa outra vez)
                If sent = TRUE then
                        VarDom_Ptr↑.Request_Ptr↑.s.aux_flags[0] ← NOT_CALLED_BEFORE
                        Exit

                Tab_Ctrl ← &(VarDom_Ptr↑.Mod_Ptr↑.ioglob_updt_rd)
                (caso esteja um pedido em decurso não é enviada a mensagem)
                If VarDom_Ptr↑.Mod_Ptr↑.ioglob_updt_rd = TRUE then
                        FIL_Out_Buffer[3] ← DRY_66
                        FIL_Out_Buffer[4] ← 0
                        FIL_Out_Buffer[5] ← '/0'
```

```
                VarDom_Ptr↑.Mod_Ptr↑.ioglob_updt_rd ← FALSE
                Repeat for Idx = 1 to 16
                        If VarDom_Ptr↑.Mod_Ptr↑.io_type[Idx] = ACTIVE then
                                VarDom_Ptr↑.Mod_Ptr↑.ioindi_updt_rd [Idx] ← FALSE
                Call send_msg (GF_Handle, 0, FIL_Out_Buffer, 2)

        Case IOINDI:
                io_num ← Call Calc_Ionum (VarDom_ptr↑.Var_Ptr↑.varname)
                Tab_Ctrl ← &(VarDom_Ptr↑.Mod_Ptr↑.ioindi_updt_rd[io_num])
                If VarDom_Ptr↑.Mod_Ptr↑.io_type[io_num] = DIGIT then
                        (caso esteja um pedido em decurso não é enviada a mensagem)
                        If (VarDom_Ptr↑.Mod_Ptr↑.ioglob_updt_rd = TRUE
                            and VarDom_Ptr↑.Mod_Ptr↑.ioindi_updt_rd[io_num] = TRUE) then
                                FIL_Out_Buffer[3] ← DAL_60
                                FIL_Out_Buffer[4] ← 1
                                FIL_Out_Buffer[5] ← io_num
                                FIL_Out_Buffer[6] ← '/0'
                                VarDom_Ptr↑.Mod_Ptr↑.ioindi_updt_rd[io_num] ← FALSE
                                Call send_msg (GF_Handle, 0, FIL_Out_Buffer, 2)
                If VarDom_Ptr↑.Mod_Ptr↑.io_type[io_num] = CONT then
                        (caso esteja um pedido em decurso não é enviada a mensagem)
                        If (VarDom_Ptr↑.Mod_Ptr↑.ioindi_updt_rd[io_num] = TRUE) then
                                FIL_Out_Buffer[3] ← DCP_62
                                FIL_Out_Buffer[4] ← 1
                                FIL_Out_Buffer[5] ← io_num
                                FIL_Out_Buffer[6] ← '/0'
                                VarDom_Ptr↑.Mod_Ptr↑.ioindi_updt_rd[io_num] ← FALSE
                        (no caso de contador pode enviar sempre o DCP_62)
                        Call send_msg (GF_Handle, 0, FIL_Out_Buffer, 2)

        Case IOINDO:
                io_num ← Call Calc_Ionum (VarDom_ptr↑.Var_Ptr↑.varname)
                Tab_Ctrl ← &(VarDom_Ptr↑.Mod_Ptr↑.ioindi_updt_rd[io_num])
                (verificar se está a haver uma escrita em simultâneo passa a máquina de )
                (estados para primeira etapa outra vez)
                If VarDom_Ptr↑.Mod_Ptr↑.ioindi_updt_wr[io_num] = FALSE and
                                VarDom_Ptr↑.Mod_Ptr↑.ioglob_updt_wr = FALSE then
                                sent ← TRUE
                VarDom_Ptr↑.Request_Ptr↑.s.aux_flags[0] ← NOT_CALLED_BEFORE
                Exit

                (caso esteja um pedido em decurso não é enviada a mensagem)
                If (VarDom_Ptr↑.Mod_Ptr↑.ioglob_updt_rd = TRUE
                        and VarDom_Ptr↑.Mod_Ptr↑.ioindi_updt_rd[io_num] = TRUE) then
                FIL_Out_Buffer[3] ← DRY_66
                FIL_Out_Buffer[4] ← 1
                FIL_Out_Buffer[5] ← io_num
                FIL_Out_Buffer[6] ← '/0'
                VarDom_Ptr↑.Mod_Ptr↑.ioindi_updt_rd[io_num] ← FALSE
                Call send_msg (GF_Handle, 0, FIL_Out_Buffer, 2)
```

6. [Finaliza]

Return (Tab_Ctrl)

8.4.4. Atendimento de Indicações MMS de Write

Função: *VOID u_write_ind(MMSREQ_IND *req_ptr)*

Descrição Funcional:

Esta função será chamada automaticamente pela função *comm_serve ( )* para o processamento de uma indicação de MMS *Write*. Como se viu, a função *comm_serve ( )* é chamada na função *Integration_Core ( )*.

Salvo algumas excepções, o processamento inicial da função é semelhante ao da função de atendimento de indicações MMS *Read*.

No caso do acesso de escrita às variáveis *COMAND_MD* ou *SUBSCR_MD*, a indicação MMS deverá ser processada em modo síncrono. Antes porém, deverão ser chamadas respectivamente as funções *Send_Filbus_Command ( )* e *Subscribe_Filbus_Info ( )*. A primeira corresponderá a enviar comandos de programação dos módulos como foi referênciado no capítulo 7. A segunda deverá proceder ao *setting* do elemento específico da estrutura *struct module_ctrl* (elementos *digit_subscr*, *alarms_subscr* ou *counter_subscr*), tal como especificado no capítulo 7.

Para atendimento de indicações MMS em modo assíncrono, será chamada na primeira etapa a função *Send_Before_Write_Resp ( )*, que será responsável pelos mapeamentos verticais, pela determinação da variável de controlo a utilizar na sincronização da resposta MMS e pelo *reset* das variáveis de controlo associadas ao mapeamento vertical, conforme foi dito na parte final do capítulo 7.

Em segunda etapa, deverá ser verificada a variável de controlo de sincronização da resposta MMS, mas também verificar se foi ou não efectuada a escrita. Basicamente, isso corresponderá a fazer a comparação das duas variáveis a presente no VMD, e que resultou de um refrescamento, tipicamente por um pedido de leitura. A excepção para esta função está na escrita de parâmetros, uma vez que neste caso será utilizada a recepção de um *ACK_91* para sincronizar a resposta MMS. A função *Verify_Equality ( )* retornará para o caso de PARIOS o valor *TRUE*.

Variáveis Globais:

```
struct module_ctrl Head_MD_List
BOOLEAN TimeOut
```

Variáveis Locais:

```
struct module_ctrl *Ap_Aux
struct read_req_info *Info_ptr
```

```
MMSREQ_IND * Req_ptr
struct named_var *Var_Name
struct named_dom_ctrl *D_Ctrl_ptr
BYTE *Var
UBYTE io_num
struct object_name Objectname
struct GTW_VarDom *VarDom_Ptr
```

Funções Chamadas:

```
struct object_name * ms_extract_varname (MMSREQ_IND *,BYTE, BYTE)          [MMS]
struct named_var *ms_find_named_var_obj (struct object_name *, BYTE)       [MMS]
VOID mp_err_resp (MMSREQ_IND *, SHORT, SHORT)                              [MMS]
VOID mv_write_resp (MMSREQ_IND *)                                          [MMS]
struct named_dom_ctrl *ms_find_named_dom (BYTE *)                          [MMS]
VOID *memcpy (VOID *, const VOID*, size_t)                                 [MSC]
UBYTE *strstr (const BYTE *, const BYTE *)                                 [MSC]
VOID Send_Filbus_Command (struct GTW_VarDom *)
VOID Subscribe_Filbus_Info (struct GTW_VarDom *)
BYTE *Send_Before_Write_Resp (struct GTW_VarDom *)
BOOLEAN Verify_Equality (struct GTW_VarDom *)
```

Descrição em Pseudo-código:

1. [Inicialização]

2. [Verificar se se trata de atendimento em segunda etapa]
   If Req_ptr↑.s.aux_flags[0] = CALLED_BEFORE **then**
   goto 12
3. [Obter um apontador para uma estrutura de informação de Write]
   Info_ptr ← CTOP (Req_ptr↑.req_info_ptr, struct write_req_info ↑)
   (Notação utilizada para o *cast* de uma variável *Convert Type OPeration*)
   (No segundo parâmetro é o novo tipo)
4. [Verificar se a variável que é pretendida ser escrita existe ou não no VMD]
   Objectname ← **Call** ms_extract_varname (Info_ptr, WRITE, 0)
   (o último parâmetro indica o indice da variável a lêr numa lista de var.)
   Var_Name ← **Call** ms_find_named_var(Objectname, Req_ptr↑.chan)
5. [Se não existir processa logo a resposta virtual e termina]
   If Var_Name = NULL **then**
   Call mv_write_resp(Req_ptr)
   Exit
6. [Se se tratarem de variáveis que não devem ser escritas]
   If (Var_Name↑.varname = ALARMS_MD **or** Var_Name↑.varname = IOGLOI_MD **or**
   (**Call** strstr (Var_Name↑.varname, "IOINDI"))) **then**
   Call mp_err_resp (Req_ptr, 7, 3)
   Exit
7. [Obter um apontador para uma estrutura do tipo controlo do domínio]
   D_Ctrl_ptr ← **Call** ms_find_named_dom(Objectname↑.domain_id)
8. [Obtem a estrutura de controlo do módulo]
   Ap_Aux ← Head_MD_List
   Repeat While Ap_Aux↑.named_dom_ctrl_ptr ≠ D_Ctrl_ptr
   Ap_Aux ← Ap_Aux↑.Next
9. [Preenche a estrura do tipo struct MMS_Rcv_Dom_Info]
   VarDom_Ptr↑.Request_Ptr ← Req_ptr
   VarDom_Ptr↑.Var_Ptr ← Var_Name
   VarDom_Ptr↑.Mod_Ptr ← Ap_Aux

```
10. [Obter os 6 primeiros caracteres do nome da variável]
        Call memcpy (Var, Var_Name, 6)
11. [Verificar se algum deles pode ser processado em modo síncrono]
        If (Var = COMAND or Var = SUBSCR) then
                Select Case (Var)
                        Case COMAND:
                                Call Send_Filbus_Command (VarDom_Ptr)
                        Case COMAND:
                                Call Subscribe_Filbus_Info (VarDom_Ptr)
                Call mv_write_resp(Req_ptr)
                Exit
12. [Verifica se está em primeira etapa ou não]
        If Req_ptr↑.s.aux_flags[0] ≠ CALLED_BEFORE then
                Req_ptr↑.s.serv_fun ← TimeOut_Verify
                Req_ptr↑.s.aux_flags[0] ← CALLED_BEFORE
                Req_ptr↑.s.cmd_info ← Call Send_Before_Write_Resp(VarDom_Ptr)
        else
                (Indicação em Segunda Etapa)
                If TimeOut = TRUE then
                        (coloca byte da tabela de controlo a '1')
                        (e os outros de situações múltiplas ?)
                        ↑(Req_ptr↑.s.cmd_info) ← UPDATED
                        (erro class = 5 service preempt problems, code = 1 Timeout)
                        Call mp_err_resp (Req_ptr, 5, 1)
                else
                        If ↑(Req_ptr↑.s.cmd_info) = TRUE then
                                If ( ← Call Verify_Equality (VarDom_Ptr) ) = TRUE then
                                        Call mv_write_resp (req_ptr)
                                else
                                        (class = 7 access problems, code = 0 other)
                                        Call mp_err_resp (Req_ptr, 7, 0)
13. [Finaliza]
        Return (VOID)
```

Função: *BYTE *Send_Before_Write_Resp(struct GTW_VarDom *VarDom_Ptr)*

Descrição Funcional:

Esta função vai proceder ao envio dos comandos à rede de *Field Bus*, para o caso dos mapeamentos assíncronos das indicações MMS *Write*.

Cada um dos casos a tratar, isto é, *PARIOS_xx*, *IOGLOO_MD* e *IOINDO_xx* deverá sofrer um processamento conforme indicado nas redes de Petri correspondentes ao processamento de indicações de escrita em modo assíncrono, conforme indicado no capítulo 7. Nesta descrição em pseudo-código está exemplificada a situação correspondente à escrita da variável *PARIOS_xx*.

A função *u_get_named_address ( )* é uma função que retorna um apontador para a localização em memória da variável em causa. Esta função tem, no entanto, de ser definida pelo utilizador. No caso concreto deste dispositivo de interligação ela simplesmente retornará o elemento apropriado da estrutura *var_acc_addr* que é apontado por *addr*. Veja-se que *addr* é por sua vez um elemento da estrutura *struct named_var*.

A função *ms_extract_write_data (req_ptr, index, type, dest, alt_acc, alt_acc_packed)* é uma função de suporte do MMS-EASE que permite obter em *dest* o valor da variável, a parir da sua representação ASN.1. No caso da implementação de baixo, o endereço de destino é o endereço físico da variável MMS. Podería ser outro, para tal procedendo-se a uma alocação prévia de memória. Assim, ao chamar a função *ms_extract_write_data (, , , , ,)* procede-se à alteração da variável do VMD. Em termos funcionais não trará problemas.

De qualquer das formas, é preciso obter o valor da variável por forma a poder com ela construir a mensagem a enviar à rede de *Field Bus*.

O apontador para os dados é inicialmente obtido na variável *Addr*. Para cada um dos tipos de variável a ser acedida terá de ser feita uma conversão de tipo em causa.

Variáveis Globais:

```
struct module_ctrl Head_MD_List
UBYTE FIL_Out_Buffer[256]
USHORT GF_Handle
```

Variáveis Locais:

```
struct module_ctrl *Ap_Aux
struct GTW_VarDom *VarDom_Ptr
BYTE *Var
UBYTE io_num
BYTE *Addr
BYTE *Tab_Ctrl
parios_xx_c_type *Parios_Ptr
ioouti_xx_c_type *Ioouti_Ptr
ioglob_md_c_type *Ioglob_Ptr
UBYTE idx
```

Funções Chamadas:

```
SHORT ms_extract_write_data(MMSREQ_IND *,SHORT,struct named_type,BYTE *,SHORT)     [MMS]
BYTE *u_get_named_addr (struct named_var *)                                        [MMS]
VOID *memcpy (VOID *, const VOID*, size_t)                                         [MSC]
BYTE Calc_Ionum (BYTE *)
SHORT send_msg (USHORT, UBYTE, UBYTE*, USHORT)                                     [FIL]
```

Descrição em Pseudo-código:

1. [Inicialização]

2. [Obter apontador para a variável de controlo do módulo do controlador]
Ap_Aux ← Head_MD_List
Repeat While Ap_Aux↑.module_type ≠ GESFIL_PC
Ap_Aux ← Ap_Aux↑.Next

3. [Constroi parte Inicial da Mensagem a Enviar]

```
        FIL_Out_Buffer[0] ← 0
        (endereço do destinatário)
        FIL_Out_Buffer[1] ← VarDom_Ptr↑.Mod_Ptr↑.address
        (endereço do controlador)
        FIL_Out_Buffer[2] ← Ap_Aux↑.address
4. [Obter o endereço físico da variável local]
        Addr ← Call u_get_named_address(VarDom_Ptr↑.Var_Ptr)

5. [Obter os 6 primeiros caracteres do nome da variável]
        Call memcpy (Var, VarDom_ptr↑.Var_Ptr↑.varname, 6)
6. [Processar para cada um dos casos]
        Select Case (Var)

          Case PARIOS:
                io_num ← Call Calc_Ionum (VarDom_ptr↑.Var_Ptr↑.varname)
                (verifica se está uma escrita em curso)
                If VarDom_Ptr↑.Mod_Ptr↑.par_updt_wr[io_num] = FALSE then
                        VarDom_Ptr↑.Request_Ptr↑.s.aux_flags[0] ← NOT_CALLED_BEFORE
                        Exit

                Tab_Ctrl ← &(VarDom_Ptr↑.Mod_Ptr↑.par_updt_wr[io_num])

                FIL_Out_Buffer[3] ← PAR_81
                FIL_Out_Buffer[4] ← 250

                Parios_Ptr ← CTOP (Addr, parios_xx_c_type)
                (Notação utilizada para o cast de uma variável Convert Type OPeration)
                (No segundo parâmetro é o novo tipo)

                (obtem-se os dados veiculados na escrita)
                Call ms_extract_write_data (VarDom_Ptr↑.Request_Ptr, 0,
                        VarDom_Ptr↑.Var_Ptr↑.type, Parios_Ptr, 0, 0)

                (passa os dados para a trama FILBUS)
                Repeat for Idx = 1 to 250
                        FIL_Out_Buffer [idx+4] ← Parios_Ptr↑.par[Idx]
                FIL_Out_Buffer [idx+5] ← '/0'

                VarDom_Ptr↑.Mod_Ptr↑.par_updt_wr[io_num] ← FALSE


          Case IOGLOO:
                ...
          Case IOINDO:
                ...
7. [Finaliza]
        Return (Tab_Ctrl)
```

8.4.5. Atendimento de 'Indicações' da Rede de Field Bus

Função: *VOID FIL_IO_Msg (VOID)*

Descrição Funcional:

Esta função irá ser chamada a partir da função *Integration_Core ( )*, após a leitura de uma mensagem da rede de *Field Bus*, e sempre que se tratarem de mensagens de entrada / saída de dados, ou seja, de mensagens cujo código esteja entre os valores 80h e FFh.

Assumem particular relevo as seguintes mensagens: *ACK_91*, *ALM_B0*, *STA_B1*, *CPI_B2*, *CPT_B3*, *DIG_B9*, *PAL_C0*, *PST_C1*, *PCI_C2*, *PCT_C3*, *PRY_C5* e *RPA_CF*. Os processamentos a efectuar para cada um dos casos são descritos na secção 7.4 do capítulo 7.

Abaixo encontram-se exemplificadas as recepções das indicações *ACK_91* e *ALM_B0*. Como é fácil de verificar, para cada caso os processamentos são muito diversos, as estruturas das mensagens *Field Bus* são bastante diferentes, etc. No entanto, nestes dois casos são ilustrados procedimentos genéricos comuns a todas as situações, que vão desde o envio de MMS *InformationReport* atá à obtenção das variáveis a ser mapeadas, passando pela actualização dessas variáveis MMS com a informação veiculada na mensagem *Field Bus* e pelo *setting* de variáveis de controlo.

Variáveis Globais:

```
struct module_ctrl Head_MD_List
UBYTE FIL_In_Buffer[256]
USHORT GF_Handle
```

Variáveis Locais:

Funções Chamadas:

Descrição em Pseudo-código:

```
1. [Inicialização]

2. [Ir procurar a correspondência entre o endereço e um domínio]
       Ap_Aux ← Head_MD_List

3. [Pesquisa na lista até encontrar um módulo com o endereço igual ao da trama]
       Repeat While Ap_Aux↑.address ≠ FIL_In_Buffer[2]
           Ap_Aux ← Ap_Aux↑.Next

4. [Para cada caso de serviço]
       Select case (FIL_In_Buffer[3])
           Case ACK_91:
                   (O serviço ACK_91 é orientado ao módulo !!!)
                   Repeat for Idx = 1 to 250
                       Ap_Aux↑.par_updt_wr[FIL_In_Buffer[Idx]]

           Case ALM_B0:
```

```
If Ap_Aux↑.md_type = FILINP then
        varname ← IOINDI + "_" + str(FIL_In_Buffer[5])
else
        varname ← IOINDO + "_" + str(FIL_In_Buffer[5])
(Com o nome da variável obter um objecto MMS do tipo named_var)
VarDom_ptr↑.Var_Ptr ← Call ms_find_named_var(Ap_Aux↑.
md_domain_name↑.objs, varname)

(Obter o enderço da variável individual)
Addr ← Call u_get_named_address(VarDom_Ptr↑.Var_Ptr)
If Ap_Aux↑.md_type = FILINP then
        Data_Ptr ← CTOP (Addr, ioinpi_xx_c_type)
else
        Data_Ptr ← CTOP (Addr, ioouti_xx_c_type)

FIL_T.Year ← FIL_In_Buffer[8]
FIL_T.Month ← FIL_In_Buffer[9]
FIL_T.Day ← FIL_In_Buffer[10]
FIL_T.Hour ← FIL_In_Buffer[12]
FIL_T.Min ← FIL_In_Buffer[13]
FIL_T.Secs ← FIL_In_Buffer[14]
FIL_T.Milsec ← FIL_In_Buffer[15]
FIL_T.250_Micr ← FIL_In_Buffer[16]
T_Stamp ← Call Convert_To_Time(FIL_T)

Data_Ptr↑.time_stamp.secs ← T_Stamp↑.secs
Data_Ptr↑.time_stamp.msecs ← T_Stamp↑.msecs
If Ap_Aux↑.md_type = FILINP then
        Data_Ptr↑.input_val ← FIL_In_Buffer[7]
else
        Data_Ptr↑.output_val ← FIL_In_Buffer[7]

(vai actualizar a variável ALARMS_MD desse módulo)
VarDom_ptr↑.Var_Ptr ← Call ms_find_named_var(Ap_Aux↑.
md_domain_name↑.objs, "ALARMS_MD")

Addr ← Call u_get_named_address(VarDom_Ptr↑.Var_Ptr)
Data_Ptr ← CTOP (Addr, alarms_md_c_type)

Data_Ptr↑.mod_address ← Ap_Aux↑.address
Data_Ptr↑.mod_type ← Ap_Aux↑.md_type
Data_Ptr↑.mod_type ← Ap_Aux↑.io_type[FIL_In_Buffer[5]]
Data_Ptr↑.alarm_qual ← FIL_In_Buffer[6]
Data_Ptr↑.value ← FIL_In_Buffer[7]
Data_Ptr↑.time_stamp.secs ← T_Stamp↑.secs
Data_Ptr↑.time_stamp.msecs ← T_Stamp↑.msecs

(envia InfoReport aos subscritores)
vname.object_tag = DOM_SPEC;
Call strcpy (vname.obj_name.dom_spec, Vardom_Ptr↑.Var_Ptr↑.
        varname)
```

```
                tname.object_tag ← VMD_SPEC
                Call strcpy (tname.obj_name.vmd_spec,Vardom_Ptr
                      ↑.Var_Ptr↑.type↑.typename)

                Repeat for Idx = 0 to 7
                      If (Ap_Aux↑.alarms_subscr [FIL_In_Buffer[5]] [Idx] and
                         mms_chan_info[Idx].ctxt.chan_state == M_ACTIVE) then
                            Call mv_info (Idx, &vname, &tname, NULL)

          Case ...:

          Default:

5. [Finaliza]
      Return (VOID)
```

8.5. Referências e Bibliografia

[FRT -] *FILBUS - Le Reseau de Terrain*, GESPAC International SA FILBUS Product Description.

[GFB - 88] *FILBUS -The GESPAC's Field Bus*, GESPAC International SA, Junho 1988.

[MSCR - 90] *Microsoft C for MS OS/2 and MS-DOS - Reference*, Microsoft Corporation, 1990.

[MSCI - 90] *Microsoft C for MS OS/2 and MS-DOS - Installing and Using the Professional Develepment System*, Microsoft Corporation, 1990.

[MSCA - 90] *Microsoft C for MS OS/2 and MS-DOS - Avanced Programming Techniques*, Microsoft Corporation, 1990.

[CMPA - 91] *Computrol's LP-40 & LP-42 ISOcomm 802.3 Controller Boards - Specifications*, AEG-Modicon, Junho 1991.

[CMPB - 91] *Computrol LP-40/LP-42 Controller Installation and Operation Manual*, AEG-Modicon, Junho 1991.

[CIS - 92] *Computrol ISOcomm Software for PC-XT and PC-AT Operating Systems User Guide*, AEG-Modicon, Abril 1992.

[CPUM - 92] *FILBUS Configuration Program User's Manual V2.3*, GESPAC International SA, Abril 1992.

[EASE - 92] *Computrol MAP 3.0 MMS-EASE Software Manual - Software Release 2.2*, Modicon Industrial Automation Systems, Março 1992.

[HFSM - 92] *FILBUS Hardware and Firmware Specification Manual V1.1*, GESPAC International SA, Abril 1992.

[SSMS - 92] *FILBUS System Software for MS-DOS Systems*, V2.3, GESPAC International SA, Abril 1992.

[INDS - 94] *FILINP-16xx Data Sheet*, GESPAC International SA, 1994.

[OUDS - 94] *FILOUT-16xx Data Sheet*, GESPAC International SA, 1994.

[CDS - 94] *FILCIO-16xx Data Sheet*, GESPAC International SA, 1994.

Capítulo 9
Considerações Finais e Conclusões

O CIM preconiza a unificação, num só processo global, de todos os subsistemas de produção auxiliada por computador. Essa integração implica cooperação e interacção entre esses diferentes subsistemas. Por isso, as redes de comunicação industriais surgem como um elemento fundamental nessa integração, uma vez que permitem fazer face a requisitos de transferência de informação cada vez mais exigentes.

As actividades em ambiente de fabrico são racionalizadas em diversos níveis hierarquicos. No que se refere à transferência de dados, cada um desses níveis apresenta requisitos diversos, levando geralmente à consideração de diferentes tipos de redes de comunicação industriais, uma vez que, não é tecnicamente possível abranger tal diversidade recorrendo a um só tipo.

No ambiente industrial é usual a consideração de três grandes tipos de redes de comunicação industriais: as redes de fábrica, as redes de célula e as redes de campo.

Esta classificação coincide sensivelmente com os três níveis inferiores da hierarquia CIM. No entanto, as valências das diferentes propostas comerciais de redes de comunicação industriais apresentam sobreposições importantes, e muitas vezes, características técnicas que lhes permitem abranger mais do que um nível da hierarquia CIM. Por outro lado, a própria organização das actividades de fabrico não pode ser estratificada de uma forma tão rígida.

Mais do que qualquer outro, o ambiente de fabrico é caracterizado por uma grande heterogeneidade no equipamento utilizado, quer em termos de proveniência quer em termos de gerações. Assim, é fundamental a existência de normas internacionais para redes de comunicação industriais. Se para os níveis superiores da hierarquia das comunicações a normalização está mais ou menos estabelecida, para os níveis inferiores diversos factores foram, e continuam a ser, impeditivos no atingir dos compromissos necessários ao estabelecimento de normas. Este facto

tem aberto espaço à proliferação de soluções proprietárias e às normas nacionais, em particular no domínio das redes do tipo *Field Bus*.

Sendo assim, o problema da integração e interligação de redes do tipo *Field Bus* no ambiente CIM tem de ser equacionado para cada caso específico. Por um lado apresentam funcionalidades e serviços diversos, por outro lado, como abrangem diferentes níveis de actividades da hierarquia de fabrico poderão ser interligadas com outras infraestruturas de comunicações a diferentes níveis na hierarquia de comunicações.

Apesar da heterogeneidade quer das redes de *Field Bus* quer dos potenciais "interlocutores", um elemento de comunicações assume um especial relevo quando se perpectiva a integração e interligação de redes de *Field Bus* no ambiente CIM. Esse elemento de comunicações é o MMS, uma vez que surge como protocolo e conjunto de serviços da camada de aplicação, quer nas redes especialmente vocacionadas para o nível fábrica, como o MAP, quer em outras arquitecturas mais vocacionadas para os níveis intermédios, como o FAIS, o PROFIBUS ou o FIP. Este é um aspecto tão mais relevante quanto, e dadas as diferenças nas arquitecturas, a interligação de redes de *Field Bus* deverá ser equacionada ao nível dos serviços da camada de aplicação.

Assim, a integração de uma rede do tipo *Field Bus* sairá facilitada se esta suportar MMS na sua camada de aplicação. No entanto, nas redes do tipo *Field Bus* francamente vocacionadas para os sensores/actuadores, isso não acontece. A rede de *Field Bus* que foi considerada neste estudo é disso um exemplo, sendo os seus serviços da camada de aplicação de concepção completamente díspar da do MMS.

A arquitectura do dispositivo de interligação proposto neste estudo, corresponde à de um servidor MMS, estruturado em termos de objectos abstractos MMS. Um dos modelos mais importantes do MMS diz respeito ao modelo de dispositivo virtual de fabrico (VMD). A modelização, no servidor de MMS, da rede de *Field Bus*, poderia ter sido feita considerando um VMD por cada módulo da rede de *Field Bus*, o que, dado o conceito de VMD e as características do módulos da rede de *Field Bus*, se afiguraría como sendo o mais natural.

No estudo que foi feito do MMS, e em particular do seu mapeamento no mundo OSI, pretendeu-se realçar o facto de que o acesso a objectos (por exemplo variáveis MMS) de diferentes VMDs exigiría pelo menos um endereço de apresentação por cada VMD. Dadas as características da rede de *Field Bus*, as associações de aplicação estabelecidas com as entidades de aplicação através desses endereços de apresentação teríam de ter carácter permanente, uma vez que não se afiguraría adequada uma gestão dinâmica dessas associações. Como também se viu, o número de associações activas poderia resultar ainda maior, uma vez que a informação de cada módulo poderá potencialmente ser utilizada por mais do que um cliente MMS.

As desvantagens deste modelo resultam essencialmente do número excessivo de canais OSI (um por cada associação de aplicação) que é necessário suportar. Se o *hardware* de comunicações poderá não ser limitativo, no caso do dispositivo de interligação a carta de comunicações suporta até 64 canais, geramente é-o a interface MMS, por razões óbvias que têm a haver com a alocação de recursos, por exemplo de memória. Por outro lado, sería exigido aos potenciais clientes MMS um número de canais que potencialmente poderíam não suportar.

Optou-se por uma solução na qual cada cliente MMS apenas precisa de suportar uma só associação de aplicação. Do lado do dispositivo de interligação existe uma entidade de aplicação, correspondente de uma forma unívoca a um VMD, sendo suportadas diversas associações de aplicação concorrentes através do endereço de apresentação dessa entidade de aplicação.

Decidiu-se por uma organização do VMD em diferentes domínios, um por cada módulo da rede de *Field Bus*. Este tipo de organização visa uma melhor racionalização dos objectos MMS de acesso a variáveis uma vez que módulos idênticos poderão ser associados a variáveis MMS de nomes iguais, uma vez que elas são específicas de domínios diferentes. Por outro lado, a organização dos módulos por domínios do VMD potencia a utilização dos serviços de gestão de domínios. Estes permitem, por exemplo, a manipulação global de toda a informação relacionada com um determinado módulo.

A estruturação e definição do conjunto das variáveis MMS utilizadas para modelizar cada um dos módulos da rede de *Field Bus* afigurou-se, essa sim, de análise bastante mais complexa, uma vez que está fortemente relacionada com opções funcionais no mapeamento vertical. Não obstante, em regra geral, a estruturação das variáveis MMS pautou-se pela maior ou menor adequação dos mapeamentos dessas variáveis nos pedidos de serviços a enviar à rede de *Field Bus*, e dos mapeamentos das 'indicações' provenientes da rede de *Field Bus* nessas mesmas variáveis MMS.

Como decorreu da análise feita no capítulo 7, dois requisitos de arquitectura permitem uma redução significativa da complexidade do dispositivo de interligação. O primeiro diz respeito à coerência do VMD com os recursos físicos que ele modeliza, ou seja, a rede de *Field Bus*. O segundo tem a haver com a confirmação remota das escritas MMS.

A rede de *Field Bus* considerada facilita, de alguma forma, a observação do primeiro dos requisitos, uma vez que os diferentes módulos podem ser programados para enviar mensagens de notificação de alterações. Estes mecanismos cobrem mesmo as alterações resultantes de actividade reflexa ao nível da rede de *Field Bus*. A esse respeito, a situação mais delicada consiste na alteração de parâmetros das E/S dos módulos como resultado de actividade reflexa. Mesmo essa situação não é crítica, uma vez que ao serem alterados os parâmetros de um módulo, este envia um serviço de notificação dessa alteração. O segundo dos requisitos implica, em certa

medida, restringir a funcionalidade do dispositivo de interligação. No entanto, e em alternativa, afigura-se razoável que seja o cliente MMS, e se o desejar, a efectuar a confirmação remota da alteração, efectuando para tal um pedido de leitura da variável que previamente escreveu.

Em termos da arquitectura do dispositivo de interligação, o cumprimento dos dois requisitos referidos permite a implementação do serviço de leitura MMS e de escrita MMS em modo síncrono, e é este o aspecto, o da utilização do MMS em modo síncrono, que torna menos complexa a implementação do dispositivo de interligação. A chegada de uma indicação de leitura de uma variável MMS é imediatamente respondida, uma vez que o seu valor está disponível no dispositivo de interligação. A chegada de um pedido de escrita MMS é também respondido síncronamente após o envio do(s) serviço(s) de escrita à rede de *Field Bus*.

A utilização do MMS em modo assíncrono implica um aumento enorme de complexidade e alguma degradação de performance, permitindo no entanto o cumprir de alguns requisitos que foi proposto respeitar: outros tipos de módulos poderão não permitir manter o VMD coerente, manutenção da quantidade de mensagens na rede de *Field Bus* abaixo dos limites de sobrecarga, não limitar as opções que a própria rede de *Field Bus* fornece, etc.

Dada a "bufferização" existente no acesso às mensagens provenientes da rede de *Field Bus*, o mapeamento dos serviços MMS em serviços remotos da rede de *Field Bus* tem de ser equacionado em termos assíncronos, na medida em que a resposta ao pedido MMS será enviada quando o pedido remoto à rede de *Field Bus* tiver sido satisfeito.

Os mecanismos de sincronização das respostas MMS tornam-se extremamente complexos uma vez que as mesmas operações MMS (de leitura ou de escrita) sobre variáveis MMS, são mapeadas numa diversidade muito grande de serviços da rede de *Field Bus*. Por outro lado, a consideração destes mapeamentos assíncronos condiciona, de alguma forma, a estruturação das próprias variáveis. Um exemplo foi dado no caso da escrita remotamente confirmada de saídas de módulos de saída.

Qualquer das duas opções para a leitura de variáveis MMS (VMD coerente ou mapeamento em serviços de consulta à rede de *Field Bus*) implica a existência dos objectos MMS de acesso a variáveis nomeadas. No caso do VMD ser coerente, só se houvesse sempre um mapeamento vertical das indicações provenientes da rede de *Field Bus* num MMS *InformationReport* para os clientes subscritores é que não sería necessário prover o VMD com os objectos MMS de acesso a variáveis, bastando nesse caso os objectos MMS correspondentes aos tipos das variáveis. No segundo caso, os objectos MMS terão de existir, uma vez que é preciso "guardar" temporariamente os valores provenientes de respostas da rede de *Field Bus*, até que seja processada realmente a resposta MMS.

A utilização da interface MMS-EASE foi considerada perante o cenário mais complexo, isto é, escritas e leituras com respostas em modo assíncrono e possibilidade de ter manipulações concorrentes (escritas e/ou leituras) à mesma variável.

A única nota de relevo na utilização desta interface diz respeito à limitação a 50 que ela impõe no número de variáveis MMS nomeadas. Esta limitação não pode ser no entanto considerada relevante. Por um lado ela não envolve aspectos de conformidade MMS. Por outro lado outras versões posteriores do MMS-EASE não são tão restritivas no número de variáveis nomeadas.

A restrição do número de variáveis MMS imposta pela interface não deve contudo ser tomada em conta na modelização do dispositivo de interligação, uma vez que soluções alternativas apresentam elas sim exigências de conformidade MMS que poderão não ser suportadas por potenciais clientes.

Uma outra limitação, não propriamente da interface MMS-EASE, mas da *stack* OSI tem a haver com o facto de nenhum dos tipos ASN.1 de representação de informação temporal (*Universal Time*, *Generalised Time*, etc) ter uma resolução inferior ao segundo. Esta rede de *Field Bus* trabalha com resoluções da ordem do quarto de milisegundo. Em alternativa a esta insuficiência, utilizou-se um mapeamento em tipos *Long Int* do ASN.1.

Em jeito de conclusão, deverá ser realçado o papel que este estudo desempenhou quer no sentido de um melhor conhecimento dos problemas que envolvem a inter-opratibilidade entre redes de comunicação industriais de tipologias diferentes, quer no sentido da validação do MMS como protocolo de comunicações da camada de aplicação especialmente vocacionado para o ambiente industrial e para a federação de diferentes subsistemas de comunicação industriais.

BIBLIOTECA F.E. PORTO

Indice de Figuras

Figura 1.1 - O Ojectivo Competitividade 4
Figura 1.2. - Célula de Fabrico Flexível 6
Figura 1.3. - Sistema de Fabrico Flexível 7
Figura 1.4 - Organização das Actividades de Fabrico 10
Figura 2.1. - Fluxos de Informação em Ambiente de Fabrico 18
Figura 2.2. - Bases de Dados Comuns 19
Figura 2.3 - Arquitectura do MR-OSI 22
Figura 2.4 - Arquitecturas MAP/TOP 24
Figura 3.1 - Modelo Hierárquico de Fabrico 28
Figura 3.2 - Hierarquia de Automação e de Comunicações 28
Figura 3.3 - Caracterização de Tráfego de Comunicações 29
Figura 3.4 - Modelo de Referência OSI e Modelo Colapsado de 3 Camadas 32
Figura 3.5 - Súmula das Principais Características de Algumas Redes de Campo 35
Figura 4.1 - Comparação das Arquitecturas de Alguns Sistemas Federadores 41
Figura 4.2 - Contexto de Utilização de Uma Bridge 45
Figura 4.3 - Contexto de Utilização de um Router 46
Figura 4.4 - Outro Contexto de Utilização de um Router 46
Figura 4.5 - Contexto de Utilização de uma Gateway 47
Figura 5.1 - Estrutura da Trama FILBUS 53
Figura 5.2 - Campo de Controlo da Trama SDLC 54
Figura 5.3 - Base de Dados Criada e Utilizada Pelo Utilitário CFG.EXE 56
Figura 5.4 - Codificação do Parâmetro Geral da Entrada 62
Figura 5.5 - Codificação dos Períodos de Actividade 63
Figura 5.6 - Codificação do Parâmetro das Mensagens Pré-definidas 64
Figura 6.1 - Elementos Constituintes da Arquitectura OSI 68
Figura 6.2 - Mapeamento e Endereçamento OSI 70
Figura 6.3 - O Modelo Cliente/Servidor do MMS 72
Figura 6.5 - Sequência Para a Descarga de um Domínio 79
Figura 6.6 - Sequência Para a Operação de Telecarga 80
Figura 6.7 - Modelização MMS 88
Figura 7.1 - Modelo de Servidor com um VMD por Módulo 96
Figura 7.2 - Modelo de Servidor com um VMD por Módulo e Mais do que uma AA por VMD 97
Figura 7.3 - Modelo de Servidor Com Um VMD e N Entidades de Aplicação 98

Figura 7.4 - Modelo de Servidor Com Um VMD e Uma Entidade de Aplicação 99
Figura 7.5 - Cenário de Leitura de Variável no Qual o VMD Está Actualizado 111
Figura 7.6 - Cenário de Leitura de Variável no Qual o VMD não Está Actualizado 112
Figura 7.7 - Cenário de Escrita de Variável sem Confirmação Remota 115
Figura 7.8 - Sequência de Transacções Referentes a um Pedido de Leitura do Estado de Uma Entrada 116
Figura 7.9 - Ex. de Cenário de MMS em Modo Assíncrono com Entrelaçamento de Mapeamentos Verticais 117
Figura 7.10 - Exemplo Restrito de Utilização de Mapeamento Assíncrono 119
Figura 7.11 - Cenário de Leitura de Parâmetro de Uma E/S de Um Módulo 122
Figura 7.12 - Rede de Petri Para a Manipulação de Variáveis MMS PARIOS_xx 128
Figura 7.13 - RdP da Leitura de Variáveis de Entrada, Síncrona ou Assíncrona Para Qualquer Modo 130
Figura 7.14 - RdP da Leitura de Variáveis de Entrada, Assíncrona no Caso de Contadores 131
Figura 7.15 - Rede de Petri Para a Leitura Remota da Variável Global 137
Figura 7.16 - Escrita Remotamente Confirmada de E/S de Saída Global 139
Figura 7.17 - Escrita Remotamente Confirmada de E/S de Saída Individual 140
Figura 7.18 - Rede de Petri Para o Atendimento de Indicações MMS Read em 1ª Etapa 150
Figura 7.19 - RdP de Processamento de Uma Indicação de MMS Read (PARIOS_xx) 151
Figura 7.20 - RdPs de Processamento de Indicações MMS Read (IOGLOI_MD) ou MMS Read (IOGLOO_MD) 151
Figura 7.21 - RdPs de Processamento de Indicações MMS Read (IOINDI_xx) 152
Figura 7.22 - RdP Para o Processamento de Uma Indicação de Leitura MMS Read (IOINDO_xx) 152
Figura 7.23 - Rede de Petri Para o Atendimento de Indicações MMS Write em 1ª Etapa 153
Figura 7.24 - RdP Para Processamento de Uma Indicação de Escrita MMS Write (PARIOS_xx) 153
Figura 7.25 - RdP Para Processamento de Uma Indicação de Escrita MMS Write (IOGLOO_MD) 154
Figura 7.26 - RdP Para Processamento de Uma Indicação de Escrita MMS Write (IOINDO_xx) 154
Figura 8.1 - Descrição Genérica do Infraestrutura de Comunicações 159
Figura 8.2 - Implementação Computrol do Modelo OSI 160
Figura 8.3 - Estrutra de Informação a Utilizar para a Estruturação do VMD 167
Figura 8.4 - RdP Referente à Multiplexação Temporal no Atendimento de Indicações 205

Índice Analítico

A

ACSE - Application Context Service Element, 69; 76

AGV - Automated Guided Veichcle, 12

AIX (ver Sistemas Operativos), 43

Alimentação no Barramento de Dados (ver Transmissão de Dados), 32

AML/2 (ver Linguagens de Programação), 43

Armazéns Automáticos, 12

ASN.1 - Abstract Syntax Notation One, 34

Assemblagem, 11

Atenuação de Sinal, 45

Automated Islands, 19

Automatização Industrial, 3; 4

Autómatos Programáveis, 40; 42; 70

B

Base de Dados

- Relacional, 40

Bases de Dados

- Informix, 41
- Ingres, 41

Bases de Dados Comuns, 19; 20

BASESTAR

- DAS, 41
- DCM, 41
- núcleo, 41

BASESTAR (ver Sistemas Federadores, 39; 40; 42

BITBUS, 30; 34; 43

Bridge, 45; 46

C

C (ver Linguagens de Programação), 43

CAD - Computer Aided Design & Drafting, 3; 8; 13; 25

CAD - Computer Aided Engineering, 3

CAD - Computer Aided Manufacturing, 3; 17; 18

CAD/CAE, 8; 28

CAE - Computer Aided Engineering, 8

CAN, 31

CAN - Controller Area Network, 30; 35

CAPP - Computer Aided Production Planning, 9; 17

CAQ - Computer Aided Quality Control, 17

CCITT - International Telegraph and Telephone Consultive Committee, 21

Célula de Fabrico, 6; 71

CGMIF - Computer Graphics Interchange Format, 25

CIM - Computer Integrated Manufacturing, 3; 7; 8; 9; 17; 18; 21; 27; 36; 39; 40; 42

Cimfast (ver Gerador de Aplicações, 40

CimView (ver Planeamento da Produção), 43

ClubFIP (ver FIP), 35

CNMA - Computer Networks for Manufacturing Applications, 20; 21; 30

Códigos de Barras, 41

Comité Internacional ISA/IEC, 31

Comunicações Industriais, 7; 14; 17-25; 27-38;

Controlo da Produção

- Controlo de Inventário, 14
- Controlo de Qualidade, 14
- Shop Floor Control, 14

Controlo de Processos

- Processos Químicos, 12
- Sensores e Actuadores, 12

Controlo de Qualidade, 5; 12; 18

- GPIB, 12
- Visão Artificial, 12

Controlo Numérico, 5; 12; 35; 40; 70

- Precisão, 5
- Repetitibilidade, 5

Correio Electrónico, 24

CSMA/CD, 30; 33; 36; 45

CSMA/DCR - Carrier Sence Multiple Access with Deterministic Collision Resolution, 35

D

DCM-Manufacturing (ver Sistemas Federadores, 39

DECFactory (ver Planeamento da Produção), 40

DECnet, 40; 42

Detecção de Erros de Transmissão, 23

Determinismo em Comunicações, 33

DFM - Design For Manufacturing, 8
Distorção de Sinal, 45

E

ECP - End Connection Pointer, 69
Efiway, 43
EMUG - European MAP User Group, 20
Engenharia da Concepção (ver CAD/CAE), 8
Esquema Conceptual Partilhado, 71
Ethernet, 30; 40; 42; 43

F

Factory Floor Automation, 17
FAIS - Factory Automation Interconnection System, 30
FDDI - Fiber Distributed Data Interface, 30
FICIM - Fieldbus Integration into CIM, 31
Field Bus, 27-38
FILBUS, 30; 34
FIP, 30; 31; 34; 36; 40; 43
    WorlFIP, 35
Flexibilidade (ver Sistemas Industriais), 5
FMS - Flexible Manufacturing Systems, 6; 7; 17
FOF - Factory Of the Future, 7
FTAM - File Transfer And Management, 24

G

Gateway, 45; 47
General Motors, 20
Gerador
    de Aplicações
        Cimfast, 40
        OS/2 Data Collection, 43
        Planteworks, 43
        Pragmakit, 43
    de Interfaces Gráficas, 41
    de Sistemas Periciais, 41
GKS INTF - Computer Graphics Metafile Interchange Format, 25
GPIB, 12

H

HART, 31
HART - Highway Addressable Remote Transducer, 30; 36
Heterogeneidade
    de Equipamento, 40
HP-Sockets, 44

I

IGES - Initial Graphics Exchange Specification, 25
Incriptação, 23
Indústria de Fabrico Discreto (ver sistemas industriais), 27
Informix (ver Bases de Dados), 41
Ingres (ver Bases de Dados), 41
Integração (ver CIM), 8; 36; 39
Inteligência Artificial
    Apoio à Decisão, 13
        Aplicações, 41
InterBUS-S, 30
ISO - International Organization for Standardization, 21; 27; 70
ISP (ver PROFIBUS), 35

J

JIT - Just In Time, 4

L

LAN - Local Area Network, 23
Linguagens de Programação
    AML/2, 43
    C, 43
    Linguagens de Quarta Geração, 43
LLC - Logical Link Control, 33

M

MAP - Manufacturing Automation Protocol, 20; 21; 23; 24; 29; 30; 40; 46; 71
Marketing, 8; 13; 14
Métodos de Controlo de Acesso ao Meio, 33
MHS - Message Handling Specification, 24
MicroVAX (ver Plataformas VAX), 42
MiniMAP, 29; 30; 31; 46
MMS - Manufacturing Mesage Specification, 67
MMS - Manufacturing Message Specification, 30; 33
    Called User, 71

Calling User, 71
Invoke ID, 71; 72
MMPM - Manufacturing Message Protocol Machine, 73
MMS-Provider, 76; 77
Modelo Cliente Servidor, 71
Modelo de Objectos, 71; 73
Alcance, 75
Classes, 74
Objectos Nomeados vs. Não Nomeados, 74
Modelo VMD, 71; 72; 73; 74
Modificadores
De Pós-execução, 72
De Pré-execução, 72
Serviço AttachToEventCondition, 73
Serviço AttachToSemaphore, 73
Objectos MMS, 72
Objecto Transacção, 72; 74
Primitivas de Serviço, 71
Receiving User, 71
Sending User, 71
Serviços
Abort, 75; 77
AttachToEventCondition, 73
Cancel, 73; 75; 77
Conclude, 75; 76
Initiate, 75; 76
Reject, 75
Utilizadores MMS, 71
MMS-Manufacturing Message Specification, 21; 23
Modelo Cliente-Servidor, 71
Modelo de Objectos, 71; 73
Modelo de Referência OSI, 20
Modelo Hierarquico do Fabrico, 27; 29
Modulink, 30
MRP I - Material Requirements Planning, 3; 9
MRP II - Manufacturing Resource Planning, 3; 9
MRP II - Manufacturing Resourse Planning, 17

N

NewWave, 44
NMT - Network Management, 21

O

ODIF - Office Document Interchange Format, 24
OpenCIM (ver Sistemas Federadores, 39
OpenCim (ver Sistemas Federadores), 44
Organização Assimétrica de Serviços de Aplicação, 71
OS/2 Data Collection (ver Gerador de Aplicações), 43
OS/400 (ver Sistemas Operativos), 43
OSI - Open Systems Interconnection, 21; 22; 31; 46; 67
ACSE - Application Context Service Element, 69; 76
AE-Invocations, 69
AP-Invocations, 69
Application Association, 69
Application Context, 69
Application Entity, 69
Application Process, 67; 69; 71
ASE - Application Service Element, 69
Camada de Aplicação, 23; 33
Serviços de Refrecamento de Bases de Dados, 33
Camada de Apresentação, 23
Camada de Ligação de Dados, 23; 33
Camada de Rede, 23
Camada de Sessão, 23
Camada de Transporte, 23
Camada Física, 22; 32
Conecções, 68; 69
ECP - End Connection Pointer, 69
Entidades, 67
Identificadores e Endereçamento, 68
SAP - Service Access Point, 67; 68; 69; 70
Serviços, 67

P

Parsing, 77
PFS-ACE (ver PFS-DAE), 43
PFS-DAE (ver Sistemas Federadores, 39; 42
PIRAMID (ver Sistemas Federadores, 39
Planeamento da Produção, 14; 40
Aplicações, 41
CimView, 43
DECFactory, 40
Gestão de Stocks, 8

Planeamento de Capacidades, 14
Plant Floor Control, 20
Planteworks (ver Gerador de Aplicações), 43
Platafarmas
VAX, 40; 42
Plataformas
ULTRIX, 40
VAX, 40
QBus, 40
PMW - Paperless Manufacturing Workplace, 44
POMS - Process Operations Management System, 44
Pragmakit (ver Gerador de Aplicações), 43
Produção Automatizada, 7
Produção Integrada por Computador
Competitividade, 3
PROFIBUS, 30; 31; 34; 35; 36
ISP - Interoperable Systems Project, 35
Programa ESPRIT, 20
Protocolos de Comunicação Prprietários, 20
PYRAMID (ver Sistemas Federadores, 42

Q

QBus (ver Plataformas VAX, 40

R

Rackbus, 30
RDA - Remote Database Access, 21
Redes Públicas de Dados, 47
Regeneração de Sinal, 45
Repetidor, 45
Robots, 3; 6; 11; 12; 20; 34
Router, 45; 46; 47
RS-232, 40; 42

S

SAP - Service Access Point, 67; 68; 69; 70
Segmentação, 23
Sistemas de Controlo de Qualidade
Aplicações, 41
Sistemas de Manutenção Preventiva, 6
Aplicações, 41
Sistemas de Transporte Automático, 6; 12; 34
Sistemas Distribuidos de Aquisição de Dados, 41
Sistemas Distribuídos de Aquisição de Dados, 4
Sistemas Federadores, 39
BASESTAR, 39; 40; 42
DCM-Manufacturing, 39
OpenCIM, 39
OpenCim, 44
PFS-DAE, 39; 42
PYRAMID, 39; 42
Sistemas Heterogéneos, 39
Sistemas Industriais, 4
Competitividade, 4
Disponibilidade, 4
Evolutibilidade, 5
Exemplos de Industrias, 9
Flexibilidade, 5; 10
de Encaminhamento, 5
de Produto, 5
de Volume, 5
Indústria de Fabrico Discreto, 9; 27
Indústria de Processos, 9; 27
Industria de Produção em Massa, 10
Industria de Protótipos, 10
Sistemas Operativos
AIX, 43
OS/400, 43
VMS, 40; 43
Stock Zero, 4

T

TCP/IP, 30; 42
Tempo Crítico, 29; 31
Token Bus, 33
Token Bus, 24; 36; 45; 46
Token Ring, 40; 43
TOP - Technical and Office Protocol, 23; 24; 25; 30
Transmissão de Dados
Alimentação no Barramento de Dados, 32
Erros de Transmissão, 33
Fibra Óptica, 32
Largura de Banda, 32
Velocidade de Transmissão, 32

U

UniTelway, 40

Universo de Discurso, 71

V

Visão Artificial, 12

VMD - Virtual Manufacturing Device, 71; 72; 73; 74

VMS (ver Sistemas Operativos), 40; 43

VTP, 24

W

WorlFIP (ver FIP), 35

X

X400 (ver correio electrónico), 24