

Faculdade de Engenharia da Universidade do Porto



**Plataforma Configurável para Gestão de
Edifícios baseada em IEC 61499**

Emanuel Guedes Pereira Damaso

Dissertação realizada no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major Automação

Orientador: Prof. Dr. Mário de Sousa

Junho de 2011

Resumo

A FEUP (Faculdade de Engenharia da Universidade do Porto) encontra-se actualmente a finalizar a implementação de um complexo e completo sistema de gestão de edifícios baseado num SCADA central, para efeitos de monitorização e controlo, e em controladores PLCs espalhados pela faculdade, com a função de actuar sobre os diferentes recursos a gerir (iluminação, elevadores, bombas de água, entre outros).

Esta dissertação surge com o objectivo de estruturar e validar uma arquitectura alternativa para esse sistema. Esta nova arquitectura baseia-se na emergente norma IEC 61499, e com ela espera-se obter maior modularidade e flexibilidade nas aplicações de controlo e supervisão, características extremamente úteis quando se trata de um sistema como o que está em causa.

Para além de ser capaz de permitir a gestão dos recursos, a nova arquitectura visa aumentar o grau de autonomia do sistema através da reformulação do processo de interacção entre as configurações determinadas pelo utilizador e as aplicações de controlo que efectuem a gestão dos recursos.

Para auxiliar na validação da arquitectura surge a ferramenta 4DIAC, uma tecnologia emergente *open-source* que visa a criação e execução de aplicações baseadas na norma IEC 61499. O estudo e a descrição do modo de funcionamento deste *framework* são também objectivos desta dissertação.

Abstract

FEUP (Univ. Porto, Engineering Faculty) is currently finishing the process of implementing a complex and complete building management system based on a central SCADA, used for monitoring and controlling purposes, and on a set of PLCs distributed around campus, used for actuating on the different manageable resources (lighting, elevators, water pumps, among others).

This thesis aims to create and validate an alternative architecture for this system. This new architecture is based on the emerging IEC 61499 standard and aims to provide greater modularity and flexibility to the supervision and control applications, faculties that are extremely useful when it comes to a system like the one this thesis targets.

Besides allowing the management of resources, the new architecture aims to provide more autonomy for the system by reforming the interaction process between the user-given configurations and the control applications that handle the resources.

As a tool for helping the validation of the new architecture the 4DIAC framework comes into the project, with it being an emerging open-source technology aimed at the creation and deployment of IEC 61499-based applications. The study and description of the framework are also objectives of this thesis.

Agradecimentos

Aos meus pais e aos meus irmãos.

A todos os amigos e colegas da faculdade, em especial aos que estiveram comigo durante todo este percurso.

Ao orientador Prof. Mário de Sousa, pela ajuda e pelo tempo despendido.

Ao Prof. José Faria e ao Bruno Moreira, pelas mesmas razões.

Índice

Resumo	iii
Abstract	v
Agradecimentos	vii
Índice	ix
Lista de Figuras	xii
Lista de Tabelas	xv
Abreviaturas	xvii
Capítulo 1	1
Introdução	1
1.1 - Sistemas de Gestão Técnica de Edifícios	2
1.2 - Plataforma de Gestão Técnica da FEUP	2
1.2.1 - Descrição de componentes	2
1.2.2 - Circuito de Iluminação	5
1.3 - Trabalho Previamente Desenvolvido	6
1.4 - Objectivos	8
1.5 - Trabalho Relacionado	9
1.6 - Estrutura do Documento	9
Capítulo 2	11
Norma IEC 61499	11
2.1 - Introdução	11
2.1 - Referência	12
2.1.1 - Sistema	12
2.1.2 - Dispositivo	12
2.1.3 - Recurso	12
2.1.4 - Function Block	13
2.1.5 - Aplicações	17
Capítulo 3 4DIAC	19
3.1 - Introdução	19

3.2 - Modo de funcionamento	19
3.3 - 4DIAC-IDE.....	21
3.3.1 - Instalação	21
3.3.2 - Utilização	22
3.3.3 - Importação/ Exportação	24
3.3.4 - Desenvolvimento de um Sistema	24
3.3.5 - Transferência de aplicações	25
3.4 - 4DIAC-RTE (FORTE)	26
3.4.1 - Compilação	27
3.4.2 - Arquitectura do código	28
3.4.3 - Adicionar elementos à biblioteca.....	29
3.4.4 - Código fonte dos Function Blocks.....	30
3.4.5 - Inicialização e Transferência	32
3.4.6 - Compilação do FORTE para ICnova	33
3.5 - Testes e validação	34
3.6 - Conclusão	34
Capítulo 4	37
Modbus.....	37
4.1 - Introdução	37
4.2 - Referência	37
4.2.1 - Princípios de funcionamento	37
4.2.2 - Modbus TCP	39
4.3 - Requisitos	40
4.4 - Arquitectura	41
4.4.1 - SIFB vs Sistema de comunicação do FORTE	41
4.4.2 - SIFB Único vs Modularidade	42
4.4.3 - Cliente	42
4.4.4 - Servidor	43
4.4.5 - Libmodbus	44
4.5 - Implementação.....	45
4.5.1 - Módulo Cliente	45
4.5.2 - Módulo Servidor	50
4.6 - Testes e Validação.....	55
4.7 - Conclusão	57
Capítulo 5	59
Sistema IEC 61499	59
5.1 - Introdução	59
5.2 - Requisitos	60
5.3 - Discussão da Arquitectura.....	60
5.4 - Descrição da arquitectura escolhida	62
5.4.1 - Aplicação de controlo genérica	64
5.4.2 - Programa para geração automática de aplicações	65
5.4.3 - Transferência de aplicações	66
5.5 - Implementação do serviço de cumprimento de horários	67
5.5.1 - Aplicação genérica	67
5.5.2 - Geração automática de aplicações	69
5.5.3 - Autoload.....	71
5.6 - Testes e validação	73
5.7 - Conclusão	75
Capítulo 6	77
Conclusão e Trabalho Futuro	77
Referências	79

Anexos	81
<i>Anexo 1</i>	81
<i>Anexo 2</i>	81
<i>Anexo 3</i>	82
<i>Anexo 4</i>	83
<i>Anexo 5</i>	84
<i>Anexo 6</i>	84
<i>Anexo 7</i>	84
<i>Anexo 8</i>	85

Lista de Figuras

Figura 1.1 - Arquitectura da plataforma de gestão técnica da FEUP	3
Figura 1.2 - Antiga interface do SCADA	4
Figura 1.3 - Esquema genérico da arquitectura do subsistema de iluminação [2]	6
Figura 2.1 - Modelo de sistema IEC 61499 [2]	12
Figura 2.2 - Modelo de dispositivo IEC 61499 [2]	13
Figura 2.3 - Modelo representativo de um Function Block genérico [2]	14
Figura 2.4 - Modelo representativo de um Basic Function Block [10]	14
Figura 2.5 - ECC exemplo [2]	15
Figura 2.6 - Máquina de estados do ECC [2]	15
Figura 2.7 - Arquitectura de um Composite Function Block genérico [2]	16
Figura 2.8 - SIFB REQUESTER genérico [10]	17
Figura 2.9- Aplicação desenvolvida para o controlo de um cilindro [15]	18
Figura 3.1 - Diagrama de funcionamento da <i>framework</i> 4DIAC	19
Figura 3.2 - <i>Screenshot</i> do modo de edição de aplicações do 4DIAC-IDE	21
Figura 3.3 - Lista de bibliotecas de FB disponíveis por defeito	22
Figura 3.4 - Configuração de um sistema teste	25
Figura 3.5 - Excerto de um relatório de transferência do 4DIAC-IDE	26
Figura 3.6 - Arquitectura do FORTE	27
Figura 3.7 - Programa CMake GUI em execução	28
Figura 3.8 - Conteúdo do ficheiro CMakeLists da pasta “src/modules/math”	29
Figura 3.9 - Função “executeEvent” da classe “FB_ADD_INT”	30
Figura 3.10 - Código fonte de um SIFB exemplo após a exportação feita pelo 4DIAC-IDE	32
Figura 3.11 - Consola após a compilação do FORTE para ICnova	34
Figura 4.1 - Tipos de memória Modbus [20]	38

Figura 4.2 - Formato de uma trama genérica do protocolo Modbus [21]	38
Figura 4.3 - Formato da trama Modbus TCP [21]	39
Figura 4.4- SIFB “mm_read_coils”	45
Figura 4.5- SIFB “mm_read_discrete_inputs”	46
Figura 4.6- SIFB “mm_read_holding_registers”	46
Figura 4.7- SIFB “mm_read_input_registers”	46
Figura 4.8- SIFB “mm_write_coils”	46
Figura 4.9- SIFB “mm_write_holding_registers”	46
Figura 4.10 - Sequência de serviço para uma inicialização mal-sucedida dos SIFBs Modbus Cliente	47
Figura 4.11 - Sequência de serviço para uma inicialização bem-sucedida dos SIFBs Modbus Cliente	47
Figura 4.12 - Sequência de serviço para uma terminação de serviço dos SIFBs Modbus Cliente	47
Figura 4.13 - Diagrama UML de actividade do processo de gestão da comunicação dos SIFBs Modbus Cliente	48
Figura 4.14 - Sequência de serviço para uma requisição de serviço bem-sucedida dos SIFBs Modbus Cliente	49
Figura 4.15 - Sequência de serviço para uma requisição de serviço mal-sucedida dos SIFBs Modbus Cliente	49
Figura 4.16 - Diagrama UML de actividade do processo de execução de serviço dos SIFBs Modbus Cliente	49
Figura 4.17 - CFB “MODBUS_TCP_SERVER”	50
Figura 4.18 - SIFB “ServerModbusMem”	50
Figura 4.19 - Diagrama UML de actividade do modo de funcionamento do SIFB ServerModbusMem	51
Figura 4.20 - SIFB “ServerslpHandler”	53
Figura 4.21 - FB “MUXModbusServer”	53
Figura 4.22 - FB “SWITCHModbusServer”	54
Figura 4.23 - Modelo do teste aos SIFB Modbus Cliente	56
Figura 4.24 - Modelo do teste da resposta a pedidos feitos pelos SIFB Modbus Cliente	56
Figura 5.1 - Diagrama da arquitectura de <i>software</i> proposta	63
Figura 5.2 - Aplicação genérica relativa ao serviço de cumprimento de horários	67
Figura 5.3 - Interface do FB “scheduleONOFF”	68

Figura 5.4 - Interface do SIFB "RTC"	69
Figura 5.5 - Diagrama UML de actividade do programa de geração de aplicações.....	70
Figura 5.6 - Interface do SIFB "BFILE_MGR"	72
Figura 5.7 - Rede interna do novo CFB "DM_KRNL"	73
Figura 5.8 - Modelo do teste à implementação do serviço de cumprimento de horários	75

Lista de Tabelas

Tabela 1 - Correspondência entre os SIFBs Modbus Cliente e os API's da biblioteca
Libmodbus 50

Abreviaturas

FEUP	Faculdade de Engenharia da Universidade do Porto
AVAC	Aquecimento, ventilação e ar condicionado
SCADA	<i>Supervisory Control and Data Acquisition</i>
PLC	<i>Programmable Logic Controller</i>
IEC	<i>International Electrotechnical Commission</i>
TCP	<i>Transmission Control Protocol</i>
PDI	Preparação da Dissertação
FBDK	<i>Function Block Development Kit</i>
PC	<i>Personal Computer</i>
IDE	<i>Integrated Development Environment</i>
FB	<i>Function Block</i>
BFB	<i>Basic Function Block</i>
CFB	Composite Function Block
SIFB	Service Interface Function Block
SFB	Simple Function Block
ECC	<i>Execution Control Chart</i>
GUI	<i>Graphical User Interface</i>
HMI	<i>Human-Machine Interface</i>
API	<i>Application Programming Interface</i>
CPU	<i>Central Process Unit</i>
IP	<i>Internet Protocol</i>
XML	<i>Extensible Markup Language</i>
ST	<i>Structured Text</i>

Capítulo 1

Introdução

A Faculdade de Engenharia da Universidade do Porto (FEUP) investiu recentemente num sistema integrado para a gestão de edifícios com o intuito de otimizar a utilização dos diferentes recursos inerentes a edifícios - iluminação, aquecimento, ventilação, bombagem, entre outros -. Este tipo de sistemas possui como principais características a automatização de processos e a flexibilização das tarefas de controlo e supervisão. Os resultados que se esperam obter fruto desta optimização são a redução de custos, redução da poluição produzida, eliminação da mão-de-obra relacionada com processos de manutenção e obtenção de certificados energéticos [1].

O sistema em causa assenta numa rede própria de dispositivos de vários tipos, entre eles sensores, actuadores e controladores. A complementar o variado leque de *hardware* encontram-se implementadas diversas componentes de *software* que lidam com a lógica programática dos controladores, asseguram a comunicação de dados pela rede, oferecem ao utilizador a possibilidade de alterar as configurações do sistema, providenciam supervisão do sistema, entre outras faculdades.

Apesar deste tipo de soluções estar já amplamente implementado ainda existe espaço para progressão no que toca às capacidades e à eficácia e eficiência do seu funcionamento. No caso concreto do sistema implementado na FEUP, a utilização de aplicações de controlo baseadas na norma IEC 61131-3 vai de encontro a certas necessidades que os dispositivos encarregados dos processos de supervisão e controlo apresentam, nomeadamente possuir um elevado grau de autonomia relativamente ao restante sistema e assegurar flexibilidade perante uma possível reconfiguração da arquitectura ou das configurações do sistema.

É neste contexto que surge a proposta que dá título a esta dissertação. Por um lado procura-se architectar e validar uma solução que sirva de base à reformulação o actual sistema de acordo com o novo *standard* IEC 61499 [2]. Por outro, visa-se tornar os processos inerentes à gestão e manutenção do sistema o mais autónomos e independentes possíveis. Este documento narra os passos tomados para atender a essa proposta, incluído discussão e definição de arquitecturas, implementação e teste de soluções, entre outros.

1.1 - Sistemas de Gestão Técnica de Edifícios

Pode-se definir um sistema integrado de gestão de edifícios como um complexo, multinível, multiobjectivo, integrado, inter-relacionado e completo gestor de informação de *design* inteligente capaz de suportar os processos associados aos recursos dos edifícios desde a fase de concepção até à fase de gestão e manutenção [3]. No caso concreto da gestão técnica, esses processos prendem-se com o controlo e supervisão de serviços como a iluminação, os elevadores, os AVACs, as bombas de água, entre outros.

Ligado à área de gestão técnica de edifícios aparece o conceito de sistema de automação de edifícios. Este tipo de sistemas usa uma arquitectura que contempla controladores digitais, redes de comunicação e até inteligência artificial, entre outros [4]. Os controladores executam aplicações que podem ser de natureza mais centralizada ou mais distribuída, sendo a função destas actuar sobre o mundo real e providenciar informação ao utilizador ou ao comando central. A rede de comunicação, que pode ser multinível, assegura a comunicação entre os diferentes componentes do sistema, utilizando para isso um ou mais protocolos próprios para o efeito.

Estes sistemas de gestão técnica são amplamente usados em edifícios contemporâneos devido às vantagens que trazem: aumentam a eficiência do uso de recursos (água, luz, gás), aumentam a segurança das pessoas (ajudam no caso da ocorrência de um incêndio), tornam o processo de supervisão mais cómodo, reduzem custos com pessoal, entre outros. Ainda assim, esta área está em franca expansão, como comprova o crescimento das soluções oferecidas no campo da domótica. Os sistemas estão cada vez mais inteligentes, mais autónomos e mais próximos do utilizador convencional.

1.2 - Plataforma de Gestão Técnica da FEUP

Ao nível de arquitectura, a plataforma de gestão actualmente implementada baseia-se num SCADA (*supervisory control and data acquisition*) central, com funções de supervisão e controlo, numa base de dados, contendo as informações necessárias ao funcionamento do sistema, numa interface Web capaz de interagir com a base de dados, numa rede de controladores do tipo PLCs (*programmable logic controllers*) distribuídos pelos diferentes edifícios da faculdade, e nos I/Os remotos (ilhas) associados a esses PLCs. Ligados a esses I/Os remotos estão os circuitos eléctricos a controlar e os sinais dos sensores dos diversos equipamentos. Esta arquitectura encontra-se ilustrada na Figura 1.1. Cada um destes componentes é descrito com maior pormenor nas secções seguintes. Deve ser mencionado que no momento da realização e escrita desta dissertação não existiam documentos (pelo menos finais e oficiais) que descrevessem o sistema, facto que impediu uma análise mais exaustiva do mesmo.

1.2.1 - Descrição de componentes

Base de Dados

A base de dados central é responsável por armazenar uma descrição útil de todos os circuitos eléctricos a controlar, incluindo o espaço, o PLC, o I/O remoto e a saída física aos

quais eles estão associados. Paralelamente, são armazenados também os horários de funcionamento de cada um desses circuitos. Um horário consiste num registo semanal com 3 períodos diários de funcionamento relativo a um circuito eléctrico. Um período diário consiste numa hora de activação, numa hora de desactivação e de um valor *set point* usado para indicar a percentagem de utilização de um equipamento (por exemplo, no caso de um equipamento de iluminação, se o *set point* for de 75% significa que 3 em cada 4 lâmpadas associadas a esse equipamento devem ser activadas).

É usado o gestor Microsoft SQL Server para aceder localmente à base de dados, permitindo que a informação sobre os horários possa ser extraída e reencaminhada para o SCADA central e permitindo ainda uma edição local do conteúdo da base de dados.

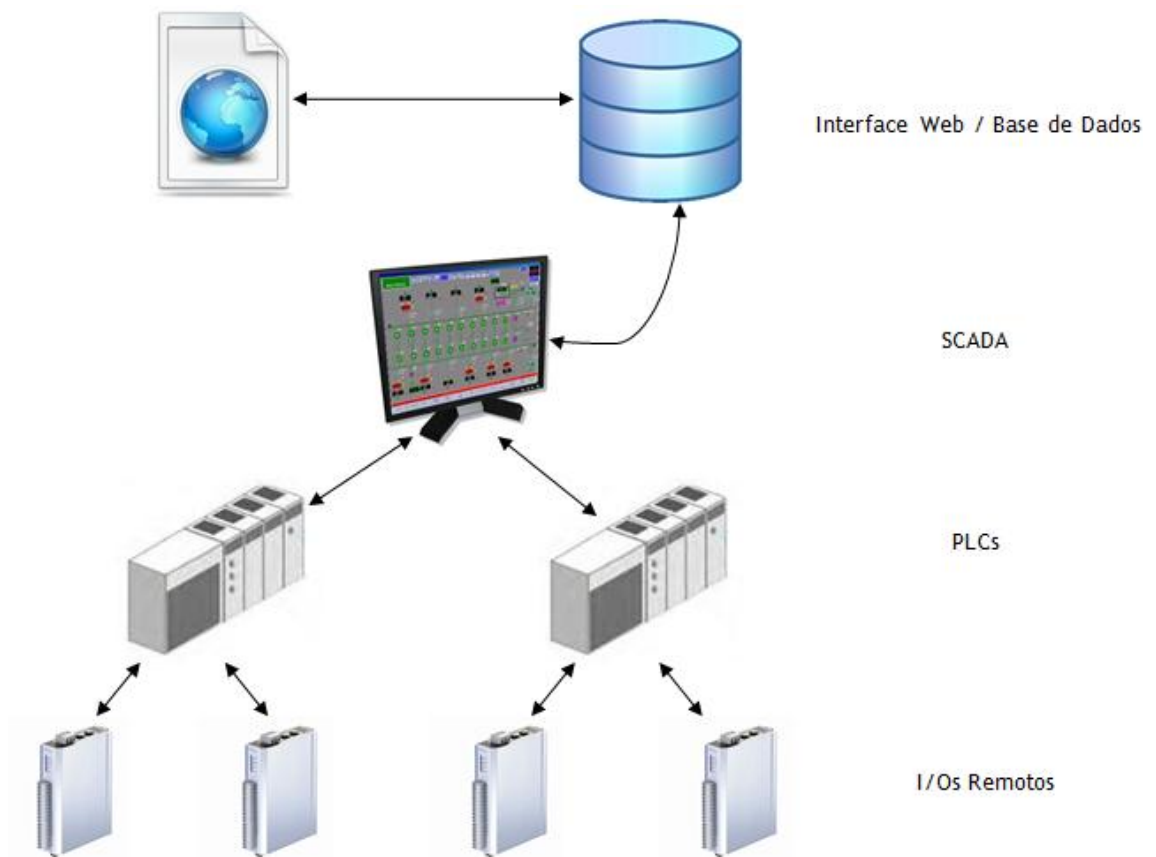


Figura 1.1 - Arquitectura da plataforma de gestão técnica da FEUP

SCADA

A gestão técnica centralizada está a cargo de um sistema SCADA que oferece a possibilidade de monitorizar e controlar os equipamentos espalhados pela faculdade. Permite comunicação com a base de dados para que sejam transferidas as configurações do sistema definidas pelo utilizador, e permite ainda redireccionar esta informação para os controladores respectivos. Estas configurações incluem os horários dos equipamentos e os alarmes, entre outra informação.

A monitorização do sistema é feita através da comunicação entre o SCADA e os dispositivos de campo (PLCs e I/Os remotos), com o SCADA a providenciar uma interface

gráfica para a apresentação que obtém dos dispositivos. Essa interface permite uma navegação entre os diferentes tipos de recursos monitorizados (iluminação, aquecimento, ventilação, etc) e entre os vários espaços que fazem parte da FEUP.

Existe ainda a possibilidade do SCADA enviar comandos em tempo real para os PLCs, comandos estes que podem ser para a activação ou desactivação de algum equipamento, entre outras operações possíveis.

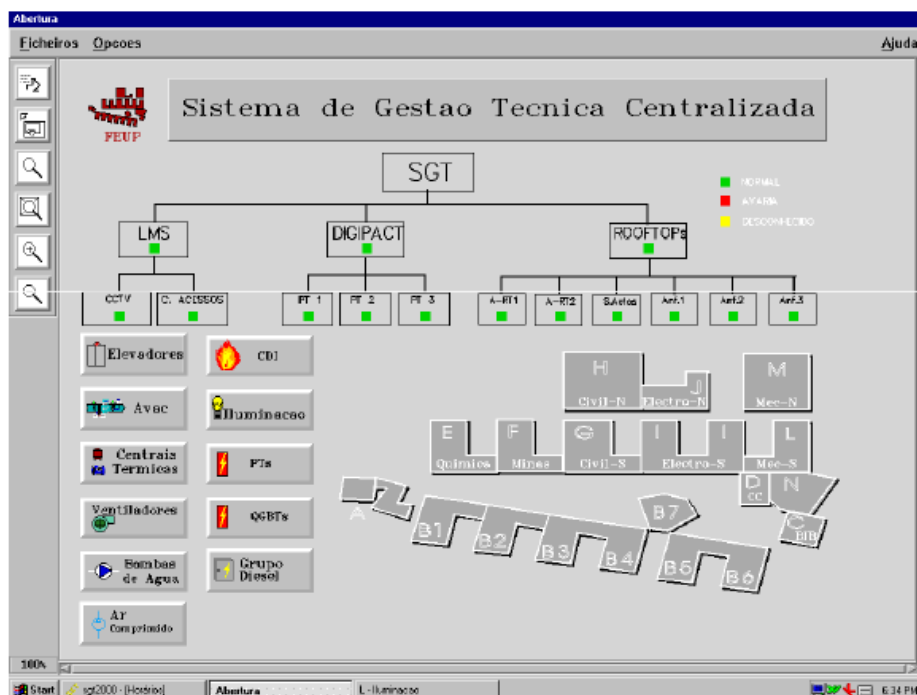


Figura 1.2 - Antiga interface do SCADA

Interface Web

Para consultar e editar a base de dados de forma remota foi desenvolvida uma interface do tipo *webpage*. Esta ferramenta, que apesar de já se encontrar a funcionar ainda não está completamente concluída, oferece ao utilizador a possibilidade de editar as configurações do sistema, com a atribuição de horários a surgir como operação mais utilizada. A interface permite ao utilizador atribuir horários a qualquer circuito eléctrico de forma individual ou em grupo. Neste último caso os circuitos podem ser agrupados por edifício, por andar ou por espaço e ser-lhes atribuído um horário de funcionamento comum, o que oferece bastante flexibilidade a todo o processo.

A interface oferece igualmente a possibilidade de serem configurados alarmes na eventualidade de um equipamento acusar alguma avaria. Entre as modalidades de alarmes estão o envio de SMS, a activação de buzinas e a notificação da avaria no SCADA.

Rede de PLCs

A lógica de controlo de campo é da responsabilidade por um vasto grupo de PLCs da marca Schneider que se encontra distribuído por toda a faculdade. Nestes controladores são executadas aplicações de controlo baseados na norma IEC 61131-3 que usam a informação

recebida do SCADA referente aos horários e alarmes dos circuitos para fazer cumprir as configurações definidas pelo utilizador. Na prática, o SCADA transfere as configurações (horários e alarmes) para espaços de memória pré-definidos nos PLCs, com as aplicações de controlo a consultarem ciclicamente esses mesmos espaços de memória de modo a verificarem se é necessário ligar ou desligar algum circuito ou a saberem qual é a atitude a tomar perante o mal funcionamento de algum equipamento. Existe ainda a possibilidade de os PLCs receberem pedidos assíncronos do controlador central SCADA visando a obtenção do valor de uma variável ou a activação ou desactivação de um determinado circuito eléctrico. Cada PLC socorre-se de uma série de I/Os remotos para ligar/desligar circuitos e ler sinais dos equipamentos. A comunicação entre as PLCs e I/Os remotos é feita usando o protocolo Modbus TCP, com os PLCs a assumirem o papel de clientes e os I/Os remotos a funcionarem como servidores.

I/Os Remotos

Como é referido no ponto anterior, cada PLC tem associado uma série de I/Os remotos de modo a expandir a sua capacidade de controlo em termos de espaço de actuação. Por um lado estas ilhas recebem os comandos dos controladores e actualizam as saídas, por outro recebem os sinais dos sensores e comunicam os mesmos aos PLCs. A configuração destes elementos consiste apenas em mapear os endereços Modbus, fazendo-lhes corresponder um *output* ou *input* físico. Enquanto entidades servidoras, o papel dos I/Os remotos é executar os pedidos enviados pelos PLCs, sendo estes pedidos do tipo escrita ou leitura de endereços de memória. Estes pedidos vão-se reflectir na actualização de saídas físicas, no caso dos pedidos de tipo escrita, ou no envio para o cliente do valor de entradas ou saídas, no caso dos pedidos do tipo leitura.

1.2.2 - Circuito de Iluminação

Tendo em conta que os diferentes subsistemas (iluminação, aquecimento, ventilação, bombagem, entre outros) possuem estruturas similares, apresenta-se aqui apenas o caso da plataforma de gestão dos recursos de iluminação. Esta exposição deve ser encarada como uma referência para os outros subsistemas.

Os elementos físicos que a plataforma de gestão do sistema de iluminação visa manipular são os circuitos eléctricos que alimentam os equipamentos. Conforme é descrito na secção relativa à base de dados, a cada circuito está associada uma série de características. O espaço em que o circuito se encontra e o seu horário de funcionamento são dois exemplos.

Cada circuito de iluminação pode possuir, em complemento à componente normal, uma parte de emergência. Na prática, este grupo de circuitos divide-se em dois subgrupos.

A comunicação entre os I/Os remotos e os PLC's consiste, por um lado, no envio de sinais de sensores e, por outro, no envio de sinais de controlo. Os sinais de controlo são homogéneos (ligar/desligar), enquanto os sinais dos sensores podem ser de vários tipos: estado do actuador, sensor de presença ou sensor de luminosidade.

Em termos de práticos, quando o PLC determina que um circuito deve ser ligado, é enviado um sinal de comando para o telerruptor respectivo, ao qual se segue a leitura do estado do actuador, para confirmar que a ordem foi bem executada. Caso o telerruptor não entre em condução o procedimento é repetido mais duas vezes. Se ao fim dessas duas vezes o

circuito não se encontrar em condução é lançado um alarme (o tipo de alarme que é activo respeita o que está estipulado na base de dados para cada situação de erro).

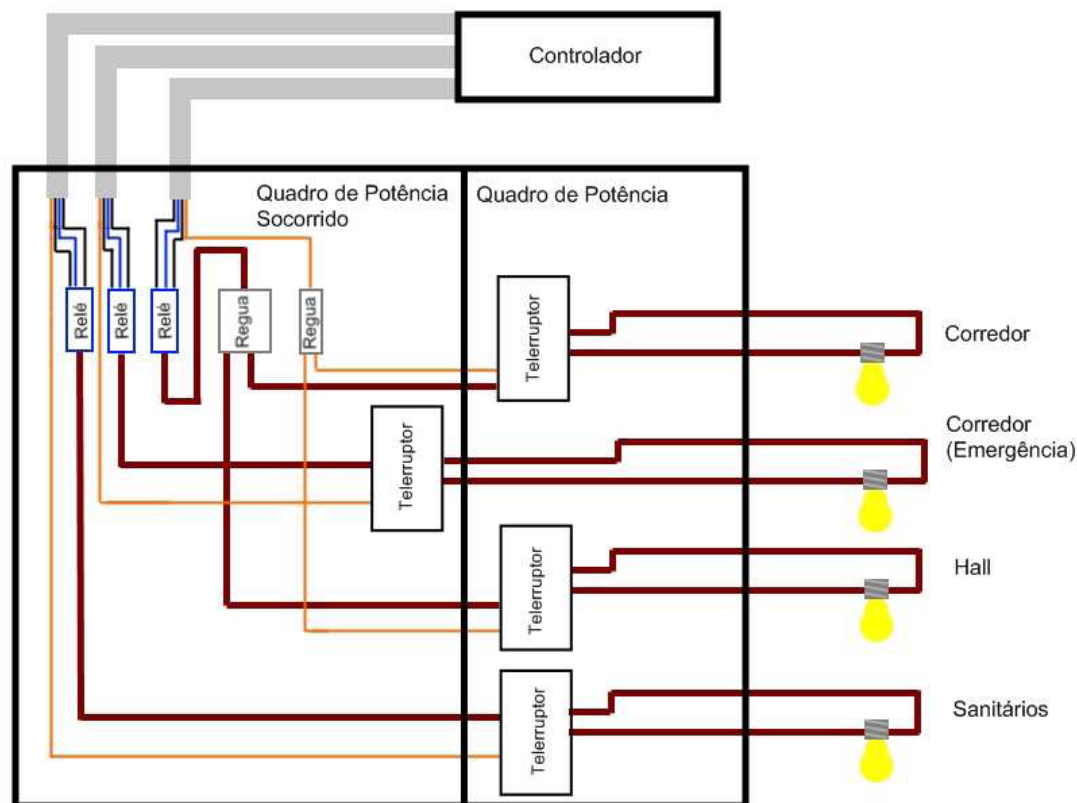


Figura 1.3 - Esquema genérico da arquitectura do subsistema de iluminação [2]

1.3 - Trabalho Previamente Desenvolvido

A realização desta dissertação foi antecedida por um trabalho introdutório desenvolvido no âmbito da disciplina de Preparação da Dissertação (PDI). Esse trabalho dividiu-se essencialmente em duas partes: estudo do sistema de gestão actualmente instalado na FEUP e preparação para o desenvolvimento de uma nova arquitectura para o sistema baseada em IEC 61499. Os resultados da primeira parte desse trabalho encontram-se documentados nas secções transactas. Os que se referem à segunda parte são de seguida descritos.

Ao se impor a criação de uma arquitectura baseada na norma IEC 61499 torna-se imperativo escolher uma *framework* que seja direccionada para essa directiva. Por *framework* entenda-se um *software* compreensivo que permita a criação e execução de um sistema desenhado pelo utilizador. Neste contexto, uma das tarefas realizadas no âmbito da disciplina de PDI foi eleger uma *framework* para ser utilizado no projecto.

Apesar a norma IEC 61499 ser relativamente recente (oficialmente saiu em 2005) existem já algumas *framework* disponíveis que satisfazem o perfil definido. Foram consideradas para utilização as seguintes ferramentas: *Function Block Development Kit (FBDK)*, *ISaGRAF* e *4DIAC*. A escolha final recaiu sobre esta última por se tratar de *software open-source*, de origem académica e por apresentar enorme potencialidade.

A iniciativa 4DIAC nasceu em Julho de 2007 no Instituto de Automação e Controlo da Universidade de Tecnologia de Viena, na Áustria. Tratando-se de um projecto dirigido por

uma comunidade académica, tem vindo a apresentar resultados extremamente positivos quer em implementações práticas, quer na área do crescimento da própria tecnologia [5][6].

A ferramenta comporta duas componentes: 4DIAC-IDE, um ambiente de desenvolvimento baseado no *software open source* Eclipse (para Windows ou Linux), e FORTE (4DIAC-RTE), um *runtime environment* implementado em C++ que tem como *target* PCs e sistemas embebidos aonde são executadas as aplicações. A primeira componente é usada para desenhar elementos e arquitectar sistemas segundo a norma IEC 61499, enquanto a segunda permite a execução desses sistemas.

Na sequência do parágrafo anterior, uma das conclusões que resultaram deste período de trabalho preparatório foi a necessidade de serem substituídos os PLCs da actual arquitectura da plataforma de gestão técnica da FEUP. Isto porque estes dispositivos, responsáveis por executarem todas as aplicações baseadas em IEC 61131 que implementam a lógica de controlo do sistema, não cumprem os requisitos impostos pelo FORTE, nomeadamente possuírem um sistema operativo compatível. A decisão tomada foi utilizar o sistema embebido ICnova AP7000 Base [7] como novo controlador, pois para além de este dispositivo cumprir os requisitos associados ao FORTE existem exemplares disponíveis na FEUP que podem ser utilizados para fins académicos.

A ICnova consiste numa plataforma de 32 bits que corre de origem Linux - Kernel v2.6.24, com várias soluções para *networking*, nomeadamente porta Ethernet 10/100 Mbit/s, porta USB-UART via CP2102, I2C e comunicação UART. O processador de 200 MHz pertence à categoria AVR32, a memória volátil SDRAM tem o tamanho de 64 MB e a não-volátil de 8 MB. Em termos de *software*, a ICnova possui compiladores de várias linguagens de programação, entre elas C++ (necessário para a utilização da ferramenta FORTE). Possui também bibliotecas que implementam a comunicação assente em TCP/IP, sendo que todo o *software* é *open-source*. O preço ronda os 100 euros.



Figura 1.4 - ICnova AP7000 Base

Uma outra conclusão resultante deste estudo prévio foi a necessidade de incorporar o protocolo Modbus na ferramenta 4DIAC, visto ela não ser suportada por defeito pela

framework. Esta incorporação é necessária para que a introdução dos novos controladores e das novas aplicações de controlo relativas à nova arquitectura não obrigue a que mais nenhuma componente da actual arquitectura implementada tenha de ser substituída (nomeadamente os I/Os remotos).

Chegou-se ainda à conclusão lógica de que seria necessário proceder à compilação do FORTE tendo a ICnova como *target*, para que o sistema embebido pudesse alojar as aplicações de controlo.

Por fim, fica a ressalva que, apesar de a opção pela ICnova não comprometer em nada o projecto, há no mercado dispositivos do mesmo perfil que possuem características técnicas superiores e que por isso seriam mais indicadas para uma implementação final do sistema a desenvolver. São exemplos desses dispositivos as soluções oferecidas pela Wago [8] e o computador embebido BeagleBoard [9].

1.4 - Objectivos

O objectivo fundamental desta tese é a elaboração e validação de uma arquitectura baseada em IEC 61499 para uma plataforma de gestão dos recursos técnicos da FEUP. Para atingir esse objectivo são definidas 3 áreas de actuação: *framework* 4DIAC, protocolo Modbus e arquitectura para a plataforma de gestão técnica da FEUP.

Na primeira área, relativa à ferramenta 4DIAC, é proposto o estudo das duas componentes da *framework* (4DIAC-IDE e FORTE), pressupondo uma análise exaustiva e compreensiva às funcionalidades, potencialidades e limitações da tecnologia. Tendo em conta a utilização da plataforma ICnova no projecto, estabelece-se igualmente como objectivo a compilação do FORTE tendo como *target* esse sistema embebido.

Na segunda parte propõe-se a implementação do protocolo Modbus na ferramenta 4DIAC, sendo os objectivos desta secção:

- Desenhar uma arquitectura que permita fornecer à ferramenta a capacidade de comunicar usando o protocolo Modbus;
- Implementar essa arquitectura;
- Testar, validar e avaliar a solução desenvolvida;
- Documentar as etapas enunciadas nos pontos anteriores.

A última secção, que é a que mais directamente se relaciona com o objectivo fundamental, prende-se com a estruturação e validação de uma arquitectura para a gestão de recursos da FEUP. Os objectivos desta secção são:

- Desenvolver uma arquitectura de controlo baseada na norma IEC 61499 capaz de servir de base para uma futura substituição da actual plataforma de gestão de recursos da FEUP e que permita aumentar o nível de autonomia do próprio sistema;
- Validar essa arquitectura através das necessárias implementações;
- Testar e avaliar as implementações desenvolvidas;
- Documentar as etapas enunciadas nos pontos anteriores;

1.5 - Trabalho Relacionado

Esta secção procura expor alguns dos desenvolvimentos e estudos recentes que têm vindo a ser feitos na área de sistemas de controlo e supervisão associados à norma IEC 61499 e compará-los com o caso abordado nesta dissertação.

Um desses estudos é descrito no artigo “Modeling of Reconfiguration Control Applications based on the IEC 61499, Reference Model for Industrial Process Measurement and Control Systems” [10], que data de 2006. Neste estudo é sugerida uma arquitectura que funcione como guia para que um sistema em execução baseado em IEC 61499 se possa adaptar perante uma necessidade de reconfiguração das aplicações que estão a correr. Da mesma forma, neste projecto há também a necessidade de o sistema de controlo se ajustar às configurações impostas pelo utilizador, nomeadamente com os horários e alarmes impostos, sendo que a perspectiva de este ajuste ser feito sem a necessidade de existir qualquer tipo de *down-time* do sistema é desejada, mas não obrigatória.

Também de 2006 consta o artigo “IEC 61499 Function Blocks for Distributed Networked Embedded Applications” [11], que aborda o desenvolvimento de um sistema de controlo baseado em IEC 61499 para a gestão de uma rede de electrónica automóvel. Este projecto funciona como modelo para a dissertação pois nele foi implementada uma arquitectura que se assemelha àquela que é desejada para a solução a desenvolver, nomeadamente a existência de múltiplos controladores embebidos unidos por uma rede de comunicação, cada um deles com uma série de sensores e actuadores associados com os quais podem interagir. As aplicações que correm nos sistemas embebidos recebem e enviam dados para um controlador central de modo a possibilitar os processos de controlo e monitorização remotos. O projecto desta dissertação difere deste *case-study* nos requisitos próprios que o sistema da FEUP impõe, para além de o controlo não ser directamente efectuado pelo utilizado mas ser antes feito em função das configurações pré-definidas.

Por último, o artigo “Migration of a PLC Controller to an IEC 61499 Compliant Distributed Control System: Hands-on Experiences” [12], de 2005, aborda a mudança de paradigma da IEC 61131-3 para IEC 61499. Trata-se de uma abordagem teórica que procura sistematizar o processo de migração dos sistemas de supervisão e controlo, abordando tanto a componente de *hardware* como de *software*. Este exercício toca naquele que é o principal desafio desta dissertação, a mudança do paradigma da estrutura de um sistema da norma antiga para a nova norma.

1.6 - Estrutura do Documento

Esta dissertação encontra-se dividida em 6 capítulos: introdução, norma IEC 61499, 4DIAC, Modbus, arquitectura baseada em IEC 61499 e conclusão e trabalho futuro.

A introdução, aonde se inclui esta secção, inclui uma contextualização do tema tratado pela dissertação e enuncia os objectivos delineados para ela.

O segundo capítulo consiste numa referência da norma IEC 61499, sendo enunciados os conceitos fundamentais em que ela assenta.

No terceiro capítulo é explicada em maior detalhe a tecnologia 4DIAC. São analisados as duas componentes da ferramenta (4DIAC-IDE e FORTE) e expõe-se o recurso sob o ponto de vista do utilizador.

O quarto capítulo aborda a incorporação do protocolo Modbus TCP na ferramenta 4DIAC. É feita uma breve referência do protocolo, discutem-se as diferentes possibilidades consideradas para resolver o problema e documenta-se os processos de implementação, teste e validação da solução.

No quinto capítulo encontra-se documentado todo o processo de geração de uma solução final que responda ao desafio que dá título à dissertação. Esse processo inclui a análise e confrontação de opções, a definição de arquitecturas e a implementação e validação dessas arquitecturas.

Por último, no sexto capítulo é analisado o trabalho efectuado e são discutidos resultados obtidos. Perspectiva-se ainda o trabalho futuro que pode suceder ao projecto.

Capítulo 2

Norma IEC 61499

2.1 - Introdução

O primeiro passo para entender o *standard* IEC 61499 é compreender o contexto em que a sua criação se insere. A antiga norma IEC 61131-3, orientada para a programação de PLCs, apresenta limitações que chocam com as necessidades emergentes do universo industrial. Os sistemas estruturados de acordo com esta norma assentam num modelo de controlo centralizado com um tipo de execução cíclica. Uma arquitectura deste género, para além de desaproveitar as potencialidades das redes de comunicação (entre outros recursos), concede às aplicações pouca flexibilidade. Outro factor limitativo prende-se com a dependência que este tipo de sistemas apresenta ao nível de *hardware*. Essa dependência leva a que a interligação de PLCs de diferentes fabricantes seja problemática e impeça frequentemente a portabilidade de *software* [13].

Na perspectiva de suplantar estas restrições e dificuldades surge a norma IEC 61499. Os sistemas estruturados segundo esta nova norma possuem uma série de características que permite que se fale numa nova geração de processos de controlo distribuído e de automação, nomeadamente:

- Interoperabilidade: capacidade de dispositivos de diferentes marcas operarem juntos para executar as funções especificadas por uma ou mais aplicações distribuídas, garantindo ao programador que desenha o sistema abstracção do *hardware*;
- Portabilidade: capacidade das ferramentas de *software* em aceitarem e interpretarem correctamente bibliotecas produzidas por outras ferramentas de *software*;
- Modularidade: tipo de estruturação que oferece uma relativa independência entre as unidades que constituem um sistema, permitindo a reutilização de *software*;
- *Event-driven*: A execução dos algoritmos é orientada a eventos, sendo devidamente apoiada por um fluxo de dados.

Estas novas faculdades fazem parte de um grupo maior de características inerentes aos sistemas gerados segundo a norma IEC 61499, características essas que fazem deste tipo de sistemas ferramentas ideais para o desenvolvimento de projectos nas mais diversas áreas de controlo e supervisão [13]. Os sectores da indústria e da domótica são algumas dessas áreas.

2.1 - Referência

Em termos concretos, o *standard* [2] define modelos que são usados na especificação, modelização e implementação de sistemas de controlo distribuídos. De seguida são descritos alguns desses modelos, nomeadamente aqueles que mais directamente dizem respeito aos utilizadores finais (estes utilizadores são os programadores que pretendem criar um sistema segundo o *standard* IEC 61499). Os modelos descritos são: o sistema, o dispositivo, o recurso, as aplicações e o Function Block (FBs).

2.1.1 - Sistema

De acordo com a norma, um sistema consiste numa série de dispositivos interligados e comunicantes entre si através de redes de comunicação que consistem em segmentos e ligadores. Os dispositivos encontram-se ligados aos segmentos através de ligadores. Ainda segundo a norma, a parte funcional do sistema é assegurada por uma ou mais aplicações que podem estar contidas em um ou mais dispositivos [2]. Conforme é ilustrada na Figura 2.1, um sistema possui um ou mais dispositivos que contêm uma ou mais aplicações que actuam sobre um determinado processo (*“Controlled Process”*), possuindo ainda a citada rede de comunicação para que os diferentes dispositivos possam trocar dados entre si. De notar ainda que nem os segmentos de comunicação, nem os dispositivos, nem os processos a controlar se encontram restringidos a determinado tipo de tecnologia (tratam-se de simples modelos).

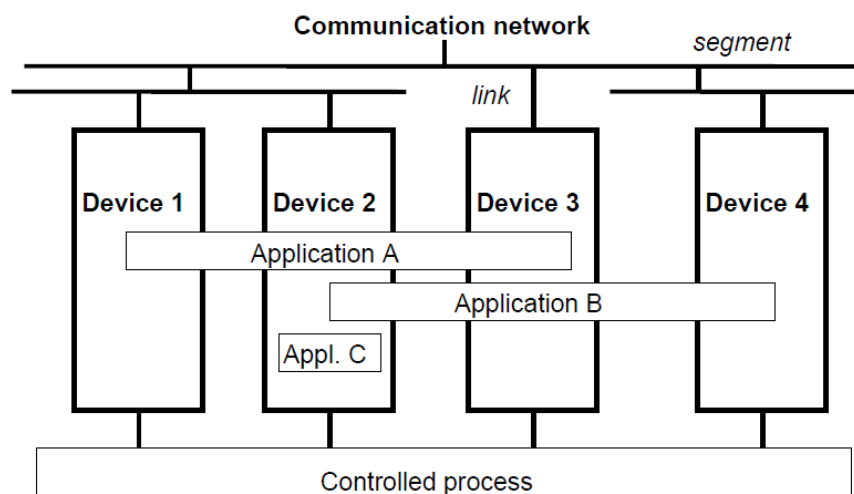


Figura 2.1 - Modelo de sistema IEC 61499 [2]

2.1.2 - Dispositivo

Um dispositivo consiste num elemento atómico da configuração de um sistema [10]. Trata-se de um meio físico é usado para albergar os recursos aonde executam as aplicações, conforme é ilustrado na Figura 2.2, oferecendo-lhes uma interface com pelo menos uma de duas entidades: rede de comunicação e processo a controlar.

2.1.3 - Recurso

Um recurso pode ser visto como uma unidade funcional, disponibilizada por dispositivo, que tem um controlo independente da sua operação. Pode ser criado, configurado, parametrizado, iniciado, apagado sem que isso afecte outros recursos do mesmo dispositivo [2]. Esta entidade fornece as ferramentas necessárias para que as aplicações possam ser executadas.

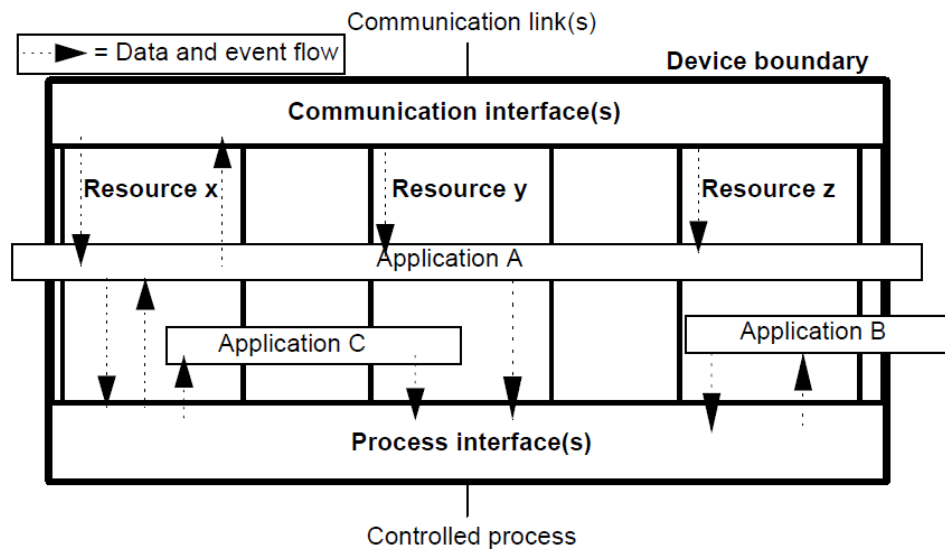


Figura 2.2 - Modelo de dispositivo IEC 61499 [2]

2.1.4 - Function Block

Um FB, segundo a própria norma IEC 61499, é a entidade básica de um componente de *software* portátil [2]. Cada FB comporta uma interface de eventos e dados de entrada e saída (Figura 2.3). Os eventos de entrada são usados para receber pedidos de execução de serviços facultados pelo FB, enquanto os de saída sinalizam o fim da execução desses pedidos. Com esta configuração é possível construir aplicações ao ligar eventos de saída a eventos de entrada, de modo a ser criada uma sequência lógica de execução. Já os dados de entrada e saída são usadas para partilhar informação entre FBs, estando cada uma deles invariavelmente associada a pelo menos um evento. Essa associação implica que quando um evento de entrada for recebido ou quando um evento de saída for emitido os dados associadas ao respectivo evento serão lidas (entrada) ou escritas (saída). Os tipos de dados disponíveis são idênticos aos que foram já tinham sido estipulados para a norma IEC 61131-3, nomeadamente os tipos inteiro, booleano e *string*, entre outros.

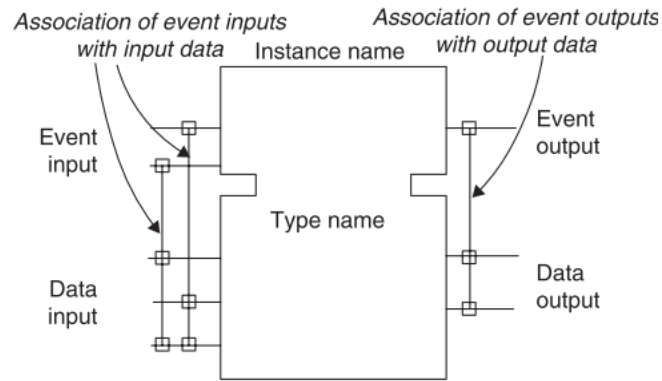


Figura 2.3 - Modelo representativo de um Function Block genérico [2]

Existem 3 tipos de FBs: Basic Function Block (BFB), Composite Function Block (CFB), Service Interface Function Block (SIFB), sendo estes de seguida sumariamente apresentados.

Basic Function Block

O tipo de FBs mais comum é o Basic Function Block, cujo modelo se encontra ilustrado na Figura 2.4. Para além de possuírem os citados *inputs* e *outputs*, cada BFB contém obrigatoriamente um *Execution Control Chart* (ECC), podendo ainda albergar variáveis internas, que são variáveis específicas do FB e estão totalmente protegidas (não acessíveis a partir do exterior), e algoritmos de execução.

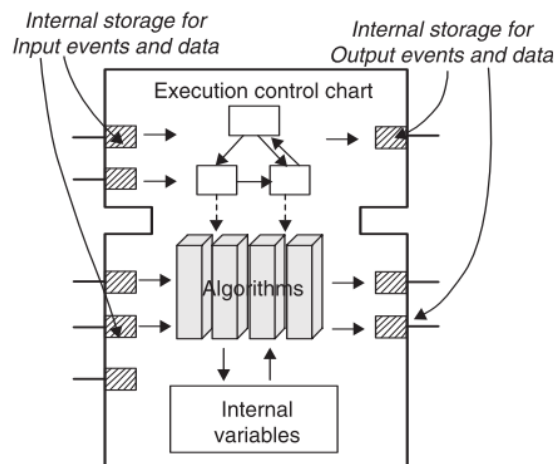


Figura 2.4 - Modelo representativo de um Basic Function Block [10]

Um ECC apresenta um modo de funcionamento do tipo máquina de estados e a sua arquitectura contempla estados, transições e acções. As acções encontram-se associadas a estados (obedecendo à relação “n” para “n”, ou seja, um estado pode possuir mais que uma acção e uma acção pode ser utilizada por mais do que um estado) e correspondem à evocação dos algoritmos do FB. As transições estabelecem ligações selectivas entre estados (através de condições booleanas), permitindo actualizar o estado activo. Existe um estado eleito como inicial e, de acordo com o modo de funcionamento implementado, só pode existir um único estado activo a cada momento, pelo que é necessário que todas as transições que saem de um estado sejam mutuamente exclusivas.

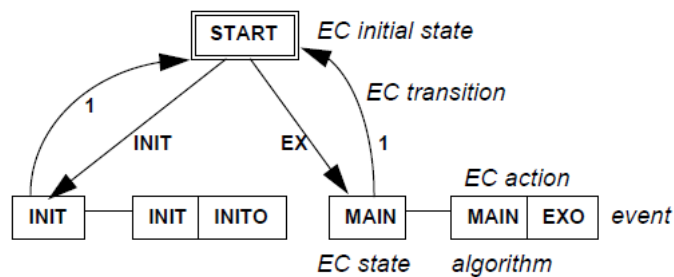


Figura 2.5 - ECC exemplo [2]

A Figura 2.5 representa um ECC exemplo. De notar que a cada algoritmo corresponde um evento de saída, o que na prática significa que, por exemplo, sempre que o algoritmo “INIT” terminar a sua execução é emitido um evento do tipo “INITO”.

Na Figura 2.6 é esquematizado o modo de funcionamento do ECC sob o ponto de vistas de máquina de estados. Inicialmente este encontra-se em repouso (“s0”), e assim que recebe um evento de entrada (“t1”) segue para um estado em que avalia a transições disponíveis (“s1”). Caso nenhuma das condições associadas às transições seja satisfeita (“t2”) o ECC volta a entrar em modo de espera (“s0”). Caso uma das transições seja permitida (“t3”) o estado do outro lado da transição passa a ser o activo e são executados todas as acções associadas a ele (“s2”). Uma vez finalizadas as acções (“t4”) voltam a ser avaliadas as transições (“s1”), fechando desta forma o ciclo em que consiste a máquina de estados.

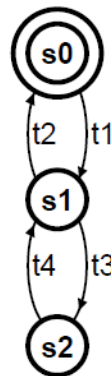


Figura 2.6 - Máquina de estados do ECC [2]

Os algoritmos contidos no FB podem ser escritos em qualquer linguagem de programação, desde que esta seja suportada pelo recurso que a executa. Perante um pedido de execução, um algoritmo deve correr um número finito de instruções, podendo recorrer à leitura das variáveis internas e aos dados de entrada e saída do FB, com as variáveis internas e os dados de saída a poderem ser ainda escritos.

Composite Function Blocks

Outro género de FBs são os Composite Function Blocks. Os elementos deste tipo, representados genericamente na Figura 2.7, denominam-se compostos por serem o resultado de uma agregação de instâncias de FBs (FBs componentes). Não possuem variáveis internas, nem ECC, nem algoritmos. Ao invés, possuem uma rede interna de FBs componentes

interligados entre si para que o CFB ofereça um serviço específico que seria impossível de oferecer por um único BFB. Essa rede consiste na união entre eventos e dados de entrada e saída. Deve ser referido que dentro dessa rede é impossível que múltiplos FBs sejam activos simultaneamente (a execução do CFB tem de ser linear), o que é a mesma coisa que dizer que um evento de saída não pode ser direccionado para mais do que um evento de entrada.

Service Interface Function Blocks

Os FBs do tipo Service Interface Function Blocks diferem dos restantes por possuírem um único algoritmo no seu interior (não possuem ECC nem as tradicionais variáveis internas). Esse algoritmo é responsável por lidar com os eventos de entrada, por actualizar as variáveis de

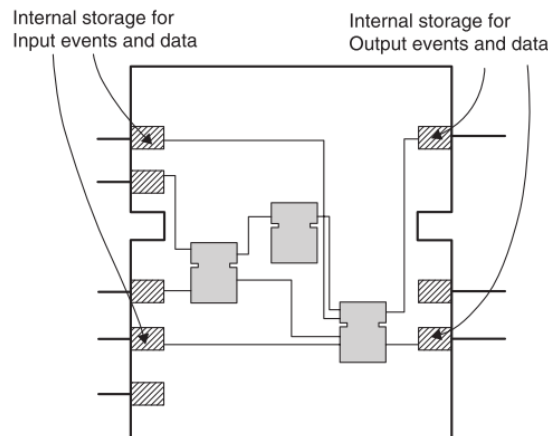


Figura 2.7 - Arquitectura de um Composite Function Block genérico [2]

saída e por emitir os eventos de saída. Para executar tarefas ele pode recorrer à leitura das variáveis de entrada.

Este género de FBs tem por função proporcionar às aplicações uma interface com os serviços de comunicação e de processo fornecidos pelo dispositivo aonde são inseridos [2]. A principal razão pela qual os BFBs não são usados para executar estas funções prende-se fundamentalmente com o grau de liberdade que elas exigem por parte do FB que as gere. Um algoritmo contido num SIFB pode a qualquer momento gerar um evento de saída (acusando, por exemplo, a recepção de uma mensagem de rede), ao passo que num tradicional BFB isso não seria possível, visto que este obedece a uma estrutura rígida dirigida pelo ECC.

A Figura 2.8 ilustra uma unidade SIFB genérica do tipo REQUESTER, que é propositadamente descrita na própria especificação da norma para ser usada como exemplo nos SIFBs que implementem serviços ligados à rede de comunicação.

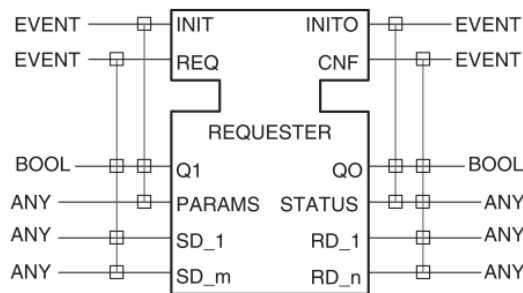


Figura 2.8 - SIFB REQUESTER genérico [10]

A juntar aos dois grupos de serviços mencionados acima (comunicação e processo), existe ainda um outro tipo de serviço que pode ser fornecido através dos SIFBs, nomeadamente os interfaces gráficos. De seguida são descritos em maior detalhe cada um destes serviços.

- *Graphical User Interface (GUI)*: Serviço de HMI, providenciando supervisão e controlo sobre as aplicações. Este serviço é essencial no caso, por exemplo, de se implementar um SCADA baseado em IEC 61499;

- *Gestão de comunicação*: Serviço que inclui os SIFBs que asseguram as comunicações entre diferentes dispositivos, fazendo uso das capacidades das redes de comunicação que os ligam. Para além de fornecerem operações comunicacionais de baixo nível, também podem implementar protocolos de cooperação de alto nível (cliente-servidor, produtor-consumidor, entre outros);

- *Gestão de processo*: Serviço de que fazem parte os SIFBs em implementam *device drivers* de todo o tipo de recursos electrónicos disponibilizados pelo dispositivo, permitindo a utilização destes por parte das aplicações. Estes recursos podem ir desde simples barramentos de sinais I/O a componentes de *software* que não fazem parte do sistema IEC 61499 (ficheiros, aplicações externas, entre outros) [14].

É importante perceber que, por princípio, os SIFBs não são implementados pelo utilizador básico. Eles devem ser fornecidos pelas ferramentas de desenvolvimento (ou outras entidades envolvidas) de modo ao utilizador os poder usar directamente numa aplicação. Como eles são altamente dependentes do ambiente e da ferramenta que os usam é natural existirem diferentes SIFBs que implementam a mesma funcionalidade, cabendo ao utilizador usar aqueles que se adequam ao ambiente/ferramenta de desenvolvimento de destino.

2.1.5 - Aplicações

Segundo a norma, uma aplicação é uma rede de instâncias de FBs cujos dados e eventos de entrada e de saída estão interconectados [2]. Assim, e de acordo com o que já foi referido anteriormente, para se construírem aplicações conectam-se diferentes instâncias de FBs, de modo a obter os já indicados fluxos de eventos e dados.

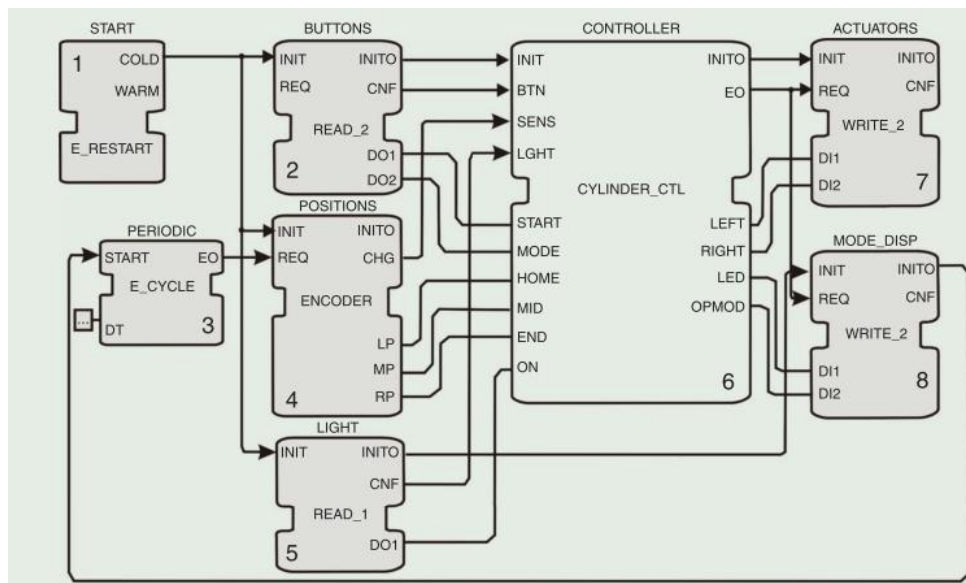


Figura 2.9- Aplicação desenvolvida para o controle de um cilindro [15]

A Figura 2.9 representa um exemplo concreto de uma aplicação inserida num sistema documentado em [15]. De notar que uma aplicação não representa um produto final pois não estipula a configuração do sistema ao nível dos dispositivos e recursos utilizados. Um sistema só se encontra pronto para ser utilizado quando todos os FBs instanciados são mapeados em dispositivos e recursos, tomando em atenção que para se estabelecer comunicação entre dois FBs de diferentes dispositivos ou recursos é necessário utilizar SIFBs específicos como intermediários.

Capítulo 3 4DIAC

3.1 - Introdução

Esta secção pretende ser uma descrição exhaustiva da tecnologia 4DIAC, realçando as suas capacidades e limitações e a sua manipulação do ponto de vista de utilizador. O capítulo começa com uma primeira secção aonde é explicado o modo de funcionamento da ferramenta, numa espécie de apresentação da tecnologia 4DIAC. De seguida são descritas e analisadas as duas componentes que formam a ferramenta. Por último são documentados os testes com vista a validar algum trabalho de implementação que foi desenvolvido no âmbito da abordagem à tecnologia 4DIAC e é feita uma conclusão para resumir o que foi apresentado nesta secção.

3.2 - Modo de funcionamento

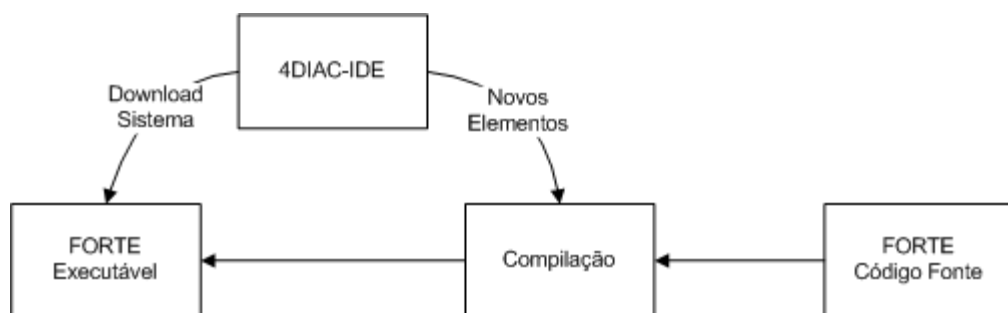


Figura 3.1 - Diagrama de funcionamento da *framework* 4DIAC

O diagrama da Figura 3.1 retrata de forma sumária o modo de funcionamento da ferramenta 4DIAC.

O FORTE consiste num *runtime* cuja função é suportar a execução de sistemas IEC 61499 (o conceito de sistema está descrito na secção 2.1.1 - Sistema). Na prática, o FORTE é o programa para onde são transferidas as aplicações que compõe um sistema, de forma a estas poderem ser executadas. Para que as aplicações sejam executadas é necessário que o FORTE contenha na sua biblioteca (a arquitectura do FORTE será explicada numa das secções seguintes) as implementações de todos os tipos de FBs usados por essas aplicações, sendo que essas implementações são adicionadas ao programa no momento da sua compilação (na figura

essa implementações encontram-se representadas pelos “Novos elementos” e pelo “FORTE - Código Base”). Daí que a compilação do FORTE seja um processo crítico na arquitectura de funcionamento da ferramenta.

A compilação do FORTE é feita utilizando 3 *inputs*: o código fonte básico do FORTE, as configurações próprias da plataforma *target* para onde se pretende compilar o *runtime*, e os ficheiros exportados pelo 4DIAC-IDE que implementam novos FBs criados pelo utilizador. O código fonte base inclui as implementações em C++ de dispositivos, recursos e FBs disponibilizados pela iniciativa 4DIAC. Ao serem instanciadas, estas implementações possibilitam a criação de aplicações, senão vejamos: uma vez em execução, o FORTE pode instanciar uma implementação de um dispositivo, contendo no seu interior uma instância de um recurso. Por sua vez, nesse recurso são instanciados os FBs necessários para desenhar uma aplicação. Este exemplo descreve de forma genérica o processo de transferência de aplicações do IDE para o *runtime* (na figura este processo está identificado como “Download Sistema”), processo esse que, logicamente, é completamente automático, de acordo com o que vai ser explicado na secção dedicada ao funcionamento do FORTE.

O segundo *input* usado na compilação refere-se ao parâmetros necessário fornecer ao processo de compilação para que a versão executável do FORTE possa correr na plataforma de destino desejada, seja ela um típico PC com Windows ou um sistema embebido com Linux. Logicamente que uma versão do FORTE compilada para uma plataforma Windows não vai funcionar numa plataforma Linux.

O último *input* referido diz aos ficheiros exportados pelo IDE. Estes ficheiros consistem na implementação em código C++ dos tipos de FBs desenhados no utilizador no IDE, podendo estes FBs serem de todos os tipos (básico, *service interface* ou composto). Para além de tipos de FBs, podem ser adicionados igualmente ao processo de compilação implementações de recursos e dispositivos, mas estas têm de ser desenvolvidas sem recurso ao IDE, pois ele não suporta esse tipo de criações. Todas estas implementações, após compilação, são utilizadas para enriquecer a biblioteca do FORTE. O que significa que, se o utilizador necessitar criar um novo tipo de FB para utilizar numa qualquer aplicação é necessário que o FB seja exportado e o seu código fonte C++ seja incorporado numa nova compilação do FORTE, ou seja, se um novo tipo de FB implica sempre uma nova compilação.

No topo da imagem encontra-se representa a aplicação 4DIAC-IDE. Conforme o que foi indicado, esta componente permite criar novos tipos de FBs, que podem ser posteriormente exportados sob o formato de código C++. Paralelamente, o 4DIAC-IDE permite ao utilizador criar sistemas baseados na norma IEC 61499 e transferir as aplicações que os constituem para um programa FORTE em funcionamento, de forma a serem executadas. Conforme vai ser explicado nas secções seguintes, a criação de sistemas consiste do desenvolvimento de aplicações e na definição de configurações, enquanto a transferência de aplicações envolve uma comunicação entre o FORTE e o 4DIAC-IDE de modo a este último enviar os pedidos necessários para que sejam criadas as instâncias que compõem, as aplicações.

De seguida são descritas em maior pormenor as duas componentes que formam a ferramenta 4DIAC.

3.3 - 4DIAC-IDE

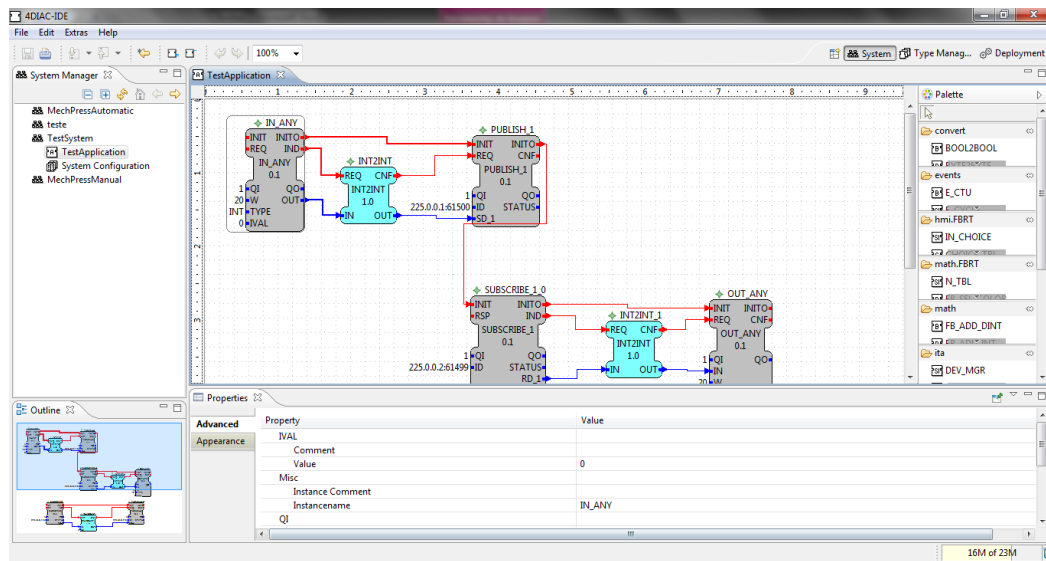


Figura 3.2 - Screenshot do modo de edição de aplicações do 4DIAC-IDE

O FORDIAC-IDE é um programa que permite a edição, importação e exportação de sistemas e elementos que seguem a norma IEC 61499. Por elementos entende-se as unidades FBs, recursos e dispositivos. Por sistema entende-se o conjunto de uma ou mais aplicações com a respectiva configuração. Para além dessas funções também permite transferir os sistemas para o *runtime*, de modo a eles serem executados.

Após uma secção aonde é explicada a forma de obter e instalar o 4DIAC-IDE, segue-se uma exposição da maneira como o programa permite gerir os elementos mencionados acima e criar sistemas.

3.3.1 - Instalação

O 4DIAC-IDE, tal como o FORTE, pode ser obtida gratuitamente na internet através das páginas oficiais da iniciativa. Existe uma versão executável do programa disponível no site www.fordiac.org sendo que, à altura da escrita desta dissertação, a versão mais recente é a v0.3. No entanto esta versão já se encontram quase obsoletas quando comparadas com aquela fornecida pelo repositório Mercurial da iniciativa [16]. A plataforma Mercurial é uma ferramenta de gestão de código fonte que é usado pelo projecto 4DIAC para controlo de versões. O conteúdo do repositório é actualizado a um ritmo quase diário, o que é sinónimo de dizer que todos os dias é gerada uma nova versão do programa. O repositório Mercurial está intimamente relacionado com a página Sourceforge da iniciativa [17], com esta a providenciar toda a documentação associada à gestão do código fonte, para além de ser o ponto de contacto entre os vários programadores da iniciativa. O website assegura também um fórum aonde os programadores podem discutir ideias e os próprios utilizadores ou curiosos podem expor dúvidas. A página sourceforge possibilita também ao utilizador subscrever a *mailing list* da iniciativa, passando este a receber notificações via correio electrónico com as mais recentes alterações aos códigos fonte dos programas. É assim aconselhado ao utilizador utilizar o código fonte disponível no repositório e não a versão disponível no *website* oficial.

Para obter o código fonte do repositório Mercurial é usada a ferramenta Tortoise, em Windows, ou o programa de linha de comandos Hg, em Linux. No primeiro caso o utilizador deve usar a opção “Clone” numa pasta de destino e usar o endereço <http://fordiac.hg.sourceforge.net/hgweb/fordiac/fordiac-ide> como fonte. No segundo deve usar o comando `$hg clone http://fordiac.hg.sourceforge.net/hgweb/fordiac/fordiac-ide` na pasta de destino. Em ambos os casos é criada uma cópia local do conteúdo do repositório.

Para executar o 4DIAC-IDE apenas é necessário ter no computador o seu código fonte e a versão “Modeling Tools” da ferramenta Eclipse. A página Sourceforge fornece um tutorial relativamente completo sobre este processo, pelo que o leitor é remetido para lá se desejar uma explicação mais extensa. Muito sucintamente, após importar o projecto 4DIAC-IDE para o Eclipse, o utilizador apenas necessita de abrir a secção “4DIAC-IDE.product” do *plugin* “org.fordiac.ide” e adicionar as dependências que não foram anexadas automaticamente. De seguida escolhe as opções “sincronizar” e “executar aplicação Eclipse” e como consequência é iniciado o processo de *loading* do 4DIAC-IDE.

3.3.2 - Utilização

Function Blocks

O FORDIAC-IDE possui algumas bibliotecas pré-existente que cobrem as operações mais básicas ao nível de FB e que, logicamente, já são suportadas pelo código fonte básico do FORTE. A lista seguinte caracteriza cada uma dessas bibliotecas.

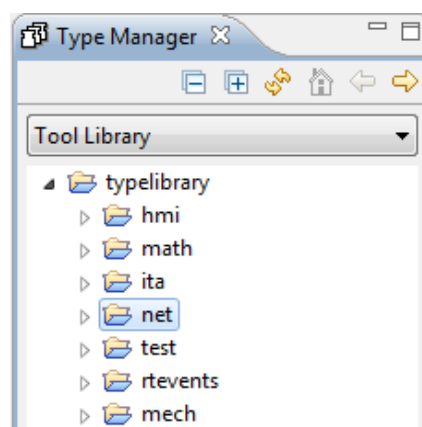


Figura 3.3 - Lista de bibliotecas de FB disponíveis por defeito

- *Interface* (“hmi”): Esta biblioteca tem origem, não no projecto 4DIAC, mas antes na iniciativa HOLOBLOC, Inc [16]. Os SIFBs que a constituem servem de ferramentas para a geração de uma *interface* homem-máquina (HMI), para efeitos de supervisão e controlo de sistema.

- Operações matemáticas (“math”): Cobre apenas as 4 operações básicas (adição, subtração, divisão e multiplicação), aceitando como dados variáveis de vários tipos. As operações matemáticas mais complexas não são contempladas por esta biblioteca.

- Gestor de Dispositivos (“ita”): Esta biblioteca não está destinada a ser utilizada directamente pelo utilizador. Nela estão contidos SIFBs que são utilizados para transferir as aplicações do IDE para o *runtime*.

- Comunicações (“net”): Os SIFBs desta biblioteca implementam os modelos de cooperação Publicador-Subscritor e Cliente-Servidor. Podem ser usados directamente pelas aplicações para trocar mensagens e dados entre diferentes dispositivos ou recursos.

- Encriptação (“test”): Pequena biblioteca que usa um par de FBs para encriptar e desencriptar mensagens. Esta encriptação é bastante simples, pelo que deve ser usada com reservas.

- Eventos (“events” e “rtevents”): Esta biblioteca assegura todas as operações que poderão ser necessárias ao nível da manipulação de eventos. É possível, entre outras funcionalidades, fundir ou dividir eventos, gerá-los periodicamente, inserir *delays* ou seleccionar um evento entre vários. No caso do grupo “rtevents” são aplicadas restrições temporais características dos sistemas tempo-real.

- Outros (“mech”): Grupo de elementos que é usado pelo projecto de demonstração que vem por defeito com o IDE. Como os elementos têm funcionalidades muito específicas, dificilmente serão utilizados noutros projectos.

Todas as unidades contidas nestas bibliotecas são reutilizáveis, e grande parte delas é mesmo editável a todos os níveis (ECC, algoritmos, entradas e saídas, etc), criando-se assim a possibilidade de serem gerados novos FBs com origem nesses FBs *standard*.

A outra maneira de se gerar um tipo de FB é criá-lo a partir do zero. O 4DIAC-IDE suporta a criação dos tipos de FBs Basic, Service *Interface* e Composite FB. Este modo de criação permite o controlo total sobre o FB gerado, sendo possível definir os eventos e dados de entrada e saída e a associação entre eventos e dados. Para além destas funções existem outras funcionalidades que são facultativas de acordo com o tipo de FB em causa. No caso dos BFBs é possível criar variáveis internas, definir o ECC e escrever os algoritmos a ele associados. Para os SIFBs é possível definir as sequências de serviço (*service sequences*) que indicam o seu modo de funcionamento. Para os CFB o utilizador pode criar a sua própria rede de instâncias FB, de acordo com o que está estipulado na norma. Para todos os casos existem também espaços próprios para criar comentários direccionados ao esclarecimento do papel de cada um dos componentes do FB.

De notar que, depois de criados, todos os novos FBs têm de ser exportados para o FORTE (de acordo com o que já tinha sido mencionado) para permitir que os sistemas que os utilizem possam executar.

Aplicações

O 4DIAC-IDE proporciona um ambiente bastante *user friendly* para a criação de aplicações, permitindo criar instâncias de FBs, proceder à ligação entre dados e entre eventos, podendo os dados de entrada ser parametrizados directamente pelo utilizador.

Outras capacidades do programa, relativas ao desenvolvimento de aplicações, prendem-se com faculdades secundárias, como por exemplo poder escolher a cor e modificar o nome de uma instância FB, esconder/mostrar as ligações de eventos/dados para proporcionar diferentes perspectivas sobre a aplicação, utilizar o método *drag-and-drop* para criar as instâncias, entre outros.

Dispositivos e Recursos

O 4DIAC-IDE possui dispositivos e recursos genéricos, parametrizáveis mas não editáveis (pelo menos não editáveis através do IDE). Os dispositivos podem ser de dois tipos: *Remotly Managed Device Type* (“RMT_DEV”) ou *Remotly Managed Window* (“RMT_FRAME”). O primeiro representa um normal dispositivo de processamento, enquanto o segundo implementa um centro de supervisão. Já os recursos podem pertencer a 4 categorias: recurso embebido (“EMB_RES”), recurso para painel gráfico (“PANEL_RESOURCE”), recurso remoto (“RMT_RES”) ou um recurso para armazenamento de elementos visual (“VIEW_PANEL”).

Canais de Comunicação

No 4DIAC-IDE um sistema completo é constituído por uma ou mais aplicações e por um modelo de configuração. Neste último é possível relacionar os diversos recursos/dispositivos entre si usando canais de comunicação (ou, segundo a designação utilizada pelo próprio programa, segmentos). Os seguintes tipos segmentos estão disponíveis: DeviceNet, EIP, Ethernet, CAN e PROFINet. A ferramenta 4DIAC está construída de tal forma que possibilita ao utilizador abstrair-se do tipo de rede de comunicação a utilizar ao longo do processo de arquitectura das aplicações. Assim, os SIFB de comunicação podem ser usados para qualquer tipo de canais. Logicamente a plataforma física em se pretende desenvolver necessita de possuir o *hardware* necessário para implementar o tipo de comunicação desejada.

3.3.3 - Importação/ Exportação

O IDE permite exportar e importar sistemas e componentes de modo a facilitar a partilha e o desenvolvimento dos mesmos. Essas funcionalidades são asseguradas pelas opções “Import” e “Export” do menu “File”. No primeiro caso o utilizador deve escolher o tipo de importação a efectuar (dispositivo, FB, recurso, segmento ou sistema) e localizar os ficheiros em causa, sendo que cada tipo de importação envolve um formato de ficheiro diferente (“.dev” para dispositivos, “.res” para recursos, “.fbt” para FBs, “.seg” para segmentos, “.sys” para sistemas). De referir que o *website* oficial da iniciativa possui para download um sistema exemplo que acompanha a versão executável do IDE o qual integra já uma série de componentes que devem ser importados, pois essa série incluir os dispositivos, recursos e segmentos e FBs já suportados pelo FORTE.

O processo de exportação assegura a geração dos ficheiros referidos acima, bastando ao utilizador seleccionar o tipo de exportação e escolher o elemento a exportar que o programa se encarrega de gerar os ficheiros apropriados.

3.3.4 - Desenvolvimento de um Sistema

Um sistema é construído no IDE através do desenvolvimento das aplicações necessárias e na definição das configurações. A primeira parte já foi analisada, enquanto a segunda pode ser explicada recorrendo à Figura 3.4.

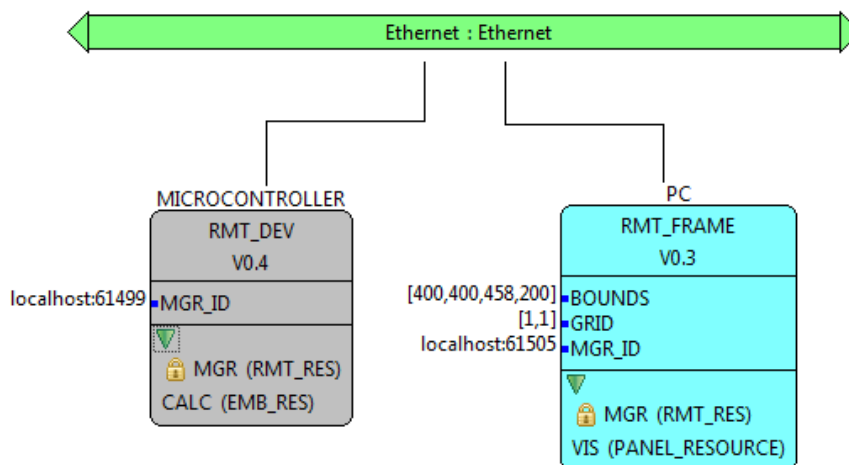


Figura 3.4 - Configuração de um sistema teste

A Figura 3.4 representa parte da configuração de um sistema teste que acompanha a versão executável do 4DIAC-IDE disponível para *download* no site da iniciativa. É possível identificar dois dispositivos (“MICROCONTROLLER” e “PC”), cada um com um recurso (“CALC” e “VIS”, respectivamente). O parâmetro “MGR_ID” é usado para indicar o código IP e a porta de escuta que corresponde ao dispositivo em causa, de modo ao IDE saber para quem deve transferir o sistema de modo a este poder ser executado. A rede Ethernet é usada para indicar de que forma os dispositivos estão interligados. Passando de um exemplo concreto para a generalidade, o utilizador pode escolher o número e o número de dispositivos que compõem o seu sistema, o número e o tipo de recursos que cada um dos dispositivos possui, o tipo de rede que interliga os dispositivos, e os endereços IP e as portas de escuta de cada um deles.

A configuração de um sistema só fica completa com o mapeamento dos FBs nos recursos. O utilizador deve indicar em cada instância de FB que faz parte das aplicações do sistema a que recurso está associado, usando para isso a opção “Map”/”Unmap”. Deve ser lembrado que as instâncias FBs alocadas em diferentes recursos e que se encontram ligadas directamente de alguma (evento ou variável) devem estar intermediadas por SIFBs de comunicação. Esta imposição reforça o pressuposto que o desenvolvimento da parte funcional do sistema pode ser feito de forma independente da configuração do sistema.

3.3.5 - Transferência de aplicações

Após um sistema ser devidamente desenhado e configurado, as aplicações que o compõem necessitam de ser transferidas para o FORTE de forma a poderem ser executadas. Para além do FORTE, o IDE permite também transferir o sistema para o *runtime* providenciado pela iniciativa HOLOBLOC, INC., denominado FBRT. Para mais informações sobre esta ferramenta, o leitor é remetido para o website oficial dessa iniciativa em www.holobloc.com.

O processo de *download* tem início na chamada para execução do *runtime*. Assumindo que foi correctamente configurado o caminho para o ficheiro executável (Extras>>Preferences>>4DIAC>>FORTE Preferences), o utilizador pode iniciar a execução do FORTE a partir do menu *Runtime Launcher*, escolhendo o código IP do dispositivo *target* (usualmente *localhost*, que é equivalente a 127.0.0.1) e a porta de escuta e confirmado a execução no botão *Launch FORTE*. Em caso de sucesso da operação surgirá uma mensagem de

confirmação, encontrando-se o programa a partir desse momento em modo de escuta. No caso de o sistema contemplar aplicações que se distribuem por mais do que um dispositivo é necessário criar mais execuções do *runtime*, sendo que as portas de escuta têm de ser diferentes umas das outras.

Conforme será explicado em maior detalhe na secção da descrição do FORTE, o *runtime* está preparado para receber e responder a uma série de mensagens pré-definidas as quais interpreta como pedidos. Cada pedido envolve uma acção e possíveis argumentos. Cada resposta indica se o pedido foi executado com sucesso ou não. No caso específico de uma transferência de aplicações (ou, como é chamado pela própria ferramenta, *deployment*) os pedidos contidos nas mensagens enviadas pelo IDE para o FORTE são geralmente do tipo criar instâncias de recursos e de FBs, criar ligação entre FBs ou parametrizar os dados de entrada de FBs, conforme é ilustrado pelo exemplo de relatório de transferência fornecido pelo IDE (Figura 3.5). A cada pedido enviado pelo IDE corresponde uma resposta que ou é um número série, para o caso de o pedido ser executado com sucesso, ou é um código de erro, para o caso de não ser executado com sucesso.



```

<!-- 127.0.0.1:61499: EMB_RES -->
<Request Action="CREATE" ID="0">
  <FB Name="EMB_RES" Type="EMB_RES"/>
</Request>

<Response ID="0"/>

<!-- 127.0.0.1:61499: EMB_RES -->
<Request Action="CREATE" ID="1">
  <FB Name="FB_ADD_INT" Type="FB_ADD_INT"/>
</Request>

<Response ID="1"/>

<!-- 127.0.0.1:61499: EMB_RES -->
<Request Action="WRITE" ID="2">
  <Connection Destination="FB_ADD_INT.IN1" Source="4"/>
</Request>

<Response ID="2">

```

Figura 3.5 - Excerto de um relatório de transferência do 4DIAC-IDE

3.4 - 4DIAC-RTE (FORTE)

Como é referido na introdução, o FORTE consiste num *runtime environment* multi-thread e portátil passível de ser executado tanto em PC como num sistema embebido, tendo já sido testado em Windows Cygwin e Linux com i386, PPC e xScale, em PCs, e em NetOS e RTOS para IPC@chip e em eCos para ARM7 [5], em sistemas embebidos.

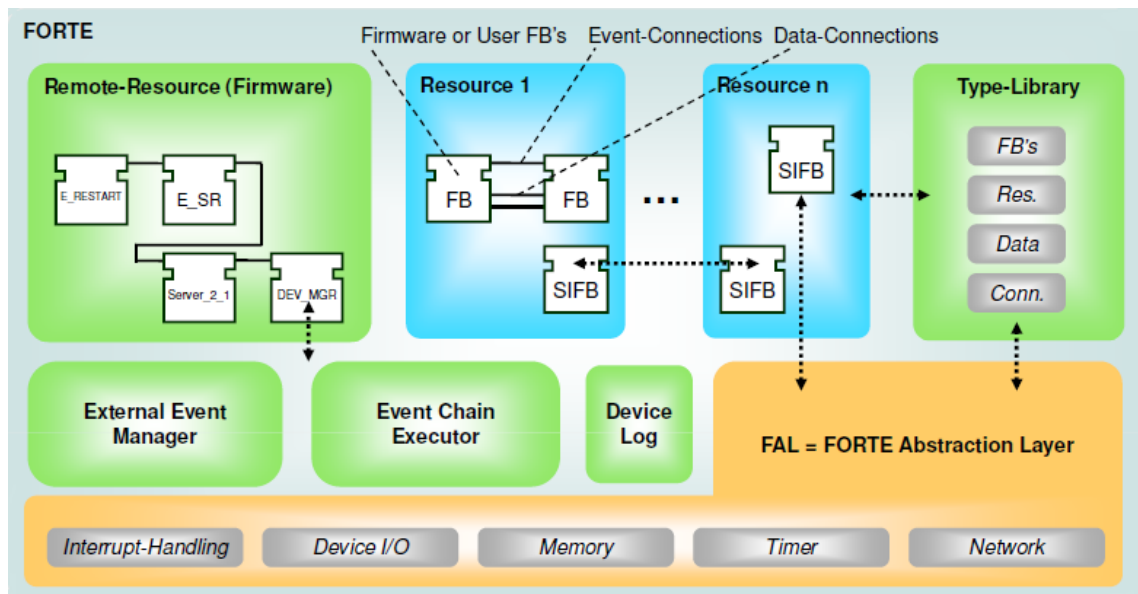


Figura 3.6 - Arquitectura do FORTE

Na prática, esta ferramenta está estruturada de forma a servir de suporte à execução de aplicações. De recordar que um sistema é um conjunto de aplicações mais uma configuração inerente ao sistema, pelo que, logicamente, só as aplicações é que são verdadeiramente executadas. O FORTE possui uma componente denominada FAL (*FORTE Abstraction Layer*) que lida com todas as funcionalidades de baixo nível fornecidas pelo sistema operativo aonde executa, nomeadamente gestão de comunicações, *threads*, memória, *interrupts*, entre outros, tal como é ilustrado pela Figura 3.6. Desta forma é fornecida às aplicações a abstracção necessária para que executem independentemente da plataforma em que estejam a correr. Conforme também é visível pela figura, o FORTE possui uma biblioteca aonde guarda os diferentes tipos de FBs, recursos, variáveis de dados e outros elementos passíveis de serem instanciados. As aplicações, que são recebidas através de um recurso (“Remote-Resource”) que é automaticamente instanciado após a iniciação do FORTE, são construídas através da criação, interligação e parametrização de instâncias de FBs contidos nessa biblioteca. Da mesma forma, podem ser instanciados recursos para conterem as aplicações. Por fim, o FORTE possui 3 módulos responsáveis pela execução das aplicações (“Event Chain Executor”), pela gestão dos *interrupts* fornecidos pela FAL (“External Event Manager”) e pela criação de relatórios dinâmicos para registo do funcionamento (“Device Log”).

3.4.1 - Compilação

Actualmente existem duas arquitecturas principais suportadas: tipo POSIX (inclui sistemas operativos que respeitem o *standard* POSIX, por exemplo o Linux) e tipo Win32 (para Windows). Existem ainda outras duas arquitecturas que correspondem a dois tipos de dispositivos singulares - plataformas da Digi e da Phytex - que, por se tratar de plataformas com sistemas operativos próprios, são de pouca utilidade. Logicamente que se pressupõe que a plataforma *host* (dispositivo aonde são feitos os processos de configuração e compilação) possui acesso a uma *toolchain* apropriada para proceder à compilação.

Apesar de ser possível executar a compilação através do Eclipse, é aconselhado utilizar o CMake [18], um *built system* que permite configurar as propriedades a conferir à versão

executável do FORTE. O CMake possui uma componente visual (CMake GUI) que oferece ao utilizador uma forma mais intuitiva de proceder à configuração, sendo aconselhado utilizar esta funcionalidade para o utilizador menos familiarizado com a ferramenta.

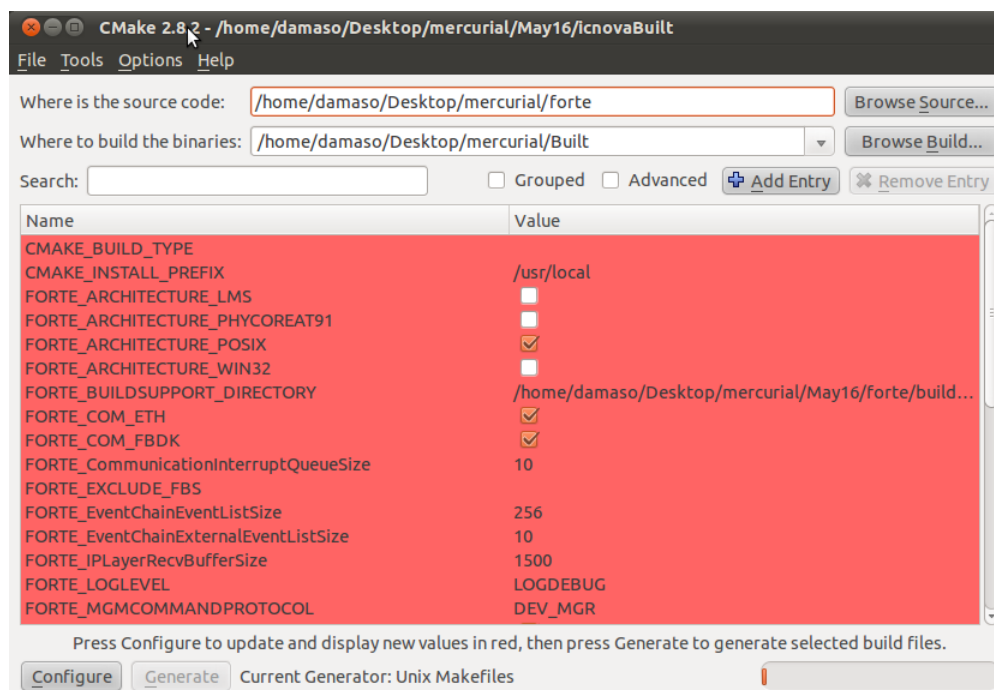


Figura 3.7 - Programa CMake GUI em execução

Ao abrir o código fonte do FORTE no CMake GUI verifica-se que é possível escolher o tipo de arquitectura da plataforma *target*, sendo esta a opção mais importante que a tomar no processo de configuração, pois todas as outras opções são secundárias e de natureza mais técnica. Por exemplo, se a plataforma alvo for um PC comum com sistema operativo Linux basta escolher a opção “FORTE_ARCHITECTURE_POSIX” e gerar os *Makefiles* para que se obtenha na pasta de destino todos os ficheiros necessários à compilação, restando usar o programa Make para gerar o ficheiro executável.

3.4.2 - Arquitectura do código

Em termos de arquitectura, o FORTE segue o conceito de *object-oriented programming*, fazendo uso de classes para representar entidades. Todos os elementos-tipo contidos na biblioteca mencionada na introdução deste capítulo se encontram modularizados, encontrando-se o código fonte agrupado em 3 grandes grupos, todos contidos na pasta “src”:

- Arquitectura (pasta “arch”), grupo que inclui o código fonte responsável pelas operações de baixo nível realizadas pelo programa, nomeadamente gestão de mensagens de *log*, do temporizador, da memória alocada dinamicamente e das ligações assentes em *sockets*. Como essas operações são dependentes do tipo de arquitectura da plataforma *target* o código está dividido entre uma parte base, comum a todas as arquitecturas, e uma parte específica, que implementa as alterações necessárias à parte comum para esta se adaptar a uma das arquitecturas em particular.

- Elementos Base (pasta “core”), grupo aonde estão incluídos os ficheiros que contêm as classes que utilizam os serviços fornecidos pelo grupo anterior e que implementam

conceitos inerentes à norma IEC 61499. Neste grupo estão contidos, por exemplo, as classes que implementam a estrutura FB do qual derivam todos os tipos de FBs criados, que efectuem a transferência de sistemas, que implementam os diferentes tipos de variáveis aceites pelos FBs, entre outros.

- Tipos de FBs (pastas “modules” e “stdfblib”), grupo que junta tipos de FBs já desenhados pela própria iniciativa 4DIAC e que já foram enumerados na secção anterior da apresentação do IDE.

O código do FORTE está estruturado de forma a permitir a geração de documentação através da ferramenta Doxygen. Este programa permite aproveitar comentários feitos nos próprios ficheiros de código de modo a criar documentos estruturados. Basta ao utilizador correr o comando \$doxygen na pasta principal do FORTE para que se gere um ficheiro denominado “index” na pasta “doc/api”. Este documento segue uma estrutura em árvore, com o utilizador a poder navegar por entre as diferentes classes que compõem a ferramenta.

Um outro documento que é importante para o utilizador é aquele que se encontra na pasta “doc/coding_rules”, denominado “forte_coding_rules”. Este PDF contém as normas que os percursos na iniciativa estabeleceram que devem ser respeitadas nos processos de criação de novos elementos ou edição de código. Contém também uma série de informações úteis que poderão ser importantes para o utilizador que começa a utilizar a ferramenta.

3.4.3 - Adicionar elementos à biblioteca

Conforme já foi referenciado neste capítulo, a criação de novos FBs no 4DIAC-IDE implica uma exportação destes para o *runtime*. Em termos práticos isso significa adicionar uma nova entrada num dos ficheiros “CMakeLists”, constantes nas pastas do FORTE associados aos tipos de FBs, com a localização e nome do elemento a adicionar. Tomando por exemplo o ficheiro associado à pasta “src/modules/math”, que contém os FBs responsáveis pelas operações aritméticas suportadas pela ferramenta, ilustrado na figura Figura 3.8, bastaria acrescentar o nome do FB à lista já existente usada como argumento às funções “forte_add_sourcefile_hcpp” (partindo do princípio que uma cópia dos ficheiros do FB se encontra na pasta). De notar que os novos tipos de FBs não são os únicos elementos passíveis de serem adicionados à biblioteca, pois novas implementações de dispositivos e recursos, entre outros, também podem ser incorporados.

```
forte_add_module(MATH "Mathematical Function Blocks")

#####
# Math FB
#####
forte_add_sourcefile_hcpp(FB_ADD_DINT FB_MUL_INT FB_SUB_DINT)
forte_add_sourcefile_hcpp(FB_ADD_INT FB_DIV_DINT FB_SUB_INT)
forte_add_sourcefile_hcpp(FB_DIV_INT FB_MUL_DINT)

if(FORTE_USE_LREAL_DATATYPE)
    forte_add_sourcefile_hcpp(FB_MUL_LREAL FB_DIV_LREAL FB_ADD_LREAL FB_SUB_LREAL)
endif(FORTE_USE_LREAL_DATATYPE)

if(FORTE_USE_REAL_DATATYPE)
    forte_add_sourcefile_hcpp(FB_MUL_REAL FB_DIV_REAL FB_ADD_REAL FB_SUB_REAL)
endif(FORTE_USE_REAL_DATATYPE)
```

Figura 3.8 - Conteúdo do ficheiro CMakeLists da pasta “src/modules/math”

3.4.4 - Código fonte dos Function Blocks

Apesar do documento Doxygen fornecer uma caracterização extensiva sobre todo o código fonte que compõe o FORTE, torna-se oportuno descrever com uma outra visão, menos técnica, o código que serve de base aos FBs, dado que para o utilizador básico o documento Doxygen pode ser algo confuso e pouco intuitivo. Como os novos FBs criados no IDE precisam de ser exportados e incorporados no FORTE, o utilizador é obrigado a lidar directamente com os seus códigos fonte para, no mínimo, conferir a inexistência de erros e comprovar a implementação dos algoritmos neles contidos. Os diferentes géneros de FBs são abordados de seguida.

Basic Function Blocks

Como a arquitectura do runtime está construída usando o conceito de object-oriented programming todos os tipos de FBs são definidos como classes. Tomando como exemplo a classe que implementa o tipo de BFB “ADD_INT”, verifica-se que ela contém uma série de variáveis estáticas que caracterizam o FB, nomeadamente número de inputs e outputs, nome das inputs e outputs, tipos das variáveis, relação eventos-variáveis, entre outros. Estas variáveis são de pouco interesse para o utilizador geral, visto o 4DIAC-IDE se encarregar de as parametrizar durante o processo de exportação do FB. As secções de código mais importantes são o conteúdo da função “executeEvent” e as funções que representam os algoritmos do FB.

```
void FB_ADD_INT::executeEvent(int pa_nEIID){
    bool bRetVal;
    do{
        bRetVal = true;
        switch(m_nECCState){
            case scm_nStateSTART:
                if(scm_nEventREQID == pa_nEIID)
                    enterStateREQ();
                else
                    bRetVal = false; //no transition cleared
                break;
            case scm_nStateREQ:
                if(1)
                    enterStateSTART();
                else
                    bRetVal = false; //no transition cleared
                break;
            default:
                DEVLOG_ERROR_2(CG_nID_CFunctionBlock, 1, "Invalid State");
                m_nECCState = 0; //0 is always the initial state
                break;
        }
        pa_nEIID = cg_nInvalidEventID; //clear after the first check
    }while(bRetVal);
}
```

Figura 3.9 - Função “executeEvent” da classe “FB_ADD_INT”

A função “executeEvent”, ilustrada na Figura 3.9, consiste na representação exacta em código C++ do ECC que comanda o modo de funcionamento do FB em causa, e é activada

aquando da recepção de um evento. O conteúdo desta função é automaticamente gerado de acordo com o que foi definido no IDE, sendo possível ao utilizador editar o código a seu entender. Essa edição deve ser efectuada com cautela, pois corre-se o risco de ir contra as especificações da norma.

Como é possível ver na figura, a função recebe como argumento um código (“pa_nEIID”) que indica qual o evento de entrada que acabou de ser recebido. A função confronta esse código com os códigos dos eventos disponíveis (tarefa que corresponde à verificação das condições booleanas das transições do ECC), e no caso de se dar uma correspondência é evocada a função responsável por actualizar o estado do ECC, por chamar os algoritmos associados às acções que esse novo estado contém, e por emitir os eventos de saída ligados a essas acções (funções “enterStateREQ” e “enterStateSTART”). A função “executeEvent” só termina a sua execução quando a variável “nRetVal” for igual a “FALSE”, o que é equivalente a dizer que nenhuma das transições do estado então activo é satisfeita.

Composite Function Blocks

Um FB do tipo composto possui, tal como um BFB, variáveis estáticas para descrever a sua interface. Possui ainda outras variáveis estáticas que são usadas para indicar quais os FBs componentes da sua rede interna e as ligações de eventos e dados entre eles.

Para que uma CFB seja correctamente adicionado ao FORTE é necessário que todos os seus FBs componentes estejam igualmente incorporados no runtime. Assim, o utilizador deve ter a precaução, quando cria um novo CFB, de verificar de todos os FBs componentes já se encontram na biblioteca do FORTE. Caso algum deles não esteja, é preciso incluí-los na nova compilação que é imposta pela criação de um novo CFB.

Service Interface Function Blocks

Antes de mais, deve referido que o desenho e implementação de SIFBs usualmente é feita pelas entidades responsáveis pela framework encarregada de os utilizar ou responsáveis pelos recursos a ser gerido pelos SIFB. Um exemplo de entidade responsável pelo framework seria a 4DIAC e um exemplo de uma entidade responsável pela gestão de recursos seria a Modbus-IDA, a associação que regula o protocolo Modbus. Neste sentido, a partir do momento em que o utilizador constrói os seus próprios SIFB deixa de ser um utilizador básico para ser alguém que expandiu as capacidades da framework que vai usar os SIFB.

Quanto ao código fonte de um SIFB exportado pelo IDE, e ao contrário dos outros tipos de FBs, com os SIFBs é esperado e imprescindível que o utilizador edite o código fonte. Eles não contêm funções para representação de algoritmos ou do ECC, pela simples razão de os SIFBs não possuírem nem algoritmos nem ECC. O que contêm é uma função “executeEvent”, que tal como no caso dos BFBs, só que desta feita a função se encontra vazia, apenas contendo o código necessário para identificar o evento que provocou a chamada da função.

Assim, é esperado que o utilizador preencha essa função com o código necessário para que o SIFB cumpra as funcionalidades desejadas, sendo possível recorrer, entre outros, à inclusão de bibliotecas para que essa tarefa seja cumprida. Como a norma e a própria ferramenta são extremamente flexíveis, as opções são vastas, pelo que a principal limitação a ter em conta é o papel que o SIFB deve desempenhar de acordo com a especificação da

norma: fornecer às aplicações interfaces com a rede de comunicação e com o processo a controlar.

```

void FORTE_TESTE::executeEvent(int pa_nEIID) {
    switch(pa_nEIID) {
        case scm_nEventINITID:
            #error add code for INIT event!
        /*
            do not forget to send output event, calling e.g.
            sendOutputEvent(scm_nEventCNFID);
        */
        break;
        case scm_nEventREQID:
            #error add code for REQ event!
        /*
            do not forget to send output event, calling e.g.
            sendOutputEvent(scm_nEventCNFID);
        */
        break;
    }
}

```

Figura 3.10 - Código fonte de um SIFB exemplo após a exportação feita pelo 4DIAC-IDE

A própria ferramenta oferece funcionalidades que podem ser utilizadas pelos SIFBs. Uma dessas faculdades é a criação de uma *thread* paralela que execute de forma independente, algo que é especialmente útil quando se deseja implementar um SIFB direccionado para a comunicação. Para que um SIFB utilize esta propriedade é necessário identificar a sua classe como sendo herdeira da classe abstracta “CThread”, implementar o método “run” e utilizar os APIs fornecidos para controlar a *thread* paralela. O método “run” consiste no código a ser executado por essa mesma *thread* paralela.

3.4.5 - Inicialização e Transferência

A evocação do FORTE é feita por recorrendo à linha de comandos (*shell*). Uma vez iniciado, o *runtime* verifica se o utilizador inseriu algum argumento na sua chamada. Se sim, esse valor será usado posteriormente na configuração do *socket* de escuta de mensagens de gestão. Caso esse argumento não exista são utilizados os valores pré-definidos (127.0.0.1 para IP - *localhost* - e 61499 para porta de escuta). Após essa verificação é criado um novo dispositivo do tipo “RMT_DEV” que vai albergar todos os recursos que forem necessários instanciar. Como consequência da criação do dispositivo é-lhe automaticamente associado um recurso do tipo “RMT_RES”, recurso este mencionado no texto de introdução desta secção sobre o FORTE, contendo unicamente um CFB denominado “DM_KRLN” responsável por todo o processo de gestão de pedidos. Este CFB incorpora dois SIFBs interligados: um “SERVER” com dois *outputs* e um *input* e um “DEV_MGR”. O primeiro elemento é responsável por receber via rede as mensagens enviadas pelo IDE e reencaminhá-las para o “DEV_MGR”, e por responder ao IDE enviando-lhe a o código de resposta fornecido pelo mesmo “DEV_MGR”. O segundo elemento tem como função executar os pedidos de gestão recebidos. Após a criação do “RMT_DEV” é configurado o IP e porta de escuta do “SERVER” e é dada ordem ao recurso para começar a executar. A partir desse momento o FORTE está pronto a receber pedidos (o que é sinónimo de dizer que está pronto para transferir e executar aplicações).

O parágrafo anterior faz referência ao conceito de “pedidos de gestão”. Estas mensagens, específicas da ferramenta 4DIAC, têm sempre como destinatário o FB de gestão (“DEV_MGR”), e servem para sinalizar operações efectuadas sobre recursos e dispositivos, entre as quais está a criação e destruição de instâncias de elementos contidos na biblioteca do FORTE. Um qualquer sistema pode ser descrito unicamente por este tipo de mensagens, sendo que a versão actual do IDE permite exportar sistema sob a forma de uma lista de pedidos de gestão. Esses pedidos de gestão ficam contidos num ficheiro com o formato “.fboot”.

Todos os pedidos de gestão possuem uma estrutura idêntica do tipo:

<Destinatário>;<Acção>

O campo <Destinatário> indica o recurso ao qual se destina o pedido descrito em <Acção>. Esta segunda parte da mensagem segue uma estrutura XML e contém o comando, o objecto e o resultado referentes ao pedido. Tomando por exemplo a mensagem seguinte:

```
EMB_RES;<Request ID="1" Action="CREATE"><FB Name="ADD" Type="ADD" /></Request>
```

O pedido consiste na criação de uma instância do FB “ADD” com o nome “ADD” associada ao recurso “EMB_RES”. Para uma referência completa sobre o funcionamento do FB “DEV_MGR” o leitor é remetido para o documento obtido através do Doxygen e para os anexos 4 a 7.

3.4.6 - Compilação do FORTE para ICnova

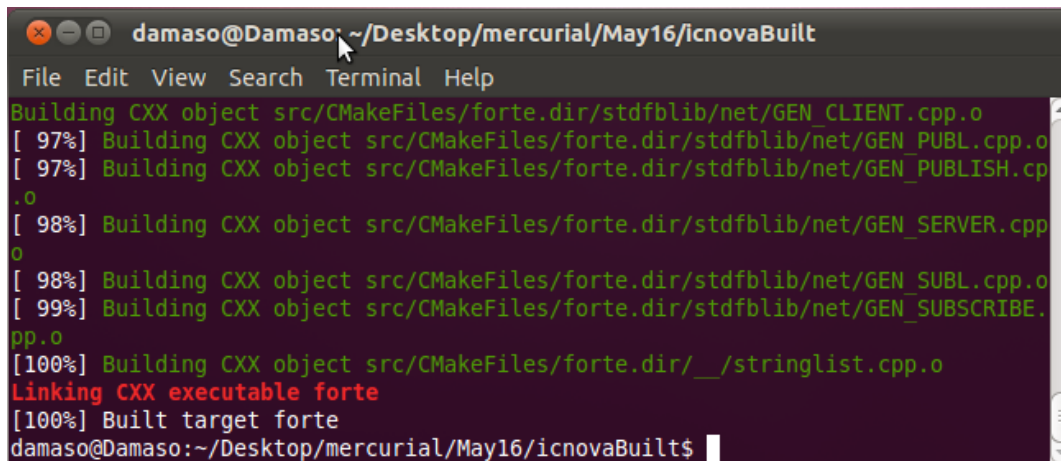
Visto a nova arquitectura contemplar a utilização do sistema embebido ICnova como novo controlador do sistema foi necessário compilar o FORTE tendo a ICnova como *target*. Esta secção pretende descrever o procedimento a tomar para efectuar essa operação.

Uma vez instalada a *toolchain* da ICnova (que tanto pode ser obtida na internet como nos DVDs fornecidos pelo fabricante), altera-se o ficheiro de configuração “CMakeLists” da pasta “/forte/src/arch/posix” de modo a ser alterada a extremidade (tradução literal da expressão *endianness*) de *little endian* para *big endian*. Esta propriedade está relacionada com a forma de gerir e guardar em memória os dados, nomeadamente a ordem utilizada para representar um tipo de dados. Por predefinição a arquitectura POSIX usa a opção “-DFORTE LITTLE ENDIAN”, mas a ICnova necessita que esta seja alterada para “-DFORTE BIG ENDIAN”. Só depois desta alteração deve ser usado o CMake-GUI para gerar os Unix Makefiles necessários ao processo de compilação, logicamente utilizando a arquitectura POSIX.

Como resultado desta operação é gerada uma nova pasta contendo todos os ficheiros de configuração necessários para proceder à compilação. O ficheiro “CMakeCache” contido nessa pasta inclui os *paths* que são necessários alterar para que o processo de compilação utilize a nova *toolchain* em vez da pré-definida. Só após a actualização do ficheiro com as localizações dos componentes da nova *toolchain* pode ser usada a ferramenta *make* para proceder à compilação. Como resultado é gerado um ficheiro executável na pasta “src” denominado “forte” que deve ser copiado para a ICnova de modo a poder ser executado.

Apesar do procedimento descrito acima ser relativamente simples e linear, as primeiras tentativas para compilar o FORTE tendo a ICnova como *target* não foram bem-sucedidas, principalmente porque não há uma documentação extensa relativa ao *cross-compiling* o que levava à parametrização incorrecta de alguma opção de compilação. Como estas primeiras

tentativas ignoravam a necessidade de ser alterada a *endianness* do FORTE, o programa era capaz de executar mas funcionava deficientemente. Estas dificuldades são aqui mencionadas para servirem de alerta a quem desejar compilar o *runtime* para uma nova plataforma. Por último, de mencionar que o anexo 8 contém uma descrição mais técnica sobre todo o processo, o que pode ser útil para o leitor caso queira, também ele, proceder à compilação.



```
damaso@Damaso: ~/Desktop/mercurial/May16/ICnovaBuilt
File Edit View Search Terminal Help
Building CXX object src/CMakeFiles/forte.dir/stdfblib/net/GEN_CLIENT.cpp.o
[ 97%] Building CXX object src/CMakeFiles/forte.dir/stdfblib/net/GEN_PUBL.cpp.o
[ 97%] Building CXX object src/CMakeFiles/forte.dir/stdfblib/net/GEN_PUBLISH.cp
.o
[ 98%] Building CXX object src/CMakeFiles/forte.dir/stdfblib/net/GEN_SERVER.cpp
.o
[ 98%] Building CXX object src/CMakeFiles/forte.dir/stdfblib/net/GEN_SUBL.cpp.o
[ 99%] Building CXX object src/CMakeFiles/forte.dir/stdfblib/net/GEN_SUBSCRIBE.
pp.o
[100%] Building CXX object src/CMakeFiles/forte.dir/__/stringlist.cpp.o
Linking CXX executable forte
[100%] Built target forte
damaso@Damaso:~/Desktop/mercurial/May16/ICnovaBuilt$
```

Figura 3.11 - Consola após a compilação do FORTE para ICnova

3.5 - Testes e validação

No seguimento da secção anterior, após a descrição do processo de compilação do *runtime* para a plataforma embebida torna-se oportuno documentar o teste realizado para validar esse mesmo processo.

Para o teste foi usado um PC a executar o 4DIAC-IDE e uma ICnova a correr o FORTE, com os dois interligados via Ethernet por uma intranet (a rede da FEUP). O teste consistiu na execução do sistema-teste “TestSystem” que acompanha a versão v0.3 do 4DIAC-IDE disponibilizada na página da iniciativa, e contempla dois dispositivos, cada um com um recurso e ligados por Ethernet. Um dos dispositivos é um PC destinado a fornecer uma HMI ao utilizador, enquanto o outro é um microcontrolador (ou sistema embebido, neste caso) que procede à parte lógica da aplicação. Na prática, o utilizador insere um valor inteiro no HMI, esse valor é enviado para o sistema embebido, o sistema embebido adiciona três unidades a esse valor e retorna-o ao PC de forma a poder ser mostrado no HMI. O teste serve para comprovar as capacidades de comunicação em rede do FORTE, para além de obviamente provar que ele foi correctamente portado para a ICnova. O teste foi bem-sucedido.

3.6 - Conclusão

Este capítulo que agora termina pretende ser uma descrição da ferramenta 4DIAC. Foram explicados os conceitos-base em que assentam o 4DIAC-IDE e o FORTE e o modo de funcionamento de cada um deles, tendo sempre em atenção que essa descrição é destinada ao utilizador básico.

Foi explicado como o IDE permite desenhar sistemas IEC 61499, criar novos tipos de FBs e transferir aplicações para o *runtime*. Foi igualmente exposta a estrutura do FORTE, sendo

destacadas as suas características mais importantes e a forma como o seu código fonte pode ser compilado para dar origem a um ficheiro executável capaz de receber e executar aplicações.

Nesse campo do processo de compilação, foi ainda descrita a forma como se proceder à compilação do *runtime* tendo a ICnova como plataforma *target*, algo indispensável para o projecto.

Por último, em jeito de avaliação da ferramenta, deve ser dito que, apesar de se tratar de uma iniciativa académica, a *framework* 4DIAC apresenta já um nível de maturidade bastante aceitável. Apesar dos *bugs* que são corrigidos com uma frequência diária, a ferramenta aparenta ser ideal para utilizar num contexto académico, dado que todo o código-fonte em que assentam as duas componentes está disponível para consulta e alteração. Tendo por base a experiência do próprio autor da tese, deve ser mencionado que o FORTE por vezes apresenta dificuldades em operações relacionadas com comunicações em rede, pois, segundo os próprios autores da iniciativa, o bom funcionamento destas depende das configurações de rede dos dispositivos que executam o FORTE.

Capítulo 4

Modbus

4.1 - Introdução

Como a lógica de controlo de campo do sistema actualmente implementado começa e termina nos PLCs apenas estes podem ser alvo de uma reforma (substituição). Este princípio implica que a proposta de sistema de gestão que resulta desta dissertação deve reutilizar, entre outros, as ilhas remotas (I/Os remotos), o que impõe ao novo sistema a utilização do protocolo Modbus TCP. Como a ferramenta 4DIAC não suportava este protocolo foi necessário corrigir essa situação. Este capítulo documenta o processo de estruturação, implementação, teste e validação de uma solução que fizesse a tecnologia 4DIAC compactível com o protocolo.

Esta secção da dissertação é formada por numa primeira parte que consiste numa referência sumária ao protocolo Modbus, de modo a contextualizar a tecnologia em causa, e uma segunda parte em que é analisada e discutida a forma como o protocolo foi implementado na ferramenta 4DIAC.

4.2 - Referência

A primeira versão do protocolo Modbus foi desenvolvida em 1979 pela empresa de produtos electrónicos Modicom (a actual Schneider), tendo como objectivo providenciar uma solução para a comunicação entre PLCs da marca. Actualmente a monitorização da evolução do protocolo encontra-se a cargo da organização sem fins lucrativos Modbus-IDA, sendo que a especificação Modbus se encontra divulgada livremente, sem custos.

Por ter sido um dos primeiros *fieldbuses* a serem introduzidos em ambientes de manufacturação automática, o protocolo Modbus é ainda hoje amplamente utilizado em sistemas industriais e, como é o caso do sistema em causa, em algumas secções da domótica.

4.2.1 - Princípios de funcionamento

O protocolo Modbus assenta no tipo de cooperação cliente-servidor. Este modelo contempla mensagens de rede que podem ser pedidos (de leitura ou escrita) ou respostas, sendo que quem toma a iniciativa de estabelecer uma comunicação é sempre o cliente. Na prática, os servidores possuem áreas de memória que os clientes podem ler e/ou escrever, recorrendo para isso a pedidos específicos. A cada pedido efectuado por um cliente deve corresponder uma resposta enviada pelo servidor, resposta esta que discrimina se o processamento do pedido foi bem-sucedido e que contém outros valores de interesse à operação (por exemplo, no caso de um pedido de leitura, a mensagem de resposta contém o valor do endereço requerido).

Primary tables	Object type	Type of	Comments
Discretes Input	Single bit	Read-Only	This type of data can be provided by an I/O system.
Coils	Single bit	Read-Write	This type of data can be alterable by an application program.
Input Registers	16-bit word	Read-Only	This type of data can be provided by an I/O system
Holding Registers	16-bit word	Read-Write	This type of data can be alterable by an application program.

Figura 4.1 - Tipos de memória Modbus [20]

A tabela da Figura 4.1 é extraída do documento “MODBUS Messaging Implementation Guide version 1.0b”, fornecido pela entidade reguladora do protocolo, e enumera as características de cada uma das áreas de memória/tabelas referidas no parágrafo anterior. As diferentes áreas são designadas por *coils* (saídas digitais), *discrete inputs* (entradas digitais), *holding registers* (saídas analógicas) e *input registers* (entradas analógicas). Logicamente as saídas apenas se pode ler, enquanto as entradas se podem ler e escrever. Os valores dos *inputs/outputs* digitais estão guardados em registos de 1 bit, enquanto os analógicos estão guardados em registos de 16 bits (words). Todos os grupos possuem endereçamentos que vão de 0 a 65535 (ou, em alguns casos, de 1 a 65536).

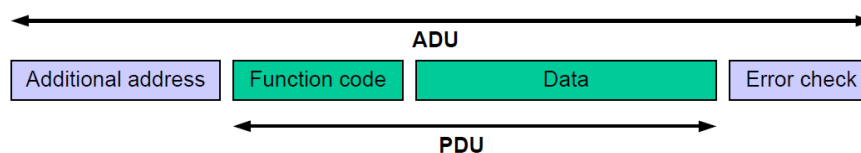


Figura 4.2 - Formato de uma trama genérica do protocolo Modbus [21]

As mensagens trocadas entre clientes e servidores são genericamente compostas por duas partes (representadas a diferentes cores na Figura 4.2). A primeira parte denomina-se PDU (*protocol data unit*) e corresponde ao resultado produzido pela camada superior (aplicação) do protocolo Modbus, sendo esta fracção independente do meio físico usado para a transmissão. A segunda parte corresponde ao cabeçalho e rodapé necessários adicionar à PDU para que a mensagem consiga chegar em segurança ao destinatário já tendo em conta o tipo de meio físico e os protocolos de mais baixo nível usados pela rede. O resultado da junção das duas partes denomina-se ADU (*application data unit*).

As tramas PDU de pedido e de resposta possuem uma estrutura característica. Ambas começam com um código que indica a função a que se referem, seguindo-se a data que serve de argumento à função. No caso de o servidor não conseguir executar um determinado pedido

com sucesso a mensagem de resposta passa a ser uma mensagem de excepção. Nesta mensagem especial o código da função obtém-se somando 128 (0x80 em hexadecimal) ao código da função à qual a resposta se refere, sendo o campo de dados ocupado com o código do erro.

Cada função, caracterizada por um código geralmente apresentado em base hexadecimal, possui um propósito próprio. A maioria das funções refere-se à leitura e escrita dos diferentes espaços de memória, sendo que algumas delas permitem ler ou escrever vários endereços num só pedido (consultar anexo 2). Existem ainda outro tipo de funções que se destinam a pedidos especiais, como por exemplo pedir a identificação do servidor.

O meio físico e respectivos protocolos subjacentes utilizados por uma rede Modbus são escolhidos tendo em conta principalmente se a rede projectada é apenas local ou se ela está englobada numa rede maior (por exemplo, a internet). No caso de uma rede local, as comunicações série RS-232 e RS-485 são as mais usadas, assentes em codificação RTU ou ASCII. Para o outro caso, usa-se Ethernet assente em TCP/IP (denominado Modbus TCP). A segunda opção é utilizada na plataforma de gestão se encontra actualmente implementada, pelo que a solução a desenvolver funcionará também segundo esta variante do protocolo. De referir que é possível uma rede englobar os dois tipos de arquitectura (série e TCP), bastando para isso utilizar *gateways* específicos que façam a conversão necessária.

4.2.2 - Modbus TCP

Modbus TCP Frame Format		
Name	Length	Function
Transaction Identifier	2 bytes	For synchronization between messages of server & client
Protocol Identifier	2 bytes	Zero for MODBUS/TCP
Length Field	2 bytes	Number of remaining bytes in this frame
Unit Identifier	1 byte	Slave Address (255 if not used)
Function code	1 byte	Function codes as in other variants
Data bytes	n bytes	Data as response or commands

Figura 4.3 - Formato da trama Modbus TCP [21]

Por não se tratar de uma comunicação série como as variantes ASCII e RTU do protocolo, o formato das tramas Modbus TCP é substancialmente diferente do desses dois modelos. Nesse formato, descrito na Figura 4.3, é possível identificar os campos que constituem a PDU (*Function Code* e *Data bytes*) juntamente com outros elementos, sendo estes: identificador da transacção, um contador que é incrementado à medida que as mensagens entre cliente e servidor são trocadas, identificador do protocolo, com valor zero para todas as mensagens Modbus, tamanho, valor que indica o número de bytes restantes na mensagem, e identificador da unidade, um código usado para quando redireccionar mensagens no caso de a rede em causa ser híbrida (série juntamente com TCP). Nota para o facto de não serem usados campos adicionais de *checksum* para o controlo de erros. Esta faculdade apenas é utilizada para as comunicações série, visto o protocolo TCP já possuir mecanismos próprios que asseguram a integridade dos dados. Uma das características do Modbus TCP é a porta 502

estar reservada para comunicações deste tipo, não sendo obrigatório ser essa a porta utilizada.

A organização Modbus-IDA providencia no seu *website*, para além da especificação do protocolo, um documento guia para a implementação de sistemas Modbus TCP [21] (por sistemas entenda-se clientes, servidores e *gateways*). Trata-se apenas de um documento guia, não um manual de implementação, mas apesar disso nele estão contidas sugestões de arquitecturas, modelos e conceitos que pretendem oferecer aos programadores/criadores de dispositivos Modbus uma base sólida para garantir interoperabilidade entre diferentes equipamentos (ver anexo 1).

4.3 - Requisitos

Os parágrafos seguintes descrevem os requisitos impostos à solução a desenvolver. Eles direccionam-se às funcionalidades quer de baixo nível quer de alto nível que a solução deve apresentar.

- Relativo ao cliente Modbus, deve ser possível ao utilizador (i.e. programa) realizar pedidos a servidores Modbus, podendo escolher o tipo de pedido (ler ou escrever), o tipo de memória e o endereço da memória. Para além disso, no caso de se tratar de um pedido de escrita o utilizador deve poder definir o valor a escrever em memória, e no caso de se tratar de um pedido de leitura deveria poder consultar o valor do endereço em causa. Para ambos os casos o utilizador deve ser informado se o seu pedido foi correctamente processado ou não. Igualmente em ambas as situações o utilizador deve conseguir escolher o IP e a porta de escuta referentes ao servidor ao qual ele deseja enviar o pedido.

- Quanto ao modo servidor Modbus, a solução deve ser capaz de criar e gerir um espaço de memória que representasse *coils*, *discrete inputs*, *holding registers* e *input registers*, dando a possibilidade ao utilizador de definir os tamanhos de cada uma das diferentes partes que compõem essa memória. Quanto à comunicação Modbus em si, o sistema desenvolvido deve ser capaz de escutar pedidos de qualquer cliente Modbus que esteja ligado em rede, processar esses pedidos acedendo à citada memória e responder ao cliente com uma mensagem de retorno concordante. Paralelamente a este último serviço deve ser possível ao programa que usa a solução consultar e editar localmente esses espaços de memória através da interface devidamente contemplada pela norma IEC 61499 (eventos e variáveis de entrada e saída).

- A solução desenvolvida deve ser robusta, de modo a não comprometer a execução das aplicações em que se insere nos casos em que existe um qualquer mau funcionamento ou avaria (como são os casos de problemas na rede ou nos dispositivos servidores Modbus a quem se possa fazer pedidos). Deve ser igualmente genérica, ou seja, as suas propriedades e faculdades devem permitir que seja utilizada sem reajustes ou redesenhos por diferentes sistemas e aplicações.

- A solução desenvolvida não deve comprometer o *uptime* do FORTE, devendo por isso utilizar com cautela a memória e as ligações de rede e lidar convenientemente com os erros que possam surgir.

- Por último, deve ser intuitivo para o utilizador trabalhar com o recurso desenvolvido. A melhor forma de responder a este requisito é ocultar ao utilizador todas as operações de baixo nível e seguir ao máximo as orientações, dadas na especificação da norma, que visam uniformizar o desenho de componentes IEC 61499.

Como os requisitos indicados acima fornecem grande liberdade ao processo de definição da arquitectura foi necessário discutir e confrontar diferentes propostas de soluções. A secção que se segue descreve esse processo, contendo uma descrição geral e a enumeração das vantagens e desvantagens de cada uma das propostas, e documentando as diferentes componentes da arquitectura escolhida.

4.4 - Arquitectura

Esta secção começa por debater a arquitectura de alto nível da solução, sendo confrontadas as diferentes abordagens consideradas de forma a serem justificadas as escolhas finais. Segue-se uma análise progressivamente mais detalhada da arquitectura da solução, juntamente com as justificações que acompanham as escolhas de natureza mais técnica.

4.4.1 - SIFB vs Sistema de comunicação do FORTE

A primeira decisão tomada no processo de definir a estrutura da solução foi escolher a forma como este se incorporava na ferramenta 4DIAC. Foram consideradas as duas opções que se seguem, sendo estas as únicas opções realisticamente viáveis.

A primeira opção consistia em criar um ou vários SIFBs capazes de gerir o protocolo Modbus de uma forma independente do sistema de comunicação já existente no FORTE, ou seja, os SIFB teriam de ser capazes de criar e gerir as *sockets* de comunicação e as mensagens trocadas, e de implementar as necessárias arquitecturas de cliente e servidor.

A segunda opção passaria por incorporar o protocolo Modbus no sistema de comunicação já existente do FORTE, paralelamente aos outros protocolos já inseridos na ferramenta. Esta escolha implicaria editar o código fonte do próprio *runtime*.

Comparando as duas hipóteses foi possível chegar às seguintes conclusões:

- Por manter uma relativa independência em relação ao FORTE, a primeira opção assegura que alterações efectuadas neste último por terceiros não interferem no bom funcionamento da solução desenvolvida, algo que a segunda opção não garante;
- Utilizando um sistema de comunicação já existente, a segunda opção assegura uma (relativa) melhor eficiência na gestão de recursos. Por outro lado esse mesmo sistema de comunicação encontra-se pouco documentado e a sua estrutura é complexa, pelo que qualquer alteração nele introduzido implica um profundo estudo dessa mesma estrutura;
- A segunda opção envolve à partida maior complexidade de implementação, visto para além de envolver o processo de desenvolvimento de código e respectivos testes, implica também a incorporação deste num outro código preexistente. Tendo em conta que a implementação do protocolo Modbus na ferramenta 4DIAC é apenas um dos objectivos da tese, o trabalho extra que a segunda opção envolve quando comparada com a primeira parece não ser justificável.
- Aceitando a possibilidade de serem utilizadas bibliotecas pré-existentes, a primeira opção torna-se substancialmente mais atractiva que a segunda. O sistema de comunicação do FORTE impossibilita a utilização de tais bibliotecas, ou pelo menos, a sua utilização implicaria uma pesada reforma de código, tal é rígida a estrutura desse sistema.

Ponderando os argumentos descritos nos parágrafos anteriores a escolha recaiu sobre a primeira opção. Para além das razões apresentadas deve ser mencionado que para que a segunda opção funcione-se plenamente seria necessário trabalhar em cooperação com os programadores do 4DIAC para que as alterações introduzidas no código fonte do *runtime*, para além de correctamente implementarem o protocolo Modbus, não corressem o risco de irem contra aquilo que os programados projectam para a ferramenta.

4.4.2 - SIFB Único vs Modularidade

O próximo passo foi decidir se se utilizaria apenas um único SIFB para implementar um híbrido servidor/cliente Modbus ou se pelo contrário se desenvolveriam um ou mais elementos para cada uma das entidades, garantindo independência entre as duas entidades. Relativamente à primeira opção, como consequência de se juntarem cliente e servidor num mesmo SIFB a complexidade deste torna-se considerável, não só ao nível da implementação como ao nível da utilização. Um dos principais conceitos da norma IEC 61499 relativamente ao desenvolvimento de FBs é que se procure criar módulos o mais simples e elementares possíveis, pelo que esta opção vai contra esse paradigma. Estas razões levaram a que a segunda opção fosse escolhida em detrimento da primeira.

Ao escolher uma opção que fornece independência entre os SIFBs cliente e servidor, a arquitectura destes passa a poder ser definida de forma igualmente independente, sendo que as próximas secções abordam os dois módulos de forma separada.

4.4.3 - Cliente

Para a arquitectura da entidade cliente foram ponderadas duas opções: a primeira contemplava usar apenas um SIFB que permitisse efectuar todo o tipo de pedidos, enquanto a segunda propunha usar vários SIFB, um por cada função base Modbus (por função base entenda-se as operações de leitura e escrita em espaço de memória que os clientes podem requerer aos servidores). A premissa associada à primeira opção implicava como consequência pouco vantajosa a necessidade de ou se utilizar uma variável para codificar a função base (consultar anexo 2) que se pretende-se associar ao pedido a efectuar ou se criar um evento de entrada para cada uma destas funções, sendo que ambas as hipóteses resultariam num módulo de utilização pouco intuitiva para o utilizador, para além de implicar uma má gestão das variáveis de entrada e saída. A segunda opção enquadra-se mais no citado paradigma IEC 61499 que visa modelizar ao máximo os elementos que servem de base às aplicações, pois ao se criar um SIFB por cada função base oferece-se ao utilizador uma forma mais simples e intuitiva de gerir o protocolo Modbus do que aquela que seria oferecida pela outra opção.

Em termos de estrutura de eventos, a especificação Modbus é clara em sugerir nestes casos a existência de um evento associado ao estabelecimento e destruição da ligação e um outro responsável por requerer o serviço fornecido pelo SIFB, neste caso o processamento de um pedido Modbus, sendo que foi esta a configuração utilizada nos diferentes SIFBs Cliente.

Assim, a arquitectura da solução final consiste em 6 SIFBs independentes, capazes de estabelecerem e gerirem uma ligação a um dado servidor, permitindo à aplicação realizar pedidos customizado por esta a partir dos dados que fornece às variáveis de entrada dos SIFBs, nomeadamente IP e porta de escuta do servidor, endereço a ler/escrever e valor a

escrever (para os tipos de pedido de escrita), e permitindo-lhe também consultar a parte fundamental da mensagem de resposta ao pedido, nomeadamente o resultado da operação associada ao pedido (se foi bem ou mal sucedida) e, no caso dos pedidos do tipo leitura, o valor do endereço que foi requerido, respeitando o que foi estipulado pelos requisitos.

4.4.4 - Servidor

A arquitectura do módulo que representa a entidade servidor tem de respeitar requisitos próprios que a faz divergir da arquitectura da entidade cliente. O módulo, para além de ter de providenciar uma interface que permita ao programa em que se insere aceder à memória Modbus, também necessita implementar um serviço de escuta e tratamento de pedidos Modbus que interage com essa mesma memória.

Assim, a memória pode ser considerada a peça chave deste módulo. Ela é única para um dado servidor e deve estar disponível, consoante diz o parágrafo anterior, tanto ao programa como ao gestor de pedidos sob protocolo Modbus. Por isso é lógico utilizar apenas um SIFB (em diante denominado “Servidor Modbus”) que a contenha, sendo esta a primeira conclusão resultante da discussão da arquitectura do módulo servidor.

Uma vez escolhida a estrutura responsável por conter a memória resta definir a interface de interacção desta com o programa e com o serviço de escuta e tratamento de pedidos. Sabendo que a o *runtime* FORTE permite *threads*, uma opção válida (e que acabou por ser a escolhida) consiste em utilizar o próprio “Servidor Modbus” para correr em *background*, através de uma *thread* independente, o serviço de escuta, garantindo a independência do módulo em relação ao programa aonde ele se possa inserir. Outra opção passaria por se criar um SIFB independente para a escuta de pedidos, o que apenas se traduziria numa maior complexidade do módulo e numa maior taxa de utilização de memória aquando da execução do programa.

O processo de definir a interface do módulo com o programa foi consideravelmente mais complexo, dado que, por definição, os FBs não aceitam mais de uma fonte (*source*) em cada variável de entrada. Como a memória Modbus é única para cada servidor (diferentes instâncias do SIFB “Servidor Modbus” representariam diferentes servidores Modbus) é necessário que a interface seja feita com um modelo relacional de “N para 1”, ou seja, como apenas pode existir uma instância do SIFB Servidor Modbus para representar cada servidor Modbus, todo o programa aonde este se insere vai ter de o partilhar, e como podem existir múltiplos pontos do programa a quererem interagir com ele dá-se o caso de muitos (“N”) quererem aceder a “um”, o que se traduz na necessidade criar uma forma de gerir esta relação. Este problema dá-se devido ao paradigma IEC 61499 não ser completamente compatível com aquele que está na base do protocolo Modbus. As soluções discutidas para implementarem a citada gestão foram a utilização de multiplexadores programáticos (FBs que aceitam vários dados de entradas e de entre eles escolhem um para transporem para dado de saída), o que ofereceria pouca flexibilidade ao módulo, ou associar um SIFB “SERVER” ao já definido “Servidor Modbus”. O “SERVER” é um SIFB já fornecido pela ferramenta 4DIAC que permite implementar um servidor simples que oferece nas suas variáveis de saída os valores que recebe de um qualquer cliente (SIFB “CLIENT”) que estabeleça uma ligação ao IP e à porta aonde está a escuta e responde a este com os valores que possui como variáveis de entrada. Esta segunda opção, que acabou por ser a escolhida, permite que qualquer parte do programa possa aceder à memória contida no “Servidor Modbus”, bastando para isso usar uma instância SIFB “CLIENT” configurada de modo a do outro lado da ligação estar o SIFB

“SERVER” associado ao SIFB “Servidor Modbus” com quem deseja interagir. A última discussão em relação à arquitectura deste módulo consistiu em escolher se se utilizava apenas um SIFB “SERVER” ou se se utilizam seis, um para cada uma das funções básicas. Acabou-se por se escolher a segunda opção pois ela permite evitar a utilização de variáveis que codifiquem a função básica envolvida na operação, o que se traduz numa maior facilidade do utilizador (e das próprias aplicações de controlo) em utilizar este recurso. Por outro lado esta escolha carrega um uso maior da rede de comunicação, algo que é aceitável pois não é previsível que um número elevado de servidores Modbus distinto sejam executados num mesmo dispositivo. Uma outra consequência da escolha desta opção é a necessidade de existirem outros FB secundários para gerirem os SIFBs “SERVER” (no processo de atribuição de valores de IP e porta de escuta) e a interacção destes com o SIFB “Servidor Modbus”. Mais concretamente é preciso incluir um SIFB que receba valores de IP e porta definidos pela aplicação e, baseados neles, configure os parâmetros de ligação dos SIFBs “SERVER”, é preciso um multiplexador programático que filtre a informação proveniente destes e a redireccione para o SIFB “Servidor Modbus”, e por último é necessário um comutador programático que reencaminhe a informação fornecida por este último SIFB para o SIFB “SERVER” certo.

Como resultado da proposta explicada acima, a arquitectura da solução final consiste num FB composto contendo 6 SIFB “SERVER”, 1 SIFB “Servidor Modbus”, 1 FB multiplexador (denominado “MUXModbusServer”), 1 FB comutador (denominado “SWITCHModbusServer”) e 1 FB “ServersIpHandler”, sendo esta unidade responsável por atribuir os códigos IP aos diferentes SIFB Server.

4.4.5 - Libmodbus

Indo ao encontro do que foi mencionado na última secção, a utilização de uma biblioteca que implemente as operações de médio/baixo nível do protocolo Modbus, numa espécie de ferramenta de *middleware*, permite reduzir substancialmente o tempo de desenvolvimento da solução, para além de ao ser utilizada uma componente deste tipo, devidamente testada e validada, ser automaticamente conferida eficiência e eficácia a esta parte da solução.

É neste contexto que surge a Libmodbus [22]. A biblioteca Libmodbus consiste numa ferramenta escrita na linguagem C que oferece ao programador o controlo de todos os níveis da comunicação Modbus, com a possibilidade de, por se tratar de uma ferramenta *open-source* abrangida pela licença LGPL v3, se poder alterar o código fonte de acordo com os requisitos do projecto em causa. Para além de implementar o protocolo Modbus TCP, a Libmodbus também permite as variantes de comunicação série deste protocolo (ASCII e RTU). Está preparada para gerar programas para serem executados em diversos sistemas operativos, entre eles Linux e Windows, mas não em simultâneo, ou seja, o processo de compilação define em que plataforma o programa pode ser executado. Desta última característica surge a única desvantagem decorrente da utilização desta ferramenta. Tendo em conta que se pretende executar o FORTE e as aplicações desenvolvidas na plataforma ICnova, e lembrando que este sistema embebido utiliza Linux como sistema operativo, está-se a limitar a utilização da solução a plataformas que possuam Linux. Um outro dispositivo que utilize o sistema operativo Windows, por exemplo, não vai conseguir utilizar os recursos desenvolvidos.

Conforme é indicado no parágrafo anterior a biblioteca oferece um controlo completo sobre todos os níveis da comunicação Modbus. O facto de esta ferramenta possuir esta característica foi uma das razões que levaram à sua incorporação neste projecto. Sob o ponto

de vista da comunicação, esta biblioteca permite ao programador assumir um controlo tanto de alto nível, oferecendo funções que já implementam todas as operações necessárias para a gestão da comunicação, como de baixo nível, com a possibilidade de alterar as propriedades mais básicas e específicas do processo, como por exemplo configurar *sockets*, definir *timouts* e gerir a memória.

A biblioteca Libmodbus é gerida por um grupo extremamente activo e produtivo, pelo que novas versões do *software* são lançadas periodicamente (muitas das quais com alterações bastantes significativas). Este facto aconselha a que futuras actualizações que sejam efectuadas à solução desenvolvida contemplem as possíveis alteração da biblioteca. A versão utilizada neste projecto foi a v2.0.3 (última versão estável).

A biblioteca não impõe uma qualificação de mestre ou escravo às entidades que a utilizam. Antes, a ferramenta Libmodbus providencia funções que são características de mestres (clientes) e/ou de escravos (servidores). A tabela do anexo 3 lista alguns dos principais APIs de alto nível fornecidos pela biblioteca, assim como as estruturas mais importantes que nela constam.

Para que os FBs possam utilizar os serviços fornecidos pela biblioteca foi necessário adicioná-la ao processo de compilação do FORTE. Porque a biblioteca está escrita em linguagem C não foi possível utilizar a função “forte_add_sourcefile_hcpp” (ver Figura 3.8), pelo que teve se ser desenhada um novo método “forte_add_sourcefile_hcpp”, similar ao anterior, mudando apenas a extensão do ficheiro a ler (de “.cpp” para “.c”).

4.5 - Implementação

Nesta secção é documentada a forma como foram aplicadas na prática as arquitecturas discutidas nos pontos anteriores. São indicados os papéis de todas a variáveis e eventos de entrada e saída dos diferentes SIFBs e é explicado o modo de funcionamento destes, o que faz com que esta secção possa ser encarada como uma guia de utilização da ferramenta sob o ponto de vista de utilizador.

4.5.1 - Módulo Cliente

Todos os 6 SIFBs que implementam as diferentes funções básicas associadas à entidade cliente possuem uma interface semelhante, conforme pode ser confirmado pelas figuras Figura 4.4 a Figura 4.9. Este facto deve-se em grande parte às rígidas regras impostas pela norma IEC 61499 para a definição da arquitectura de SIFBs. Por outro lado esta homogeneidade foi propositadamente procurada pois ela traduz-se numa maior facilidade de utilização da solução por parte do utilizador.

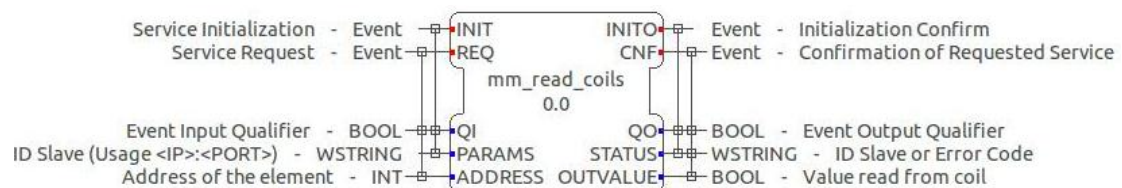


Figura 4.4- SIFB “mm_read_coils”

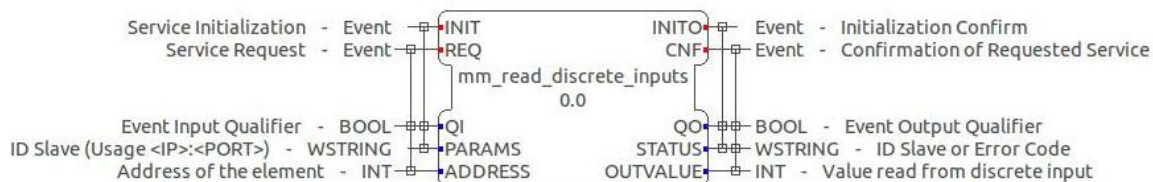


Figura 4.5- SIFB "mm_read_discrete_inputs"

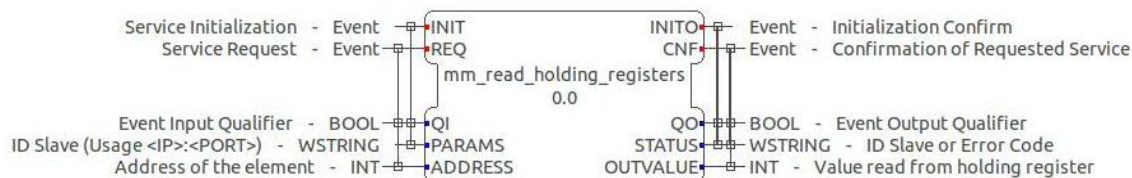


Figura 4.6- SIFB "mm_read_holding_registers"

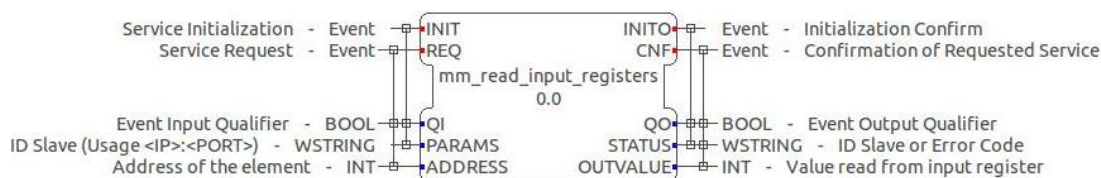


Figura 4.7- SIFB "mm_read_input_registers"

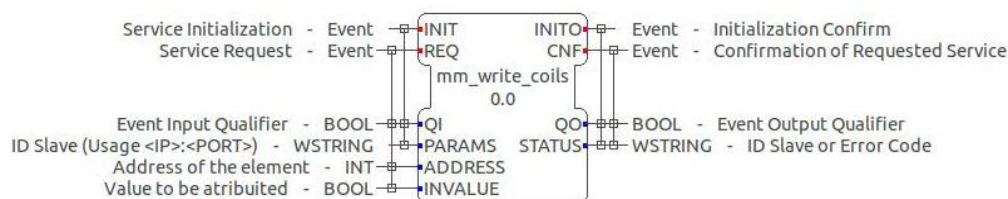


Figura 4.8- SIFB "mm_write_coils"

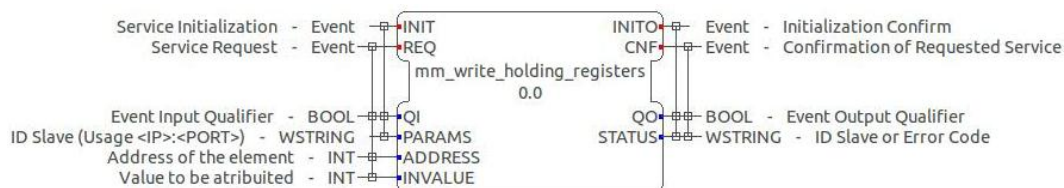


Figura 4.9- SIFB "mm_write_holding_registers"

Os SIFBs encontram-se divididos, em termos funcionais, em duas partes: a parte de gestão da comunicação e a parte de execução do serviço. Essa distinção é feita através dos dois pares de eventos de entrada e saída: "INIT" e "INITO" encarregam-se da ligação enquanto "REQ" e "CNF" tratam do serviço associado ao SIFB em causa. Ao primeiro par estão associadas as variáveis de entrada "QI" e "PARAMS" e de saída "QO" e "STATUS", enquanto o segundo está relacionado com "QO" e "STATUS" e as variáveis de endereço e de escrita e leitura de valores. "QO" está associado com ambos os pares de eventos pois qualifica o sucesso ou insucesso da execução do pedido ("QI" não se relaciona também com ambos porque, como será explicado mais à frente, este executa um tipo único de serviço). "PARAMS" é usado pelo programa para passar ao SIFB a informação necessária para este

estabelecer a ligação cliente-servidor, enquanto “STATUS” contém informação útil para a aplicação, sendo que o papel desta variável será explicada em maior detalhe mais à frente. Cada SIFB possui ainda mais duas variáveis adicionais: “ADDRESS”, que é usado pelo programa para indicar que endereço do servidor deseja ler ou escrever, e “INVALUE” ou “OUTVALUE”. “INVALUE” consta dos SIFBs que constroem e enviam pedidos de escrita para permitirem ao programa definir o valor a inserir em “ADDRESS”, enquanto “OUTVALUE” está presente nos SIFBs que constroem e enviam pedidos de leitura com a função de fornecer ao programa o valor lido em “ADDRESS”.

As próximas secções explicam em maior detalhe os processos que são accionados em função dos dois eventos de entrada mencionados.

Gestão da comunicação

Comum a todos os SIFBs é a forma como é gerida a comunicação Modbus. O evento “INIT” é o elemento-chave deste processo, sendo responsável por comandar o estabelecimento e destruição da ligação cliente-servidor consoante a indicação da aplicação aonde o SIFB se encontra inserido.

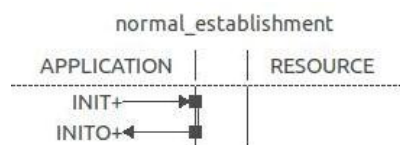


Figura 4.11 - Sequência de serviço para uma inicialização bem-sucedida dos SIFBs Modbus Cliente

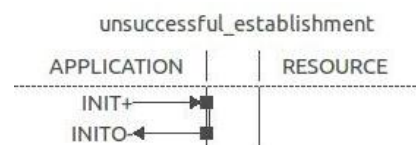


Figura 4.10 - Sequência de serviço para uma inicialização mal-sucedida dos SIFBs Modbus Cliente

O primeiro caso, de estabelecimento de comunicação, dá-se ao ser despoletado um evento do tipo “INIT+” (o sinal mais significa que o valor da entrada booleana “QI” é “TRUE”) que se traduz num comando ao SIFB para estabelecer uma ligação a um servidor, sendo este identificado pelos parâmetros indicados na entrada “PARAMS”, sob a estrutura <IP>:<Porta de escuta> (por exemplo “127.0.0.1:502”). De acordo com o sucesso ou insucesso dessa operação é emitido um evento “INITO”, confirmando a execução da operação, acompanhado pela actualização da variável de saída QO que indica se o processo foi bem-sucedido (“TRUE”) ou não (“FALSE”), conforme é representado nas figuras Figura 4.11 e Figura 4.10, respectivamente.

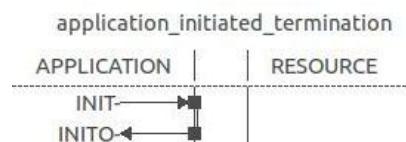


Figura 4.12 - Sequência de serviço para uma terminação de serviço dos SIFBs Modbus Cliente

No caso de destruição da ligação, é recebido um evento do tipo “INIT-” (“QI” é “FALSE”) o qual o SIFB interpreta como um comando para romper a ligação que tinha criado anteriormente. Quando o processo é completo é invariavelmente emitido um evento “INITO-”, conforme ilustrado na Figura 4.12.

O processo de gestão da comunicação é explicado de uma forma mais clara pelo diagrama UML da Figura 4.13.

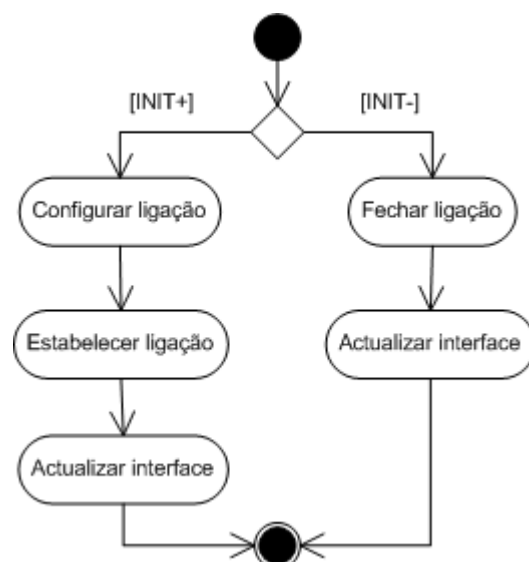


Figura 4.13 - Diagrama UML de actividade do processo de gestão da comunicação dos SIFBs Modbus Cliente

Como é ilustrado pelo fluxograma o processo de gestão possui, graças às funcionalidades fornecidas pela biblioteca Libmodbus, uma arquitectura relativamente linear, o que resultou numa implementação bastante simples.

Em termos práticos, ao receber um evento “INIT+” o programa começa por extrair os códigos IP e porta de escuta do servidor a contactar e verifica as suas validades. No caso de essa validade não se verificar é actualizada a saída “STATUS” com o código de erro “0”, é enviado um evento “INITO-” e o algoritmo termina. Caso os códigos sejam válidos, o programa usa-os para configurar a ligação, usado para isso os APIs apropriados fornecidos por Libmodbus. De seguida tenta-se estabelecer a ligação, podendo este processo resultar em sucesso ou insucesso. No primeiro caso é actualizada a saída “STATUS” com uma cópia da entrada “PARAMS” para que a aplicação a jusante do SIFB possa usar essa informação, e enviado um evento “INITO+”, enquanto no segundo é atribuído à saída “STATUS” o código de erro e é emitido um evento “INITO-”. Em ambos os casos o programa termina depois de executar as respectivas instruções.

A recepção de um evento “INIT-” sinaliza um pedido para que seja destruída a ligação estabelecida. Este processo consiste apenas na evocação do API responsável por desligar a ligação, que se encarrega de executar todas as instruções necessárias ao cessamento da ligação, seguida da atribuição do código “0” a “STATUS” e a emissão de um evento “INITO-” que confirma a execução do comando.

Execução do serviço

A componente dos diferentes SIFBs cliente responsável por executar o serviço de envio de pedidos Modbus, apesar de não ser igual em todos, apenas varia ligeiramente de SIFB para SIFB. Em todos eles o evento “REQ” é responsável por sinalizar essa execução, sendo que este não se encontra associado ao qualificador de entrada “QI” pois só fornece um tipo de serviço por SIFB (só envia pedidos Modbus, ao contrário de “INIT” que cria e destrói ligações cliente-servidor).

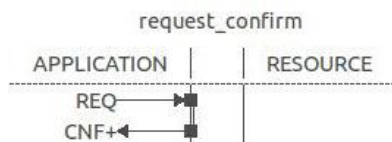


Figura 4.14 - Sequência de serviço para uma requisição de serviço bem-sucedida dos SIFBs Modbus Cliente

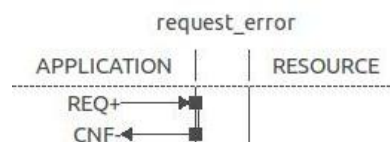


Figura 4.15 - Sequência de serviço para uma requisição de serviço mal-sucedida dos SIFBs Modbus Cliente

Assim que é recebido um evento “REQ” o algoritmo contido no SIFB utiliza os dados de entrada para construir um pedido e enviá-lo pela rede. Caso a mensagem de resposta indique que o pedido foi executado com sucesso é enviado um evento “CNF+”, caso contrário é emitido um evento “CNF-”, conforme é ilustrado pelas sequências de serviço das figuras Figura 4.14 e Figura 4.15.

De forma genérica, o modo de funcionamento dos algoritmos de execução de serviço dos diferentes SIFB é esquematizado no diagrama UML da Figura 4.16.



Figura 4.16 - Diagrama UML de actividade do processo de execução de serviço dos SIFBs Modbus Cliente

Em termos funcionais, a recepção de um evento “REQ” resulta na chamada de um API próprio de cada SIFB, de acordo com a Tabela 1. Como os API’s se encarregam de executar todos os passos necessários à construção e envio de pedidos e à recepção da resposta o restante algoritmo apenas envia um evento “CNF+” de confirmação para as operações bem-sucedidas ou “CNF-” para as mal-sucedidas, actualizando em ambos os casos a saída “STATUS” (para pedidos correctamente executados “STATUS” é actualizada para “0”, enquanto que para os que resultam em erro é-lhe atribuído o código do erro). Para além disso, nas operações de leitura, é ainda actualizada a variável de saída “OUTVALUE” com o valor lido em “ADDRESS”.

SIFB	API
mm_write_coils	force_single_coil
mm_write_holding_registers	preset_single_register
mm_read_coils	read_coil_status
mm_read_discrete_inputs	read_input_status
mm_read_holding_registers	read_holding_registers

mm_read_input_registers	read_input_registers
-------------------------	----------------------

Tabela 1 - Correspondência entre os SIFBs Modbus Cliente e os API's da biblioteca Libmodbus

4.5.2 - Módulo Servidor

De acordo com o que foi explicado na secção de definição da arquitectura, o módulo servidor consiste num complexo CFB, denominado “MODBUS_TCP_SERVER” e ilustrado na Figura 4.17, que contém uma série de componentes interligados, cada um com uma função própria. A interface simples que possui só é possível porque toda a complexidade que o CFB possui está escondida no seu interior. Possui apenas um par de eventos (“INIT”/”INITO”) que são usados para inicializar e terminar o servidor. Possui 3 variáveis de entrada: “QI”, o qualificador usado para indicar se o evento INIT recebido sinaliza a inicialização (*TRUE*) ou a terminação (*FALSE*) do serviço, ID, que contém os códigos IP e porta de escuta que servirão de base aos SIFB “SERVER” contido no interior do CFB, e “PARAMS”, que contém os valores necessários para a inicialização da memória Modbus. Quando a variáveis de saída, existem apenas duas: “QO”, que indica se a indica o resultado dos processos de inicialização e terminação, e “STATUS”, que após a inicialização possui uma cópia da variável de entrada “PARAMS”.

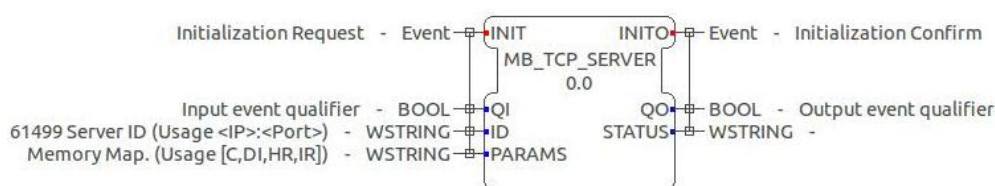


Figura 4.17 - CFB “MODBUS_TCP_SERVER”

As secções que se seguem documentam o processo de implementação dos diferentes componentes que formam “MODBUS_TCP_SERVER”, explicando igualmente o modo de funcionamento de cada um deles. Numa parte final é ainda abordada a forma como eles se interligam dentro do próprio CFB.

ServerModbusMem

O SIFB responsável por abrigar a memória Modbus e executar o serviço de escuta de pedidos denomina-se “ServerModbusMem”, e a sua interface é ilustrada na figura Figura 4.18. Como já foi referida, a memória Modbus é peça central na solução, pelo que, por associação, este SIFB pode ser considerado o componente chave do módulo Servidor.

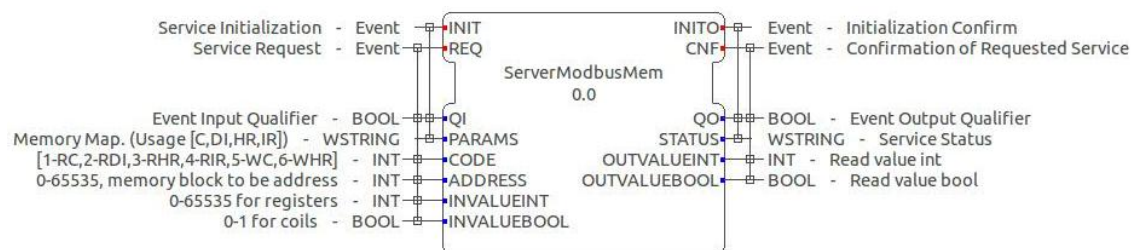


Figura 4.18 - SIFB “ServerModbusMem”

O modo de funcionamento deste SIFB é substancialmente mais complexo do que o dos elementos que foram até aqui apresentados. O diagrama UML de actividade da figura 4.19 - Diagrama UML de actividade do modo de funcionamento do SIFB “ServerModbusMem” esquematiza-o de uma forma sumária e simplificada.

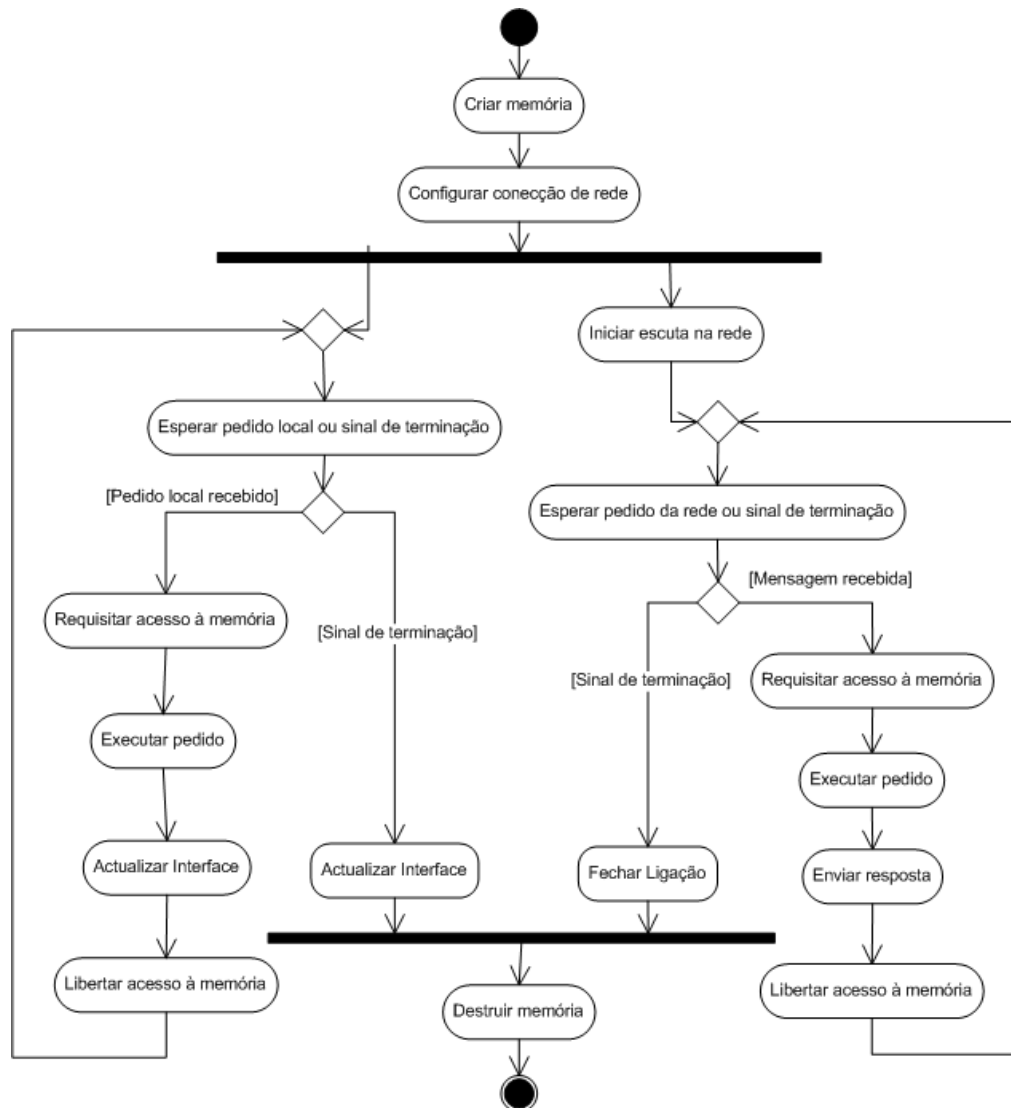


Figura 4.19 - Diagrama UML de actividade do modo de funcionamento do SIFB “ServerModbusMem”

Este SIFB entra em funcionamento quando recebe um evento de entrada “INIT+”, começando por criar a memória Modbus de acordo com a informação contida na variável “PARAMS” e por configurar a ligação que permitirá escutar pedidos Modbus vindos da rede. De seguida é criada a *thread* responsável por escutar e cumprir esses pedidos, representada na figura pelo ramo direito que tem origem no *fork*. Após este passo é emitido um evento “INITO+”, se nenhum dos passos anteriormente descrito resulte em erro, caso contrário o evento enviado é do tipo “INITO-“. Entretanto a nova *thread*, após iniciar o processo de escuta de pedido, entra num ciclo que só termina quando é recebida ordem de finalização. Este ciclo consiste sequencialmente em esperar a recepção de um pedido, requisitar acesso à memória quando um pedido chega, aceder à memória executando o pedido, construir e enviar a resposta e libertar o acesso à memória. É imperativo utilizar um método de gestão ao

acesso à memória pois este recurso é partilhado tanto pela *thread* filho como pela *thread* pai. Neste projecto foi utilizado o algoritmo de Dekker [23], que é um método de simples implementação e de validade comprovada. Paralelamente à *thread* filho corre a *thread* pai responsável por atender aos pedidos locais, que é sinónimo de reagir à recepção de um evento “REQ”, que sinaliza um pedido do programa de ler ou escrever um qualquer endereço da memória Modbus. Representado no diagrama UML pelo ramo esquerdo originário do *fork*, esta rotina começa por esperar o evento “REQ”, seguindo-se um pedido de permissão de acesso à memória. Após essa permissão ser concedida são utilizadas as variáveis de entrada para identificar o tipo de procedimento que deve ser tomado. A variável “CODE” identifica o tipo de memória e de operação (leitura/escrita) a que se refere o pedido, enquanto “ADDRESS” representa o endereço de memória envolvido. As outras duas variáveis de entrada associadas a “REQ” são usadas nos pedidos de escrita (“INVALUEINT” para *holding registers* e “INVALUEBOOL” para *coils*). De seguida são actualizadas as variáveis de saída, com “QO” a identificar se a operação foi bem-sucedida (“TRUE”) ou não (“FALSE”) e uma das outras duas variáveis (“OUTVALUEINT” e “OUTVALUEBOOL”) a conter o valor lido, no caso de se tratar de um pedido de leitura (a primeira variável é usada quando o valor lido é de um registo enquanto a segunda é usado quando o valor é de um bit). Após esta actualização é enviado um evento “CNF” e é libertado o acesso à memória.

O SIFB pode ser desligado, bastando para isso o programa enviar para este um evento “INIT-”. Este acontecimento é representado no diagrama como a transição “Sinal de terminação”, e tem como consequência a interrupção dos serviços de escuta de pedidos locais e de rede. Uma vez terminados esses serviços, destrói-se (limpa-se) a memória Modbus e emite-se um evento “INITO-” que sinaliza a desactivação do SIFB.

Relativamente às sequências de serviço que ilustram o modo de funcionamento “ServerModbusMem” podem ser consultadas as figuras mencionadas para os SIFBs do módulo cliente pois são igualmente válidas o componente aqui abordado.

ServersIpHandler

Conforme foi referido na secção de apresentação do CFB “MODBUS_TCP_SERVER” uma das variáveis de entrada consiste numa porção de texto que contém um código IP e um número de porta. Estes dados são usados pelo SIFB “ServersIPHandler”, ilustrado na figura para gerar os códigos que vão alimentar as variáveis de entrada ID dos 6 SIFBs SERVER” contidos no interior de “MODBUS_TCP_SERVER”. Na prática, a recepção de um evento “INIT” dá origem à actualização das diversas variáveis de saídas de acordo com o seguinte algoritmo: para a variável “RC” é feita uma cópia exacta do conteúdo de “IPPORT”, para “RDI” é feita uma cópia de “IPPORT” com a diferença de se adicionar uma unidade ao valor da porta, para “RHR” usa-se o mesmo procedimento que para “RDI” só que se adicionar duas unidades em vez de uma, e assim sucessivamente até “WHR”. Como é de esperar, após ser concluído o processo de actualização é enviado um evento “INITO” que sinaliza o fim do processo.

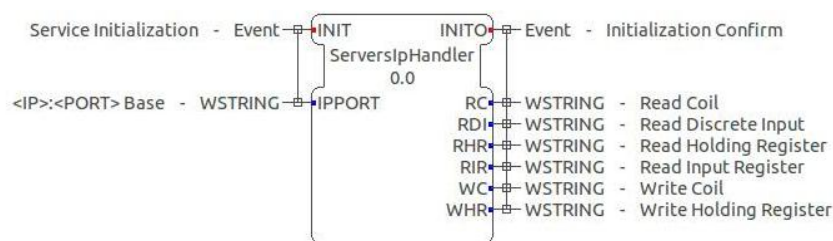


Figura 4.20 - SIFB "ServersIpHandler"

SIFBs SERVER

São utilizados dois tipos de SIFBs "SERVER" no CFB "MODBUS_TCP_SERVER". Um tipo para tratar dos pedidos de escrita e outro os de leitura. No primeiro caso são recebidas via rede duas variáveis que acompanham o evento "IND": uma corresponde ao endereço de memória aceder e outra o valor a escrever nessa memória. No segundo caso é recebida apenas uma variável a acompanhar o evento "IND", correspondendo ao endereço de memória a ler, mas por outro lado é enviada uma variável associada ao evento "RSP", variável essa que representa o valor lido no endereço indicado.

Este recurso é fornecido de origem pela ferramenta 4DIAC, pelo que para mais explicações sobre o modo de funcionamento dos SIFBs "SERVER" o leitor é encaminhado para o documento Doxigen fornecido pela própria ferramenta.

MUXModbusServer

Foi necessário desenvolver um recurso que mediasse a interação entre as saídas dos diferentes SIFBs "SERVER" e as entradas do SIFB "ModbusServerMem". Esse recurso, denominado FB "MUXModbusServer" e ilustrado na figura Figura 4.21, consiste num multiplexador programático de funcionamento bastante simples. Perante a recepção de um dos seis eventos de entrada são actualizadas as variáveis de saída com uma cópia do endereço associado ao evento accionado e com o código da função desejada, necessária para a jusante o ModbusServerMem possa interpretar o pedido que recebe.

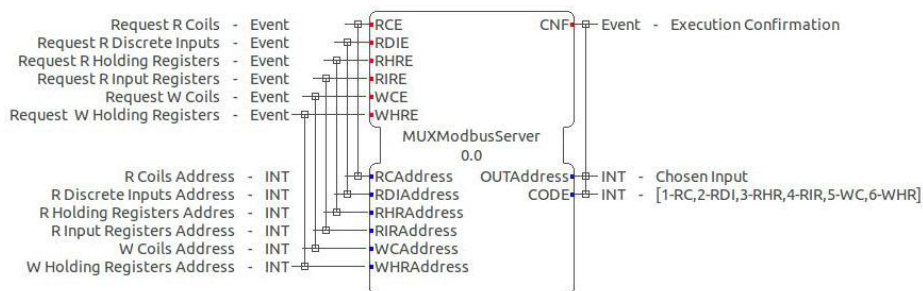


Figura 4.21 - FB "MUXModbusServer"

SWITCHModbusServer

De forma semelhante ao que acontece com a gestão da interação entre as saídas dos SIFBs "SERVER" e as entradas do SIFB "ModbusServerMem" foi necessário criar um elemento

que gerisse a interface entre as saídas do último e as entradas dos primeiros. É neste contexto que surge o FB “SWITCHModbusServer”, elemento cuja função é orientar o evento de saída CNF de “ModbusServerMem” para o SIFB “SERVER” devido, auxiliando-se do código que codifica o tipo de pedido para executar essa função.

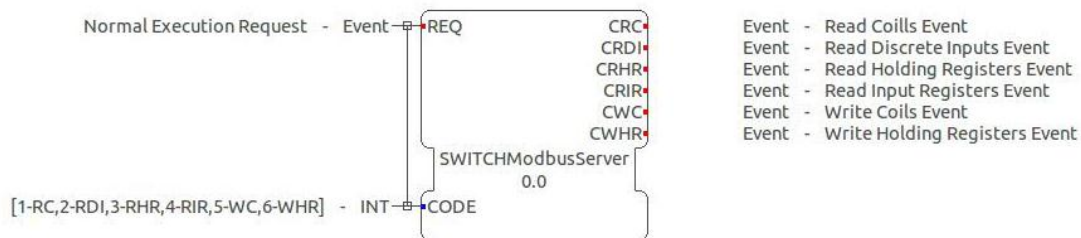


Figura 4.22 - FB “SWITCHModbusServer”

Estrutura interna

A arquitectura das ligações entre eventos e dados de diferentes componentes do “MB_TCP_SERVER” é relativamente complexa, e como resultado as imagens obtidas do 4DIAC da rede de componentes são tão pouco elucidativas ao ponto de não estarem presentes neste documento. A forma mais eficaz de analisar a arquitectura é assumir uma perspectiva de mais alto nível, realçando não os traçados individuais de cada conexão mas antes os diferentes fluxos de sinais que regem o modo de funcionamento do CFB, pelo que os próximos parágrafos procuram fazer isso mesmo.

No que se refere aos processos de inicialização e finalização do recurso, o evento de entrada INIT responsável por controlar esses processos é repercutido ao longo de todos os elementos que necessitam de inicialização e finalização através do par de eventos “INIT”/“INITO”, começando pelo SIFB “ServerModbusMem”, seguindo-se o SIFB “ServersIpHandler” e por fim os diferentes SIFBs “SERVER”. Após este percurso o sinal é orientado para o evento de saída “INITO”.

Quando ao fluxo de sinal associado à qualificação de eventos, representado pelo par “QI”/“QO”, começa na entrada “QI” do CFB, acompanhando o sinal “INIT” até ao SIFB “ServerModbus”, sendo que depois distribuído pelos diferentes SIFBs “SERVER”. A variável “QO” do último deste SIFBs é ligada à saída “QO” do CFB, assegurando que a sinalização do sucesso ou insucesso do processo de inicialização funciona correctamente.

O fluxo de sinal que advém da variável ID do CFB é direccionada para a entrada “IPPORT” do SIFB “ServersIpHandler”, com as saídas deste a serem distribuídas pelos respectivos SIFBs “SERVER”.

As variáveis de saída dos diferentes SIFBs “SERVER” são encaminhadas para as respectivas variáveis de entrada do FB “MUXModbusServer”, de modo a serem filtradas. As variáveis de saída deste FB são direccionadas para as entradas do SIFB “ServerModbusMem”. De igual forma, as saídas “OUTVALUEINT” e “OUTVALUEBOOL” são direccionadas para os SIFBs “SERVER” correspondentes, fechando o ciclo inerente à recepção e execução de um pedido local, sendo o “SWITCHModbusServer” usado para, como já foi referido, direccionar o evento de saída CNF para o SIFBs Server apropriado.

4.6 - Testes e Validação

Os testes efectuados sobre as soluções desenvolvidas assentaram em 3 fases de avaliação: primeiro avaliou-se a solução que implementa o cliente Modbus, depois a solução que implementa o servidor Modbus, e por fim testou-se a performance das duas ferramentas com estas a operarem uma com a outra.



Figura 4.23 - EtherTRACK I/O ET-8DI2-8DO2-H

Para testar os SIFBs que implementam a solução cliente Modbus recorreu-se a um dispositivo servidor Modbus EtherTRACK I/O modelo ET-8DI2-8DO2-H e uma ICnova executando o programa FORTE, com os dois ligados em rede através da intranet da FEUP. As aplicações de teste descarregadas para o FORTE consistiam na escrita e leitura sucessiva de diferentes espaços de memória Modbus do servidor. Para o caso das *coils* a aplicação começava por utilizar o “mm_write_coils” para escrever num determinado endereço e posteriormente usava o “mm_read_coils” para resgatar o valor escrito, de modo a comprovar a validade das operações. Esse método também foi usado para a memória do tipo *holding registers*, que também é passível de ser escritos, usando para isso o par de SIFBs “mm_write_holding_registers” e “mm_read_holding_registers”. Para os restantes SIFBs “mm_read_discrete_inputs” e “mm_read_input_registers” foram executadas aplicações que procedessem à leitura de endereços de memória do servidor que continham valores definidos na altura da configuração do dispositivo. No fundo, todos estes testes foram do tipo “leitura de valores escritos”. Foram realizadas séries de 5 testes deste género para cada um dos diferentes tipos de memória. Em todos os casos, os pedidos de escrita foram bem-sucedidos (as mensagens de resposta recebidas pelo cliente indicavam isso mesmo), assim como os de leitura (os valores retornados foram aqueles que tinham sido escritos anteriormente), daí os testes terem sido considerados como bem-sucedidos.

Foram feitos testes adicionais com o objectivo de incorporar a solução em cenários não-ideais. Num desses testes foi adicionado mais um dispositivo EtherTRACK ao modelo representado na Figura 4.24, com a aplicação executada pelo a utilizar uma mesma instância do SIFB “mm_write_coils” para enviar pedidos aos dois servidores. Este teste, que atestou a capacidade da solução de gerir as ligações aos servidores, foi igualmente satisfatória. Uma última prova consistiu em correr uma aplicação que contivesse várias instâncias do SIFB “mm_write_coils” todas configuradas de forma a enviarem pedidos para um mesmo servidor. Como todos os pedidos foram realizados satisfatoriamente, se conclui-se que o teste também tinha sido bem-sucedido.

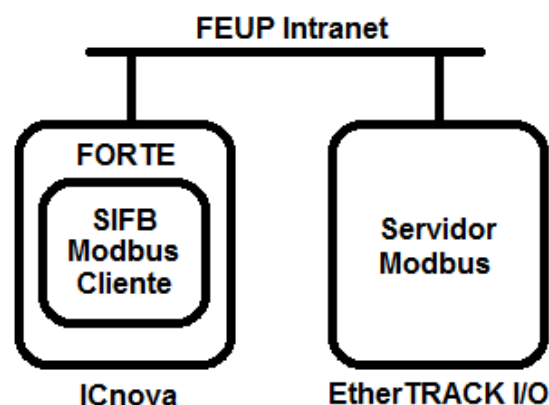


Figura 4.24 - Modelo do teste aos SIFB Modbus Cliente

Relativamente aos testes da solução que implementa o servidor Modbus, este foram divididos em duas partes: teste da resposta a pedidos de feitos pelos SIFBs cliente desenvolvidos e teste da resposta a pedidos feitos fora do âmbito da norma IEC 61499. Ambos os testes usaram dois PCs interligados através da intranet da FEUP.

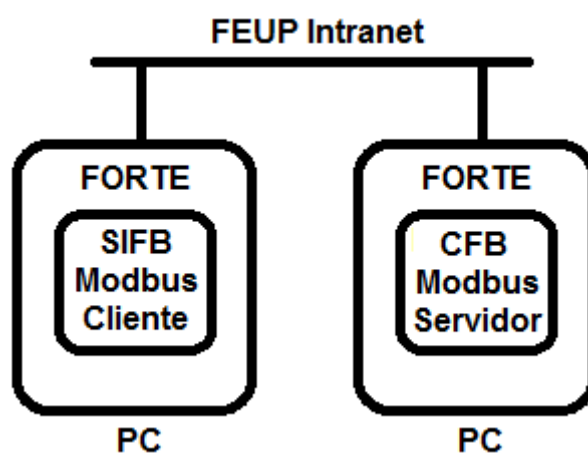


Figura 4.25 - Modelo do teste da resposta a pedidos feitos pelos SIFB Modbus Cliente

No primeiro teste, cujo modelo se encontra representado na Figura 4.25, cada um dos PCs continha o FORTE em execução. Para um dos *runtimes* foi transferida uma aplicação que consistia no CFB “MODBUS_TCP_SERVER”, ou seja o servidor Modbus desenvolvido. Para o outro foi transferida uma aplicação que utilizava os diferentes SIFB que foram desenvolvidos para implementar o cliente Modbus para enviar pedidos para o CFB “MODBUS_TCP_SERVER” a executar no outro PC. Desta forma uma das aplicações implementava um servidor Modbus enquanto o outro representava um cliente Modbus.

Tal como num dos testes descritos anteriormente, foi utilizada a técnica de escrever um determinado valor num espaço de memória e ir recuperá-lo posteriormente, sendo este processo repetido múltiplas vezes e para cada um dos 4 tipos de memória. Em termos de resultados, os pedidos de escrita foram correctamente executados pelo servidor e os pedidos de leitura retornaram os valores anteriormente escritos, pelo que os testes foram considerados bem-sucedidos. Em momento algum foi registado algum erro no processo de troca de mensagens.

Para testar a resposta a pedidos com origem em entidade exteriores à norma IEC 61499 visto, foi usado um simulador Modbus Poll [24], que permite emular um cliente Modbus. Este programa foi executado num PC, enquanto o FORTE foi executado no outro. Para essa instância em execução do FORTE foi transferida a aplicação já utilizada no teste anterior que continha o “MODBUS_TCP_SERVER”. O procedimento de teste propriamente dito foi idêntico ao descrito acima, com o simulador a ser utilizado para escrever e ler nos diferentes espaços de memória do servidor. Também neste teste os resultados foram satisfatórios.

4.7 - Conclusão

O presente capítulo documentou todo o processo que envolveu implementar o protocolo Modbus TCP na ferramenta 4DIAC. Começou-se por fazer uma breve referência ao protocolo, para que o leitor ficasse contextualizado o suficiente para entender a tarefa que foi executada. De seguida foram enunciadas e debatidas as diferentes possibilidades que foram ponderadas para a arquitectura, sendo justificado o porquê de a escolha final recair sobre uma gama de 6 SIFBs independentes entre si para implementar a entidade cliente e sobre um complexo CFB para representar a entidade servidor. A fase de implementação foi a etapa a ser documentada na secção seguinte, contendo esta uma descrição dos módulos implementados, descrição essa feita com o necessário detalhe técnico sem que ao mesmo tempo se torne de difícil entendimento para o leitor. Por último, foram descritos os testes a que a solução desenvolvida foi submetida de forma a esta poder ser validada.

Confrontando os requisitos descritos no início do capítulo com os resultados obtidos pode-se afirmar que a solução desenvolvida enquadra-se perfeitamente no que era pretendido nesta parte da dissertação. A solução não só cumpre os requisitos como apresenta um potencial de crescimento assinalável, faltando ainda assim testar a solução desenvolvida num ambiente mais real de modo a poderem ser avaliados algumas das suas características que não puderam ser testadas. Seria interessante testar o seu comportamento perante uma rede com baixa disponibilidade ou num sistema que exigisse uma maior taxa de troca de mensagens, para verificar a percentagem de pacotes perdido de modo a testar a eficiência da solução.

Avaliando em termos técnicos a solução, a eficácia desta ficou provada com os testes realizados. Ao serem simulados diferentes cenários foi possível avaliar as diferentes funcionalidades da solução, comprovando que todas essas partes respondiam de forma satisfatória. Quando à eficiência, e na ausência de uma outra ferramenta que sirva de comparação, apenas pode ser dito que o processo de desenvolvimento de código teve sempre em consideração este factor, racionando da melhor forma possível os recursos disponíveis.

Uma última referência para a arquitectura das duas metades da solução. No caso do módulo cliente, a opção de utilizar vários SIFBs revelou-se feliz pois ofereceu ao módulo uma flexibilidade que lhe permite ser facilmente incorporado em qualquer aplicação. Já no caso do módulo servidor, essa flexibilidade não está tão marcada. Tal como foi enunciado na secção 4.4.3 - Cliente, os conceitos que estão na base do protocolo Modbus e da norma IEC 61499 não são completamente compatíveis, pelo que esse facto não foi sentido no desenvolvimento do módulo cliente mas foi claramente identificado no processo de estruturação do módulo servidor. Tal como essa mesma secção documenta, diferentes abordagens foram avaliadas, cada uma com as suas vantagens e desvantagens tidas em

consideração. Por fim, foi escolhida uma arquitectura que satisfaz os requisitos e acaba por não comprometer muito ao nível da flexibilidade.

Capítulo 5

Sistema IEC 61499

5.1 - Introdução

O principal desafio abordado neste projecto foi desenhar e implementar uma arquitectura base para uma nova plataforma, estruturada segundo a norma IEC 61499, que surgisse como uma alternativa ao actual sistema utilizado para a gestão técnica da FEUP. Para assistir a esse processo realizou-se um estudo da tecnologia 4DIAC que resultou no capítulo 3 e desenhou-se e implementou-se um recurso que conferisse a capacidade de comunicar via protocolo Modbus a essa tecnologia, exercício este que resultou no capítulo 4. Este novo capítulo documenta a forma como, utilizando os recursos mencionados, foi desenvolvida uma solução que respondesse a esse desafio principal.

É importante nesta introdução mencionar os limites da nova plataforma que se pretende architectar. Tendo em conta a plataforma actualmente implementada, o campo de acção da nova arquitectura restringe-se à aos controladores de campo e à base de dados. Quanto aos controladores, a nova arquitectura impõe que os PLCs sejam substituídos por dispositivos com outras características (tal como já foi mencionado no capítulo introdutório) e que sejam desenhados novas aplicações de controlo. Em relação à base de dados, esta apenas pode ser consultada para extrair as configurações determinadas pelo utilizador (não é permitido editar a informação lá contida). Logicamente, os novos controladores e as respectivas aplicações podem interagir com o SCADA e com os I/Os remotos para poderem implementar os serviços de controlo e supervisão.

Em termos de estrutura, este capítulo começa por enumerar os requisitos que foram apresentados à solução a desenvolver, seguindo-se a descrição e discussão de propostas para a arquitectura da solução, que conduz à fase de implementação da arquitectura escolhida. Por último são descritos os testes efectuados para validar o recurso desenvolvido e os respectivos resultados obtidos, sendo todas estas subsecções discutidas na parte final de conclusões.

5.2 - Requisitos

Desenhar uma arquitectura não é sinónimo de implementar uma solução final. Neste sentido, o que é exigido como produto final desta dissertação não é uma ferramenta completa para ser utilizada na FEUP como plataforma de gestão de recursos. Esse produto deve consistir numa nova arquitectura, direccionada à uma nova plataforma de gestão de recursos, devidamente validada que sirva de base a uma futura implementação real. Os requisitos de seguida apresentados são relativos exactamente a essa arquitectura.

- A nova arquitectura deve permitir que as aplicações de controlos assegurem os mesmos serviços que são fornecidos pelas aplicações do sistema actualmente implementado, nomeadamente controlo dos circuitos eléctricos de acordo com as configurações definidas pelo utilizador (horários, alarmes, *setpoint*) e interface com o SCADA central;

- Essas novas aplicações de controlo têm de ser estruturadas segunda a norma IEC 61499;

- A nova arquitectura deve permitir um elevado grau de autonomia, de forma a diminuir a necessidade intervenção humana;

- Deve ser assegurado que a informação contida na base de dados principal do sistema actualmente implementado possa ser utilizada no processo de controlo dos equipamentos.

- O processo de controlo deve-se adaptar de forma dinâmica às configurações que vão sendo dadas pela base de dados (daí a expressão no título da dissertação “plataforma configurável”).

5.3 - Discussão da Arquitectura

Depois de uma fase de análise e avaliação, foram eleitas duas possíveis arquitecturas de dessem resposta ao objectivo principal da tese: a primeira baseia-se na geração automática de aplicações enquanto a segunda envolve a reconfiguração dinâmica de parâmetros de parâmetros. Os parágrafos que se seguem explicam de forma sucinta cada uma dessas opções e avaliam as respectivas vantagens e desvantagens.

A opção de gerar automaticamente as aplicações de controlo implica o desenvolvimento de uma ferramenta que crie de forma automática as aplicações de controlo destinadas às ICnova's. Em termos de funcionamento, após a geração de uma nova série de aplicações de controlo estas iriam substituir as que estivessem a ser executadas pelos dispositivos embebidos. Essa ferramenta de geração de novas aplicações utiliza como *inputs* a informação disponibilizada pela base de dados e uma série aplicações de controlo genéricas. Da combinação das duas resultam as aplicações finais, capazes de executar os serviços necessários tendo em conta as configurações determinadas pelo utilizador. Com esta opção cada um dos controladores possui independência dos restantes, pois cada um deles apenas realiza controlo local. Após serem geradas, as novas aplicações substituem aquelas que estivessem a ser executadas nos controladores.

A outra opção contempla a utilização, em cada controlador, de uma aplicação imutável em termos de estrutura mas que aceita dados via rede. Essas configurações consistem nas configurações determinadas pelo utilizador e que estão contidas na base de dados. Neste caso cada controlador possui uma aplicação customizada de acordo com os serviços que fornece. As aplicações de controlo necessitariam de utilizar SIFBs que realizassem a interface entre a aplicação de controlo e a configurações recebidas (a solução lógica seria usar uma base de dados local e um respectivo gestor). Seria igualmente necessário desenvolver uma ferramenta

que garantisse a actualização das configurações locais enquanto as aplicações se encontrassem em execução. Para implementar esta última funcionalidade seria necessário utilizar um controlador central que comunicasse com todos os outros aonde fosse executada uma aplicação com o único propósito de distribuir as configurações controladores, mantendo desta forma todo o processo dentro da norma IEC 61499, ou então usar um método como a partilha ou transferência de ficheiros, o que faria com que o processo saísse um pouco do âmbito da norma. Esta opção é, no seu núcleo, idêntica àquela que rege a plataforma actualmente implementada na FEUP.

Escolher uma hipótese entre as duas foi uma tarefa árdua. A primeira hipótese garante maior flexibilidade, enquanto a segunda praticamente anula o *downtime* das aplicações de controlo. A primeira assegura melhor portabilidade, enquanto a segunda oferece um processo de implementação mais fácil. A primeira apresenta maior potencial de evolução, enquanto a segunda faz um uso mais eficiente dos recursos dos dispositivos embebidos. Analisada mais ao pormenor, a avaliação das duas opções resultou nas seguintes conclusões:

- A primeira opção permitiria adicionar novas faculdades aos programas de controlo ou reformar as antigas de uma maneira mais fácil do que a segunda, pois como as próprias aplicações incluem as configurações dadas pelo utilizador, seriam evitados problemas de compatibilidade entre as partes de processo de controlo e dados de controlo após serem introduzidas alterações numa das partes. A expressão dados de controlo é usada para nomear a parte das aplicações que guardas a informação necessária à execução dos serviços, enquanto processo de controlo é a denominação usada para referir a parte das aplicações que usa os dados de controlo para executar os serviços. Como na segunda opção existe esta separação entre processo de controlo e dados de controlo a questão da cooperação entre as partes já tem de ser levada em conta num processo de reformulação. Daí que a primeira hipótese possa ser considerada mais flexível que a segunda;

- A perspectiva de evitar quer a reiniciação do FORTE quer a interrupção do processo de controlo aquando de uma actualização dos dados de controlo é extremamente apelativa. A opção de utilizar parâmetros de configuração oferece tal possibilidade, pois ela possibilita que os dados de controlo possam ser actualizados enquanto o FORTE está em execução. A implementação desta faculdade envolveria a actualização das bases de dados locais de forma independente da parte das aplicações responsáveis pelo processo de controlo, algo perfeitamente exequível. A opção de gerar automaticamente aplicações de controlo não permite esta actualização de dados enquanto o FORTE está *online* (em execução), pois para que a actualização de horários se fizesse reflectir no sistema de controlo era necessário que novas aplicações fossem geradas e descarregadas para os controladores. Por outro lado, e indo ao encontro do trabalho de investigação documentado em [10], é possível efectuar alterações estruturais às aplicações baseadas em IEC 61499 enquanto estas estão *online*. Nesse sentido, seria possível, numa versão mais avançada da opção de gerar automaticamente as aplicações, actualizar os dados de controlo das aplicações em execução sem necessitar que estas fossem substituídas por outras, o que implicaria *downtime*.

- Apesar de não ser uma questão fulcral, visto que aplicação da solução desenvolvida encontra-se para já restringida à FEUP, a portabilidade do sistema de controlo ganha uma certa importância quando se pensa na possibilidade de vir a ser alterado alguma componente de *hardware*, em especial o controlador embebido. Caso um novo dispositivo fosse escolhido para substituir a ICnova, por questões de preço ou de performance, o ideal seria não ser necessário realizar qualquer tipo de alterações ao nível de *software*. A utilização de um

gestor de base de dados compromete esse desejo e, por consequência, a segunda opção passa a apresentar problemas de portabilidade. Vale a pena explicar que o gestor de base de dados não ajuda na portabilidade de uma solução final pois esta passa a depender de *software* estranho à tecnologia 4DIAC que pode ou não estar disponível na nova plataforma para onde se pretende portar a solução.

- A hipótese de se gerar as aplicações automaticamente implica desenvolver uma aplicação genérica que se adapte à informação contida na base de dados. Implica também desenvolver uma ferramenta que execute esse processo de adaptação e faça a transferência das aplicações para os vários dispositivos da forma mais autónoma possível. Do outro lado, as aplicações são escritas directamente e, apesar de terem de ser personalizadas para cada controlador de acordo com os serviços que este fornece, elas seriam muito idênticas entre si, podendo muitas porções do código serem reutilizadas por diferentes aplicações. Neste sentido, a solução que envolve a reconfiguração de parâmetros aparenta ser de mais fácil implementação que a outra.

- A questão do potencial de evolução está intimamente ligadas com as três primeiras características discutidas nos parágrafos anteriores (flexibilidade, capacidade de evitar *downtime* e portabilidade). Analisando esses 3 itens conclui-se que a opção de gerar aplicações automaticamente parece a mais promissora. Por potencial de evolução entenda-se a perspectiva que a solução oferece no sentido de ser melhorada para lá do que é exigido pelos requisitos específicos desta dissertação, prevendo uma possível adaptação do sistema a um novo projecto ou o surgimento de novos requisitos ao sistema actual.

- Nesta fase de discussão de propostas não é fácil prever a eficiência que cada uma das opções é capaz de oferecer, mas ainda assim é possível afirmar a utilização de aplicações com bases de dados locais garante que os dados do processo estejam compactados, algo que não pode ser dito sobre a outra hipótese, aonde esses dados estão forçosamente contidos em estruturas IEC 61499 que, dadas as suas complexidades, envolver mais uso de memória. Novamente de realçar que estas conclusões são algo conjunturais, pois só depois de uma fase de implementação poderiam ser efectuados testes que as comprovassem ou desmentissem.

Perante as conclusões expostas nos pontos anteriores foi escolhida a solução de gerar automaticamente as aplicações de controlo como a que seria usada no projecto. Os principais factores que pesaram nesta decisão foram o potencial que essa solução apresentava juntamente com o desafio de implementar um sistema que não envolvesse qualquer tipo de base de dados local. Assim todas as operações efectuadas pelas aplicações são transparentes, por oposição à opção de utilizar SIFBs que ocultem essas mesmas operações.

5.4 - Descrição da arquitectura escolhida

Em termos de *hardware*, a nova arquitectura proposta apenas implica a substituição dos PLCs por novos controladores embebidos, de acordo com já foi descrito anteriormente. A base de dados, o SCADA e os I/Os remotos permanecem inalteráveis, apesar de numa implementação futura estes (principalmente a base de dados) possam ser alterados de forma a ser tirado o maior partido das faculdades oferecidas pela nova arquitectura.

A arquitectura de *software* da solução proposta, esquematizada na Figura 5.1, funciona da seguinte forma: a base de dados fornece a informação relativa às configurações definidas pelo utilizador ao programa de geração de aplicações. Este módulo, usando também aplicações genéricas pré-definidas, cria aplicações finais de controlo. Essas aplicações de

controlo são então distribuídas pelos controladores e executadas com auxílio do FORTE. De notar que esta descrição apenas se refere à gestão das aplicações de controlo a serem executadas pelos controladores embebidos porque esse processo é o único a ser reformulado quando se compara com a arquitectura do sistema actualmente implementado na FEUP.

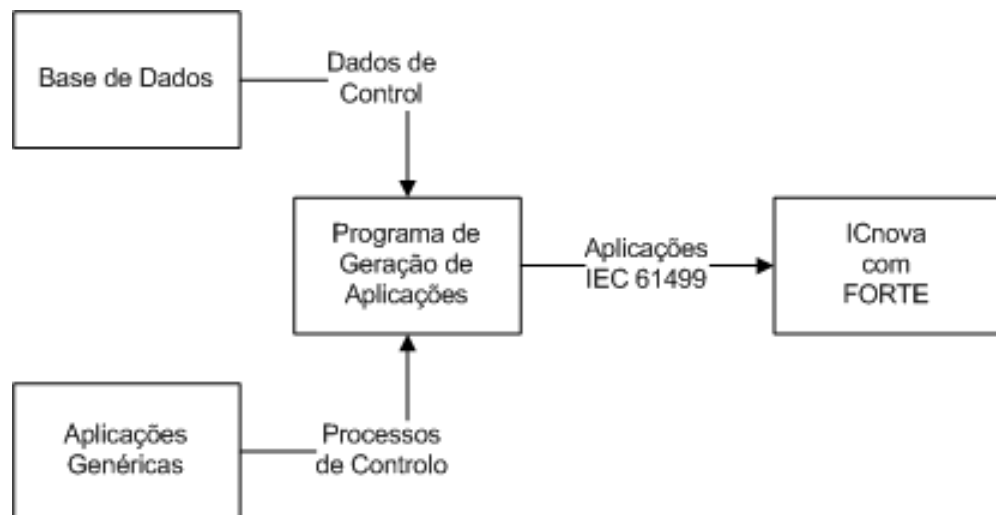


Figura 5.1 - Diagrama da arquitectura de *software* proposta

Cada aplicação genérica implementa um dos serviços que são fornecidos pelos controladores (cumprimento de horários, a interface com o SCADA, a gestão de alarmes, entre outros) e está estruturada, logicamente, segundo a norma IEC 61499. Tomando como exemplo o serviço de cumprimento de horários, a aplicação genérica associada a este serviço implementa a comunicação entre a aplicação e os I/Os remotos, mas não possui nenhum horário concreto (hora de activação, hora de desactivação, identificação do circuito, identificação do I/O remoto associado ao circuito, etc.). Logicamente que a escolha desta arquitectura implica o desenvolvimento de todas estas aplicações genéricas.

Para serem obtidas aplicações finais a partir da combinação das aplicações genéricas e da informação fornecida pela base de dados esta arquitectura contempla um programa destinado unicamente para esse propósito. Ele é responsável por juntar as partes de controlo de processo e de dados do processo de forma a serem geradas as aplicações finais capazes de serem utilizadas pelos programas FORTE executados nos diversos controladores espalhados pela FEUP.

Com o propósito de validar a arquitectura sugerida, foi architectado e implementado (apenas) o serviço de cumprimento de horários, pelas seguintes razões:

- O serviço de cumprimento de horários pode ser considerado o mais crucial e aquele que está na base das aplicações de controlo;
- O cumprimento de horários implica a utilização dinâmica da informação da base de dados (pois esta pode ser alterada) e obriga a que as aplicações interajam com os I/Os remotos usando o protocolo Modbus TCP;
- Todos os serviços são, no seu essencial, regidos pelo mesmo princípio de funcionamento (actuação de acordo com as configurações da base de dados e/ou SCADA e interacção via Modbus TCP com os I/Os remotos), pelo ao ser implementada e validada uma aplicação que assegure o serviço de cumprimento de horários fica provado que a arquitectura que suporta

essa aplicação também suportaria as outras aplicações que implementassem os restantes serviços.

- A implementação de todos os serviços seria impossível de efectuar dados os limitados recursos e o tempo disponíveis para a realização da dissertação.

No fundo, os referidos serviços são homogéneos ao ponto de se poder garantir que ao se conseguir implementar um deles (neste caso o cumprimento de horários) usando a nova arquitectura, se prova que todos podem ser implementados com sucesso segundo essa nova arquitectura. Esta opção só é possível porque o objectivo desta tese é desenhar e validar a arquitectura de uma nova plataforma/sistema e não implementar todos os serviços da gestão técnica da FEUP.

Os pontos que se seguem descrevem a arquitectura desenhada para possibilitar a implementação do citado serviço. Obviamente este desenho tem por base a arquitectura ilustrada na Figura 5.1.

5.4.1 - Aplicação de controlo genérica

De acordo com os requisitos expostos no início deste capítulo, as aplicações de controlo finais devem ser capazes de conter informações fornecida pela base de dados e fazê-la reflectir sobre a lógica de controlo dos equipamentos. No caso do serviço de cumprimento de horários, essa informação consiste em referências temporais (dia da semana e hora) para activação e desactivação de equipamentos e em códigos identificativos dos circuitos eléctricos a controlar. Neste contexto, a aplicação de controlo genérica referente ao cumprimento de horários deve implementar totalmente o processo de controlo (comunicação com I/Os remotos e gestão dos horários) e deve estar preparada para incorporar essa informação relativo aos horários. Estes dois requisitos deram origem a duas fases no processo de desenho da aplicação genérica, fases essas descritas de seguida.

Dados de controlo

Descartada que foi a opção de utilizar um gestor de bases de dados local para lidar com os dados de controlo, o caminho tomado para incorporar esta informação na aplicação final de controlo foi criar um novo tipo de FB alojasse essa informação. Na prática essa informação consiste numa série de horas de ligar e desligar para cada um dos circuitos associados a um determinado controlador, à qual se junta a identificação interna do circuito, para definir a que equipamento se refere cada horário. Perante esta premissa, a arquitectura da aplicação genérica contempla um novo FB (designado “scheduleONOFF”) que aceita como *inputs* um par de referências temporais (dia da semana e hora) que correspondem aos momentos de ligar e desligar um determinado equipamento. Como funcionalidades, esse FB é ser capaz de alertar a aplicação quando chegam os momentos de ligar e desligar.

Com um FB específico para conter a informação dos momentos de ligar e desligar um equipamento restou determinar, no que se refere à informação dos horários, de que maneira a aplicação lidaria com a identificação interna de cada equipamento, sabendo que cada horário está associado a uma dessas identificações e cada identificação consiste num código IP de um I/O remoto e no número de endereço de memória Modbus correspondente à saída física do circuito eléctrico. Para incorporar essa informação na aplicação foi decidido associar a cada instância do FB descrito no parágrafo anterior um novo CFB que usasse o recurso Modbus documentado no capítulo anterior, inserindo directamente neste último a informação

referente à identificação interna. É usado um CFB e não directamente um dos SIFBs que compõe o módulo cliente Modbus para que a ligação cliente-servidor seja automaticamente criada antes e destruída depois da execução de um pedido.

Esta arquitectura escolhida para o modelo de gestão dos dados de controlo pode não assegurar uma optimização do uso da memória, visto a informação estar distribuída por vários FBs, mas oferece uma estrutura simples, robusta e flexível.

Processo de controlo

Com a estrutura de dados definida, restou desenhar a forma como as aplicações executam as operações de controlo. Como os requisitos impostos obrigam a que as aplicações de controlo sejam capazes de ligar e desligar equipamentos de acordo com a informação contida nos horários, a estrutura definida para o processo do controlo consiste numa operação cíclica de regulação. Na prática, a aplicação analisa periodicamente todas as instâncias dos FBs “scheduleONOFF” para verificar se é necessário ligar ou desligar algum equipamento. Caso ocorra uma correspondência são utilizadas as funcionalidades oferecidas pelo recurso Modbus desenvolvido para enviar os comandos de controlo para os I/Os remotos.

Para que esta arquitectura seja possível é necessário fornecer aos FBs “scheduleONOFF” a data e hora em tempo real. Daí nasce a necessidade de se criar um SIFB que, usando os recursos fornecidos pelo sistema operativo que aloja o FORTE, consiga indicar essa data e essa hora.

Os pontos fortes desta arquitectura são a sua simplicidade, flexibilidade e escalabilidade, se bem que a opção por realizar *poolings* periódicos (questionar em ciclo as estruturas que alojam os dados de controlo) não vai completamente ao encontro dos conceitos em que se baseia a norma IEC 61499 (esta privilegia fluxo de eventos em detrimento de execuções cíclicas).

5.4.2 - Programa para geração automática de aplicações

O próximo passo dado no processo de definição da arquitectura do *software* do serviço de cumprimento de horários foi desenhar a forma como iriam ser geradas as aplicações finais a partir da interacção entre a aplicação genérica e os horários da base de dados. As duas únicas formas viáveis para desenhar uma aplicação associada à tecnologia 4DIAC sem usar directamente o IDE consistem em escrever directamente código sob a forma texto, daí se ter recorrido a um programa escrito em C++.

Uma das formas de criar uma aplicação sem usar o IDE passa por escrever o ficheiro XML que contém toda a informação de uma aplicação. A implementação desta escolha seria razoavelmente complicada, dada a complexidade do ficheiro, e para além disso seria necessário usar o 4DIAC-IDE posteriormente para a parte de transferência da aplicação, algo que não é desejado.

A outra opção passa por criar um ficheiro do tipo “.bfile” que contenha as mensagens necessárias para que o FORTE crie e execute a aplicação desejada. Esta segunda opção apresenta-se com mais exequível do que a primeira porque o tipo de ficheiro “.bfile” possui uma estrutura bastante mais simples do que a ficheiro XML e porque é possível emular a transferência da aplicação sem recorrer ao 4DIAC-IDE. Para ser rigoroso, também na primeira opção seria possível emular essa transferência, mas essa implementação seria mais custosa visto que esse processo teria de ser uma clonagem perfeita da funcionalidade do 4DIAC-IDE

para existir a compactibilidade necessária com o FORTE. A segunda opção permite que essa implementação seja feita com maior liberdade visto que a interacção com o *runtime* é directa.

O programa gestor da base de dados principal do sistema actualmente implementado permite exportar os horários sobre a forma de tabelas contidas em ficheiros “.txt”. Como consequência, a arquitectura do programa C++ torna-se extremamente linear: ele utiliza o modelo definido pela aplicação genérica e adiciona-lhe o número necessário de instâncias *scheduleONOFF* (e o CFB associado) para incorporar todas as entradas da tabela contida no ficheiro “.txt” usado como *input*. Logicamente que o programa C++ gera um número de aplicações igual ao número de controladores referenciados no ficheiro “.txt” (de modo a cada aplicação ser transferida para o devido controlador).

5.4.3 - Transferência de aplicações

O último passo deste processo fase consistiu em desenhar a forma com uma aplicação contida num *bootfile* seria transferida para FORTE, isto porque a ferramenta 4DIAC não possuía ainda essa funcionalidade.

O procedimento escolhido para responder a esse desafio envolveu alterar o CFB “DM_KRNL” de forma a ser criada uma forma alternativa de interagir com o “DEV_MGR”. O modelo actual funciona de maneira ao SIFB “SERVER”, contido no interior do CFB, receber as mensagens via rede e redireccioná-las para o SIFB gestor. A arquitectura sugerida envolve um novo SIFB que extrai as mensagens de um *bootfile* local. De relembrar que o processo de *deployment* no FORTE foi devidamente explicado na secção 3.4.5 -

A acompanhar o novo SIFB estão outros componentes capazes de implementar um funcionamento mutuamente exclusivo entre este novo elemento e o já pré-existente “SERVER”, de modo ao FORTE aceitar a aplicação ou por via rede ou por configuração local. Desta forma a solução desenvolvida torna-se genérica e passa a poder ser usada em qualquer projecto.

Foi necessário definir a forma como o FORTE saberia se deveria aceitar a aplicação via rede ou por ficheiro local. A decisão tomada foi incorporar no CFB “DM_KRLN” cuja única função é seleccionar uma opção entre as duas, com o factor de decisão a residir na existência ou não de um *bootfile* num determinado *path* pré-definido. Assim, logo no momento de iniciação de execução, o FORTE verifica a presença do ficheiro nessa localização e activa o modo de *deployment* local se o ficheiro existir ou activa o modo de *deployment* por rede se não existir. Uma alternativa a este processo seria o *runtime* receber, via argumentos de linha de comandos, um *path* um nome de ficheiro que o utilizador usaria para indicar ao FORTE que desejava executar um aplicação contida num ficheiro local. Essa hipótese foi excluída porque, para além de ser de maior complexidade de implementação, implicaria alterações significativas no código fonte do FORTE, algo que não é desejado sem o envolvimento directo de um dos responsáveis da iniciativa 4DIAC, sob a pena de se perder estar a restringir o alcance de utilização da solução (tornar a implementação pouco genérica). Deve ser referido que a solução escolhida tem como desvantagem a necessidade de se proceder a uma recompilação do FORTE quando é pretendido alterar o *path* do *bootfile*.

5.5 - Implementação do serviço de cumprimento de horários

Esta secção documenta os processos de implementação da arquitectura para o serviço de cumprimentos de horários que foi descrito na secção anterior. É explicada a forma como foram construídos a aplicação genérica e o programa de geração de aplicações que estão contemplados na arquitectura, e é também descrita a forma como se desenvolveu uma solução que possibilita a transferência de uma aplicação para o FORTE de forma a ela poder ser executada.

5.5.1 - Aplicação genérica

A implementação da aplicação genérica seguiu a estrutura definida na secção anterior, e encontra-se representada na Figura 5.2. Os parágrafos que se seguem explicam cada um dos elementos que compõem a aplicação e ainda descrevem a forma de funcionamento da mesma.

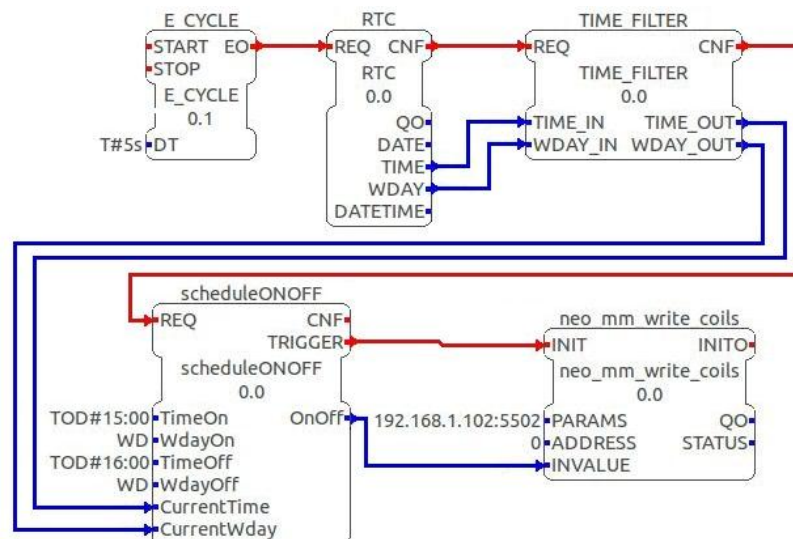


Figura 5.2 - Aplicação genérica relativa ao serviço de cumprimento de horários

Dados de controlo

De acordo com a arquitectura sugerida no capítulo anterior, a solução de incorporar a informação dos horários na aplicação passou por desenvolver um tipo de FB que alojasse esses dados. Esse novo FB, designado “scheduleONOFF”, tem a interface ilustrada na Figura 5.3. Usa dois pares de variáveis de entrada que especificam marcas temporais (hora e dia da semana) relativas aos momentos de activação ou desactivação de um equipamento, e aceita igualmente um outro par de variáveis que serve de elemento de comparação. Na prática, quando o FB recebe um evento “REQ” é comparado o par de variáveis “CurrentTime” e “CurrentWday” com os outros dois pares, testando-se deste modo a necessidade de ser executada alguma operação de controlo. Se nenhuma das comparações resultar numa correspondência é apenas emitido um evento “CNF”. Se se der uma correspondência com as variáveis de activação (“TimeOn” e “WdayOn”), juntamente com o evento “CNF”, é emitido

um evento “TRIGGER” com a variável “OnOff” a “TRUE”. Se essa correspondência se der com as variáveis de desactivação o procedimento é idêntico ao anterior, com a diferença a residir no valor de “OnOff” (desta vez “FALSE”).

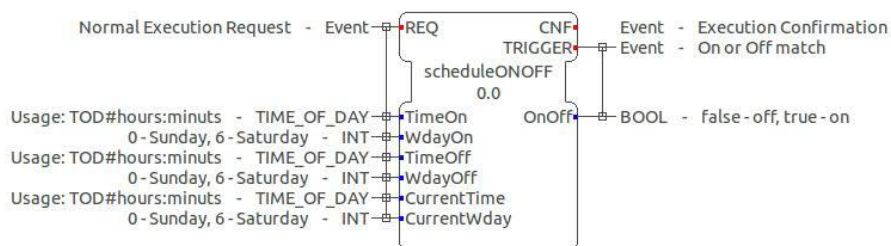


Figura 5.3 - Interface do FB "scheduleONOFF"

Os restantes dados de controlo são alojados pelo CFB desenhado a partir do recurso “mm_write_coils” documentado no capítulo anterior. Este novo CFB apenas possui um evento de entrada e um evento de saída, consequência da arquitectura para ele definida, pois como a ligação Modbus é criada antes e destruída após a execução de um pedido não existe uma separação entre os processos de gestão da ligação e execução do serviço. De resto a interface do CFB “neo_mm_write_coils” é idêntica à do “mm_write_coils” (com a excepção da ausência do qualificador de entrada “QI” no primeiro, visto ele não ser necessário), com os dados de controlo restantes (IP e porta de escuta do servidor Modbus, e endereço de memória associado à saída física do circuito eléctrico a controlar) a serem configuradas como dados de entrada, de acordo com o que é ilustrado na Figura 5.2 (os valores atribuídos aos dados de entrada na figura são para efeitos exemplificativos).

Processo de controlo

Uma vez estabelecida a relação entre o FB “scheduleONOFF” e o CFB “neo_mm_write_coils” foi desenhada a parte da aplicação responsável por fornecer a hora e data em tempo real e por gerar o fluxo de eventos necessário para que a rotina de verificação de horários seja executada periodicamente. A solução passou por criar um novo SIFB, denominado “RTC”, que utilizasse os recursos fornecidos pelo sistema operativo Linux para expor nas suas variáveis de *output* as gradezas temporais devidamente actualizadas. Optou-se por inclui as grandezas hora (“TIME”), dia da semana (“WDAY”), data (“DATE”), e data juntamente com hora (“DATETIME”), sendo que estas últimas duas variáveis apenas são usadas para que o recurso possa ser reaproveitado noutros projectos, assegurando a sua flexibilidade. O modo de funcionamento deste SIFB é bastante simples, contendo apenas um evento de entrada que, ao ser despoletado, força a que seja feita uma chamada ao sistema operativo através do *system call* “time”, com este a retornar toda a informação temporal necessária para a actualização das variáveis de saída, fornecida pelo sistema operativo Linux de forma transparente. Após ser feita a actualização é emitido um evento “CNF” a sinalizar o final da operação.

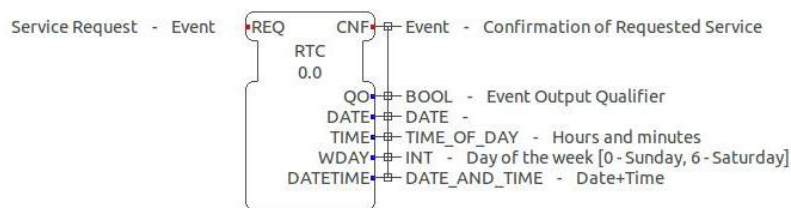


Figura 5.4 - Interface do SIFB "RTC"

O último passo no desenho da aplicação genérica foi architectar a forma como o SIFB "RTC" seria gerido. De acordo com o que fora estabelecido, o "RTC" deve ser questionado periodicamente de modo a fornecer datas e horas actualizadas que possam ser usadas pelos vários elementos "scheduleONOFF". A decisão tomada foi associar ao evento de entrada REQ do "RTC" um FB "E_CYCLE" (que já vem incluído na ferramenta 4DIAC), de modo a este último gerar eventos de saída com uma periodicidade definida pelo utilizador (nos testes que serão documentados na secção seguinte foi utilizada uma frequência de 5 segundos). A jusante de "RTC" é incluído um FB que funcione como filtro com o intuito de interromper o fluxo de eventos no caso dos novos valores das variáveis temporais serem iguais aos últimos (FB "TIME_FILTER"). A estrutura completa da aplicação genérica encontra-se ilustrada na Figura 5.2. Nota que os diferentes elementos estão parametrizados com valores de teste, pelo que todos esses valores podem ser alterados. Na realidade as únicas diferenças entre a aplicação genérica e uma aplicação final é o número de instâncias do par "scheduleONOFF"/"neo_mm_write_coils" e a citada parametrização das variáveis.

5.5.2 - Geração automática de aplicações

Um dos pontos mais sensíveis e desafiantes durante o processo de implementação foi criar um programa na linguagem C++ que utilizasse como *inputs* a informação contida na base de dados e a aplicação genérica e que gerasse como *output* os diferentes ficheiros ".fboot" que representam as aplicações finais de controlo. O diagrama UML da Figura 5.5 esquematiza o modo de funcionamento do programa, sendo de destacar a presença de uma base de dados auxiliar usada para estabelecer a correspondência entre os códigos de identificação dos equipamentos que estão indicados no ficheiro ".txt" dos horários e os códigos IP do controlador e I/O remoto e endereço de memória Modbus neste último. Esta base de dados auxiliar foi criada propositadamente para fornecer esse serviço.

A primeira parte do programa consiste em abrir o ficheiro que contém os horários e abrir a base de dados auxiliar (por abrir entenda-se configurar as variáveis do tipo apontadores necessários para interagir com esses elementos), de modo a estes recursos poderem ser utilizado. Depois desta fase inicial o programa entra num ciclo que possui tantas interações quantas o número de entradas no ficheiro dos horários. Este ciclo começa por extrair os dados da entrada que se segue nesse ficheiro e verifica se essa informação é relativa a um controlador que já possui um ficheiro aplicação criado. Se for, é aberto o ficheiro do controlador e são-lhe adicionadas as mensagens necessárias para incluir na aplicação o horário em causa. Se não, é criado um novo ficheiro de controlador, é adicionado a este um cabeçalho pré-definido, e só então são incluídas as mensagens relativas ao horário. Em ambos os casos, após o fim das respectivas operações, é verificado se o ficheiro dos horários já não contém entradas por atender (o que significa que se chegou ao final do ficheiro). Se nem

todos os horários tiverem sido atendidos é extraída a informação da próxima entrada e o ciclo volta ao início. Por outro lado, se se tiver atingido o fim do ficheiro é executada a rotina de finalização, aonde todos os ficheiros de controladores criados são reabertos de forma a serem inseridos neles os rodapés necessários. Com esta última operação é finalizado o processo de geração de aplicações para os diferentes controladores usando os dados fornecidos pela base de dados central actualmente implementada.



Figura 5.5 - Diagrama UML de actividade do programa de geração de aplicações

Alguns dos passos referidos no parágrafo anterior requerem uma explicação mais detalhada, como é o caso do processo adição do cabeçalho aos ficheiros dos controladores. Nesta etapa são inseridas nos ficheiros as mensagens associadas à criação e parametrização dos componentes das aplicações que não estão dependentes da informação fornecida pelo ficheiro que contém os horários. Esses componentes consistem no “E_CYCLE”, no “RTC” e no “TIME_FILTER”, sendo incluídas nesta fase as mensagens responsáveis por estabelecer as ligações entre eventos e variáveis respeitantes a estes elementos. É nesta fase que é igualmente inserida uma mensagem de criação de um recurso do tipo “BEM_RES” (esta mensagem trata-se da primeira a ser inserida no ficheiro, visto que este recurso criado no *runtime* é o destinatário de todas as mensagens restantes).

Uma outra secção do programa que merece esclarecimento é a fase de inserção das mensagens referentes a uma entrada do ficheiro dos horários. Por cada uma destas entradas são criados 21 pares “sheduleONOFF”/”neo_mm_write_coils”, isto porque cada entrada contém a informação semanal de controlo de um determinado circuito, e como cada dia contém 3 períodos perfaz o total de 21 (7*3) pares. Estas mensagens consistem nos pedidos de criação e parametrização destes elementos e no estabelecimento das necessárias ligações entre eventos e variáveis.

Por último é executada a rotina de inserção das mensagens de rodapé, com estas a consistirem em pedidos de criação das ligações necessárias em falta e numa última instrução que sinaliza o início da execução do recurso (no fundo sinaliza o início da execução da aplicação desenhada).

5.5.3 - Autoload

A implementação de um sistema alternativo para *deployment* de uma aplicação contida num *bootfile* local implicou o desenvolvimento de um novo SIFB que extraísse as mensagens do ficheiro e as direccionasse para o SIFB gestor. Implicou igualmente a criação de outros elementos auxiliares que permitissem seleccionar entre os modos de *deployment* local e remote (via rede). Todas estas operações ocorrem no interior do CFB “DM_KRNL”, sendo este utilizado pelo FORTE, aquando do seu início, para realizar a transferência de aplicações. As duas partes do processo de implementação são descritas de seguida, às quais se segue a exposição novo CFB “DM_KRNL”.

BFILE_MGR

A designação que dá título a esta subsecção deve-se ao novo SIFB desenhado propositadamente para lidar com o *bootfile*. Este elemento possuindo dois pares de eventos para separar as rotinas de inicialização das rotinas de serviço.

Durante a inicialização o algoritmo tenta abrir o *bootfile* em modo de leitura, de forma a verificar a presença deste na localização pré-definida. Caso a operação seja bem-sucedida é emitido o respectivo evento “INITO+”, caso o ficheiro não seja encontrado é emitido “INIT-”.

A rotina de serviço, despoletada pela recepção de um evento “IND” (é indiferente o valor do qualificador), começa por extrair a próxima linha do *bootfile*, que corresponde a um pedido. Essa linha de texto é então trabalhada de modo a serem separadas as duas partes nela contida, o destinatário e a acção. Cada uma dessas metades é atribuída à respectiva

variável de saída, sendo de seguida é emitido um evento “IND-” que sinaliza o fim da rotina de serviço.

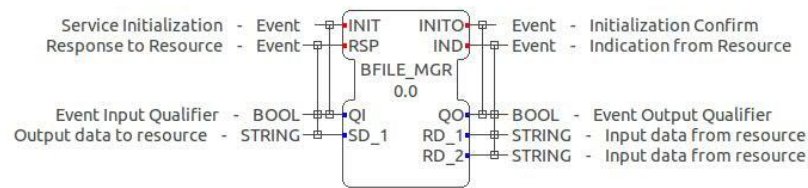


Figura 5.6 - Interface do SIFB "BFILE_MGR"

Seleccção do método de transferência

Os elementos usados para possibilitar a escolha entre os dois métodos de transferência de aplicações consistem num novo SIFB, denominado “CHECK_BFILE”, e num mistura de novos e pré-existentes BFBs de multiplexadores e comutadores programáticos (à imagem do que foi usado na implementação servidor Modbus, documentada na secção 4.5.2 - *Módulo Servidor*).

O SIFB “CHECK_BFILE” tem por função verificar a presença do *bootfile* na localização pré-definida. Para isso executa um algoritmo semelhante à rotina de inicialização do “BFILE_MGR”, actualizando o seu qualificador “QO” de saída de acordo com o resultado da verificação. Usando esta variável os elementos a jusante podem decidir se activam um ou outro modo de *deployment*.

Os multiplexadores usados foram criados de raiz, pois nenhuma das soluções oferecidas pela ferramenta se ajustava à função a desempenhar. Um deles, denominado “MUX_IN_MGR”, aceita dois pares de variáveis de texto de entrada e selecciona um deles para a saída de acordo com o qualificador de entrada “QI”. O outro, chamado “E_MUX”, tem a mesma função mas em vez de usar pares de variáveis de texto usa variáveis únicas booleanas.

Os comutadores utilizados já existiam na ferramenta e denominam-se “E_SWITCH”. Têm por função emitir um de dois eventos após a recepção de uma indicação no seu evento de entrada, sendo o qualificador de entrada “QI” responsável por escolher qual dos dois eventos deve ser accionado.

Novo DM_KRNL

O resultado da reforma de “DM_KRNL” é ilustrado na Figura 5.7. De notar que na imagem apenas estão representadas as conexões correspondentes a ligações entre eventos visto que se fossem inseridas as conexões relativas a dados a imagem se tornaria imperceptível.

O primeiro elemento da rede a ser executado é o SIFB “CHECK_BFILE”, de modo à sua variável de saída QO estar actualizada pois ela é usada quer pelos multiplexadores quer pelos comutadores nas suas execuções. Segue-se a activação do “DEV_MGR” e a execução do comutador “E_SWITCH”. Este usa a informação fornecida por “CHECK_BFILE” para decidir se deve activar “BFILE_MGR” ou “SERVER_1_2”. Para encerrar o processo de inicialização é feita a activação do elemento seleccionado.

Finalizada a fase de inicialização o CFB entra em modo de recepção de pedidos. No caso de ter sido “SERVER_1_2” a ser activado o sistema entra em pausa até ser recebida uma

mensagem via rede. Essa mensagem é reencaminhada para “DEV_MGR” e este, após a processar, envia uma confirmação para o SIFB de comunicação. Como “SERVER_1_2” nunca é finalizado o CFB está sempre preparado para iniciar um novo ciclo de recepção e execução de pedidos.

No caso de ter sido “BFILE_MGR” a ser activado a extracção e execução das mensagens contidas no *bootfile* é feita de imediato. O “BFILE_MGR” extrai uma mensagem de cada vez, envia a informação para o “DEV_MGR”, e este executa o pedido e responde com um sinal de confirmação, à imagem do que acontece com o modo de *deployment* através da rede. Uma diferença que vale a pena referir entre os dois modos de operação é que, enquanto no *deployment* via rede é sempre possível executar novos pedidos, no *deployment* local usando *bootfile* é impossível executar novos pedidos após serem cumpridas as mensagens constantes no ficheiro.

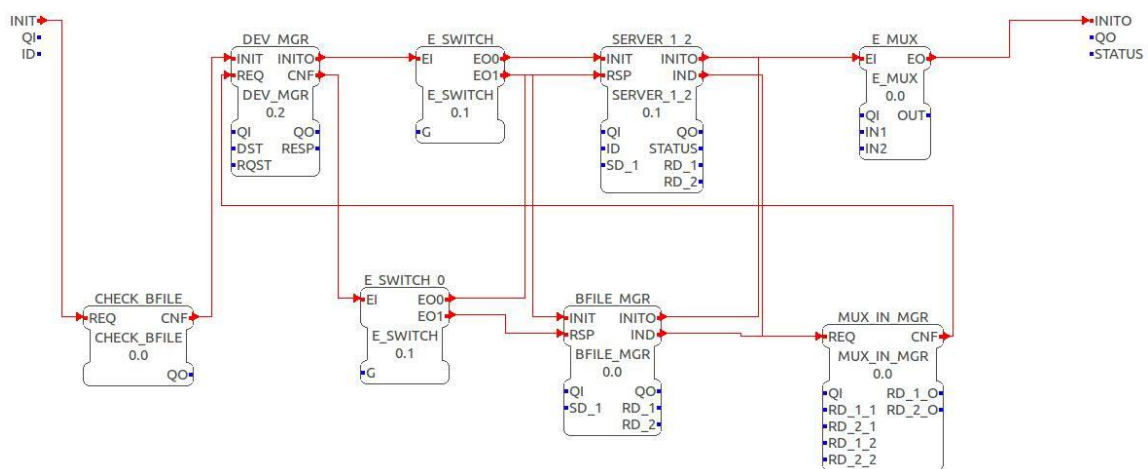


Figura 5.7 - Rede interna do novo CFB “DM_KRNL”

5.6 - Testes e validação

A validação da arquitectura e da solução desenvolvidas foi feita com a realização de um único exame abrangente. Esse exame consistiu numa primeira fase de geração de aplicações de controlo, usando para isso o programa gerador de aplicações, o modelo da aplicação genérica desenvolvida e um ficheiro contendo os horários a cumprir. Posteriormente uma dessas aplicações de controlo foi transferida para o FORTE contido num controlador ICnova de modo a se poder avaliar a eficácia do processo de controlo dos equipamentos, pois é a partir dessa avaliação que se pode concluir se a arquitectura e a solução desenvolvidas cumprem os requisitos ou não.

Os horários usados no teste tiveram como particularidade o facto de serem completamente fictícios. Com esta opção foi possível ganhar maior controlo sobre o teste sem que este fosse comprometido porque esses horários fictícios emulavam perfeitamente horários reais. Essa informação estava contida no ficheiro “.txt” de acordo com aquilo que seria obtido se os dados fossem extraídos directamente da base de dados real. Esse ficheiro continha horários relativos a equipamentos associados a 5 controladores diferentes, logo o programa gerador de aplicações criou 5 aplicações distintas.

Uma vez geradas, foi necessário provar que as aplicações funcionavam correctamente. Uma delas foi escolhida para ser executada pelo FORTE contido numa ICnova ligada à rede intranet da FEUP. O ficheiro que continha a aplicação foi colocado no *path* pré-definido na memória da ICnova de forma a que, quando fosse iniciado, o *runtime* transferisse a aplicação automaticamente. O processo de *deployment* demorou mais de 10 minutos, algo que seria problemático numa implementação real da solução. Essa demora deve-se às características limitadas da plataforma ICnova, visto que a mesma aplicação demorou pouco mais de 10 segundos a ser transferida para o FORTE sendo este executado num PC comum. Logicamente que a duração do processo de *deployment* depende do número de horários contidos na aplicação a transferir, sendo que a aplicação referente ao teste efectuado possuía 230.

De acordo com o que já foi explicado, cada horário individual está direccionado para um determinado circuito eléctrico controlado pelos I/Os remotos. Na prática, essa associação é feita através dos códigos IP do controlador e do I/O remoto e do endereço Modbus aonde esse circuito eléctrico está mapeado. No teste efectuado nenhum desses endereços de memória foi usado para mapear saídas físicas, pois essa opção levaria a um teste muito mais complexo e que necessitava de muitos mais recursos (I/Os remotos) e não acrescentava nada ao teste. Desta forma apenas as memórias Modbus internas dos servidos são alteradas, sem que essa alteração se reflecta no mundo físico, o que dá origem ao conceito de circuito virtual.

Perante este contexto, cada um dos 230 horários era referente a um circuito virtual que seja localizava num de 2 I/Os remotos utilizados no teste, de acordo com modelo ilustrado na Figura 5.8. Para além disso, as referências temporais para a activação e desactivação de circuitos foram escolhidas de forma a em cada minuto um circuito virtual necessita-se de ser activado ou desactivado, o que na prática significava que a aplicação de controlo, uma vez em execução, enviava de minuto a minuto um pedido de escrita num dos endereços do espaço de memória Modbus referente às coils que visa activar ou desactivar um circuito virtual.

Os 230 horários correspondiam a quase 5000 pares de FBs “sheduleONOFF”/“mm_write_coils”. Essas quase 5000 instâncias do FB “shedulesONOFF” eram todas analisadas periodicamente (sempre que o “TIME_FILTER” emitia um evento de saída) para se poder comparar as referências temporais que continham com as que eram fornecidas pelo “RTC”, de modo a verificar se era necessário ligar ou desligar algum equipamento. Este teste demonstrou que essa análise era feita tão rápida (muito menos que os 5 segundos utilizados para questionar o RTC) que não comprometia em nada o funcionamento da aplicação. Este facto prova que a arquitectura é escalável.

A aplicação foi executada durante um dia completo, com a sua monitorização a ser feita através da consola de execução do FORTE, aonde era mostrada a informação dos pedidos que iam sendo enviados aos I/Os remotos, incluído as respostas destes. Ao fim desse período de aproximadamente 24 horas a aplicação de controlo continuava em perfeito funcionamento, com as mensagens a serem correctamente enviadas e executadas.

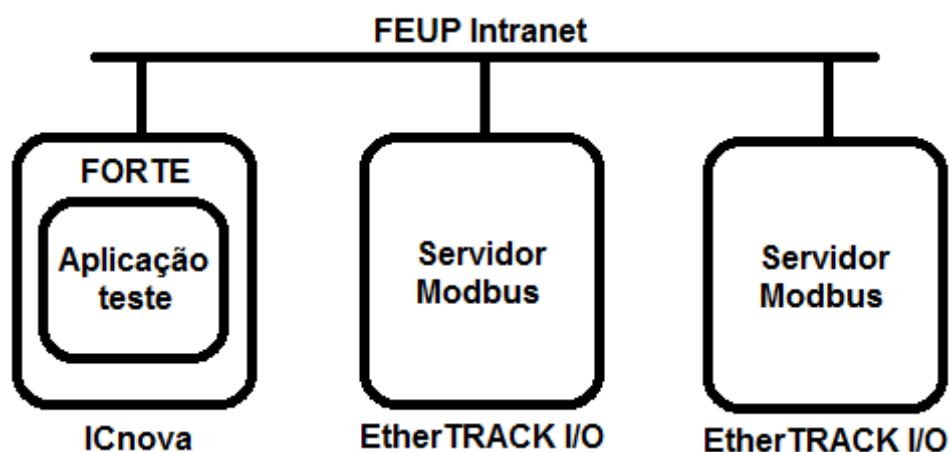


Figura 5.8 - Modelo do teste à implementação do serviço de cumprimento de horários

5.7 - Conclusão

Ao longo deste capítulo foram documentados os passos que levaram à criação e validação de uma nova arquitectura para a plataforma de gestão técnica da FEUP tendo por base a norma IEC 61499. Esses passos incluíram a confrontação de diferentes possibilidades para a estruturação dessa nova arquitectura, a sua descrição e uma implementação minimalista que servisse para a validar.

Para além da nova arquitectura e da implementação do serviço de cumprimentos de horários segundo esta nova arquitectura, também deve ser realçada a faculdade *autoload* que foi adicionada à ferramenta 4DIAC, faculdade essa que os próprios programadores da iniciativa já tinham identificado como sendo prioritária para a *framework*. Dentro a implementação do serviço de cumprimento de horários, é de realçar a criação de um programa gerador de aplicações de controlo e de uma aplicação genérica que serve de modelo para esse programa. Desta forma os restantes serviços que no futuro venham a ser implementados já podem ter por base um exemplo devidamente validado. Ainda assim deve ser dito que o modelo representado pela aplicação genérica apenas cumpre as funcionalidades básicas inerentes ao fornecimento do serviço de cumprimento de horários, visto ele não contemplar processos de controlo de erros, leitura de *inputs* dos I/Os remotos, entre outros. A implementação que foi desenhada não fornece essas funcionalidades dado que o propósito que esteve na base do seu desenvolvimento foi a validação da arquitectura sugerida, e não a criação de uma solução final para utilização no mundo real.

Um ponto referido na secção de testes foi o facto de o processo de *deployment* ter demorado muito tempo. Esta desvantagem pode ser neutralizada recorrendo a um outro controlador com características melhores que a ICnova, ou então fazendo com que o *deployment* de uma nova aplicação não implique a suspensão da execução do FORTE. A funcionalidade *autoload* desenvolvida implica que o FORTE tenha de ser reiniciado para que a nova aplicação seja transferida, algo que pode ser evitado se essa aplicação for transferida para um novo recurso dentro do mesmo dispositivo. Quando a transferência terminasse era parado o recurso antigo e executado o novo. Esta hipótese merece ser avaliada no contexto de um trabalho futuro.

Uma das grandes vantagens da arquitectura escolhida é a independência de execução que é conferida a cada controlador. Por exemplo, na eventualidade da ocorrência de um corte na

rede de comunicação os controladores continuam a sua execução normalmente (partindo do princípio que a comunicação entre controladores e I/Os remotos permanece válida), assim como no caso em que se dá um corte na energia de alimentação dos controladores. Neste caso, ao reiniciarem, os controladores podem voltar a executar as devidas aplicações visto elas estarem guardadas num ficheiro local.

Por último, deve ser mencionada a flexibilidade que a ferramenta 4DIAC e a própria norma IEC 61499 oferecem à arquitectura sugerida. A facilidade com que se cria automaticamente uma aplicação usando um programa C++ básico é de realçar, sendo esta talvez a melhor característica da norma: ao ter na sua base modelos e não implementações concretas, torna-se simples utilizar os recursos oferecidos por diferentes tecnologias.

Capítulo 6

Conclusão e Trabalho Futuro

Confrontando os objectivos propostos com os resultados produzidos pode-se afirmar que o trabalho desenvolvido no âmbito desta tese foi completamente satisfatório. Para além da estruturação e validação de uma nova arquitectura baseada na norma IEC 61499, foi igualmente desenhada uma solução que incorporasse o protocolo Modbus na ferramenta 4DIAC e analisada e descrita esta *framework* que apresenta grande potencial.

Outros objectivos menores foram atingidos, nomeadamente a compilação bem-sucedida de uma versão executável do FORTE tendo a ICnova como plataforma de destino, o desenvolvimento de uma implementação base para o serviço de cumprimento de horários executado pelos controladores e a adição da faculdade *autoload* ao *runtime*.

Tendo em conta que um dos grandes desafios desta dissertação foi utilizar uma norma que, para além de ser relativamente recente e ainda com pouca expressão no que toca a implementações fora do âmbito académico, apresenta grandes potencialidades para responder aos desafios presentes e futuros dos sistemas de controlo, torna-se oportuno confrontar a impressão geral que resultou da sua utilização com a opinião já existente sobre a aplicação da antiga norma IEC 61131-3. A nova norma apresenta mais flexibilidade em todas as suas fases de utilização (desenho de FBs, desenho de sistemas, execução de aplicações) e garante verdadeira e abrangente portabilidade (algo que, na antiga norma, apenas era atingido a um nível muito pequeno, com os seus FBs limitados). O novo *standart* (e a tecnologia 4DIAC) permite muito facilmente incorporar novos recursos nos sistemas desenvolvidos através da utilização de SIFBs (exemplo disso mesmo é a solução Modbus desenvolvida), algo que não é possível usando a antiga norma. Por outro lado, a antiga norma assegura todas as funcionalidades básicas que são fornecidas pela nova, e talvez aí esteja a explicação do porquê de a nova norma não possuir um nível de implementação maior. Como a maioria dos sistemas industriais são de complexidade reduzida, não se torna vantajoso migrar para um novo paradigma que acaba por não produzir assim tantos dividendos. No entanto, como a nova norma permite reduzir os tempos de desenvolvimento de sistemas (pelo menos a longo prazo), é natural que essa migração acabe mesmo por ser feita. De destacar igualmente uma das grandes vantagens que advém da utilização do novo *standard*, no caso a independência que os sistemas em si baseados apresentam em relação ao *hardware* aonde são

executados. Este facto permite reduzir em muitos os custos de desenvolvimento, para além de tornar a tarefa de interligação de plataformas de diferentes marcas num exercício simples. De notar também que as *frameworks* disponíveis para o desenvolvimento e execução de sistemas baseados em IEC 61499 são maioritariamente não-proprietárias, algo que também ajuda na redução de custos.

A utilização da tecnologia 4DIAC revelou-se uma escolha acertada. Para isso contribuíram as funcionalidades que ela oferece e o apoio que é prestado no fórum oficial da iniciativa pelos próprios programadores da ferramenta. Apesar de ser um projecto de origem académica apresenta já uma maturidade assinalável que a pode levar, a curto prazo, a uma presença forte no universo comercial dos sistemas de controlo distribuído.

Logicamente que a arquitectura que resultou da realização desta dissertação é apenas o ponto de partida para o desenvolvimento de uma solução abrangente e completada visando a plataforma de gestão técnica da FEUP. Neste sentido, é possível perspectivar trabalhos futuros com origem nesta dissertação, como por exemplo um projecto que vise uma implementação mais completa dos serviços de cumprimento de horários, ou então dos restantes serviços que são assegurados pela plataforma de controlo actualmente instalada na FEUP. Para além destas opções mais lógicas e primárias, é também possível prever o nascimento de projectos que procurem adicionar novas funcionalidades ao sistema, como por exemplo implementar um servidor Web em cada controlador para efeitos de consulta e comando remotos, ou então a migração, dentro dos controladores, de um sistema operativo normal para um de tempo-real. Uma outra opção que surge como resultado desta dissertação é a possibilidade de ser reformado o SCADA de acordo com a nova norma IEC 61499.

Relativamente à implementação do protocolo Modbus na ferramenta 4DIAC, a solução desenvolvida, apesar de assegurar as funcionalidades principais em envolvem o protocolo, não pode ser considerada verdadeiramente completa pois não contempla alguns dos serviços associados ao protocolo, nomeadamente a troca de pedidos referentes a operação que não a escrita e leitura de simples endereços de memória. Neste contexto, um futuro projecto pode proceder à implementação das faculdades em falta. Da mesma forma seria importante incorporar a solução na infra-estrutura de comunicação do FORTE (actualmente uma e outra estão completamente separados), de modo a aumentar a sua flexibilidade [26].

Por fim, deve ser realçada a forma como a dissertação abrangeu tanto a área de investigação e desenvolvimento como a área de projecto e implementação, algo que contribuiu crucialmente para a aquisição de novas competências por parte do autor. O contacto com novas tecnologias foi igualmente importante, sendo de destacar a utilização do conceito de *cross-compiling* e da ferramenta CMake.

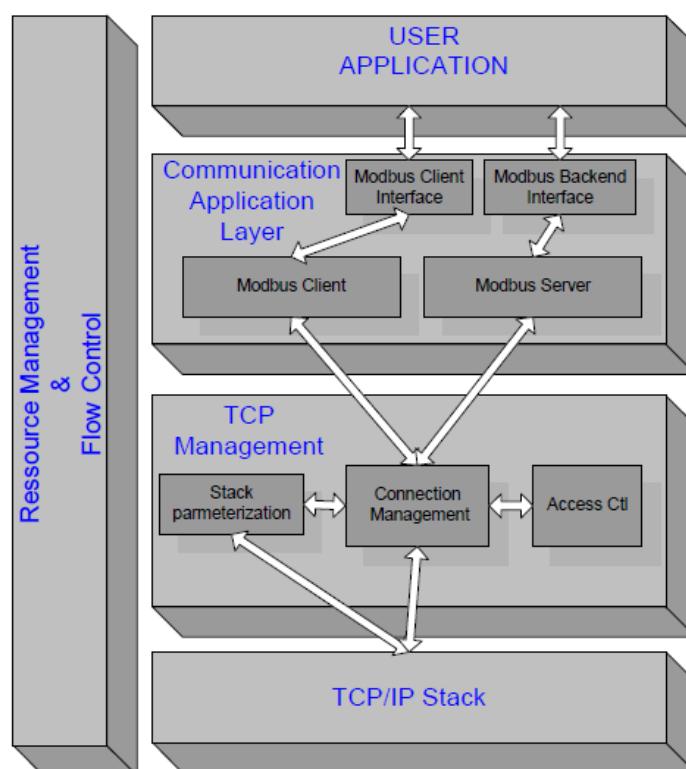
Referências

- [1] C. de Almeida, “Integração de sistemas de gestão técnica e de gestão da manutenção”, 2010.
- [2] International Electrotechnical Commission, International Standard IEC 61499-1, Function Block Architecture Part 1, Geneva, 2005.
- [3] H.-Y. Huang, J.-Y. Yen, S.-L. Chen, and F.-C. Ou, “Development of an Intelligent Energy Management Network for Building Automation,” *IEEE Transactions on Automation Science and Engineering*, vol. 1, Jul. 2004, pp. 14-25. 2004).
- [4] Gao Manru, Su Wei and Xue Lijun, “Engineering Design of Intelligent Building Management System (IBMS)”, CCTAE, 2010.
- [5] A. Zoitl, T. Strasser, and A. Valentini, “Open Source Initiatives as basis for the Establishment of new Technologies in Industrial Automation : 4DIAC a Case Study,” *Mechatronics*, 2010, pp. 3817-3819.
- [6] T. Strasser, M. Rooker, G. Ebenhofer, A. Zoitl, C. Sunder, A. Valentini, A. Martel, “Framework for Distributed Industrial Automation and Control (4DIAC)”, *6th IEEE International Conference on Industrial Informatics*, pp.283-288, 13-16 July 2008.
- [7] Controlador ICnova AP7000 Base. Disponível em http://www.ic-board.de/product_info.php?language=en&info=p75_ICnova-AP7000-Base.html.
- [8] Controladores Wago. Disponível em <http://www.wago.com/cps/rde/xchg/SID-338CDAE2-3B5A0E36/wago/style.xsl/gle-351.htm>
- [9] Controlador BeagleBoard. Disponível em <http://beagleboard.org/>.
- [10] T. Strasser, I. Müller, C. Sünder, O. Hummer, H. Uhrmann, “Modeling of Reconfiguration Control Applications based on the IEC 61499, Reference Model for Industrial Process Measurement and Control Systems”, DIS’06.
- [11] M. Hirsch, V. Vyatkin, H. -M. Hanisch, “IEC 61499 Function Blocks for Distributed Networked Embedded Applications”, 2006.
- [12] T. Hussain, G. Frey, “Migration of a PLC Controller to an IEC 61499 Compliant Distributed control System: Hands-on Experiences”, ICRA 2005.
- [13] H.-M. Hanisch, V. Vyatkin, “Automation Systems by Using the New International Standard IEC 61499: A Developer’s View”, *The Industrial Information Technology Handbook*, Cap. 66, 2005.
- [14] J. Christensen, “IEC 61499 - A Standard for Software Reuse in Embedded, Distributed Control Systems”. Disponível em <http://knol.google.com/k/iec-61499#>.

- [15] V. Vyatkin, “The IEC 61499 Standard and Its Semantics,” *Industrial Electronics*, pp. 40-48.
- [16] Repositório Mercurial da iniciativa 4DIAC. Disponível em <http://fordiac.hg.sourceforge.net/hgweb/fordiac/>.
- [17] Página Sourceforge da iniciativa 4DIAC. Disponível em http://sourceforge.net/scm/?type=hg&group_id=199151.
- [18] Ferramenta CMake. Disponível em <http://www.cmake.org/>.
- [19] Ferramenta Doxygen. Disponível em www.doxygen.org/.
- [20] Modbus-IDE, “Modbus Application Protocol V1.1b”, pág 6. Disponível em http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf
- [21] Modbus-IDA, “Modbus Messaging on TCP/IP Implementation Guide V1.0b”, pág. 4. Disponível em http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf
- [22] Ferramenta Libmodbus. Disponível em <http://libmodbus.org/>.
- [23] J. Dekker, “Algoritmo de Dekker”. Disponível em http://en.wikipedia.org/wiki/Dekker%27s_algorithm.
- [24] Ferramenta Modbus Poll. Disponível em http://www.modbustools.com/modbus_poll.asp.
- [25] 4DIAC, “FORTE Communication Infrastructure”, 2010. Disponível em http://sourceforge.net/apps/mediawiki/fordiac/index.php?title=FORTE_Communication_Infrastructure

Anexos

Anexo 1



Anexo 1 - Arquitectura conceptual para o Serviço de mensagens MODBUS [21]

Anexo 2

			Function Name	Function Code
Data Access	Bit access	Physical Discrete Inputs	Read Discrete Inputs	2
		Internal Bits or Physical Coils	Read Coils	1
			Write Single Coil	5
			Write Multiple Coils	15
	16-bit access	Physical Input Registers	Read Input Register	4
		Internal	Read Holding	3

		Registers or Physical Output Registers	Registers	
			Write Single Register	6
			Write Multiple Registers	16
			Read/Write Multiple Registers	23
			Mask Write Register	22
			Read FIFO Queue	24
	File Record Access		Read File Record	20
			Write File Record	21
Diagnostics			Read Exception Status	7
			Diagnostic	8
			Get Com Event Counter	11
			Get Com Event Log	12
			Report Slave ID	17
			Read Device Identification	43
Other			Encapsulated Interface Transport	43

Anexo 2 - Funções permitidas no protocolo Modbus [20] (Nota: A negrito encontram-se as 6 funções básicas em que assenta o protocolo)

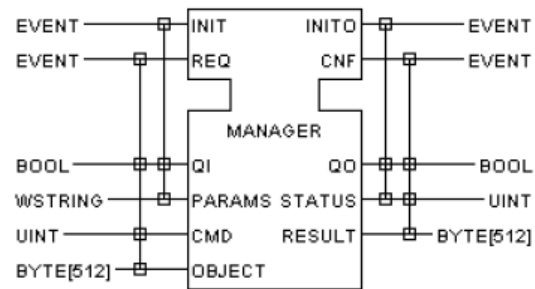
Anexo 3

Tipo	Nome	Descrição
Estrutura	modbus_param_t	Este elemento contém toda a informação necessária para o estabelecer de uma comunicação cliente-servidor, quer para o caso em que um cliente contacta um servidor para realizar um pedido, quer para quando um servidor inicia a escuta de pedidos. Dessa informação faz parte, para o caso de uma ligação TCP, a identificação do <i>socket</i> , do IP e da porta, entre outros dados (por exemplo <i>flags</i> e apontadores).
	modbus_mapping_t	Esta estrutura, associada ao servidor Modbus, guarda os valores dos diferentes espaços de memória acessíveis aos clientes.
API	modbus_init_tcp	Método utilizado para configurar uma estrutura do tipo modbus_param_t para um tipo de ligação TCP. Utiliza como factores de configuração dois códigos, um relativo ao IP e outro relativo à porta. No caso do servidor, o IP usado deve ser o do próprio dispositivo, no caso do cliente deverá ser o IP do servidor a que deseja efectuar pedidos.

	<code>modbus_connect</code>	Método cuja função é estabelecer uma ligação, segundo as informações contidas na estrutura <code>modbus_param_t</code> que recebe como argumento. É utilizada pelas entidades cliente que desejam fazer pedidos a servidores.
	<code>modbus_init_listen_tcp</code>	Método utilizado pelas entidades servidor para escutar pedidos de mestres via TCP. As informações da ligação são novamente fornecidas pela estrutura <code>modbus_param_t</code> que serve de argumento à função.
	<code>modbus_mapping_new</code>	Função que inicializa uma estrutura do tipo <code>modbus_mapping_t</code> de acordo com as configurações recebidas nos argumentos, configurações essas que incidem sobre o tamanho de cada um dos diferentes tipos de memória.
	<code>modbus_manage_query</code>	Função que trata um pedido recebido por um servidor, construindo e enviando de seguida uma resposta concordante. Para isso consulta (e altera, caso seja necessário) a estrutura <code>modbus_mapping_t</code> que lhe é passada em argumento.
	<code>force_single_coil</code>	Método utilizado pelos clientes para enviar um pedido de escrita num endereço de memória do tipo <i>coil</i> a um servidor.
	<code>preset_single_register</code>	Método utilizado pelos clientes para enviar um pedido de escrita num endereço de memória do tipo <i>holding register</i> a um servidor.
	<code>read_coil_status</code>	Método utilizado pelos clientes para enviar um pedido de leitura de um endereço de memória do tipo <i>coil</i> a um servidor.
	<code>read_input_status</code>	Método utilizado pelos clientes para enviar um pedido de leitura de um endereço de memória do tipo <i>discrete input</i> a um servidor.
	<code>read_holding_registers</code>	Método utilizado pelos clientes para enviar um pedido de leitura de um endereço de memória do tipo <i>holding registers</i> a um servidor.
	<code>read_input_registers</code>	Método utilizado pelos clientes para enviar um pedido de leitura de um endereço de memória do tipo <i>input register</i> a um servidor.

Anexo 3 - APIs e estruturas da biblioteca Libmodbus

Anexo 4



Anexo 4 - Managment Function Block genérico [2]

Anexo 5

Value	Command	Semantics
0	CREATE	Create specified object
1	DELETE	Delete specified object
2	START	Start specified object
3	STOP	Stop specified object
4	READ	Read data from access path
5	WRITE	Write data to access path
6	KILL	Make specified object unrunnable
7	QUERY	Request information on specified object

Anexo 5 - Valores e semântica da Input "CMD" do FB gestor [2]

Anexo 6

Value	Status	Semantics
0	RDY	No errors
1	BAD_PARAMS	Invalid PARAMS input value
2	LOCAL_TERMINATION	Application-initiated termination
3	SYSTEM_TERMINATION	System-initiated termination
4	NOT_READY	Manager is not able to process the command
5	UNSUPPORTED_CMD	Requested command is not supported
6	UNSUPPORTED_TYPE	Requested object type is not supported
7	NO_SUCH_OBJECT	Referenced object does not exist
8	INVALID_OBJECT	Invalid object specification syntax
9	INVALID_OPERATION	Commanded operation is invalid for specified object
10	INVALID_STATE	Commanded operation is invalid for current object state
11	OVERFLOW	Previous transaction still pending

Anexo 6 - Valores e semântica da Input "STATUS" do FB gestor [2]

Anexo 7

CMD	OBJECT	RESULT
CREATE	type_declaration	data_type_name
	fb_type_declaration	fb_type_name
	fb_instance_definition	fb_instance_reference
	connection_definition	connection_start_point
	access_path_declaration	access_path_name
DELETE	data_type_name	data_type_name
	fb_type_name	fb_type_name
	fb_instance_reference	fb_instance_reference
	connection_definition	connection_definition
	access_path_name	access_path_name
START	fb_instance_reference	fb_instance_reference
	application_name	application_name
STOP	fb_instance_reference	fb_instance_reference
	application_name	application_name
KILL	fb_instance_reference	fb_instance_reference
QUERY	all_data_types	data_type_list
	all_fb_types	fb_type_list
	data_type_name	type_declaration
	fb_type_name	fb_type_declaration
	fb_instance_reference	fb_status
	connection_start_point	connection_end_points
	application_name	fb_instance_list
	access_path_name	access_path_declaration
READ	access_path_name	accessed_data
WRITE	access_path_data	access_path_name

Anexo 7 - Sintaxe dos comandos aceites pelo FB gestor [2]

Anexo 8

- Instalar Mercurial (<http://mercurial.selenic.com/>);
- Copiar link do forte no repositório do mercurial (<http://fordiac.hg.sourceforge.net/hgweb/fordiac/forte/>);
- Criar uma nova directoria (p.e. “fordiac”) e fazer cd para ela;
- Executar o comando \$hg clone <http://fordiac.hg.sourceforge.net/hgweb/fordiac/forte/>;
- cd para a pasta “fordiac/forte/src/arch/posix”;
- Abrir ficheiro “CMakeLists.txt”;
- Alterar a opção da linha “forte add definition” de “-DFORTE LITTLE ENDIAN” para “-DFORTE BIG ENDIAN”
- Instalar “CMake” (<http://www.cmake.org/>)
- Instalar “CMake-GUI”
- Executar “CMake-GUI”
- Utilizar directorias: source code - “fordiac/forte/” e built directory - “fordiac/icnova”. Clicar em “Configure” (para o caso do cmake-gui pedir confirmação para a pasta de destino, escolher a opção “Unix Makefiles” e “Use default native compilers”)

- Escolher a opção FORTE_ARCHITECTURE_POSIX
- Clicar em “Configure” novamente
- Clicar em “Generate”
- cd para a pasta “fordiac/icnova”
- Abrir o ficheiro “CMakeCache.txt”

Este próximo passo visa configurar os processos de compilação e linkagem para a plataforma icnova. Será necessário aceder à toolchain da icnova, pelo que a indicação <Path to AVR32 toolchain> deve ser substituída pela localização apropriada.

- Substituir as linhas:

```
CMAKE_AR:FILEPATH=/usr/bin/ar -> CMAKE_AR:FILEPATH=<Path to AVR32 toolchain>
/avr32/usr/bin/avr32-linux-ar
CMAKE_CXX_COMPILER:FILEPATH=/usr/bin/c++ -> CMAKE_CXX_COMPILER:FILEPATH=
<Path to AVR32 toolchain>/avr32/usr/bin/avr32-linux-g++
CMAKE_C_COMPILER:FILEPATH=/usr/bin/gcc -> CMAKE_C_COMPILER:FILEPATH= <Path
to AVR32 toolchain>/avr32/usr/bin/avr32-linux-gcc
CMAKE_LINKER:FILEPATH=/usr/bin/ld -> CMAKE_LINKER:FILEPATH=<Path to AVR32
toolchain>/avr32/usr/bin/avr32-linux-ld
CMAKE_NM:FILEPATH=/usr/bin/nm -> CMAKE_NM:FILEPATH=<Path to AVR32
toolchain>/avr32/usr/bin/avr32-linux-nm
CMAKE_OBJCOPY:FILEPATH=/usr/bin/objcopy -> CMAKE_OBJCOPY:FILEPATH=<Path to
AVR32 toolchain>/avr32/usr/bin/avr32-linux-objcopy
CMAKE_OBJDUMP:FILEPATH=/usr/bin/objdump -> CMAKE_OBJDUMP:FILEPATH=<Path
to AVR32 toolchain>/avr32/usr/bin/avr32-linux-objdump
CMAKE_RANLIB:FILEPATH=/usr/bin/ranlib -> CMAKE_RANLIB:FILEPATH=<Path to
AVR32 toolchain>/avr32/usr/bin/avr32-linux-ranlib
CMAKE_STRIP:FILEPATH=/usr/bin/strip -> CMAKE_STRIP:FILEPATH=<Path to AVR32
toolchain>/avr32/usr/bin/avr32-linux-strip
```

Nota: Esta configuração sugerida é válida na altura da escrita deste documento. Pelo facto de o source code do forte ser actualizador quase diariamente são aconselhadas precauções na execução destas alterações.

- Ainda na pasta “fordiac/icnova” executar o comando “make”;
- No caso de a compilação ser bem sucedida, na pasta “/fordiac/icnova/src” deverá ser criado o desejado ficheiro executável chamado “forte”.