SISTEMAS DE SUPERVISÃO E CONTROLO DE AUTÓMATOS:

SOLUÇÕES BASEADAS EM OPC E IEC 60870-5-104

Daniel Jasson Ferreira Rocha

Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

2013

Esta dissertação deve ser mantida confidencial por um período de 3 anos.

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Disciplina de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores

Candidato: Daniel Jasson Ferreira Rocha, Nº. 1080445, 1080445@isep.ipp.pt

Orientação científica: Paula Maria Marques Moura Gomes Viana, pmv@isep.ipp.pt

Empresa: EFACEC

Supervisão: Luís Miguel Ferreira Morgado, luis.morgado@efacec.com

Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

2 de Agosto de 2013

Agradecimentos

À Engenheira Paula Maria Marques Moura Gomes Viana pelo apoio dado ao nível de conhecimentos da área de redes de comunicação, pela sua orientação e conselhos.

Agradeço ainda aos Engenheiros Paulo Paixão e Luís Morgado da empresa EFACEC pela orientação e conselhos dados, bem como a transmissão de conhecimentos sobre a área e ajuda prestada.

Também tenho a agradecer à empresa Softing pela ajuda, conselhos e informação transmitida para a realização dos clientes OPC.

Resumo

Atualmente, no segmento metro-ferroviário, há uma tendência para que todos os equipamentos que constituem os sistemas auxiliares de uma estação (escadas mecânicas, elevadores, bloqueadores, validadores de bilhética, ventiladores, bombas, entre outros) sejam dotados de inteligência. Tipicamente, um conjunto de equipamentos são ligados a um autómato que permite o controlo local e remoto e é vulgar que, sendo de fabricantes diferentes, suportem tecnologias distintas.

Um sistema de supervisão que permita o acesso à informação disponibilizada por cada um dos autómatos, ou à atuação sobre um deles, terá por isso que implementar e suportar diversos protocolos de comunicação de forma a não ficar limitado a um tipo de tecnologia.

De forma a diminuir os custos de desenvolvimento e operação de um sistema de supervisão e controlo e facilitar a integração de novos equipamentos, com diferentes características, têm sido procuradas soluções que garantam uma mais fácil comunicação entre os diversos módulos intervenientes. Nesta dissertação são implementadas soluções baseadas em clientes OPC-DA e OPC-AE e no protocolo IEC 60870-5-104, permitindo que os sistemas de supervisão e de controlo comuniquem com os equipamentos através destes três módulos.

Os principais aspectos inovadores estão associados à implementação de uma arquitetura multiprotocolo usando as novas tendências de supervisão e controlo baseadas em soluções distribuídas.

Palavras-Chave

Clientes OPC DA e AE, Mestre IEC 60870-5-104, Sistema de supervisão e de controlo.

Abstract

Currently, in the metro-rail segment, there is a tendency for all equipment that constitute the auxiliary systems of a station (mechanical stairs, elevators, tickets validators, fans, pumps) are endowed with intelligence. Typically, a set of equipment connected to a PLC (Programmable Logic Controller) which allows for local and remote control and it is normal, being from different manufacturers support different technologies.

A monitoring system that allows access to the information provided by each of the PLC or the actuation of one of them, will have to deploy and support various communication protocols in order not to be limited to one type of technology.

To reduce the costs of developing and operating a supervision and control system and facilitate the integration of new equipment with different characteristics have been sought solutions that ensure easier communication between the different modules involved. This dissertation will be implemented based solutions OPC-DA and OPC AE clients and IEC 60870-5-104 protocol, enabling the supervision and control systems to communicate with the devices through these three modules.

The main innovative aspects are associated to implement a multiprotocol architecture using new trends monitoring and control based on distributed solutions.

Keywords

IEC 60870-5-104 master, OPC DA and AE clients, Supervision and control system.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

AE	–	Alarms & Events
AGV	–	Automated Guided Vehicle
APCI	–	Application Protocol Control Information
APDU	–	Application Protocol Data Unit
ASCII	–	American Standard Code for Information Interchange
ASDU	–	Application Service Data Unit
AS-I	–	Actuator Sensor Interface
CAN	–	Controller Area Network
CF	–	Control Field
CiA	–	CAN in Automation
CIM	–	Computer Integrated Manufacturing
CNC	–	Computer Numerical Control
COM	–	Component Objetc Model
CS	–	CheckSum
CT	–	Command Type
DA	–	Data Access
DCE	–	Distributed Computing Environment
DCOM	–	Distributed Component Object Model
DLL	–	Dynamic-Link Library
DP	–	Decentralized Peripherals
DX	–	Data eXchange
E/S	–	Entradas/Saídas
ERP	–	Enterprise Resource Planning

FMS	–	Field Message Specification
GDS	–	Generic Data Slave
HDA	–	Historical Data Access
HDLC	–	High Level Data Link Control
HSE	–	High Speed Ethernet
HTTP	–	Hypertext Transfer Protocol
IDE	–	Integrated Development Environment
IEC	–	International Electrotechnical Commission
IHM	–	Interface Human Machine
IP	–	Internet Protocol
JNI	–	Java Native Interface
LPCI	–	Link Protocol Control Information
LPDU	–	Link Protocol Data Unit
LSB	–	Least Significant Byte
MES	–	Manufacturing Execution Systems
MSB	–	Most Significant Byte
ODVA	–	Open DeviceNet Vendor Association
OLE	–	Object Linking and Embedding
OPC	–	OLE for Process Control
OSI	–	Open Systems Interconnection
P/N	–	Positive/Negative
PA	–	Process Automation
PLC	–	Programmable Logic Controller
PROFIBUS	–	PROcess Field BUS
PROFINet	–	Profibus for Ethernet
QDS	–	Quality Descriptor
RTU	–	Remote Terminal Unit

S/E	–	Select / Execute
SBO	–	Select Before Operate
SCADA	–	Supervisory Control and Data Acquisition
SDK	–	Software Development Kit
SOAP	–	Simple Object Access Protocol
SQ	–	Structure Qualifier
TCP	–	Transmission Control Protocol
UA	–	Unified Architecture
UDP	–	User Datagram Protocol
URL	–	Uniform Resource Locator
WSDL	–	Web Service Description Language
XML	–	eXtensible Markup Language

1. INTRODUÇÃO

A utilização de PLC´s (*Programmable Logic Controller*) em ambientes fabris permitirá otimizar um conjunto de operações com vista à melhoria do processo de fabrico, segurança de operação, entre outros. Com o aumento do número de dispositivos instalados numa indústria, torna-se essencial a interligação entre os diversos sistemas, de forma a permitir a troca de informação e a coordenação de operações. Assim, surgiram as redes de comunicação industriais para possibilitar a partilha e a troca de informação entre os diferentes níveis hierárquicos, bem como o controlo e monitorização dessa informação.

Nesta dissertação, serão abordados alguns dos protocolos de comunicação vocacionados para redes industriais, sendo analisadas as suas características principais, limitações, requisitos e tipo de aplicações/ambientes para que estão vocacionados, tendo como objetivo final a implementação de alguns módulos direcionados para os sistemas de supervisão e de controlo que a EFACEC utiliza.

1.1. CONTEXTUALIZAÇÃO

Este projeto surgiu da necessidade da empresa EFACEC dotar os seus sistemas de supervisão e de controlo com vários módulos protocolares de forma a permitir a comunicação com os diversos dispositivos do segmento metro-ferroviário (escadas mecânicas, elevadores, bloqueadores, validadores de bilhética, ventiladores, bombas, entre outros), não ficando restrito aos protocolos implementados por cada um dos equipamentos.

Desta forma, pretendeu-se analisar alguns protocolos de comunicação para autómatos e selecionar os que melhor se aplicam aos sistemas de supervisão e de controlo usados pela EFACEC.

1.2. OBJECTIVOS

O objectivo principal desta dissertação consiste no desenvolvimento e implementação de módulos de *software* que permitam a comunicação dos sistemas de supervisão e de controlo com os diversos equipamentos do segmento metro-ferroviário. Desta forma, torna-se possível a implementação de funções de controlo e a possibilidade de monitorização dos diversos equipamentos, não ficando restrito ao tipo de protocolos implementados por cada um dos equipamentos. Dada a complexidade inerente a este objectivo, sentiu-se a necessidade de o subdividir em múltiplas tarefas de realização mais simples:

- Análise do estado de arte:
  - Protocolos de comunicação industrial existentes;
  - Possíveis arquiteturas para a implementação dos módulos protocolares nos sistemas de supervisão e de controlo.
- Especificação da arquitetura e requisitos do sistema:
  - Análise da arquitetura atual dos sistemas de supervisão e de controlo;
  - Seleção dos protocolos necessários para o desenvolvimento dos módulos;
  - Seleção das bibliotecas/*software* necessários para o desenvolvimento dos módulos.

- Desenvolvimento e implementação dos módulos protocolares no sistema de supervisão;
- Testes e resultados:
  - Testes e resultados dos módulos desenvolvidos;
  - Correção de possíveis falhas existentes.

1.3. ORGANIZAÇÃO DO RELATÓRIO

No capítulo 1 é apresentada uma introdução a esta dissertação, sendo descritos os objectivos principais. No capítulo seguinte, 2, são apresentados alguns conceitos fundamentais relacionados com protocolos de comunicação para redes industriais, sendo apresentadas algumas soluções a áreas de aplicação. No 3º capítulo são discutidas possíveis arquiteturas para a implementação dos clientes OPC e do mestre IEC 60870-5-104 nos sistemas de supervisão. No capítulo 4 faz-se uma apresentação do trabalho implementado, descrevendo as opções tomadas, arquiteturas e funcionalidades de cada um dos módulos. No último capítulo, o 5º, são reunidas as principais conclusões e perspectivados futuros desenvolvimentos.

2. REDES DE COMUNICAÇÃO INDUSTRIAIS

A evolução tecnológica permitiu automatizar um conjunto de funções realizadas em ambientes industriais através da instalação de sensores e actuadores e de sistemas de controlo. Apesar das vantagens introduzidas nos sistemas produtivos, o aumento do número de dispositivos começa a colocar problemas às típicas soluções centralizadas devido à quantidade de fios necessários, aos erros de ligações dos fios e dimensões físicas. A Figura 1 representa um sistema centralizado, onde se consegue verificar que todos os dispositivos (estações de controlo e monitorização, computadores e dispositivos de campo) se encontram ligados diretamente ao PLC (*Programmable Logic Controller*).

As arquiteturas distribuídas ou descentralizadas (Figura 2) apresentam-se como solução alternativa em que os dispositivos se encontram interligados entre si através de redes, permitindo a redução do número de cabos necessários, tornando o sistema mais flexível e de mais fácil manutenção, com as consequentes vantagens económicas.

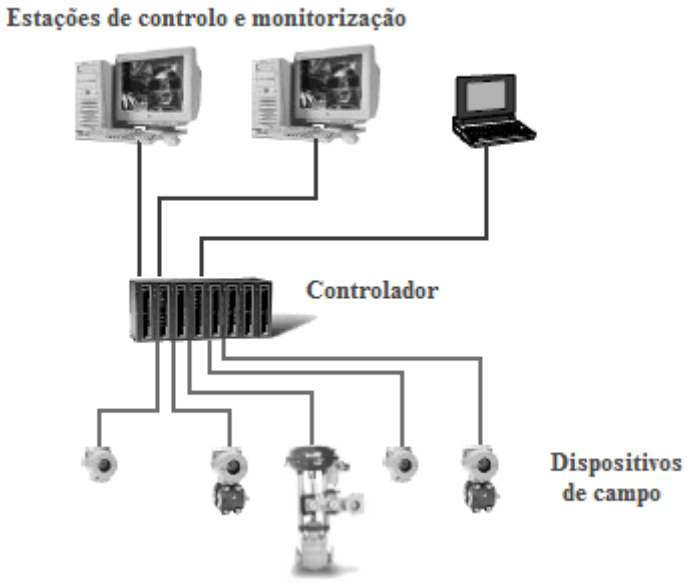


Figura 1 Sistema Centralizado

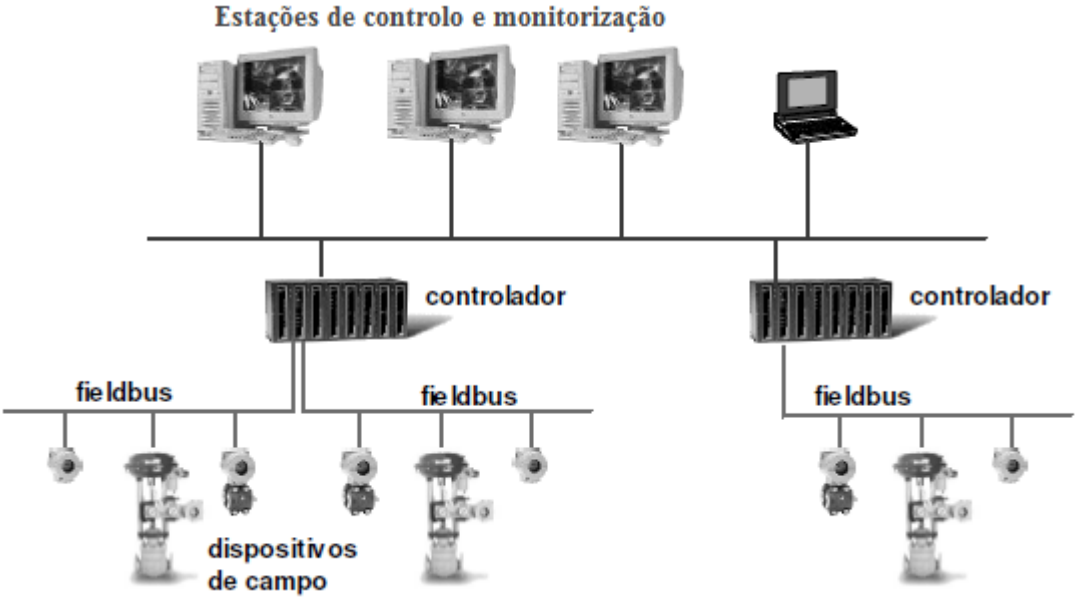


Figura 2 Sistema Descentralizado [1]

Atualmente, a tecnologia dominante são as redes *fieldbus*. Estas redes interligam todos os dispositivos através de um barramento com comunicação digital, bilateral e em tempo real e apresentam uma arquitetura descentralizada.

Estes sistemas são baseados em protocolos abertos permitindo a transmissão de dados e a ligação de equipamentos de diversos fabricantes. Assim, o utilizador fica livre para escolher o fabricante, e com a flexibilidade de expandir ou modificar facilmente o sistema [2].

Este capítulo irá demonstrar a importância de um sistema descentralizado numa indústria, bem como as tecnologias associadas a esse sistema. Serão abordados diversos protocolos existentes para a implementação de um sistema deste género.

2.1. ORGANIZAÇÃO FUNCIONAL EM CAMADAS

A quantidade de dispositivos (*field level*) em aplicações industriais aumentou de forma significativa com o decorrer dos anos, criando a necessidade de definir hierarquias e estruturas nos mais diversos ambientes industriais.

O suporte de comunicação de um ambiente industrial típico é formado por diversos níveis hierárquicos, constituindo uma estrutura que envolve, desde as tarefas administrativas, até ao controlo da operação das máquinas e equipamentos de produção.

Numa unidade de produção podem existir diferentes redes de comunicação, com a finalidade de optimizar a integração e informatização de todos os seus constituintes. A pirâmide CIM (*Computer Integrated Manufacturing*) é uma tentativa para diferenciar as diversas funções, requisitos e equipamentos existentes, definindo para tal um conjunto de níveis hierárquicos, como demonstra a Figura 3 [3].

Neste modelo, as redes de comunicação de acordo com a exigência requerida ao sistema de comunicação poderão ser divididas nos seguintes níveis hierárquicos:

- Nível de Gestão
- Nível de Controlo
- Nível de Campo/Processo
- Nível de Entradas/Saídas

Figura 3 Pirâmide CIM [3]

No nível de Gestão, nível mais elevado, é centralizada toda a informação dos níveis mais baixos. Este nível normalmente é dedicado à partilha de recursos (por grupos de utilizadores). Os controladores normalmente usados são servidores ou estações de trabalho. O volume de tráfego na rede normalmente é elevado, sendo os tempos de resposta curtos, não tendo no entanto características de tempo real. Nestas áreas são normalmente utilizadas redes TCP/IP (*Transmission Control Protocol / Internet Protocol*) com suporte físico Ethernet.

O nível de Controlo, também conhecido como nível de célula, tem como objetivo interligar dispositivos fabris de controlo e supervisão, tais como, controladores de célula, sistemas de transporte, armazéns automáticos, sistemas de inspeção e teste, PLC´s, entre outros. Importante salientar que, este nível deve garantir bons mecanismos de controlo de erros, por forma a que não seja necessário parar a produção por dificuldades de comunicação ou erros de transmissão. O protocolo de rede mais usado neste nível é também o TCP/IP com suporte físico Ethernet.

O nível de Campo/Processo, também conhecido como nível das máquinas, tem como objetivo interligar dispositivos como CNC´s (*Computer Numerical Control*), robôs, AGV´s (*Automated Guided Vehicle*), sistemas de visão, PLC´s, entre outros. Neste nível são

usadas as redes *fieldbus* que permitem respostas em tempo real, isto é, tempos de resposta de alguns milissegundos e com grande tolerância a falhas. Este tipo de redes têm como objetivo interligar diferentes unidades inteligentes que necessitam de cooperar entre si para realizar uma determinada operação, sendo por isso os tempos de resposta um aspeto crítico.

Nas redes *fieldbus* é normalmente utilizada uma hierarquia do tipo *master*/*slave*; em que o *master* controla todas as comunicações fazendo ciclicamente *poolling* aos *slaves*. Desta forma consegue-se eliminar as colisões na rede obtendo-se assim tempos de resposta mais rápidos.

O nível de Entradas/Saídas é o nível mais baixo da hierarquia e tem como função interligar os dispositivos pouco inteligentes ou sem inteligência, tais como os sensores e actuadores. Neste nível, são também utilizadas redes *fieldbus*, sendo aqui normalmente designadas por redes *fieldbus* de baixo nível.

O volume de dados, associado a cada um dos níveis, bem como os tempos e a frequência de transmissão, apresentam diferenças significativas, tal como ilustrado na Tabela 1. Como se pode verificar, à medida que se desce no nível hierárquico, o volume de dados a circular pela rede será menor. O mesmo se verifica quanto aos tempos de transmissão. A consideração destes fatores é importante para a posterior escolha de tipo de rede *fieldbus* a implementar.

Tabela 1 Características fundamentais de cada um dos níveis hierárquicos

	Volume de Dados	Tempo de Transmissão	Frequência de Transmissão
Nível de Gestão	MBytes	Hora / Minuto	Dia / Turno
Nível de Controlo	kBytes	Segundos	Horas / Minutos
Nível de Campo/Processo	Bytes	100 us … 100 ms	10 ms … 100 ms
Nível de Entradas/Saídas	Bits	1 ms … 10 ms	Milissegundos

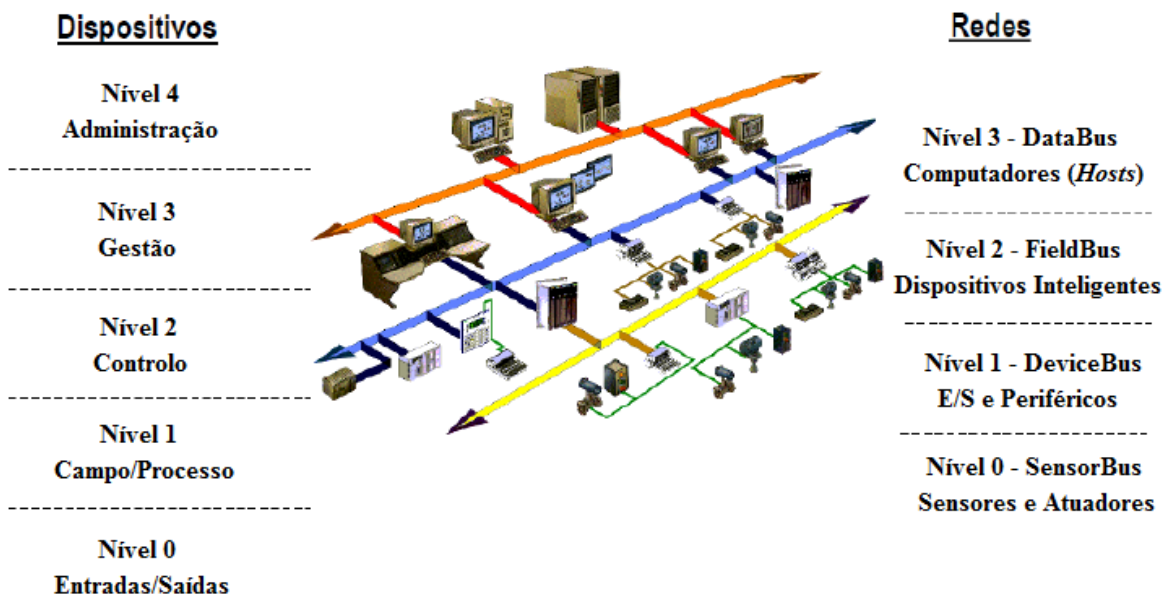


Figura 4 Classificação das redes industriais e estrutura funcional [4]

Existem no mercado diversas soluções para implementação de redes de comunicação na indústria. Estas encontram-se agrupadas conforme o nível em que são aplicadas. Normalmente o nível de gestão é subdividido em dois níveis, o nível de administração e o nível de gestão. Na Figura 4 apresentam-se os níveis hierárquicos e, do lado direito, o tipo de rede utilizada para interligar os níveis hierárquicos. A Figura 5 apresenta algumas das tecnologias disponíveis para a implementação de cada um dos tipos de rede.

A rede *Sensorbus*, rede de nível mais baixo, é geralmente usada para interligar sensores e actuadores com transmissões/recepções na ordem de bits de dados, tais como interruptores. Os tempos de reação são na ordem de milissegundos e as distâncias máximas suportadas são de 200 metros.

A rede *Devicebus*, rede que se encontra compreendida entre a rede *Sensorbus* e *Fieldbus*, é geralmente usada para interligar dispositivos com pontos discretos, dados analógicos ou uma mistura dos dois. Isto quer dizer, que as transmissões/recepções de dados são na ordem de bytes. Os tempos de reação são na ordem de milissegundos e as distâncias máximas suportadas são de 500 metros.

As redes *Fieldbus* permitem interligar os dispositivos de Entradas/Saídas mais inteligentes, com transmissões/recepções com grandes blocos de dados (kBytes). Os

equipamentos ligados nessa rede possuem inteligência para desempenhar funções específicas de controlo, como o controlo de fluxo de informações e processos. Os tempos de transmissão de dados são longos, na ordem de centena de segundos, e as distâncias máximas suportadas são de cerca de 10 Km.

A rede *Databus* possibilita a comunicação entre os sistemas informáticos de gestão e administração. Os tempos de reação são da ordem dos minutos, as distâncias máximas de cerca de 100 km e a natureza das informações trocadas são dados com grande volume de informação. O tipo de rede encontrada neste nível é a rede Ethernet.

A escolha de um tipo de rede tem que ser baseada nos tipos de dispositivos a serem controlados, sendo o fator distância também um aspeto importante a considerar. Outros fatores que influenciam a decisão são a velocidade de transmissão, a arquitetura da rede e o número de dispositivos. Na Tabela 2 e na Tabela 3 encontram-se resumidas as características principais de algumas das redes de comunicação industrial mais conhecidas.

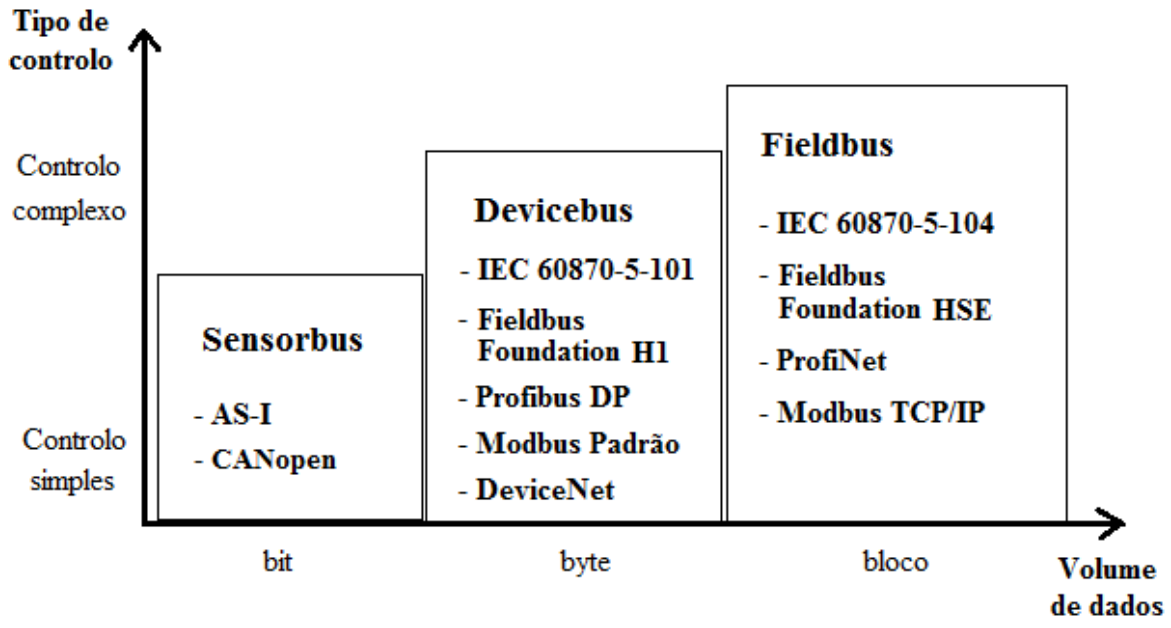


Figura 5 Organização das redes nos diferentes níveis

Tabela 2 Características de algumas redes *fieldbus* – Parte 1

Características das redes					
Rede	**Fabricante**	**Norma**	**Topologia**	**Meio Físico**	**Propriedade**
Profibus DP/PA	Siemens	EN 50170 / DIN 19245 part 3(DP) / 4(PA), IEC 1158-2(PA)	Barramento, Estrela, Anel	Par trançado, fibra	Livre
Modbus RTU/ASCII	Modicon	EN 1434-3 (camada 7), IEC 870-5 (camada 2)	Barramento, Estrela, Árvore	Par trançado	Livre
Modbus Plus	Modicon		Barramento	Par trançado	Proprietário
Fieldbus Foundation H1	Fieldbus Foundation	ISA SP50/IEC 61158	Estrela, Barramento	Par trançado, fibra	Livre
Fieldbus Foundation HSE	Fieldbus Foundation	IEC 61158, IEEE 802.3u, RFC para IP, TCP e UDP	Estrela	Par trançado, fibra	Livre
DeviceNet	Allen-Bradley	ISO 11898 & 11519	Tronco/Derivações	Par trançado	Livre
LonWorks	Echelon Corp.		Estrela, Barramento, Anel	Par trançado, fibra	Livre
Interbus-S	Phoenix Contact, Interbus Club	DIN 19258, EN 50.254	Derivações em “T”	Par trançado, fibra	Livre
AS-I	AS-I Consortium		Barramento, Anel, Estrela, Árvore	Cabo de 2 fios	Livre
CAN Open	CAN In Automation	CiA	Barramento, Tronco/Derivações	Par trançado	Livre

Tabela 3 Características de algumas redes *fieldbus* – Parte 2

Características das redes					
Rede	**Número máx. de dispositivos**	**Método Comunicação**	**Distância Máxima**	**Velocidade transmissão**	**Diagnósticos**
Profibus DP/PA	127	Master/Slave	24 Km (fibra)	9600 bits/s a 12 Mbps	Estação, Módulo, Canal
Modbus	250	Master/Slave	350 m	300 bps a 38.4 kbps	
Modbus Plus	64	Ponto-ponto	450 m / segmento	1 Mbps	
Fieldbus Foundation H1	240/segmento, 65000 segmentos	Master/Slave, Notificação de Eventos	1900 m	31.25 kbps	Monitorização da rede, Status
Fieldbus Foundation HSE	-	Master/Slave, Notificação de Eventos	100 m; 2000 m (fibra)	100 Mbps	
DeviceNet	64	Master/Slave, Multi-master, Ponto-ponto	500 m	500 kbps, 250 kbps, 125 kbps	Monitorização do barramento
LonWorks	32000/domínio	Ponto-ponto	2000 m	1.25 Mbps full duplex	
Interbus-S	256	Master/Slave	400 m / segmento, 12.8 Km	500 kbps full duplex	Erros e Quebra de cabo
AS-I	31 slaves	Master/Slave com polling cíclico	100 m	167 kbps	Falha no dispositivo
CAN Open	126	Master/Slave, Multi-master, Ponto-ponto, Multi-cast	10 m a 1000 m	10 kbps a 1 Mbps	Controlo de erros, mensagens de emergência

2.2. PROTOCOLOS DE COMUNICAÇÃO INDUSTRIAL

2.2.1. IEC 60870-5

A norma IEC (*International Electrotechnical Commission*) 60870-5 é uma norma internacional usada para a comunicação entre sistemas de controlo. Por outras palavras, esta solução é principalmente utilizada para a interligação entre sistemas de gestão e supervisão, como o SCADA (*Supervisory Control and Data Acquisition*), com dispositivos inteligentes, como o PLC. Esta norma encontra-se subdividida em diversas especificações:

- IEC 60870-5-1 – Transmissão de tramas;
- IEC 60870-5-2 – Serviços de transmissão de dados;
- IEC 60870-5-3 – Estrutura genérica dos dados;
- IEC 60870-5-4 – Definição e codificação da informação;
- IEC 60870-5-5 – Aplicação de funções básicas;
- IEC 60870-5-6 − Orientações para testes de conformidade para o protocolo IEC 60870-5;
- IEC 60870-5-101 – Protocolo de transmissão, norma para controlo de sistemas de comunicação básico;
- IEC 60870-5-102 – Protocolo de transmissão orientado para sistemas elétricos de potência;
- IEC 60870-5-103 − Protocolo de transmissão, usado para a interface informativa da proteção dos equipamentos;
- IEC 60870-5-104 − Protocolo de transmissão para acesso à rede

A rede IEC 60870-5 apresenta algumas diferenças em relação a outras soluções *fieldbus*, devido ao facto de suportar eventos produzidos com data e hora, sincronização de data e hora, e reporte de eventos por ordem de prioridade.

2.2.1.1. IEC 60870-5-101

O IEC 60870-5-101 [5] é um protocolo de comunicação série aberto, que utiliza as normas RS232 e RS485. Foi normalizado pela IEC e é predominante no mercado europeu. É direcionado para aplicações SCADA, mas também pode ser utilizado noutros sistemas. Este protocolo implementa três das camadas do modelo OSI (*Open Systems Interconnection*):

- Física (camada 1) − especifica que o protocolo pode ser implementado com configurações, ponto a ponto, multiponto, linear, estrela e anel;
- Ligação lógica (camada 2) – especifica o tipo de serviços de transmissão de dados existentes. O protocolo IEC 60870-5-101 proporciona três tipos de serviços básicos: transmissão sem confirmação, transmissão com confirmação e pergunta/resposta. Os serviços de transmissão de dados podem ser balanceados ou não-balanceados;
- Aplicação (camada 7) − define a estrutura dos dados, denominada por ASDU (*Application Service Data Unit*).

Esta solução permite velocidades de transmissão até 64 kbps e métodos de comunicação em *master/slave* e *multi-master*. Admite um ou dois bytes para endereçamento, sendo 255 e 65535, respetivamente, os endereços de *broadcast* em cada um dos casos.

A camada de ligação lógica especifica dois tipos de transmissão de dados [5]:

- **Transmissão não-equilibrada** – a estação que controla (*master*), ou seja a estação de monitorização e supervisão, como o sistema SCADA, gera o fluxo de dados fazendo *polling* às estações controladas (*slaves*). Neste caso, a estação de controlo inicia todas as transferências de mensagens, enquanto que as estações controladas podem transmitir apenas em resposta a uma mensagem da estação de controlo. Suporta mensagens sem confirmação, com confirmação e pergunta/resposta;

- **Transmissão equilibrada** − Quando se utiliza a transmissão balanceada, cada estação pode iniciar a transferência de mensagens. As estações podem atuar simultaneamente como estações de controlo e estações controladas, sendo chamadas de estações combinadas. Só podem ser usadas topologias ponto-ponto. Não suporta a comunicação pergunta/resposta.

A informação propriamente dita, medições analógicas e digitais, é formatada na camada de aplicação e está contida nas estruturas conhecidas como ASDU. Uma ASDU contém um campo para as causas de transmissão, que pode ser usado para indicar possíveis erros, tais como, ASDU inexistente, objeto desconhecido, etc.

A camada de ligação lógica adiciona informação extra, designada por LPCI (*Link Protocol Control Information*) aos dados da ASDU. Na norma IEC 60870-5-101, uma APDU (*Application Protocol Data Unit*) é igual à ASDU. O conjunto formado pela APDU e LPCI é denominado por LPDU (*Link Protocol Data Unit*), tal como se ilustra na Figura 6. Este tipo de trama é de tamanho variável, podendo contudo existir tramas com tamanho fixo caso não seja transmitida a ASDU ou então constituídas por um único caracter, como acontece no caso da confirmação [6].

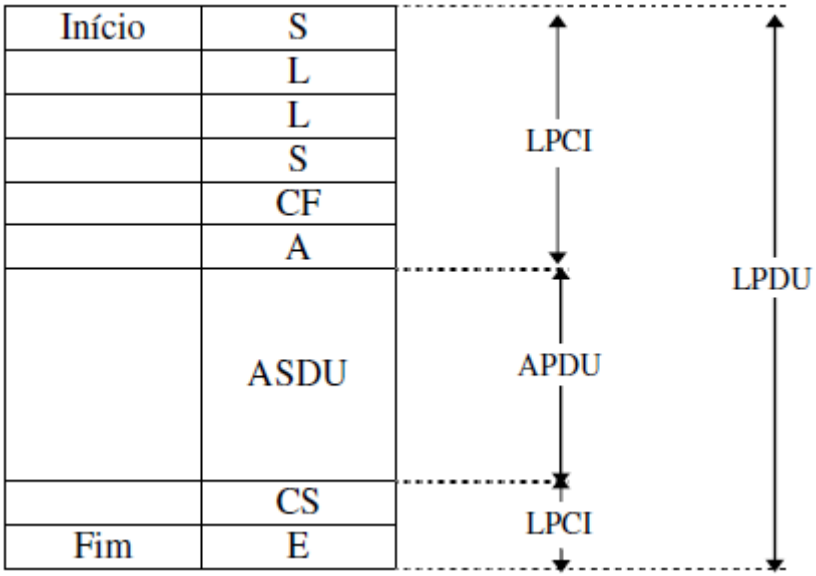


Figura 6 Formatação da camada de ligação lógica [6]

Os campos que constituem a LPCI são:

- S − Caracter de início da trama;
- L − Comprimento da trama (0 a 255). Aparece duplicado;
- CF (*Control Field*) − Este campo contém informações que caracterizam a direção da mensagem, o tipo de serviço proporcionado e suporta funções que suprimem perda ou duplicação de mensagens;
- A − Endereço;
- CS (*Checksum*) – Verificação da integridade da mensagem;
- E − Caracter de fim da trama.

O protocolo suporta a funcionalidade de sincronismo de data e hora. Para isso, a estação de controlo e a estação controlada necessitam de estar sincronizadas. A sincronização fornece uma sequência correta cronológica dos eventos.

A execução de um comando no protocolo IEC 60870-5-101 obriga à execução de uma sequência de passos que incluem a solicitação de seleção do ponto a ser comandado, a resposta de confirmação da seleção do ponto, o envio da solicitação do comando a ser executado e a resposta de confirmação da execução do mesmo (SBO − *Select Before Operate*).

2.2.1.2. IEC 60870-5-104

O protocolo IEC 60870-5-104 é uma extensão do protocolo IEC 60870-5-101, implementando as funcionalidades da camada de aplicação e introduzindo novas tecnologias ao nível das camadas física, ligação lógica, rede e transporte (Tabela 4). Utiliza a interface TCP/IP para redes Ethernet [7].

Tabela 4 Estrutura do protocolo IEC 60870-5-104

Selecção dos ASDUs do IEC 60870-5-101 e IEC 60870-5-104	Aplicação (camada 7)
RFC 793 (Protocolo de controlo de transmissão − TCP)	Transporte (camada 4)
RFC 791 (Protolo de Internet − IP)	Rede (camada 3)
RFC 894 (Transmissão de datagramas IP sobre redes Ethernet)	Ligação Lógica (camada 2)
IEEE 802.3	Física (camada 1)

A camada de transporte é orientada às ligações permitindo a implementação de mecanismos de controlo de erros e de duplicados, bem como a sua correção. É utilizada a solução TCP/IP tendo sido definido o porto 2404 para o servidor.

A ASDU não necessita de caracteres de início ou de fim, devido ao cabeçalho APCI (*Application Protocol Control Information*) englobar um caracter de início, tamanho da ASDU e campo de controlo. Ao conjunto do ASDU e APCI dá-se a designação de APDU [7].

O funcionamento é de certa forma semelhante ao protocolo IEC 60870-5-101, nomeadamente na implementação de mecanismos de sincronização de data e hora para o histórico dos eventos. A diferença mais relevante relaciona-se com o suporte de comunicação de cada uma das variantes: comunicação série no IEC 60870-5-101 e TCP/IP no IEC 60870-5-104.

Devido ao facto de se implementar um módulo mestre IEC 60870-5-104 na presente dissertação, serão descritos pormenores mais detalhados nesta subsecção sobre o protocolo, nomeadamente a constituição da sua trama.

Uma trama IEC 60870-5-104 (APDU) pode ser constituída apenas pelo APCI, ou então pelo conjunto APCI e ASDU. O comprimento máximo que uma APDU pode ter é de 253 octetos. Isto significa que a ASDU apenas poderá conter no máximo 247 octetos. O APCI é sempre iniciado pelo octeto START, seguido do comprimento da APDU (Figura 7).

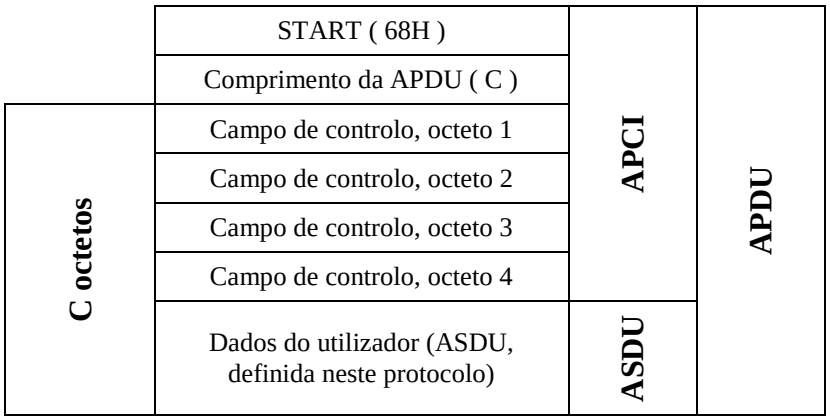


Figura 7 Constituição de uma trama IEC 60870-5-104

O protocolo implementa 4 bytes para controlo. Os campos de controlo são usados para implementar uma transferência de informação numerada (*I-format*), funções de supervisão numeradas (*S-format*) e funções de controlo não numeradas (*U-format*). A seleção do tipo de controlo a usar é realizada através dos dois bits menos significativos do primeiro octeto. Caso, o bit menos significativo seja "0", significa que se trata de um *I-format*. Se os valores dos dois bits for "01" trata-se de uma *S-format*, caso seja "11" trata-se de uma *U-format*, como demonstra a Figura 8. O *U-format* contém um campo STARTDT para indicar a inicialização da troca de dados, um campo STOPDT para parar a troca de dados e TESTFR para testar a conexão.

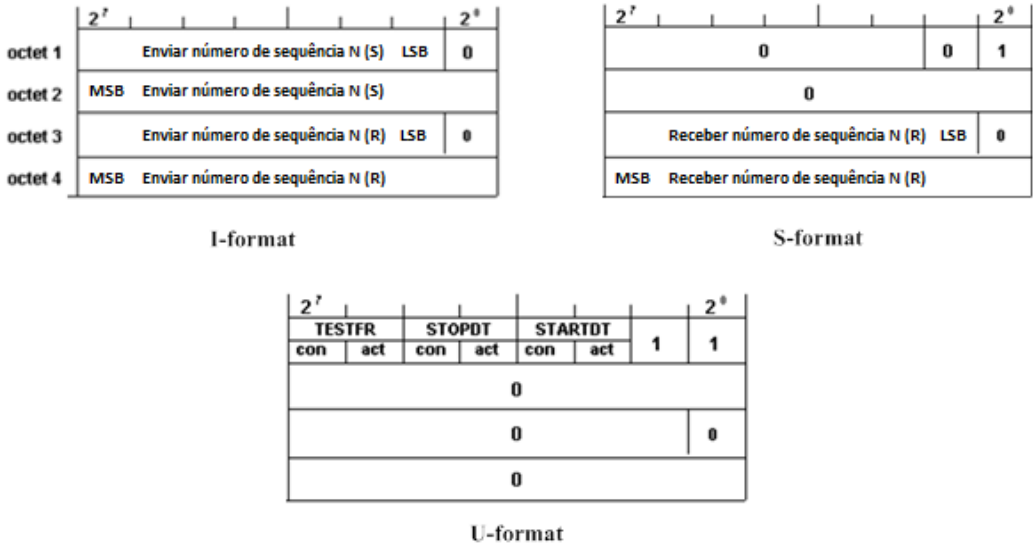


Figura 8 Tipos de controlo

A Figura 9 demonstra a estrutura da ASDU [8]. Uma ASDU é constituída por:

- **Identificador de tipo** (8 *bits*) – usado para identificar qual o tipo de pedido/resposta da mensagem. A norma define uma gama de valores entre 1 a 127. Os valores entre 128 a 135 são reservados e 136 a 255 são para uso especial.
- ***SQ*** (***Structure Qualifier*** − 1 *bit*) – define como se encontra estruturado o campo do objeto de informação.

  Caso o bit se encontre a "0", significa que a trama será constituída por um ou mais campos de "objetos de informação" ("endereço do objeto de informação" + "elementos da informação").

  Caso o bit se encontre a "1", significa que a trama será constituída por apenas um campo de "objeto de informação", em que este é constituído por um "endereço do objeto de informação" e por uma sequência/conjunto de "elementos de informação".
- **Número de objetos** (7 *bits*) – define o número de objetos de informação existentes.
- ***T*** (***Test*** − 1 *bit*) – indica se é uma ASDU gerada durante uma condição de teste.
- ***P/N*** (***Positive/Negative*** − 1 *bit*) – define se a mensagem é uma confirmação positiva ou negativa.
- **Causa de transmissão** (5 *bits*) – define a causa da transmissão.
- **Endereço do originador** (8 *bits*) – campo opcional (0 a 255), usado apenas para redireccionamento de ASDU´s.
- **Endereço comum da ASDU** (16 *bits*) – define o endereço de um dispositivo. Este campo pode usar 1 ou 2 bytes, sendo os valores 255 ou 65535 usados para *broadcast*. O valor "0" não é usado. O "octeto 1" é o byte menos significativo e o "octeto 2" é o byte mais significativo.
- **Endereço do objeto de informação** (24 *bits*) – define o endereço de destino no qual se pretende realizar o pedido/resposta. Este campo pode usar 1, 2 ou 3 octetos. O "octeto 1" é o byte menos significativo e o "octeto 3" é o byte mais significativo.
- **Elementos da informação** (variável) – são os dados propriamente ditos, que podem ser um ou mais valores e que podem estar agrupados ou não.

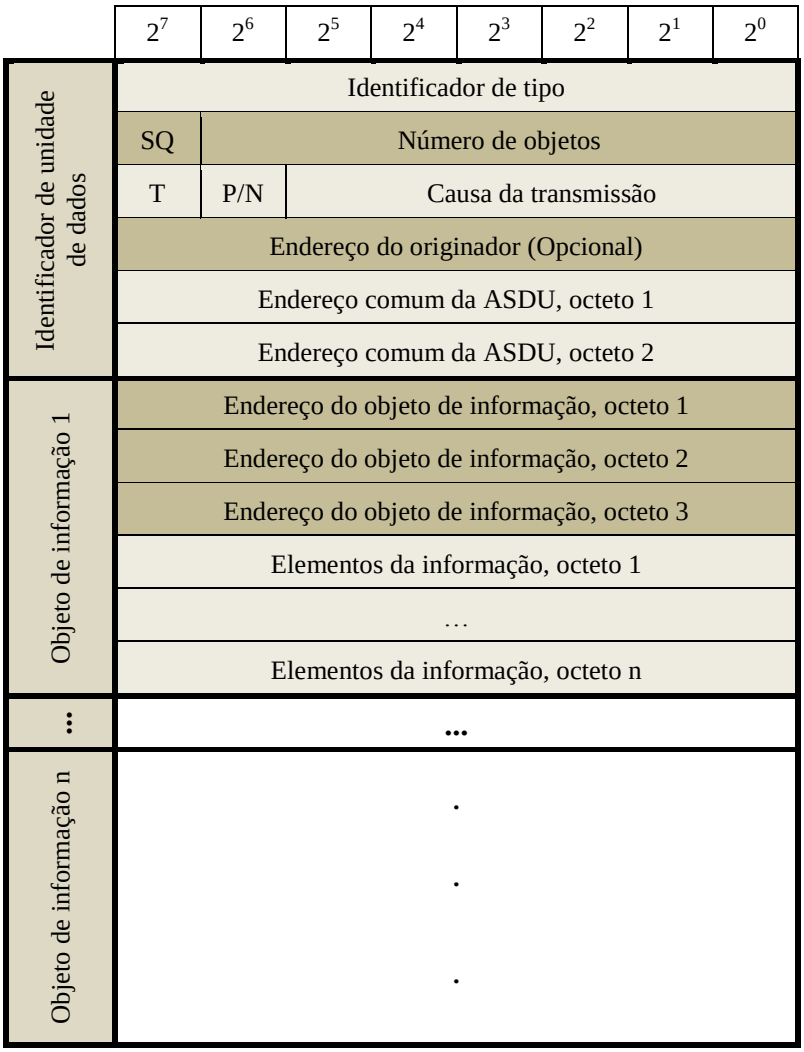


Figura 9 Estrutura de uma ASDU

2.2.2. FIELDBUS FOUNDATION

O Fieldbus Foundation [9] é uma arquitetura aberta com o objetivo de interligar dispositivos de controlo e automação industrial. Esta rede engloba diversas funcionalidades que incluem processamento distribuído, diagnóstico avançado e redundância. O Fieldbus Foundation permite o acesso a muitas variáveis, não só relativas ao processo, mas também do diagnóstico dos sensores e actuadores, dos componentes electrónicos, do desempenho, entre outras [10].

Este protocolo apresenta dois tipos de aplicação (Figura 10):

- **Fieldbus Foundation H1** − é uma rede de transmissão de dados em tempo real, que segue a norma IEC 61158 e tem uma velocidade de comunicação de 31,25 kbps para a comunicação com dispositivos de campo e controlo, tais como, sensores e actuadores. Este protocolo permite funções de diagnóstico e pode usar o cabo de par trançado, em que os sinais e a alimentação vão no mesmo fio, ou fibra ótica. Admite topologias em barramento ou em estrela, com um alcance máximo até 1900 metros [11].

- **Fieldbus Foundation HSE** (*High Speed Ethernet*) – é um protocolo de transmissão de dados, para redes Ethernet, que segue a norma IEC 61158 e tem uma velocidade de comunicação de 100Mbps, que é idealmente usada como *backbone*. O objetivo deste protocolo é interligar os dispositivos que são usados principalmente para aplicações de monitorização e controlo, tais como, PLC, servidores, estações de trabalho, entre outros [12].

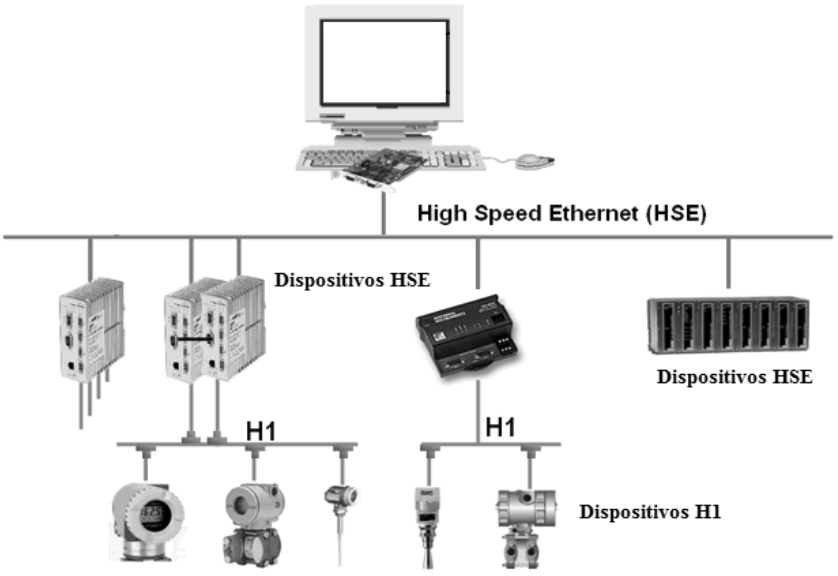


Figura 10 Exemplo de uma rede Fieldbus Foundation

2.2.3. PROFIBUS (PROCESS FIELD BUS)

A rede PROFIBUS (*PROcess Field BUS*) [13] encontra-se descrita pelas normas DIN 19245 incorporada na norma EN 50170 e também IEC 61158 e IEC 61784. O PROFIBUS é uma norma aberta e destaca-se por abranger os diversos níveis hierárquicos numa indústria. Para isso, oferece características diversas de protocolos de comunicações (Figura 11), tais como [4]:

- PROFIBUS DP (*Descentralized Peripherical*): é o mais usado dentre os protocolos; é caracterizado pela velocidade, eficiência e baixo custo de ligação. Foi projetado especialmente para a comunicação entre sistemas de automação e periféricos distribuídos;

- PROFIBUS FMS (*Field Message Specification*): é um protocolo de comunicação geral para as tarefas de comunicações solicitadas. FMS oferece muitas funções sofisticadas de aplicações para comunicação entre dispositivos inteligentes;

- PROFIBUS PA (*Process Automation*): este protocolo define os parâmetros e blocos de funções dos dispositivos de automação de processo, tais como transdutores de medidas, válvulas e IHM (*Interface Human Machine*);

- PROFINet (*Profibus for Ethernet*): comunicação entre PLC e PC usando Ethernet/TCP-IP;

- PROFISafe: para funções relacionados com segurança;

- PROFIDrive: para funções relacionados com o controlo de movimento.

- Os meios de transmissão mais usuais são o RS485 e a fibra ótica, sendo o RS485 mais comum. O protocolo PROFIBUS utiliza o método de comunicação do tipo *master/slave*, podendo ainda ser usado o método *multi-master*. As topologias apresentadas por este sistema podem ser do tipo linear, estrela ou anel. As velocidades de transmissão podem ir dos 9600 bps aos 12 Mbps, com distâncias de 24 Km por fibra ótica.

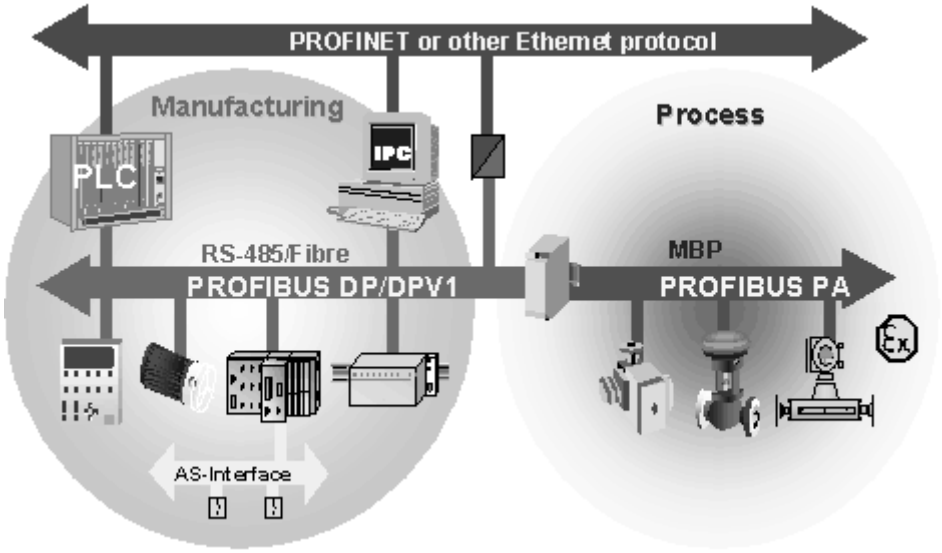


Figura 11 Exemplo de uma rede Profibus [14]

Para controlar a rede, o mestre deve conhecer toda a informação sobre os equipamentos, nomeadamente os seus endereços, configuração de E/S (Entradas/Saídas) e parâmetros de operação. Cada equipamento deve ter um ficheiro GDS (*Generic Data Slave*) com informação para descrever as funcionalidades, características e parâmetros de configuração desse equipamento. Uma ferramenta de configuração é usada para combinar o ficheiro GDS e a informação do utilizador numa base de dados mestre que é utilizada para estabelecer a comunicação e iniciar a troca de dados com o *slave* Profibus [2].

2.2.4. MODBUS

O Modbus [15] foi desenvolvido e publicado pela Modicon em 1979 para o uso nos seus PLC, sendo um dos protocolos normalizados mais antigos. Atualmente, a Modicon, parte do grupo da Schneider Electric, disponibilizou as especificações e normas que definem o Modbus em domínio público. Por esta razão é utilizado em milhares de equipamentos existentes (variadores de velocidade, robôs, máquinas especiais, autómatos programáveis industriais, entre outros) e é uma das soluções de rede mais baratas a serem utilizadas em automação industrial [4] [2].

Para a implementação do Modbus são usualmente utilizadas as normas de comunicação RS232 ou RS485, com topologias do tipo linear, estrela ou árvore. O método de comunicação é do tipo *master/slave*, permitindo velocidades de transmissão de 300 bps a 38,4 kbps no Modbus padrão e 1 Mbps no Modbus Plus.

Existem dois modos de transmissão série no protocolo Modbus [16]:

- RTU (*Remote Terminal Unit*) − cada byte da mensagem é enviado como 2 caracteres hexadecimais de 4 bit. Este é o modo mais usado para a transmissão de dados e com melhor desempenho;

- ASCII (*American Standard Code for Information Interchange*) − cada byte da mensagem é enviado como 2 caracteres ASCII de 4 bit. Este modo é usado quando a ligação de comunicação física ou as capacidades do dispositivo não permitem a conformidade com as exigências do RTU.

A aproximação Modbus ao modelo OSI aparece ao nível das camadas 1, 2 e 7, respectivamente camada física, ligação lógica e aplicação [2]:

- **Nível físico (nível 1):** par trançado, máximo de 19200 bps, RS232/RS485/Anel de corrente;

- **Ligação Lógica (nível 2):** acesso à rede por mecanismo tipo *master/slave*. Controlo de erros por CRC16 (Modo RTU). Num método de acesso tipo *master/slave*, a iniciativa do envio de mensagens está restringida ao *master*. Se uma resposta for requerida, os *slaves* respondem à solicitação do *master*, ou então limitam-se a executar as ações pedidas pelo *master*. O *master* pode dirigir-se individualmente aos *slaves*, ou difundir mensagens dirigidas a todos os *slaves* (*broadcast*), inserindo nas mensagens o endereço 00;

- **Aplicação (nível 7):** definiram-se neste nível as funções de leitura e escrita de variáveis (*bits*, *words*, E/S), diagnóstico e estatísticas de ocorrência da rede.

A solução Modbus suporta diversas alternativas (Figura 12) [4]. O Modbus TCP/IP é usado para comunicação entre sistemas de supervisão e controladores lógicos programáveis. O protocolo Modbus é encapsulado no protocolo TCP/IP e transmitido

através de redes Ethernet. O Modbus Plus é usado para a comunicação entre controladores lógicos programáveis, módulos de E/S, chaves de partida eletrônica de motores, interfaces homem máquina, entre outros. O meio físico é o RS485 com taxas de transmissão de 1 Mbps e controlo de acesso ao meio por HDLC (*High Level Data Link Control*). O Modbus Padrão é usado para a comunicação dos PLC com os dispositivos de entrada e saída de dados, dispositivos electrónicos inteligentes como relés de proteção, controladores de processo, actuadores de válvulas, transdutores de energia, entre outros. O meio físico é o RS232 ou RS485 em conjunto com o protocolo *master/slave*.

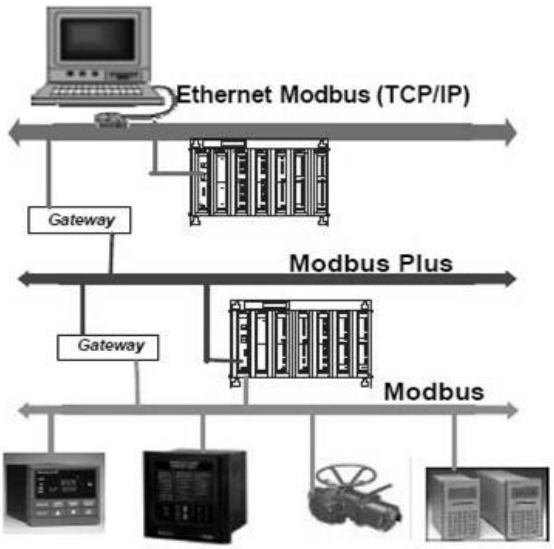


Figura 12 Exemplo de uma rede Modbus [4]

2.2.5. DEVICENET

O DeviceNet [17] surgiu em 1994 baseado na tecnologia CAN (*Controller Area Network*), tendo as especificações sido desenvolvidas pela ODVA (*Open DeviceNet Vendor Association*). A norma de comunicação DeviceNet baseia-se na camada física 2 do modelo OSI e na técnica de transporte CAN. As soluções apresentam como vantagens a possibilidade de remover e substituir equipamentos em redes sob tensão e sem um equipamento de programação, ou ainda a possibilidade de fornecer a alimentação aos equipamentos através do próprio cabo de rede [2].

Esta solução proporciona comunicações fiáveis e possibilita a troca de informações entre sistemas de fabricantes diferentes. O DeviceNet permite a integração em redes de dispositivos, tais como, disjuntores, relés de proteção de motores, arrancadores suaves, módulos de iluminação, fins-de-curso, sensores de proximidade e fotoelétricos. Como é baseado no protocolo CAN, para além de permitir a comunicação entre dispositivos, ainda permite realizar diagnósticos de falhas.

O protocolo DeviceNet encontra-se especificado nas normas ISO 11898 e 11519 e permite métodos de comunicação do tipo *master/slave*, *multi-master* e ponto-ponto, com um alcance máximo de 500 metros e velocidades de comunicação até 500 kbps.

2.2.6. AS-I (ACTUATOR SENSOR INTERFACE)

A *AS-International*, organização que apoia o AS-I (*Actuator Sensor Interface*) [18], formou-se em 1991 através de um consórcio de 11 empresas europeias que desenvolveram a norma. Hoje em dia, esta associação encontra-se aberta a qualquer fornecedor ou utilizador desta tecnologia [2].

O objetivo é ligar entre si sensores e actuadores de diversos fabricantes, utilizando um cabo com dois fios, capaz de transmitir dados e alimentação simultaneamente (tipicamente 24VDC nominal e 2A), alcançando distâncias até 100 metros (300 metros com repetidores). Como os dados digitais são codificados num sinal sinusoidal, com uma pequena largura de banda, o AS-I realiza uma filtragem a esse sinal, eliminando as frequências fora dessa largura de banda. Assim, o AS-I pode operar em ambientes com ruído eléctrico, sem haver erros de transmissão.

A tecnologia AS-Interface é compatível com qualquer outro *bus* de campo ou rede. Existem *gateways* para ligação a CANopen, Profibus, RS485 e RS232. A topologia implementada pode ser do tipo barramento, anel, estrela ou árvore, com um método de comunicação *master/slave* (um *master* por rede) com *pooling* cíclico e com uma velocidade de transmissão de dados de 167 kbps.

O mestre possibilita as funções de diagnóstico, monitorização contínua da rede, reconhecimento de falhas e atribuição de endereço correto quando um nó é removido para manutenção [4].

Existem duas versões da AS-I: a versão 2.0 e a 2.1. A versão 2.0 suporta até 31 *slaves* num barramento, enquanto que a versão 2.1 suporta até 62 *slaves*. Como cada *slave* pode ter 4 entradas ou saídas, o número máximo de elementos binários que podem ser ligados aos *slaves* na versão 2.0 é de 124 (Figura 13) e na versão 2.1 é de 248 [4].

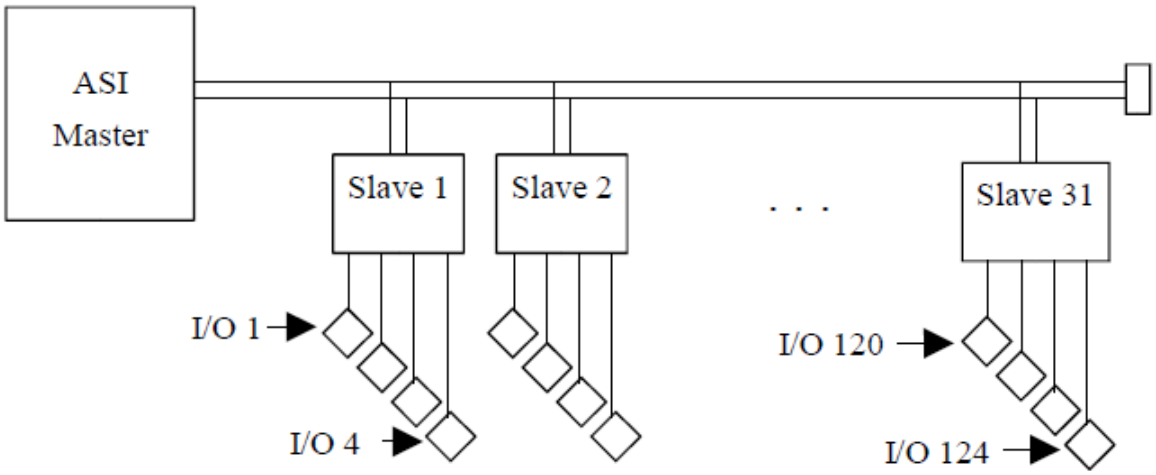


Figura 13 Configuração típica do AS-I [4]

2.2.7. CANOPEN (CONTROLLER AREA NETWORK)

Desenvolvido originalmente pela BOSCH em 1983 para a indústria automóvel. A comunicação na rede CANopen é realizada por eventos, o que reduz o tráfego na rede. Possíveis conflitos que podem ocorrer são evitados através de uma definição de níveis de prioridade. Uma rede CANopen [19] segue a norma CiA (*CAN in Automation*) e possibilita a sua configuração sem a necessidade de um dispositivo *master*. Este protocolo apenas define os níveis 1 e 2 do modelo OSI e permite que cada dispositivo possa enviar mensagens sempre que o bus estiver livre. O uso deste protocolo permite a ligação máxima de 126 dispositivos numa rede [2].

O CANopen permite a interligação de sensores, sensores inteligentes, válvulas pneumáticas, leitores de códigos de barras, variadores de velocidade, interfaces de diálogo operador, etc. É muito usado em aplicações de controlo de movimento, robótica, equipamentos médicos, gruas, transportes e linhas de produção.

O meio físico utilizado é o cabo de par trançado, com dois pares de fios, CAN-H, CAN-L, CAN-GND. A sua topologia é linear com um sinal diferencial em dois condutores e um comum. O cabo pode incluir também dois condutores extra para alimentação.

Este protocolo pode ser implementado com uma topologia em barramento ou em tronco/derivações, com métodos de comunicação *master/slave*, *multi-master*, ponto-ponto e *multicast*, e as velocidades de transmissão podem atingir até 1 Mbps.

Uma das grandes vantagens no uso do CANopen reside no facto de existir mecanismos de detecção e tratamento de erros, resultando numa probabilidade muito pequena de erros não detectados.

2.3. SISTEMAS DE SUPERVISÃO DE REDES INDUSTRIAIS

As diferenças entre os protocolos de comunicação impedem a interoperabilidade entre sistemas de supervisão, automação e controlo, de diferentes fabricantes. A especificação OPC (OLE for *Process Control*) [20] define um conjunto de objetos, interfaces e métodos a ser utilizados no controlo de processo e sistemas de automação de forma a facilitar a interligação entre plataformas que usam diferentes protocolos [21].

2.3.1. CONCEITOS FUNDAMENTAIS DO OPC

A especificação OPC apareceu com o intuito de eliminar *drivers* necessários para a comunicação entre o servidor e o cliente, que na maioria das vezes são proprietários, criando assim uma dependência ao fornecedor proprietário. Utilizando OPC, apenas será necessário um *driver* para a comunicação.

A tecnologia OPC é baseada nas tecnologias Microsoft OLE (*Object Linking and Embedding*) COM (*Component Objetc Model*) e DCOM (*Distributed Component Object Model*), que são definidas pela OPC Foundation [20]. São usadas as soluções TCP ou UDP (*User Datagram Protocol*) tendo sido definido o porto 135 para o servidor. Apesar de ter como base a tecnologia DCOM da Microsoft existem também disponibilizados portos DCOM para outros sistemas operativos, tais como o Linux.

Normalmente, a tecnologia OPC é usada para as interfaces de aplicações de automação industrial, tais como sistemas IHM e SCADA, para receberem os dados dos dispositivos e para fornecerem essa informação e eventos para aplicações de gestão e monitorização (Figura 14) [22].

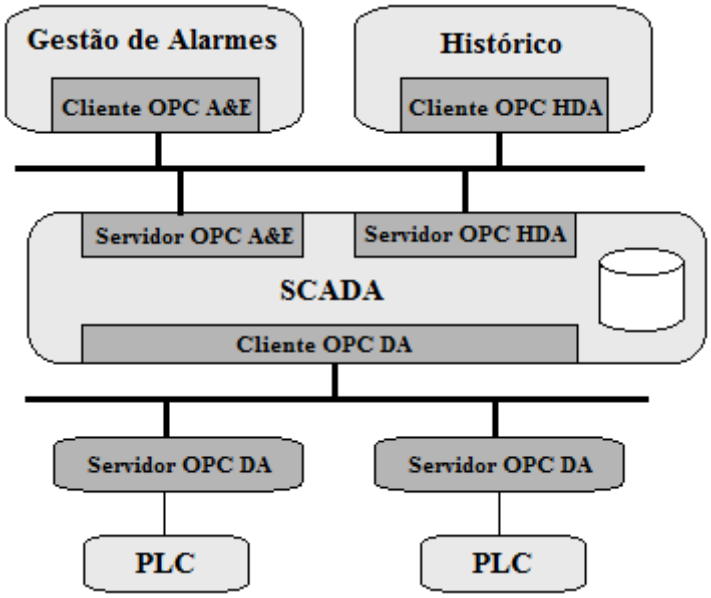


Figura 14 Uso típico dos servidores e clientes OPC [22]

Resumidamente, a especificação OPC estabelece as regras para que sejam desenvolvidos sistemas com interfaces normalizadas para comunicação dos dispositivos de campo (PLC, sensores, actuadores, entre outros) com sistemas de monitorização, supervisão e gestão, tais como, SCADA (*Supervisory Control and Data Acquisition*), MES (*Manufacturing Execution Systems*) e ERP (*Enterprise Resource Planning*), interligando desta forma os diferentes níveis hierárquicos [4].

A arquitetura da especificação OPC contém dois tipos de interfaces: a interface OPC *Custom* e a OPC *Automation*. As interfaces OPC *Custom* são projetadas para serem utilizadas com linguagens de programação que utilizam apontadores, como C/C++, enquanto que, para linguagens mais simples, como *Visual Basic* e *Delphi*, devem ser utilizadas as *interfaces* OPC *Automation*. Nestas últimas, existe um componente a mais no servidor OPC, chamado *Automation Wrapper*, que encapsula e realiza a gestão entre as linguagens sem apontadores e a interface OPC *Custom*, conforme apresentado na Figura 15 [4].

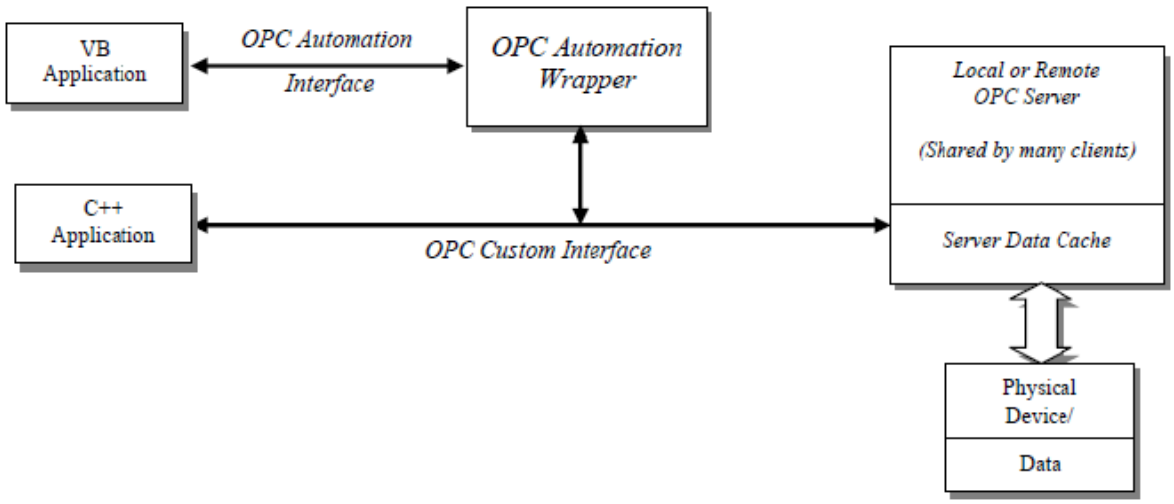


Figura 15 Arquitetura típica do OPC [23]

A especificação OPC inclui as seguintes variantes [24]:

- **OPC DA (*Data Access*)** – principal especificação do OPC; fornece a funcionalidade de transferência de dados em tempo real e de forma contínua de PLC´s para sistemas IHM, supervisão e similares;
- **OPC AE (*Alarms & Events*)** – fornece notificações de alarmes e eventos, como alarmes de processo, ações do operador, mensagens de informação, etc.;

- **OPC *Batch*** – permite mecanismos de troca de informações e condições operacionais atuais em equipamentos que implementam este tipo de controlo. É uma extensão da OPC DA;

- **OPC DX (*Data eXchange*)** – é uma extensão do OPC DA, e fornece mecanismos para troca de dados entre diferentes servidores OPC DA através de redes *fieldbus*, incluindo serviços de configuração, diagnóstico e monitorização;

- **OPC HDA (*Historical Data Access*)** – fornece mecanismos consistentes e uniformes de acesso a histórico de dados;

- **OPC *Security*** – fornece mecanismos de controlo de acesso a clientes, para as informações de processo, oferecendo desta forma uma proteção contra modificações não autorizadas de parâmetros do mesmo;

- **OPC XML/DA** – é a extensão da OPC DA, fornecendo mecanismos consistentes e flexíveis para apresentação dos dados usando a linguagem XML (*eXtensible Markup Language*), permitindo a sua apresentação em navegadores *web* via Internet/Intranet [4];

- **OPC CD (*Complex Data*)** – é outra extensão da OPC DA e permite aos servidores a descrição e representação de formatos de dados mais complexos, tais como, estruturas binárias e documentos XML. Vem sempre associada à DA ou à XML/DA;

- **OPC *Commands*** – grupo desenvolvido para a criação de interfaces que permitem aos clientes e servidores a identificação, envio e monitorização de comandos que são executados num dispositivo;

- **OPC UA (*Unified Architecture*)** – é uma especificação recente, que não necessita do uso da Microsoft COM, oferecendo segurança e fiabilidade para o acesso a dados em tempo real, histórico da informação e eventos. A especificação tem como objetivo unificar todas as especificações OPC, principalmente as OPC DA, HDA e AE. Para a sua implementação os dispositivos ou sistemas têm que ser compatíveis com o OPC UA.

2.3.2. ESPECIFICAÇÃO OPC PARA ACESSO AOS DADOS

A especificação OPC-DA é usada para o acesso aos dados num servidor OPC-DA. As funções de um servidor OPC são as de efetuar a aquisição dos dados e disponibilizar a informação aos clientes OPC, que nele se conectarem. Na especificação OPC o conceito de *namespace* representa o conjunto de dados disponibilizados pelo servidor OPC e pode ser representado de duas maneiras: de forma hierárquica ou plana [25].

Num *Namespace* Hierárquico (*Hierarchical Namespace*), os itens no servidor OPC estão organizados de forma hierárquica, ou seja, em forma de árvore. Na Figura 16 encontra-se representado um exemplo de uma estrutura de um *namespace* Hierárquico num servidor OPC-DA. Os dados de um servidor OPC podem ser designados por itens ou *tags* e descrevem as entradas, saídas, contadores ou temporizadores dos autómatos. Os grupos (*group*) servem para agrupar os diversos itens conforme a necessidade do cliente. Dentro desses grupos encontram-se os itens OPC. Os itens podem ser expansíveis, sendo denominados por *branches*. Aos que se encontram no nível mais baixo da hierarquia dá-se a designação de folhas (*leafs*). Assumindo o exemplo da Figura 16, o item denominado por "watch" é um *branch*, enquanto que o item "value 1" é um *leaf*. O caminho para o item "value 1" é representado por `"watch.device 1.value 1"`. Para se efetuar a leitura do item "value 1", o pedido terá que ter a forma "`watch.device 1.value 1`" (caminho + nome do item).

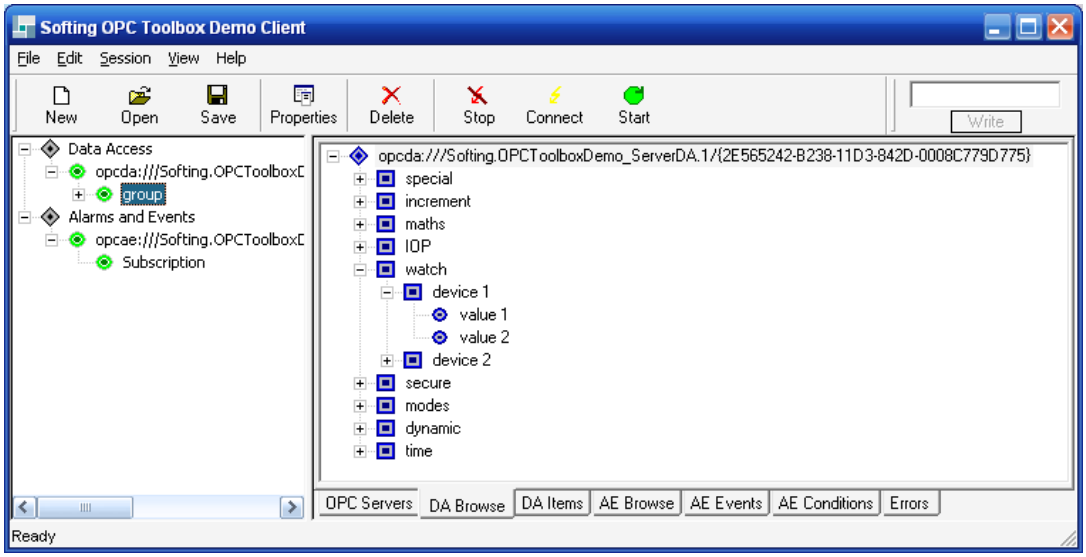


Figura 16 Simulador cliente OPC Softing – estrutura do *namespace* hierárquico do servidor

No caso do *Namespace* Plano (*Flat Namespace*), os itens encontram-se na raiz do servidor, existindo apenas itens não expansíveis (*leafs*). Na Figura 17 encontra-se representada a estrutura de um *namespace* plano.

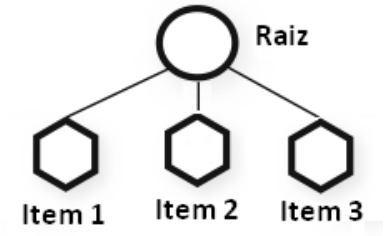


Figura 17 Estrutura de um *namespace* plano [25]

O servidor OPC pode usar um *namespace* estático ou dinâmico. No caso mais comum, o *namespace* de um servidor encontra-se disponível após a sua inicialização, sendo denominado por *namespace* estático. Contudo, em situações em que a informação está armazenada numa base de dados, os pedidos do cliente não são obtidos diretamente, sendo necessária a consulta a essa base de dados. Nestes casos, o *namespace* é designado como sendo dinâmico. Outro exemplo de utilização desta aproximação está relacionado com questões de desempenho que pode ser limitado se todos os itens estiverem disponíveis no *filesystem*.

Um servidor OPC-DA pode ser identificado por um URL (*Uniform Resource Locator*) com o formato: `opcda://<IP do Dispositivo>/<Nome do Servidor OPC>/{<ClassID>}`. Cada um dos itens no servidor OPC pode ter diversos atributos dos quais se destacam (Figura 18):

- ***Item*** – caminho e nome do item;
- ***Value*** – valor/estado do item. O valor do item é do tipo VARIANT que é definido pelo objeto COM;
- ***Quality*** – qualidade do valor do item que pode ter a classificação de boa (*good*), má (*bad*) ou incerta (*uncertain*). O servidor marcará o valor como mau se, por exemplo, não existir ligação com o dispositivo. O valor pode ser incerto se existir uma ligação com o servidor, mas o valor lido não faz qualquer sentido. A qualidade pode ainda

indicar outros tipos de informação do estado dos valores dos itens. Esta informação pode ser consultada no Anexo C;

- ***Time Stamp*** – tempo (data + hora) em que o valor do item foi obtido;

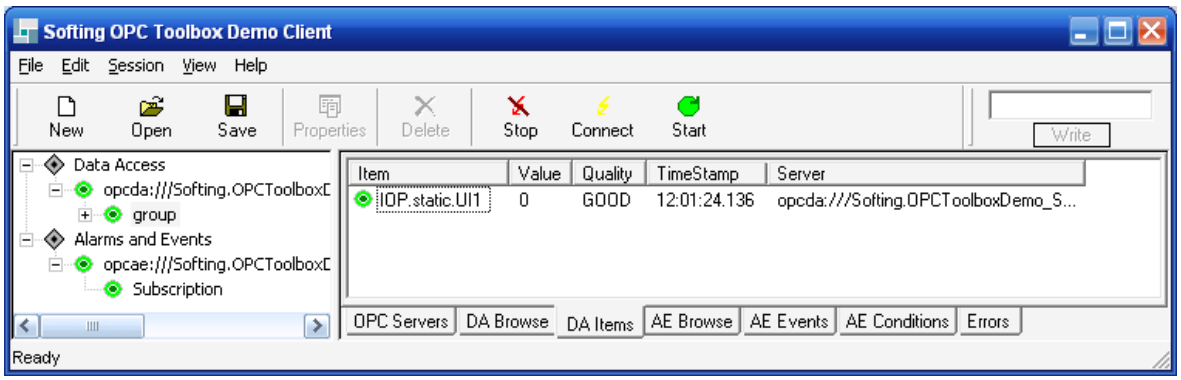


Figura 18 Simulador Softing cliente OPC – características dos itens no servidor OPC

Os componentes COM/DCOM possuem interfaces que contêm os seguintes objetos:

- ***OPCServer*** – realiza toda a gestão da conexão com o cliente e retorno dos dados. Permite ao cliente aceder aos itens disponíveis no servidor e aos métodos para a gestão do cliente de objetos *OPCGroup*.

- ***OPCGroup*** – realiza o agrupamento e gestão de estados dos objetos *OPCItem*. Este objeto é responsável pela criação, alteração e remoção dos grupos no servidor OPC, permitindo desta forma, a organização dos dados pelos clientes OPC. Disponibiliza ainda métodos de escrita e leitura dos itens.

- ***OPCItem*** – representa os itens num servidor. Um valor (*value*), uma qualidade (*quality*) e um tempo (*time stamp*) são associados a cada item. Porém, um item não pode ser acedido diretamente como um objeto pelo cliente OPC. O acesso aos itens é realizado através do objeto *OPCGroup* que implementa o *OPCItem*.

- ***OPCClient*** – o conjunto de interfaces do objeto *OPCClient* permite a comunicação e troca de informação de um cliente com um servidor.

A ligação a um servidor OPC pode estar num de três tipos de estados: desconectado (não existe ligação), conectado (existe ligação ao servidor e aos itens definidos) e ativado

(quando for realizada a ligação será efetuada uma monitorização constante aos itens previamente definidos).

A leitura de itens pode ser realizada a partir da cache interna do servidor OPC ou diretamente do dispositivo. Isto quer dizer que, quando se realiza um pedido deste género o cliente pode especificar se deseja pedir o valor diretamente ao dispositivo ou à cache. Para pedidos à cache, o utilizador pode especificar um tempo máximo desde a última atualização dos itens. Se esse valor tiver ultrapassado, a cache é atualizada antes de ser enviada a resposta.

Os pedidos podem ser feitos através de quatro tipos de comunicação:

- **Leitura/Escrita Síncrona** – o cliente efetua um pedido e aguarda pela resposta do servidor OPC (modo bloqueante);
- **Leitura/Escrita Assíncrona** – o cliente efetua um pedido, e uma função (*handler*) será responsável por verificar se recebeu a resposta do servidor OPC (modo não bloqueante);
- ***Refresh*** – o cliente realiza uma leitura regular a todos os itens num determinado grupo, independentemente dos itens terem sofrido ou não alteração no seu valor;
- ***Subscription*** – quando este modo é ativado, o servidor realiza leituras cíclicas determinadas pela taxa de atualização aos itens num determinado grupo e envia a informação para o cliente caso uma mudança de valor ou estado ocorrer. O cliente por sua vez tem a opção de definir a taxa de atualização com que os dados serão lidos pelo servidor.

2.3.3. ESPECIFICAÇÃO OPC PARA ALARMES E EVENTOS

A especificação OPC-AE é usada para alarmes e eventos gerados por um servidor OPC-AE. O principal objetivo desta especificação é notificar os sistemas de supervisão e de gestão ou estações, quais as áreas do processo que necessitam de atenção imediata. Importante referir que o servidor OPC-AE é independente do servidor OPC-DA. Contudo, existem situações (por exemplo: autómatos) que os itens/*tags* contidos no servidor OPC-

DA são iguais para o servidor OPC-AE. A tecnologia OPC-AE especifica alarmes e eventos, em que os alarmes são considerados como condições anormais, enquanto que os eventos são considerados como ocorrências detectáveis. O tipo e a forma em que os eventos e condições são definidos são da responsabilidade do fabricante do servidor. Os servidores OPC-AE são constituídos por um conjunto de interfaces e objetos, que fornecem aos clientes mecanismos para notificá-los da ocorrência de eventos e alarmes. As interfaces disponibilizam ainda, serviços que permitem aos clientes determinar os eventos e alarmes suportados pelo servidor, e obter os seus estados atuais. A especificação define os seguintes objetos COM:

- *OPCEventServer* – fornece métodos para a criação dos objetos *OPCEventSubscription* e *OPCEventAreaBrowser* (Figura 19). Permite ainda consultar as categorias dos eventos e parâmetros associados ao evento.

- *OPCEventSubscription* – fornece um meio para os clientes acederem à área de eventos implementada pelo servidor. Permite a configuração de filtros e outros atributos para a notificação de eventos. A implementação deste objeto é opcional.

- *OPCEventAreaBrowser* – usado para gerir os eventos e notificar os clientes.

Figura 19 Relação do objeto OPC Event Server

A especificação AE define três tipos de eventos:

- **Eventos simples (*Simple Events*)** – são considerados eventos simples, eventos tais como, falha num dispositivo, mudanças de estado nos itens, entre outros.

- **Eventos *tracking* (*Tracking Events*)** – os eventos *tracking* são gerados quando existe intervenção por parte do utilizador, como por exemplo, escrita num item.

- **Eventos de condição (*Condition Events*)** – os eventos de condição são gerados quando as condições impostas se encontram fora dos limites. Por outras palavras, pode-se aplicar uma gama de valores às variáveis medidas (condições) e sempre que a variável ultrapasse essa gama de valores é gerado um evento. Importante mencionar, que pode ser imposta à mesma variável diversas condições, designadas por subcondições. Por exemplo, na leitura de um sensor analógico, se for imposta uma condição de 10 a 50, sempre que a variável ultrapassar estes valores, será gerado um evento do tipo condição.

Tabela 5 Atributos retornados pelos eventos

Parâmetros	Descrição	Simples	Tracking	Condição
Fonte	Item ao qual ocorreu o evento.	✓	✓	✓
Tempo	Quando ocorreu o evento.	✓	✓	✓
Tipo	Tipo do evento (simples, *tracking*, condição).	✓	✓	✓
Categoria	Tipo de categoria do evento.	✓	✓	✓
Severidade	Nível de severidade do evento (valores entre 1 a 1000), quanto maior o valor mais grave é considerado o evento.	✓	✓	✓
Mensagem	Descrição do evento.	✓	✓	✓
ID Actor	Utilizador responsável pelo evento.	✗	✓	✓
Nome da Condição	Nome da condição do evento.	✗	✗	✓
Nome da Subcondição	Nome da subcondição do evento.	✗	✗	✓
Mudança da Máscara	Indica quais as propriedades da condição que modificaram para causar o envio da notificação.	✗	✗	✓
Novo estado	Novo estado da condição (*Active*, *Enable*, ACK, *Disconnected*).	✗	✗	✓
Qualidade	Qualidade da condição (Anexo C). Quando a qualidade de uma condição é alterada, gera um evento.	✗	✗	✓
ACK	*Flag* para indicar se a condição necessita de reconhecimento (*acknowledgment*) deste evento.	✗	✗	✓
Tempo Ativo	Indica o tempo que a condição ficou ativa (*Active*) ou o tempo da transição da atual subcondição.	✗	✗	✓
Cookie	*Cookie* definido pelo servidor, associado ao evento e usado pelo cliente durante o reconhecimento (*acknowledgment*).	✗	✗	✓

A Tabela 5 apresenta os atributos retornados por cada um dos eventos. A implementação de mecanismos de *acknowledgement* e filtros podem ser usados. O objeto *OPCEventSubscription* permite a aplicação dos seguintes filtros:

- **Tipo** – filtro para tipo de evento (simples, *tracking*, condição).
- **Categoria** – filtro para categorias de eventos.
- **Severidade mínima** – envia para o cliente notificações acima do valor definido para o filtro.
- **Severidade máxima** – envia para o cliente notificações até ao valor definido para o filtro.
- **Área** – filtro para grupos (áreas) de eventos.
- **Fonte** – filtro para itens (fontes).

No OPC-AE existem dois tipos de estruturação de dados (Figura 20):

- ***Event Area*** – agrupa os diferentes itens por áreas de processo e é semelhante ao *namespace* hierárquico do OPC-DA. Esta estrutura é sempre do tipo hierárquico e é constituída por áreas e fontes (itens). Por exemplo, na Figura 20 a), a fonte denominada por "Temperatura" encontra-se na área "Edifício da produção ► Linha 1 ► Dispositivo" e tem associada uma condição denominada por "Condição de Nível".
- ***Filter Space*** – usada para estruturação lógica dos eventos. Os três tipos de eventos (simples, *tracking* e condição) são associados a categorias de eventos. Por exemplo, na Figura 20 b), a categoria "Categoria 1" contém um evento de condição denominado por "Condição de Nível" e os atributos denominados por "Valor mínimo" e "Valor máximo".

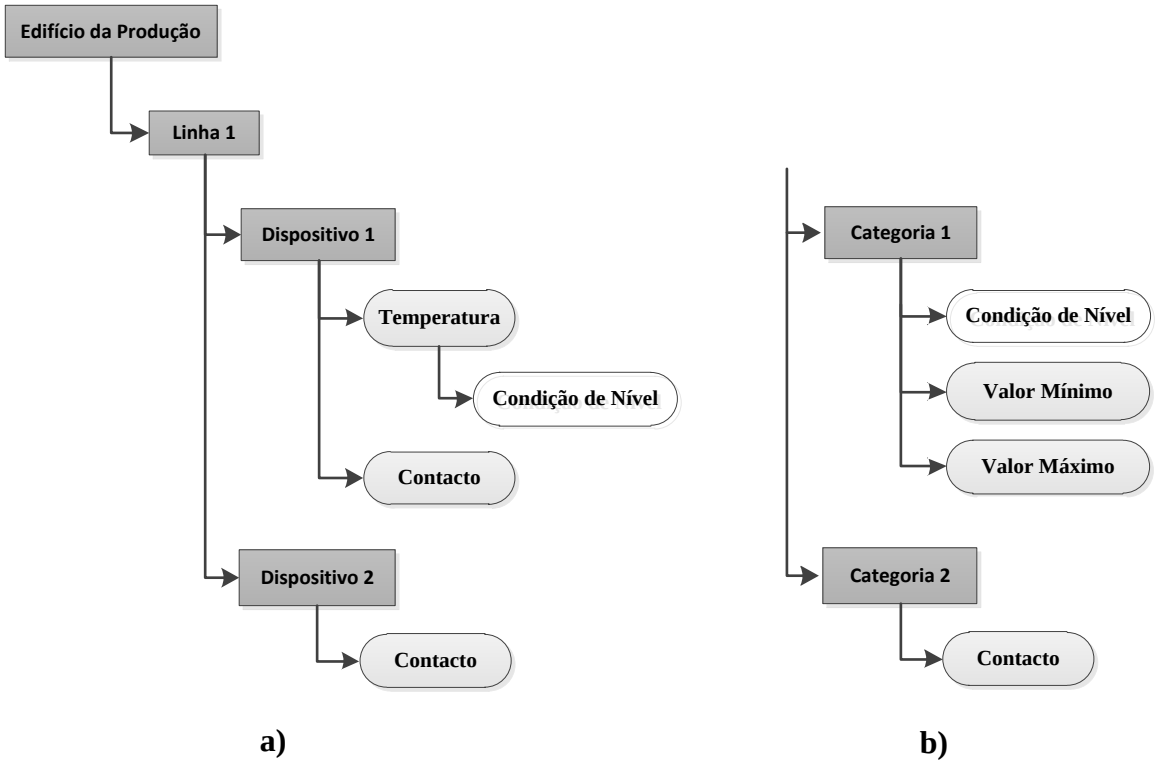


Figura 20 Estrutura de dados num servidor OPC-AE. a) *Event Area* b) *Filter Space*

3. ARQUITETURA DO SISTEMA DE SUPERVISÃO

De maneira a dotar os sistemas de supervisão e de controlo da empresa com protocolos capazes de controlar e monitorizar o estado dos autómatos, foram considerados alguns aspetos relevantes na definição da arquitetura a utilizar. Um dos fatores principais na seleção dos protocolos a utilizar foi o nível hierárquico a que pertencem. Visto que o sistema deve comportar aspetos que vão desde o nível de controlo até ao nível de gestão, optou-se por uma solução baseada em *fieldbus* e OPC.

A opção pela especificação OPC foi baseada no facto de esta apresentar a capacidade de leitura, escrita e monitorização das variáveis de um autómato (especificação DA), bem como possibilitar a monitorização de eventos e alarmes gerados por estes (especificação AE). A seleção do protocolo IEC 60870-5-104 foi motivada pela robustez da sua trama, bem como pela implementação de mecanismos contra falhas e perdas de dados.

Um dos requisitos principais definidos foi a implementação dos módulos protocolares em Linux (Red Hat Enterprise Linux 5), sistema operativo utilizado pelos sistemas de supervisão e de controlo da EFACEC. Neste capítulo serão apresentadas algumas soluções

e arquiteturas para a implementação dos clientes OPC-DA e OPC-AE para este sistema operativo, e ainda para a implementação do *master* do IEC 60870-5-104.

3.1. SOLUÇÕES E ARQUITETURAS OPC PARA LINUX

A especificação OPC Clássico (OPC-DA e OPC-AE) é baseada nas tecnologias Microsoft OLE COM e DCOM que são disponibilizadas pelas plataformas Windows. Por esta razão, a integração numa plataforma em Linux requer algum esforço. Para integrar a especificação OPC-DA em sistema Linux terá que se optar por uma das seguintes soluções possíveis [26]:

1. Desenvolver o cliente OPC-DA em ambiente Windows e através de um mecanismo (por exemplo: *sockets*) permitir ao utilizador em ambiente Linux aceder à aplicação;

2. Construir um cliente XML/DA (Linux) e adquirir uma *gateway/bridge* XML/DA para DA (Windows);

3. Implementar a especificação DA em ambiente Linux com a biblioteca OpenOPC e instalar a *gateway* do OpenOPC numa máquina em Windows;

4. Construir um cliente Java (Linux) com o "Java COM bridge" (Windows);

5. Instalação do serviço DCOM em Linux − Não é recomendável devido à necessidade de diversas dependências e de determinadas versões de pacotes instalados. Para a implementação de uma solução deste tipo, existe um *software* comercializado pela Software AG, denominado de "EntireX DCOM". Ainda é possível encontrar uma outra solução similar a esta, com a vantagem de ser livre, que implementa os serviços DCE (*Distributed Computing Environment*) e DCOM no Linux [27]. O problema desta última solução é que os pacotes novos existentes nos sistemas operativos Linux, não são compatíveis com esta solução.

No Anexo A são apresentadas algumas bibliotecas que implementam a especificação OPC. Para a implementação da solução OPC em Linux destacaram-se as seguintes bibliotecas livres:

- **PyOPC** – Esta solução é referente à implementação do cliente XML/DA e necessita de uma *gateway/bridge* XML/DA para DA instalada num sistema operativo Windows. A linguagem utilizada para a sua implementação é o Python. Esta solução não foi alvo de prova de conceito [28].

- **OpenOPC** – Disponibiliza um cliente OPC-DA para Linux e uma *gateway* instalada numa máquina com sistema operativo Windows. A linguagem utilizada para a sua implementação é o Python [29].

- **UTGARD** – Implementa o cliente OPC-DA com 100% de Java puro, não necessitando de bibliotecas JNI (*Java Native Interface*) ou DLL (*Dynamic-Link Library*). A *gateway/bridge* é realizada pela biblioteca *j-Interop* que é usada juntamente com a biblioteca UTGARD, que permite a implementação do protocolo DCOM, através do Java COM *bridge*. Esta solução não foi alvo de prova de conceito [30].

Todas as bibliotecas que foram apresentadas necessitam sempre de uma máquina Windows para servir de *gateway/bridge* para os serviços COM/DCOM disponibilizados pela plataforma Windows (OpenOPC e UTGARD) ou para tradução de XML/DA para DA (PyOPC), como demonstra na Figura 21.

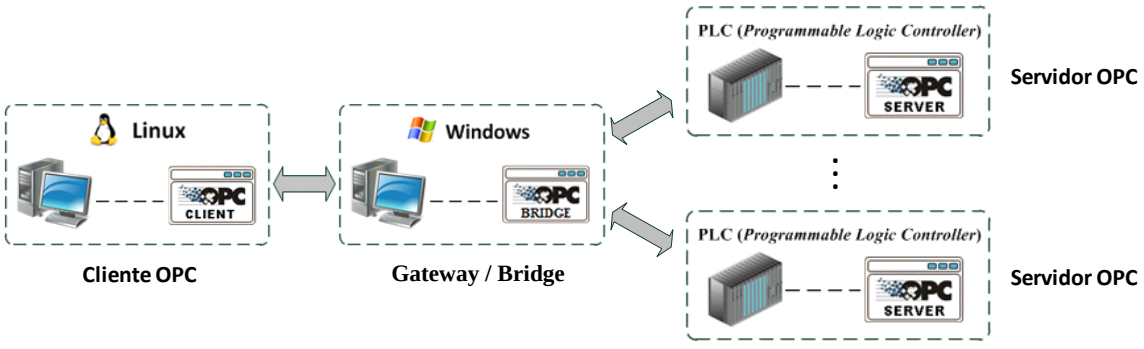


Figura 21 Arquitetura para a especificação OPC-DA em sistema Linux

As soluções apresentadas apenas implementam a especificação OPC-DA, sendo possível a sua integração em sistemas Linux. Ao passo que a especificação OPC-AE terá que ser integrada em sistema Windows, visto que não foram encontradas soluções que implementem esta especificação em ambientes Linux. Para isso, terá que ser usado um mecanismo, por exemplo *sockets* TCP, para que o cliente em ambiente Linux possa aceder à aplicação desenvolvida em sistema Windows.

De forma a tornar possível a implementação dos clientes OPC-DA e OPC-AE num sistema Linux optou-se por analisar as seguintes arquiteturas:

- Desenvolvimento de clientes OPC-DA e OPC-AE em Windows e através de um mecanismo de *sockets* TCP, o cliente em sistema Linux aceder à aplicação;
- Desenvolvimento de cliente OPC-XML/DA em sistema Linux e desenvolver o cliente OPC-AE e instalar a *gateway/bridge* (tradução de XML/DA para DA) em sistema Windows;
- Desenvolvimento de cliente OPC-DA em sistema Linux com a biblioteca OpenOPC e desenvolver o cliente OPC-AE e instalar a *gateway/bridge* em sistema Windows;

3.1.1. ARQUITETURA OPC-DA E OPC-AE EM SISTEMA WINDOWS E INTERFACE EM SISTEMA LINUX

Uma possível solução para utilizar um cliente OPC em Linux a comunicar com servidores OPC-DA e OPC-AE, seria o desenvolvimento destas duas especificações numa máquina em Windows e aceder a estas duas especificações através de uma interface proprietária desenvolvida em sistema Linux.

Na máquina Windows seriam desenvolvidos dois clientes OPC (DA e AE) para comunicar com os servidores OPC (DA e AE) dos PLC. Das soluções existentes para a implementação destas especificações optou-se pelo *software* proprietário "OPC Classic Toolkits" da empresa Softing [31].

Desta forma, é possível implementar os clientes OPC-DA e OPC-AE com o *software* "OPC Classic Toolkits" numa máquina Windows. Posteriormente será necessário o

desenvolvimento de uma interface no sistema Linux, que permita aceder à solução desenvolvida em Windows, isto para possibilitar a comunicação entre o sistema em Linux com os servidores OPC dos PLC.

Nesta solução pode optar-se por implementar os clientes OPC-DA e OPC-AE com os SDK (*Software Development Kit*) da Softing, ou então, implementar o cliente OPC-AE com o SDK da Softing e o cliente OPC-DA com uma solução livre, como por exemplo o *JEasyOPC Client* [32]. Na Figura 22 é possível visualizar a arquitetura para esta solução.

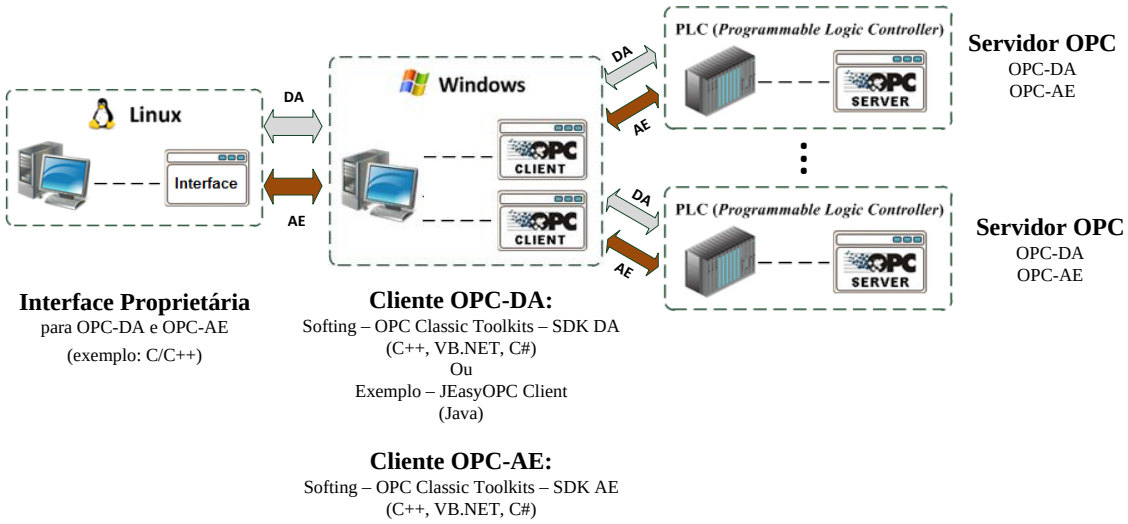


Figura 22 Arquitetura OPC-DA e OPC-AE em sistema Windows

3.1.2. ARQUITETURA OPC-XML/DA EM SISTEMA LINUX E OPC-AE EM SISTEMA WINDOWS

Uma solução alternativa passa pela implementação da especificação OPC-XML/DA em Linux (Red Hat Enterprise Linux 5), uma vez que se baseia em serviços *web* e a especificação OPC-AE em Windows, com os SDK da Softing.

Os serviços *web* são componentes que permitem às aplicações enviar e receber dados em formato XML. Assim, torna-se possível que os recursos de uma aplicação de um *software* se encontrem disponíveis sobre a rede de uma forma normalizada. Os serviços

Web são baseados em tecnologias normalizadas, em particular o XML e o HTTP (*Hypertext Transfer Protocol*). Desta forma, permitem disponibilizar serviços interativos na Web, podendo ser acedidos por outras aplicações usando, por exemplo, o protocolo SOAP (*Simple Object Access Protocol*). O modo de acesso ao serviço Web é descrito por um ficheiro com uma linguagem em WSDL (*Web Service Description Language*), que é baseada em XML.

A especificação XML/DA é baseada nas normas dos serviços Web, como o XML, SOAP e WSDL, e normaliza as mensagens SOAP trocadas entre o cliente e o servidor. A normalização dessas mensagens permite a implementação em sistemas operativos diferentes. O SOAP é baseado em XML de forma a encapsular as mensagens num formato adequado para a transmissão, usando o protocolo HTTP.

Como o cliente é implementado com a especificação OPC-XML/DA e o servidor com a especificação OPC-DA, será necessário a implementação de uma ponte (*gateway/bridge*) para a sua conversão. A implementação desta ponte tem que ser realizada numa plataforma Windows, já que a especificação OPC-DA utiliza os serviços COM/DCOM do Windows para a comunicação com o servidor.

De uma forma geral, esta é a solução com maior custo financeiro. Isto é devido ao facto de necessitar do SDK XML/DA, de uma Ponte (*software* "OPC Easy Connect Suite" da Softing [33] para a conversão de XML/DA para DA) e do SDK AE.

A ponte é instalada numa máquina em Windows, bem como a especificação OPC-AE. O *software* "OPC Easy Connect Suite" da Softing é uma aplicação gráfica que permite encontrar e adicionar servidores OPC-DA (Figura 23). A função principal desta ponte será a conversão da especificação XML/DA para DA, enviando essa informação para o servidor. Ainda permite configurar outro tipo de parâmetros, como por exemplo, realizar um armazenamento dos dados que circulam na ponte, configurar a porta utilizada para a ponte, entre outros. A arquitetura descrita para esta solução encontra-se representada na Figura 24.

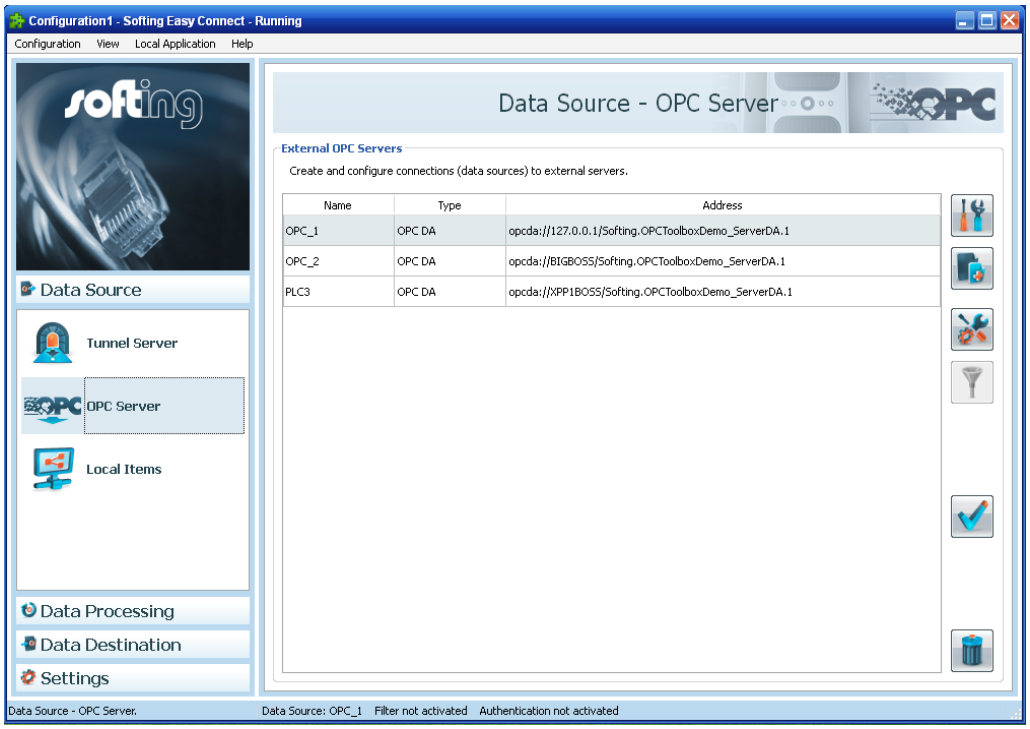


Figura 23 "OPC Easy Connect Suite" da Softing

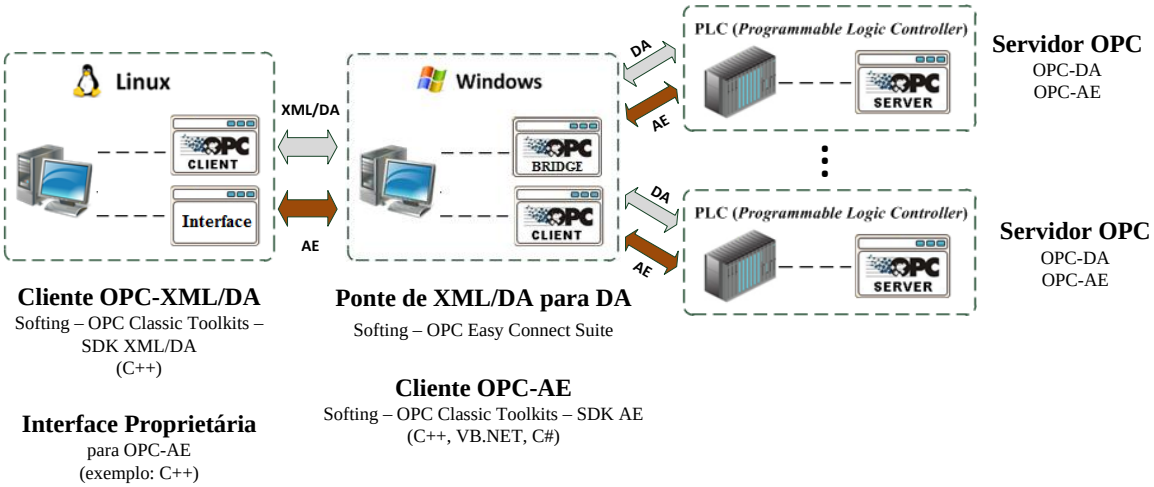


Figura 24 Arquitetura OPC-XML/DA em sistema Linux e OPC-AE em sistema Windows

3.1.3. ARQUITETURA OPC-DA EM SISTEMA LINUX E OPC-AE EM SISTEMA WINDOWS

Uma outra possível implementação para o cliente OPC seria desenvolver o cliente OPC-DA em sistema Linux e através de uma *interface* proprietária aceder ao cliente OPC-AE desenvolvido com o SDK AE da Softing em sistema Windows. A biblioteca OpenOPC permite realizar a implementação do cliente OPC-DA diretamente no sistema em Linux, necessitando a instalação de uma *gateway* (incluída com a biblioteca) num sistema em Windows. Esta *gateway* é usada para aceder aos serviços COM/DCOM e enviar a informação para o servidor OPC-DA do PLC. A arquitetura desta solução encontra-se representada na Figura 25.

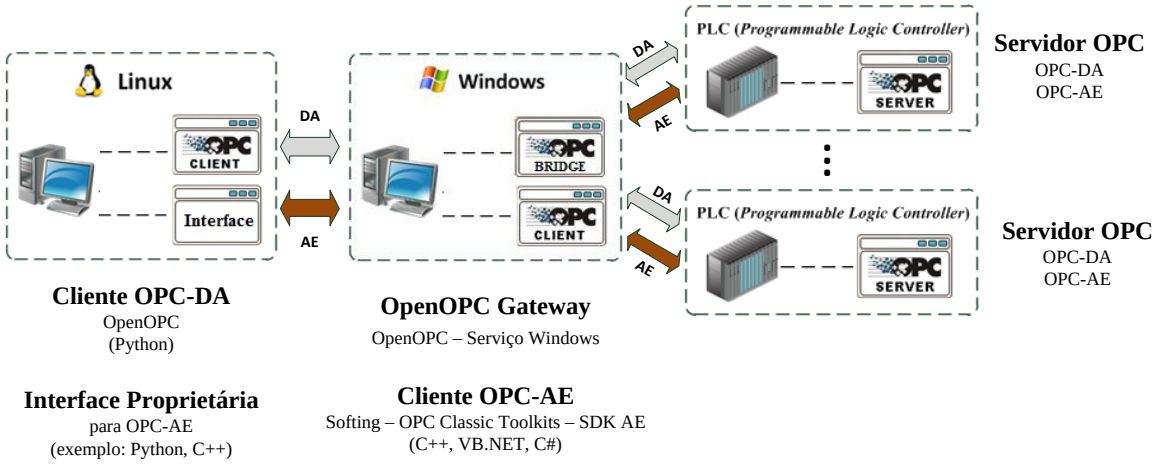


Figura 25 Arquitetura OPC-DA em sistema Linux e OPC-AE em sistema Windows

A Figura 26 mostra um exemplo de escrita de dados num servidor OPC-DA através da biblioteca OpenOPC em sistema Linux. Para este exemplo, na linha de comandos, insere-se o IP da *gateway* (IP da máquina Windows: 192.168.122.200), o IP do PLC (192.168.122.191), nome do servidor OPC-DA do PLC (neste caso foi usado um simulador de um servidor OPC-DA com o nome: Softing.OPCToolboxDemo_ServerDA), o item DA a escrever (IOP.static.I1) e o valor a escrever (neste caso, "1").

```
[daniel@redhat1boss OpenOPC]$ /usr/bin/python26 opc.py -m open -H 192.168.122.20
0 -h 192.168.122.191 -s Softing.OPCToolboxDemo_ServerDA -w IOP.static.I1 1
IOP.static.I1     Success
```

Figura 26 Escrita de dados com o cliente OPC-DA

Na Figura 27 mostra um exemplo de leitura de dados num servidor OPC-DA através da biblioteca OpenOPC em sistema Linux. Para este exemplo, na linha de comandos, insere-se o IP da *gateway* (IP da máquina Windows: 192.168.122.200), o IP do PLC (192.168.122.191), nome do servidor OPC-DA do PLC (neste caso foi usado um simulador de um servidor OPC-DA com o nome: Softing.OPCToolboxDemo_ServerDA) e o item DA a ler (IOP.static.I1).

```
[daniel@redhat1boss OpenOPC]$ /usr/bin/python26 opc.py -m open -H 192.168.122.20
0 -h 192.168.122.191 -s Softing.OPCToolboxDemo_ServerDA -r IOP.static.I1
IOP.static.I1     1     Good     04/04/13 18:05:10
```

Figura 27 Leitura de dados com o cliente OPC-DA

3.1.4. COMPARAÇÕES ENTRE AS ARQUITETURAS APRESENTADAS

Através de testes efetuados para cada uma das arquiteturas obteve-se um tempo de acesso aos dados relativamente instantâneo para as arquiteturas do OPC-DA e OPC-AE em sistema Windows e OPC-XML/DA em sistema Linux. Relativamente à arquitetura que implementa a biblioteca OpenOPC, verificou-se um tempo de acesso aos dados superior. Um dos fatores para este atraso, está relacionado com a integração da *gateway* no sistema Windows.

Os custos associados à arquitetura do OPC-DA e OPC-AE em sistema Windows são referentes aos SDK do OPC-DA e OPC-AE, em que o OPC-DA poderá ser desenvolvido com uma biblioteca livre ou não. Como neste caso foram considerados os SDK da Softing, os custos serão relacionados aos SDK DA e AE da Softing. Em contrapartida, o suporte oferecido pela empresa é melhor. Os custos de implementação da solução com a biblioteca OpenOPC e do SDK AE são menores, comparativamente à arquitetura que implementa a especificação OPC-XML/DA. Isto deve-se ao facto do OpenOPC ser uma biblioteca livre, enquanto que o SDK XML/DA é comercial e ainda necessita de uma *bridge/gateway* para o acesso aos dados no servidor OPC-DA.

3.2. ARQUITETURA DO MESTRE IEC 60870-5-104

O protocolo IEC 60870-5-104 define dois tipos de dispositivos − os dispositivos a serem controlados (escravos) e os dispositivos que controlam (mestres). Como se pretende controlar e monitorizar os dispositivos, será então implementado o mestre IEC 60870-5-104.

A implementação do mestre IEC 60870-5-104 não implica quaisquer restrições para a sua integração nos sistemas de supervisão e de controlo. Por este motivo, o mestre será integrado diretamente no sistema de supervisão (Linux) que irá interagir com os diversos dispositivos escravos que se encontram na rede. A arquitetura para este tipo de implementação encontra-se representada na Figura 28.

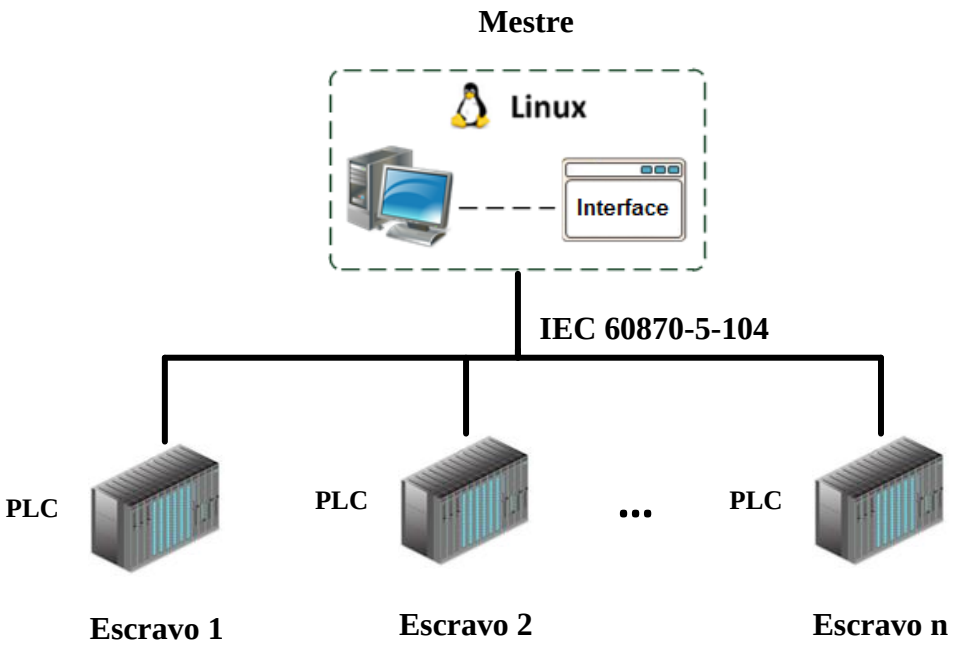


Figura 28 Arquitetura do mestre IEC 60870-5-104 no sistema de supervisão

4. IMPLEMENTAÇÃO DOS MÓDULOS PROTOCOLARES DE SUPERVISÃO

Para o desenvolvimento do cliente OPC optou-se por usar a solução que agrega as especificações DA e AE em Windows (Figura 22). A seleção da arquitetura foi baseada em diversos fatores, nomeadamente no tempo de acesso aos dados, nos custos associados e suporte oferecido. Os clientes OPC-DA e OPC-AE serão dois programas independentes, visto que os dispositivos poderão conter apenas um dos servidores, ou ambos.

De uma forma resumida, a integração dos clientes OPC-DA e OPC-AE contém duas componentes fundamentais:

1. O desenvolvimento de duas *interfaces* proprietárias na máquina em Linux, que obriga ao uso de *sockets* para a passagem dos dados para a máquina em Windows.

2. O desenvolvimento de duas *interfaces* (OPC-DA e OPC-AE da Softing) na máquina em Windows, responsáveis pela recepção de dados e pelo processamento e envio dos mesmos. O processamento de dados refere-se à interpretação das mensagens recebidas e implementação dos clientes OPC-DA e OPC-AE.

Nas secções seguintes discutem-se os aspectos relativos às implementações dos clientes OPC-DA e OPC-AE no sistema em Windows através da biblioteca da Softing e do mestre IEC 60870-5-104 diretamente no sistema de supervisão (sistema em Linux). As *interfaces* proprietárias OPC-DA e OPC-AE para o sistema de supervisão encontram-se fora do âmbito desta dissertação, visto que será a empresa EFACEC a desenvolve-las.

4.1. BIBLIOTECA SOFTING

Para o desenvolvimento dos clientes OPC-DA e OPC-AE serão usados os SDK DA e AE do *toolbox* da Softing (membro da OPC Foundation), que já implementam as *interfaces* com o objeto *OPCClient* dos componentes COM/DCOM. A versão utilizada pelos SDK para o desenvolvimento do produto é a versão 4.31. Os SDK da Softing fornecem um conjunto de ferramentas (classes, métodos, construtores e propriedades) que são utilizadas para facilitar a programação de clientes e servidores OPC. Utilizando uma abordagem similar a alguns outros fornecedores, os SDK simplificam todas as funcionalidades OPC encapsulando-as em algumas DLL. Disponibiliza também um IDE (*Integrated Development Environment*) que permite a geração de um projeto usando uma *interface* do tipo *wizard*, criando o código base das funcionalidades selecionadas, como ilustra a Figura 29.

O *toolbox* da Softing utilizado para o desenvolvimento dos clientes OPC tem as seguintes funcionalidades:

- Suporta as especificações: DA 1.0, DA 2.05, DA 3.0, AE 1.10, XML-DA 1.01 para Windows, XML-DA 1.0 para Linux.
- Pode ser desenvolvido nos sistemas operacionais: Windows 2000, Windows XP, Windows 2003 Server, Windows Vista, Windows Server 2008, Windows 7 e Linux nas linguagens C#, Visual Basic .NET e C++.

- Pode ser desenvolvido nas plataformas: NET Framework 1.1, 2.0, 3.5 e 4.0, com a Microsoft Visual Studio 2003, 2005, 2008 e 2010 e Microsoft Visual Studio 2005 Express Edition 2005, 2008 e 2010.

- Contém documentação e amostras de programas com o código fonte tanto de clientes OPC como de servidores OPC.

- Disponibiliza *wizards* para a criação de clientes e servidores OPC (versão demonstrativa).

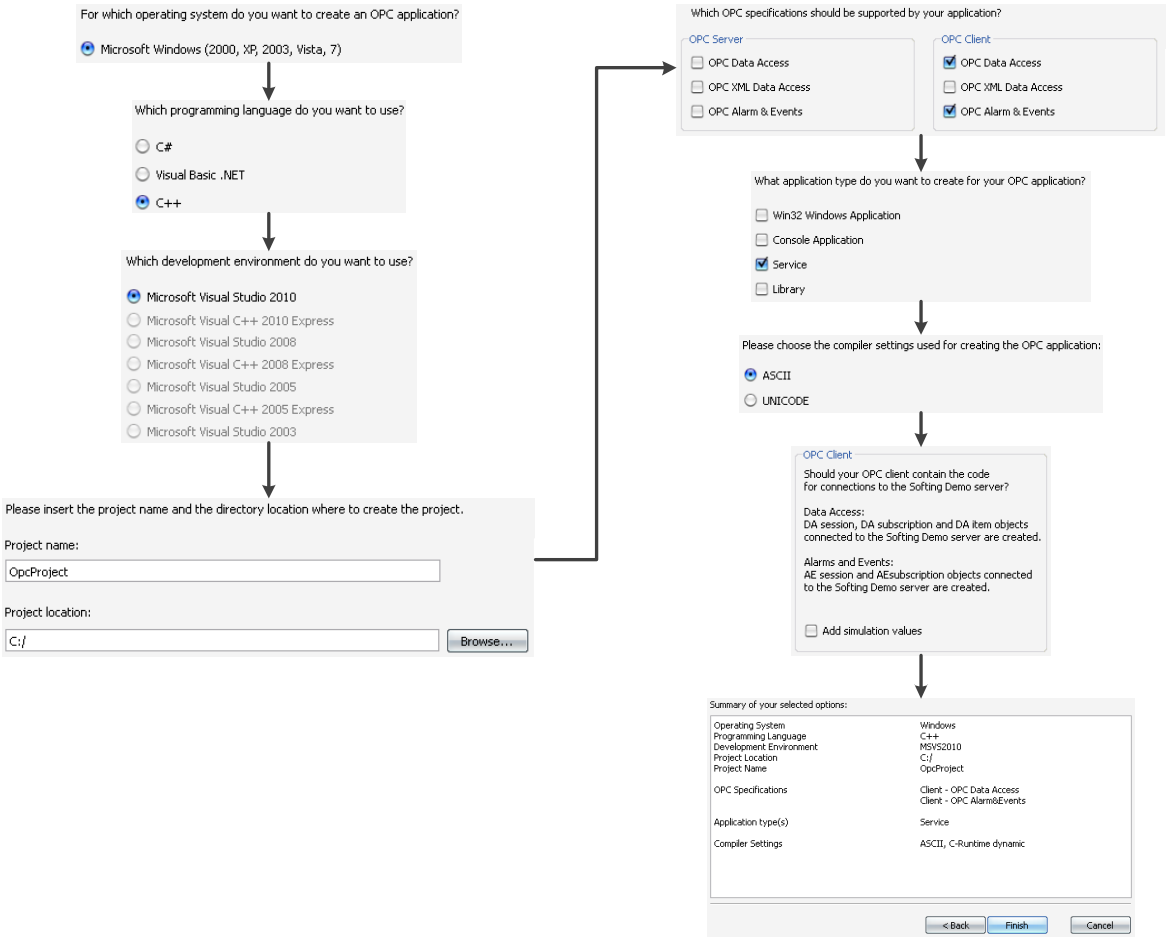


Figura 29 Passos do *wizard* do Softing OPC *toolbox*

4.2. SERVIÇO WINDOWS DO CLIENTE OPC-DA

Para a implementação do cliente OPC-DA em sistema Windows foi desenvolvido um programa que irá funcionar como um serviço do Windows ficando, desta forma, o seu funcionamento transparente para o utilizador (Figura 30). O serviço foi desenvolvido com um mecanismo de *sockets* TCP para permitir a passagem de dados entre a máquina Linux e a máquina Windows.

Para o desenvolvimento do serviço Windows foi usado o Microsoft Visual Studio 2010, com a linguagem em C++ do SDK DA da Softing. Para auxiliar o desenvolvimento do cliente OPC-DA foi usado o simulador de servidor OPC-DA disponibilizado pela Softing. O serviço desenvolvido suporta a versão 2.0 do OPC-DA pelo facto de esta ser a versão implementada em alguns dos dispositivos.

Para permitir a comunicação entre o cliente e o servidor OPC são necessárias as configurações das seguranças DCOM e da *firewall* (porta 135 TCP) para possibilitar as especificações OPC-DA e OPC-AE acederem aos serviços COM/DCOM disponibilizados pelas plataformas da Microsoft.

Para tornar o serviço modular optou-se por implementar um mecanismo que permite a ligação de múltiplos clientes (sistemas em Linux ou em Windows) e múltiplos pedidos por cliente (leitura, escrita, monitorização, entre outros). O *software* desenvolvido implementa mecanismos para tratar potenciais problemas como *threads* bloqueadas. O bloqueio de *threads*, por exemplo, é tratado utilizando mecanismos de *timeout* e de verificação de falha na comunicação com o *socket*. A Figura 31 apresenta o fluxograma do funcionamento do serviço Windows desenvolvido e no Anexo D descreve-se o processo para proceder à sua instalação.

Por razões de processamento foi aplicado o mecanismo denominado por *select* para aceitar ligações TCP. O *select* é um mecanismo síncrono, ou seja, aguarda por chegada de dados, e não bloqueante, isto permite reduzir o processamento do serviço e aplicar um mecanismo de *timeout*.

Nome	Descrição	Estado	Tipo de arranque	Iniciar sessão como
OPC DA Service	OPC DA Client Service	Iniciado	Automático	Sistema local

Figura 30 Serviço Windows OPC-DA

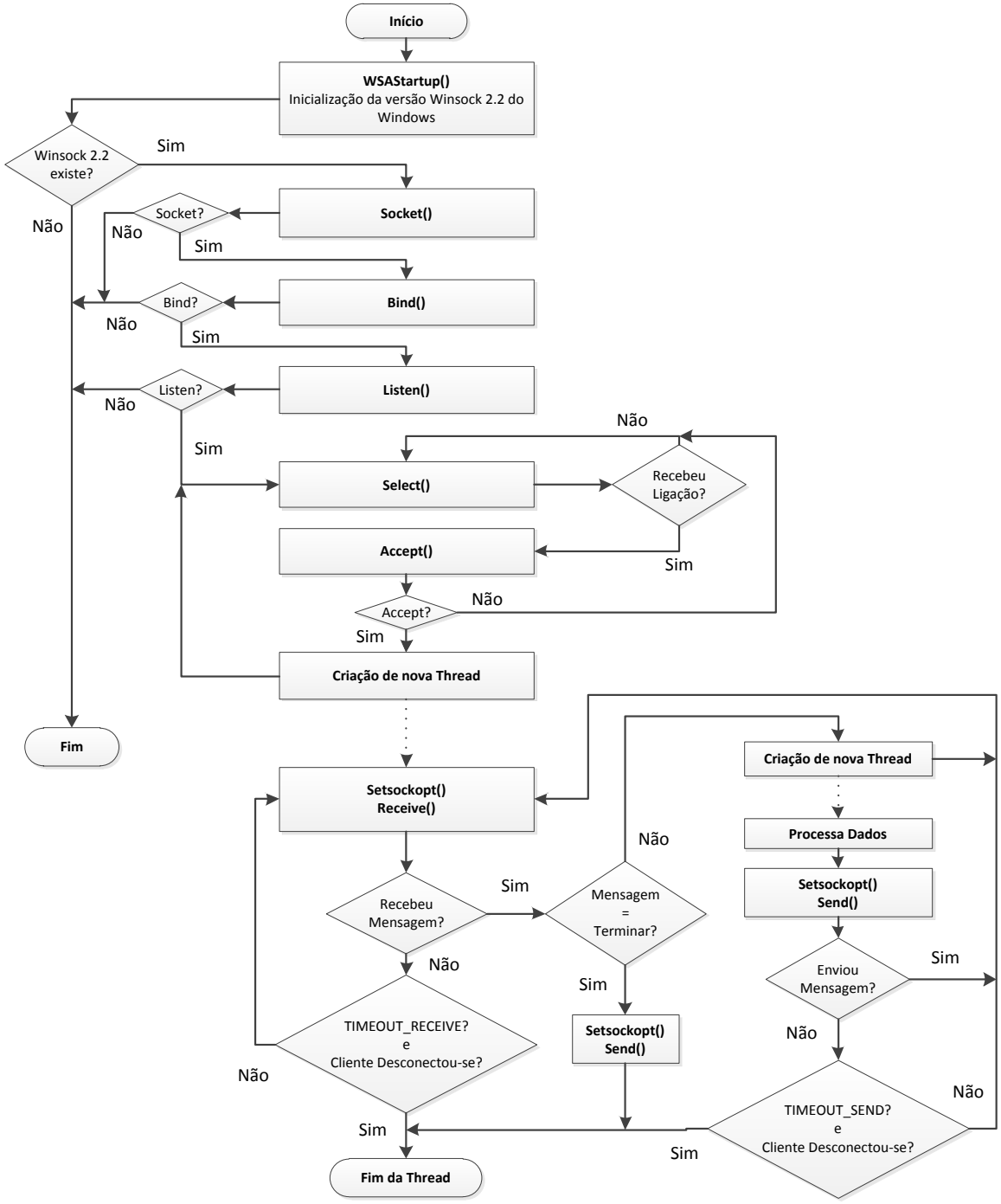


Figura 31 Fluxograma do funcionamento do serviço Windows

Ao ser inicializado pela primeira vez, o serviço irá criar dois tipos de ficheiros:

- **Ficheiro de inicialização (*Settings.ini*):** contém algumas das variáveis principais para o sistema (Tabela 6). O ficheiro depois de ser criado pode ser editado, conforme as necessidades do utilizador. Em cada re-início do serviço é verificada a

existência ou não deste ficheiro e dos campos necessários para a inicialização das variáveis do programa. No caso destas condições não serem satisfeitas, o ficheiro será criado e/ou os campos inicializados com valores por defeito.

- **Ficheiro de histórico de eventos e erros do serviço (*LogFile.log*):** registo de eventos e erros do serviço Windows. A mensagem gerada contém o formato: `<data> <hora> - <evento/error>: <mensagem>`. Determinados erros, gerados pelo sistema operativo, serão impressos com o código do erro e a respetiva mensagem do sistema operativo Windows (Figura 32).

Tabela 6 Variáveis definidas no ficheiro de inicialização

Ficheiro de inicialização (*Settings.ini*)		
Campos da secção [Settings]	**Valores por defeito**	**Descrição**
MaxThreads	100	Define o número máximo de *threads* inicializadas pelo sistema.
BufferSize	150	Define o número máximo de caracteres para a troca de mensagens entre o serviço Windows e o cliente.
TimeoutSend	5	Tempo máximo, em segundos, para o envio de dados para o cliente.
TimeoutReceive	15	Tempo máximo, em segundos, para a recepção de dados provenientes do cliente.
MaxClients	10	Número máximo possível de clientes conectados ao serviço.
PortNumber	27015	Número da porta usada pelo serviço para possibilitar a ligação de clientes ao serviço.

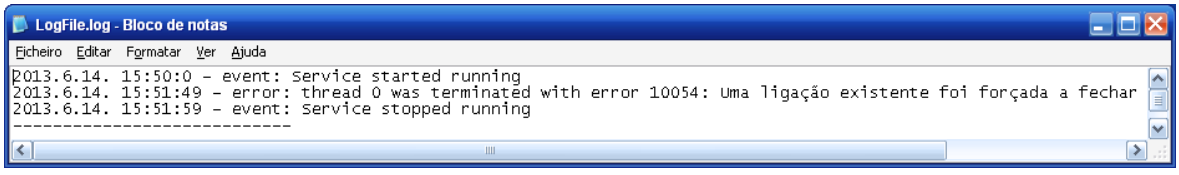


Figura 32 Ficheiro de histórico de eventos e erros do serviço

Os utilizadores para requisitarem pedidos OPC ao serviço Windows desenvolvido terão que enviar um comando específico (em forma de *string*) para que este seja processado. Com a exceção dos comandos relacionados com caminhos e nomes de itens, o serviço trata os pedidos como *case sensitive*. As funcionalidades implementadas no serviço são:

- **Listagem de servidores OPC-DA 2.0 num dispositivo** − serão retornados os URL dos servidores OPC-DA 2.0 contidos no dispositivo selecionado;

- **Ligação a um servidor OPC-DA de um dispositivo** − o URL do servidor inserido pelo utilizador será guardado para a criação de sessões com o servidor para requisitar os seguintes pedidos:

  - **Listagem dos itens contidos no servidor:** serão retornados os itens contidos num determinado caminho inserido pelo utilizador (*namespace* do servidor). De mencionar que os itens contidos num determinado caminho podem ser expansíveis;

  - **Listagem de informações sobre o servidor:** serão retornadas as informações relativas ao servidor, que contêm o seu estado, uma descrição, informação do vendedor, versão do produto, tempos de inicialização, tempo atual, etc.;

  - **Leitura de itens no servidor:** serão retornadas as leituras requisitadas pelo utilizador (estado de entradas, saídas, contadores, registos, entre outros). Nesta opção, o utilizador insere o caminho das variáveis que pretende ler, bem como o modo que pretende utilizar (R ou RS − modo síncrono; RA − modo assíncrono);

  - **Escrita de itens no servidor:** o utilizador requisita a escrita de itens no servidor (estado de saídas, contadores, registos, entre outros). Nesta opção, o utilizador insere o caminho dos itens que pretende escrever e o seu valor, bem como o modo que pretende utilizar (W ou WS − modo síncrono; WA − modo assíncrono);

  - **Monitorização de itens no servidor:** o utilizador irá receber notificações de eventos (*subscriptions*) correspondentes a mudanças de estado dos itens e o seu valor atual (M ou MS − modo síncrono; MA − modo assíncrono);

- **Remoção de itens da monitorização:** remove o item inserido pelo utilizador da monitorização atual (MR ou MRS − modo síncrono; MRA − modo assíncrono);

- **Cancelar a monitorização:** cancela a monitorização atual (síncrona ou assíncrona).

Na Tabela 7 são apresentados os diferentes comandos e possíveis combinações a serem recebidos pelo serviço para a execução de cada uma das funcionalidades. Na listagem de itens, o utilizador ao inserir apenas o comando `L` irá requisitar um pedido de listagem de todos os itens (expansíveis ou não) da raiz do servidor. O utilizador para requisitar os itens contidos num determinado caminho terá que enviar o comando `L <caminho>`. Como os itens/caminhos podem conter espaços entre as palavras, o utilizador deverá introduzir aspas no caminho desejado `L "<caminho>"`.

Para realizar a leitura de itens terá que enviar para o serviço o comando `R <caminho + nome do item>`. Para a leitura de múltiplas variáveis, como, por exemplo, dos itens "I1" e "I2" contidos em IO ► inputs, o serviço deverá receber o comando `R <caminho + nome do item> <quantidade>` (`R IO.inputs.I1 2`), válido também para os comandos `RS` e `RA`. O comando de escrita de itens funciona de forma similar com a adição do valor a ser escrito `W <caminho + nome do item> <valor> <quantidade>`.

A monitorização de variáveis é realizada por um vector que contém as variáveis a serem monitorizadas. Sempre que um item do vector altere o seu estado no servidor OPC, o serviço recebe essa notificação, bem como o seu valor/estado atual. Para acrescentar itens à monitorização é realizado pelo comando `M <caminho + nome do item> <quantidade>`. Caso o utilizador necessite de retirar um dos itens da monitorização, deverá de enviar o comando `MR <caminho + nome do item>`. Este comando apenas remove um item de cada vez. Para cancelar a monitorização atual deverá enviar o comando `MQ`.

Tabela 7 Funcionalidades do serviço OPC-DA

Funcionalidade	Comandos	Algumas combinações para os comandos	Exemplos de comandos
Listagem de servidores OPC-DA 2.0	INFO	INFO <IP do Dispositivo>	INFO 192.168.10.1
Ligação a um determinado servidor OPC-DA	opcda	opcda://<IP do Dispositivo>/<Nome do Servidor OPC>/{<ClassID>}	opcda://192.168.10.1/ServerDA.1 /{2E542-B238-11D3-842D-0008C77}
Listagem dos itens do servidor	L	L L <caminho> L "<caminho>"	L L IO.inputs L "IO.all inputs"
Listagem de informações do servidor	S	S	S
Leitura de itens no servidor	R RS RA	R <item> R <item> <quantidade> R "<item>" RS <item> RA <item> <quantidade>	R IO.input.I1 R IO.input.I1 2 R "IO.all inputs.I1" RS IO.input.I1 RA IO.input.I1 2
Escrita de itens no servidor	W WS WA	W <item> <valor> W <item> <valor> <quantidade> W "<item>" <valor> W <item> "<texto>" WS <item> <valor> WA <item> <valor> <quantidade>	W IO.input.I1 1 W IO.input.I1 1 2 W "IO.all inputs.I1" 1 W IO.words.MW1 "DA" WS IO.input.I1 1 WA IO.input.I1 1 2
Monitorização de itens no servidor	M MS MA	M <item> M <item> <quantidade> M "<item>" MS <item> MA <item> <quantidade>	M IO.input.I1 M IO.input.Q1 2 M "IO.all inputs.I2" MS IO.input.I3 MA IO.input.I1 2
Remoção de itens da monitorização	MR MRS MRA	MR <item> MRS "<item>" MRA <item>	MR IO.input.I1 MRS "IO.all inputs.I2" MRA IO.input.I2
Cancelar a monitorização	MQ	MQ	MQ

Para a implementação dos pedidos de leitura, escrita e monitorização na versão 2.0 do OPC-DA é necessário criar uma sessão, uma subscrição e inicializar os itens a serem requisitados (Figura 33). O serviço foi desenvolvido de maneira a realizar os pedidos de leitura diretamente ao dispositivo e não à cache, visto que se pretende adquirir dados em tempo real. No Anexo E encontram-se todos os métodos, classes e enumerações usados pelo serviço.

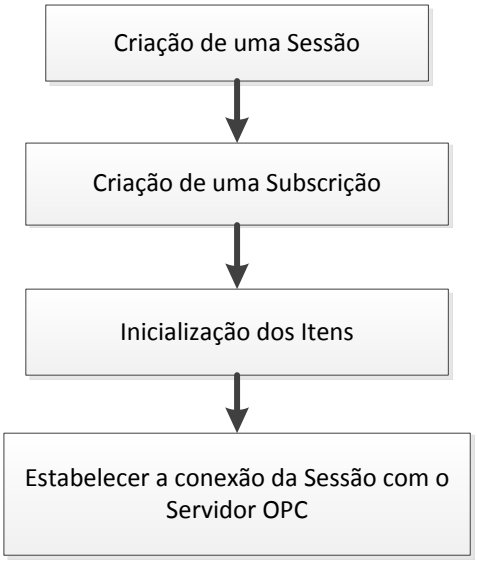


Figura 33 Estabelecimento de uma ligação com o servidor OPC

Na Figura 34 apresenta-se o código referente à leitura síncrona do servidor OPC, onde *SERVER* é o URL do servidor OPC, *MAX_AGE* é o tempo para a atualização da cache (como se pretende ler diretamente do dispositivo, o valor será "0"), *NUMBER_OF_ITENS* é a quantidade de itens a ler e *ITEM_NAME* é referente ao item a ler.

Quando é executado um comando de leitura (Figura 35), o servidor OPC retorna o valor atual do item requisitado (*value*), a qualidade à qual o item foi adquirido (*quality*) e o *time stamp*. O comando de leitura `subscription->read(MAX_AGE, itensToProcess, valuesToProcess, processResult, MODE)))` necessita dos seguintes parâmetros:

- MAX_AGE − tempo dos itens serem atualizados na cache (caso o valor seja 0, a leitura é feita diretamente ao dispositivo);
- itensToProcess − vetor com os itens a serem lidos;
- valuesToProcess − onde a informação retornada da leitura será guardada;
- processResult − retorna se a leitura foi realizada com sucesso;
- MODE – Modo síncrono ou assíncrono.

```
// Criação da sessão e subscrição (session e subscription)
MyDaSession* session = new MyDaSession(_T(SERVER));
MyDaSubscription* subscription = new MyDaSubscription(MAX_AGE, session);
std::vector<DaItem*> itensToProcess;
std::vector<ValueQT*> valuesToProcess;
std::vector<long> processResult;

itensToProcess.assign(NUMBER_OF_ITENS, NULL);
valuesToProcess.assign(NUMBER_OF_ITENS, NULL);
processResult.assign(NUMBER_OF_ITENS, E_FAIL);
// Inicialização dos itens (Preenchimento do vetor itensToProcess)
for (unsigned int i = 0; i < NUMBER_OF_ITENS; i++)
  itensToProcess[i] = new MyDaItem(_T(ITEM_NAME), subscription);
// Estabelece a ligação da sessão com o servidor OPC
session->connect(TRUE, FALSE, NULL);
```

Figura 34 Código referente ao estabelecimento de uma ligação com o servidor

```
// Comando para leitura síncrona dos itens contidos em itensToProcess
long result = subscription->read(MAX_AGE, itensToProcess, valuesToProcess,
                                 processResult, NULL)))
if(SUCCEEDED(result)) {
  for (unsigned int i = 0; i < NUMBER_OF_ITENS; i++) {
       if (SUCCEEDED(processResult[i])) {
            // O valor do item itensToProcess[i] foi lido com sucesso
            // Informação contida em valuesToProcess[i]
            std::string text = "";
           text = std::string(READ_OK);
           text += std::string(OpcClient.itensToProcess[i]->getId().c_str());
           text += std::string(";");
           text += std::string(const_cast<Variant&>(
                   (OpcClient.valuesToProcess[i])->getData()).toString());
           text += std::string(";");
           text += std::to_string((long long)(
                   (OpcClient.valuesToProcess[i])->getQuality()));
           text += std::string(";");
           text += std::string(
                   ((OpcClient.valuesToProcess[i])->getTimeStamp()).toString());
       }
       else
            // Não foi possível ler o valor itensToProcess[i]
   } // end for
} // end if
else
   // Não foi possível ler os valores requisitados
```

Figura 35 Código referente à leitura síncrona

Para executar um comando de escrita é necessário fornecer ao servidor o valor a escrever, sendo o *quality* e o *time stamp* irrelevantes para a escrita. Nesta situação, normalmente é colocado o valor da qualidade a 65535 (*QUALITY_NOT_SET*). Para realizar uma monitorização de dados, apenas é necessário ativar a subscrição depois do estabelecimento da ligação da sessão com o servidor OPC: `subscriptions->connect(TRUE, TRUE, NULL)`. No Anexo E encontram-se os métodos usados pelas classes necessários para os pedidos OPC.

Na Tabela 8 e na Tabela 9 são apresentadas as mensagens retornadas pelo serviço Windows para os utilizadores conectados a ele. A razão pela qual se optar por códigos diferentes de sucesso de mensagens é, devido ao facto do serviço Windows suportar múltiplos pedidos. Por exemplo, se o utilizador inserir três tipos de comandos diferentes:

- R IO.input.I1
- W IO.output.Q1
- MQ

e por alguma razão, a resposta de cada um dos comandos tiver algum atraso, então o resultado será a recepção de diversos códigos de sucesso para cada um dos comandos. Nestas situações, se os códigos de sucesso fossem iguais para todos os comandos, seria difícil a identificação de cada uma das respostas recebidas para os comandos efectuados.

Para a realização de testes do serviço Windows foi desenvolvido um programa simples para comunicar com este. O programa de testes verifica, num ficheiro de inicialização, qual o IP e o número da porta associados à máquina Windows onde se encontra o serviço, e estabelece uma ligação com o serviço. O programa encontra-se dividido em duas *threads*: uma será responsável pela recepção dos dados recebidos do serviço Windows, e pela apresentação dos resultados na consola; a outra será responsável pela leitura dos comandos de teste num ficheiro de texto e pelo envio desses mesmos comandos para o serviço Windows.

Tabela 8 Códigos retornados dos comandos antes da ligação com o servidor OPC-DA

Primeiro MENU		
Código Retornado	**Descrição**	**Exemplo de comando**
0	Cliente terminou ligação com o serviço com sucesso.	Q
1	URL do servidor detetado com sucesso. O serviço passa para o **Segundo Menu.**	opcda://1.1.1.1/ServerDA.1/{2EC-779D-775}
2;<IP>	Comando info realizado com sucesso.	Info 1.1.1.1
3;<servidor>	Retorna URL do servidor OPC-DA.	Info 1.1.1.1
6	Faltam barras no URL do servidor (o comando necessita de 4 barras no total).	opcda:/~~/~~1.1.1.1~~/~~ServerDA.1/{2EC-779D-775}
7	Faltam chavetas no "ClassID" (2 chavetas).	opcda://1.1.1.1/ServerDA.1/~~{~~2EC-779D-775~~}~~
8	Falta 1 barra a seguir a "opcda:/".	opcda:/~~/~~1.1.1.1/ServerDA.1/{2EC-779D-775}
9	O endereço IPv4 inserido está incorreto.	opcda://1.1.1~~.1~~/ServerDA.1/{2EC-779D-775} info 1.1.1~~.1~~
10	As chavetas no comando opcda não se encontram no campo ClasseID.	opcda://1.1.1.1/{ServerDA.1}/2EC-779D-775
16;<IP>	O dispositivo não contém servidores DA 2.0.	Info 1.1.1.1
17;<IP>	Não se conseguiu conectar ao dispositivo.	Info 1.1.1.1
55	Não conseguiu detetar o servidor.	opcda://1.1.1.1/ServerDA.1/{2EC-779D-775}
60	Comando inserido inválido.	xpto
61	Não foi possível criar thread para processar o comando info.	Info 1.1.1.1

Tabela 9 Códigos retornados dos comandos após a ligação com o servidor OPC-DA

Segundo MENU		
Código Retornado	**Descrição**	**Exemplo de comando**
0	Retornou com sucesso ao **Primeiro Menu.**	end
1	Conclusão da listagem de itens.	L
2;<nome>;<caminho>	Indica que o item é expansível (*branch*), o nome do item e o caminho+nome do item.	L
3;<nome>;<caminho>	Indica que o item não é expansível (*leaf*), o nome do item e o caminho+nome do item.	L
6;<item>;<valor>; <qualidade>;<data> <hora>	Indica que a leitura síncrona foi feita com sucesso, o caminho+nome do item, valor, qualidade, data+hora.	Rs IO.input.I1
7;<item>	Não conseguiu realizar a leitura síncrona ao "caminho+item" do servidor.	Rs IO.input.I1
8	Não conseguiu realizar a leitura síncrona aos itens do servidor.	Rs IO.input.I1
11;<item>;<valor>; <qualidade>;<data> <hora>	Indica que a leitura assíncrona foi feita com sucesso, o caminho+nome do item, valor, qualidade, data+hora.	Ra IO.input.I1
12;<item>	Não conseguiu realizar a leitura assíncrona ao "caminho+item" do servidor.	Ra IO.input.I1
13	Não conseguiu realizar a leitura assíncrona aos itens do servidor	Ra IO.input.I1
16;<item>;<valor>; <qualidade>;<data> <hora>	Indica que a escrita síncrona foi feita com sucesso, o caminho+nome do item, valor, qualidade, data+hora.	Ws IO.input.I1
17;<item>	Não conseguiu realizar a escrita síncrona ao "caminho+item" do servidor.	Ws IO.input.I1
18	Não conseguiu realizar a escrita síncrona aos itens do servidor.	Ws IO.input.I1
19	Não conseguiu realizar escrita síncrona. Tipo do valor a escrever incompatível.	Ws IO.input.I1
21;<item>	Indica que a escrita assíncrona foi feita com sucesso, o caminho+nome do item.	Wa IO.input.I1
22;<item>	Não conseguiu realizar a escrita assíncrona ao "caminho+item" do servidor.	Wa IO.input.I1
23	Não conseguiu realizar a escrita assíncrona aos itens do servidor.	Wa IO.input.I1
24	Não conseguiu realizar escrita assíncrona. Tipo do valor a escrever incompatível.	Wa IO.input.I1
25	Utilizador não inseriu valor a escrever.	Ws IO.input.I1 Wa IO.input.I1
28;<item>;<valor>; <qualidade>;<data> <hora>	O item modificou o estado (monitorização).	M IO.input.I1 Ma IO.input.I1
32	A monitorização foi cancelada.	MQ
35;<item>	Removeu o item (caminho+nome) com sucesso da monitorização.	Mrs IO.input.I1 Mra IO.input.I1

36;<item>	Não conseguiu remover o item (caminho+nome) da monitorização.	Mrs IO.input.I1 Mra IO.input.I1
40;<estado>	Indica o estado do servidor (1-ligado ou 0-desligado).	S
41;<informação>	Informação sobre o fabricante do servidor OPC.	S
42;<versão>	Informação sobre a versão do servidor.	S
43;<data> <hora>	Indica quando (data e hora) o servidor foi inicializado.	S
44;<data> <hora>	Indica o tempo (data e hora) da última transmissão de dados para o cliente.	S
45;< data> <hora>	Indica a data e a hora atual do servidor.	S
46;<n° de subscrições>	Número total de subscrições que foram criadas no servidor.	S
47;<desempenho>	Desempenho do processador do servidor.	S
48;<linguagem>	Lista de linguagens suportadas pelo servidor.	S
49;<estado>	Descrição textual do estado atual do servidor.	S
50	Pedido de informações sobre o servidor falhou.	S
60	Comando inserido inválido.	xpto
61	Não foi possível criar thread para processar o comando.	(para qualquer um dos pedidos)

A Figura 37 ilustra um exemplo de dois tipos de retorno para um pedido de listagem de servidores OPC-DA 2.0 através do comando "info" (Figura 36). Quando o programa de testes se conecta ao serviço Windows, imprime a mensagem "*Succefully connected*". Na primeira situação, quando se digita o comando `info <IP>` obteve-se a mensagem `17;<IP>`, isto significa que não se conseguiu conectar ao dispositivo com o endereço `<IP>`. Na segunda situação, conectou-se ao dispositivo (para efeitos de simulação, instalaram-se servidores OPC-DA 2.0 numa máquina Windows) e retornou todos os servidores OPC-DA que encontrou `3;<opcda>`, retornando no final uma mensagem de sucesso para o endereço IP `2;<IP>`. Os restantes testes e respetivos resultados encontram-se no Anexo F.

```
info 192.168.122.191
info 192.168.122.193
```

Figura 36 Teste para comando "info"

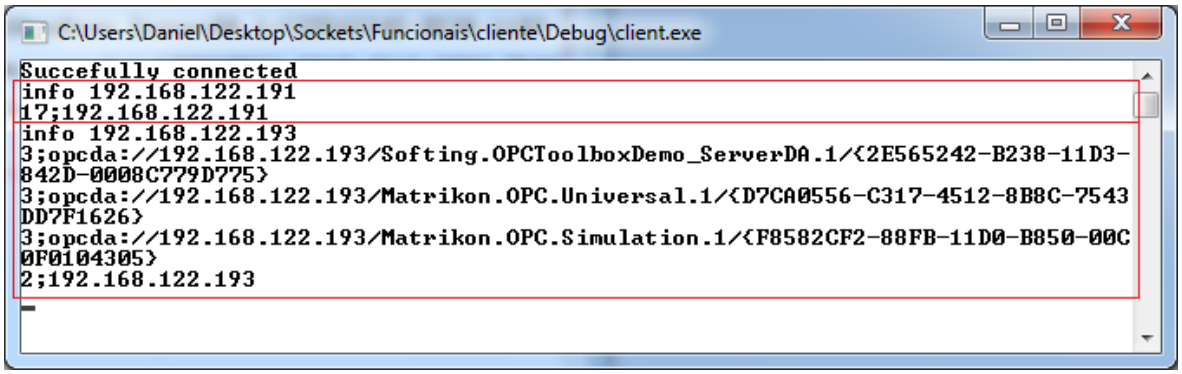


Figura 37 Retorno do comando "info"

4.3. SERVIÇO WINDOWS DO CLIENTE OPC-AE

O serviço Windows desenvolvido para a implementação do cliente OPC-AE contém alguns dos mecanismos desenvolvidos para o OPC-DA e o modo de funcionamento é de certa forma semelhante. Nomeadamente, os mecanismos de múltiplos clientes e múltiplos pedidos, ficheiros criados na inicialização, entre outros. O único valor que será modificado do ficheiro de inicialização será o da porta usada pelo serviço.

Para o desenvolvimento do serviço Windows foi usado o Microsoft Visual Studio 2010, com a linguagem em C++ do SDK AE da Softing. Para auxiliar o desenvolvimento do cliente OPC-AE foi usado o simulador de servidor OPC-AE disponibilizado pela Softing.

Como no serviço OPC-DA, recebe os pedidos por parte dos utilizadores, processando-os de forma a implementar o cliente OPC-AE. A seguir são apresentadas as funcionalidades implementadas no serviço:

- **Listagem de servidores OPC-AE 1.0 num dispositivo** − serão retornados os URL dos servidores OPC-AE 1.0 contidos no dispositivo selecionado;

- **Ligação a um determinado servidor OPC-AE de um dispositivo** − o URL do servidor inserido pelo utilizador será guardado para a criação de sessões com o servidor para requisitar os seguintes pedidos:

- **Listagem dos itens contidos no servidor:** serão retornados os itens contidos num determinado caminho inserido pelo utilizador (*Event Area*). De mencionar, que as áreas normalmente são elementos expansíveis;

- **Listagem do esquema de eventos no servidor:** será retornado ao utilizador, a listagem do esquema de todos os eventos definidos no servidor (*Filter Space*). De mencionar, que normalmente as categorias são elementos expansíveis;

- **Listagem de informações sobre o servidor:** será retornado ao utilizador, as informações relativas ao servidor. As informações são referentes ao estado, descrição, informação do vendedor, versão do produto, tempos de inicialização, tempo atual, entre outros, do servidor.

- **Listagem de condições associadas a um item:** será retornado todas as condições associadas ao item requisitado.

- **Monitorização de todos os eventos no servidor:** sempre que é gerado um evento (*simple, tracking, condition*) o serviço irá receber a informação relativa a esse evento e enviá-la ao utilizador;

- **Monitorização de eventos com aplicação de filtros:** permite ao utilizador definir filtros para a monitorização dos eventos;

- **Remoção de elementos dos filtros de monitorização:** remove um determinado elemento do filtro (categoria, área, fonte) da monitorização atual;

- **Cancelar a monitorização:** cancela a monitorização atual.

Na Tabela 10 são apresentados os diferentes comandos e possíveis combinações a serem recebidos pelo serviço para a execução de cada uma das funcionalidades. Na listagem de itens, o utilizador ao inserir apenas o comando `L` irá requisitar um pedido de listagem de todos os itens (expansíveis ou não) da raiz do servidor. O utilizador para requisitar os itens contidos numa determinada área terá que enviar o comando `L <caminho>`.

Como os itens/áreas podem conter espaços entre as palavras, o utilizador deverá introduzir aspas para o caminho desejado `L "<caminho>"`.

A monitorização de eventos com aplicação de filtros permite adicionar novos filtros à monitorização atual. Para isso o utilizador terá que enviar um novo comando `M` com o filtro desejado (área, categoria, tipo de evento, severidade mínima, severidade máxima ou fonte). No filtro do tipo de evento (filtro número 3), o utilizador seleciona quais os eventos que deseja receber (`s` → simples, `t` → *tracking*, `c` → condição).

Tabela 10 Funcionalidades do serviço OPC-AE

Funcionalidade	Comandos	Algumas combinações para os comandos	Exemplos de comandos
Listagem de servidores OPC-AE 1.0	INFO	INFO <IP do Dispositivo>	INFO 192.168.10.1
Ligação a um determinado servidor OPC-AE	opcae	opcae://<IP do Dispositivo>/<Nome do Servidor OPC>/{<ClassID>}	opcae://192.168.10.1/ServerAE.1/{2E542-B238-11D3-842D-008C7}
Listagem dos itens do servidor	L	L L <caminho> L "<caminho>"	L L Producao.plc1 L "Producao.plc 1"
Listagem do esquema de eventos do servidor	LS	LS	LS
Listagem de informações do servidor	S	S	S
Listagem de condições associadas a um item	C	C <item> C "<item>"	C Producao.plc1.I1 C "Producao.plc 1.I1"
Monitorização de todos os eventos do servidor	E	E	E
Monitorização de eventos com a aplicação de filtros	M	M 1 <area> M 1 "<area>" M 2 <categoria> M 3 <tipo de evento> M 4 <severidade mínima> M 5 <severidade máxima> M 6 <fonte>	M 1 Producao.plc1 M 1 "Producao.plc 1" M 2 9 M 3 sct M 4 10 M 5 100 M 6 Producao.plc1.I1
Remoção de elementos dos filtros da monitorização	MR	MR 1 <area> MR 2 <categoria> MR 6 <fonte>	MR 1 Producao.plc1 MR 2 9 MR 6 Producao.plc1.I1
Cancelar a monitorização	MQ	MQ	MQ

Para realizar uma monitorização de eventos com a aplicação de filtros é necessário primeiro a criação de uma sessão e de uma subscrição, estabelecer a ligação com o servidor, atribuir filtros à subscrição e por fim ativá-la (Figura 38). No Anexo G encontram-se todos os métodos, classes e enumerações usados pelo serviço. Na Tabela 11 e na Tabela 12 são apresentadas as mensagens retornadas pelo serviço Windows para os utilizadores conectados a ele.

Figura 38 Passos necessários para a monitorização de eventos com aplicação de filtros

Tabela 11 Códigos retornados dos comandos antes da ligação com o servidor OPC-AE

Primeiro MENU		
Código Retornado	**Descrição**	**Exemplo de comando**
0	Cliente terminou ligação com o serviço com sucesso.	Q
1	URL do servidor detetado com sucesso. O serviço passa para o Segundo Menu.	opcae://1.1.1.1/ServerAE.1/{2EC-779D-775}
2;<IP>	Comando info realizado com sucesso.	Info 1.1.1.1
3;<servidor>	Retorna URL do servidor OPC-AE.	Info 1.1.1.1
6	Faltam barras URL do servidor (o comando necessita de 4 barras no total).	opcae:~~/~~1.1.1.1~~/~~ServerAE.1/{2EC-779D-775}
7	Faltam chavetas no "ClassID" (2 chavetas).	opcae://1.1.1.1/ServerAE.1/~~{~~2EC-779D-775~~}~~
8	Falta 1 barra a seguir a "opcae:/".	opcae:~~/~~1.1.1.1/ServerAE.1/{2EC-779D-775}
9	O endereço IPv4 inserido está incorreto.	opcae://1.1.1~~.1~~/ServerAE.1/{2EC-779D-775} info 1.1.1~~.1~~
10	As chavetas no comando opcae não se encontram no campo ClasseID.	opcae://1.1.1.1/~~{~~ServerAE.1~~}~~/2EC-779D-775
16;<IP>	O dispositivo não contém servidores AE.	Info 1.1.1.1
17;<IP>	Não se conseguiu conectar ao dispositivo.	Info 1.1.1.1
55	Não conseguiu detetar o servidor.	opcae://1.1.1.1/ServerAE.1/{2EC-779D-775}
60	Comando inserido inválido.	xpto
61	Não foi possível criar thread para processar o comando info.	Info 1.1.1.1

Tabela 12 Códigos retornados dos comandos após a ligação com o servidor OPC-AE

Segundo MENU		
Código Retornado	**Descrição**	**Exemplo de comando**
0	Retornou com sucesso ao Primeiro Menu.	end
1	Conclusão da listagem de itens.	L
2;<nome>;<caminho>	Indica que o elemento retornado é expansível (normalmente trata-se de uma área), o nome do elemento e o caminho+nome do elemento.	L
3;<nome>;<caminho>	Indica que o elemento não é expansível (normalmente trata-se de um item/fonte), o nome do elemento e o caminho+nome do elemento.	L
4;<nome>;<condição>	Indica que o item contém uma condição, o nome do item e a condição associada a esse item.	L
8	A monitorização foi cancelada.	MQ
10;<estado>	Indica o estado do servidor (1-ligado ou 0-desligado).	S
11;<info>	Informação sobre o fabricante do servidor OPC.	S
12;<versão>	Informação sobre a versão do servidor.	S
13;<data> <hora>	Indica quando (data e a hora) o servidor foi inicializado.	S
14;<data> <hora>	Indica o tempo (data e hora) da última transmissão de dados para o cliente.	S
15;< data> <hora>	Indica a data e a hora atual do servidor.	S
16;<nº de subscrições>	Número total de subscrições que foram criadas no servidor.	S
17;<desempenho>	Desempenho do processador do servidor.	S
18;<linguagem>	Lista de linguagens suportadas pelo servidor.	S
19;<estado>	Descrição textual do estado atual do servidor.	S
20	Pedido de informações sobre o servidor falhou.	S
25;<item>;<condição>	Indica que o item (caminho+nome) contém uma condição.	C IO.inputs.I1
26;<item>	Indica que o item (caminho+nome) não contém nenhuma condição.	C IO.inputs.I1
30;<fonte>	Indica o item/fonte (caminho+nome) ao qual ocorreu o evento.	M 3 sct E
31;<data> <hora>	Indica a data e a hora em que ocorreu o evento.	M 3 sct E
32;<tipo de evento>	Tipo do evento (CONDITION, SIMPLE, TRACKING).	M 3 sct E
33;<categoria>	Categoria a que pertence o evento (valor inteiro).	M 3 sct E
34;<severidade>	Indica o nível de severidade (gravidade) do evento.	M 3 sct E

35;<mensagem>	Mensagem descritiva do evento.	M 3 sct E
36;<ID Actor>	Identificação do utilizador (caso seja atribuído um).	M 3 sct E
37;<condição>	Nome da condição do evento (eventos CONDITION).	M 3 sct E
38;<subcondição>	Nome da subcondição do evento (eventos CONDITION).	M 3 sct E
40;<ACT> <ENA> <ACK> <DIS>	Estado do evento: ativo (ACT), *enable* (ENA), se necessita de reconhecimento (ACK), desconectado (DIS).	M 3 sct E
41;<qualidade>	Qualidade do evento CONDITION (Anexo C).	M 3 sct E
42;<reconhecimento>	Indica se é necessário o reconhecimento (acknowledge) do evento (valor inteiro: 1-sim ; 0-não).	M 3 sct E
43;<data> <hora>	Indica a data e a hora, em que o servidor permaneceu ativo.	M 3 sct E
50	Listagem do esquema de eventos completa.	LS
51;<categoria>	Categoria do evento (em texto).	LS
52;<atributo>	Atributos da categoria do evento (em texto).	LS
53;<condição>	Condições do atributo (em texto).	LS
56	Filtro inválido (por ex: valor do filtro "8" não existe).	M 8 10
57;<filtro>	Não conseguiu remover/encontrar o elemento (área, categoria, item) do filtro.	MR 6 IO.input.I1
58;<filtro>	Removeu com sucesso o elemento (área, categoria, item) do filtro.	MR 6 IO.input.I1
60	Comando inserido inválido.	xpto
61	Não foi possível criar thread para processar o comando.	(para qualquer um dos pedidos)

Para a realização de testes foi usado o mesmo programa que foi criado para testar o serviço OPC-DA. No ficheiro de inicialização apenas se altera a porta de comunicação. A Figura 40 demonstra um exemplo de dois tipos de retorno para um pedido de listagem de servidores OPC-AE 1.0 através do comando "info" (Figura 39). Os restantes testes e respetivos resultados encontram-se no Anexo H.

```
info 192.168.122.191
info 192.168.122.193
```

Figura 39 Teste para comando "info" do serviço AE

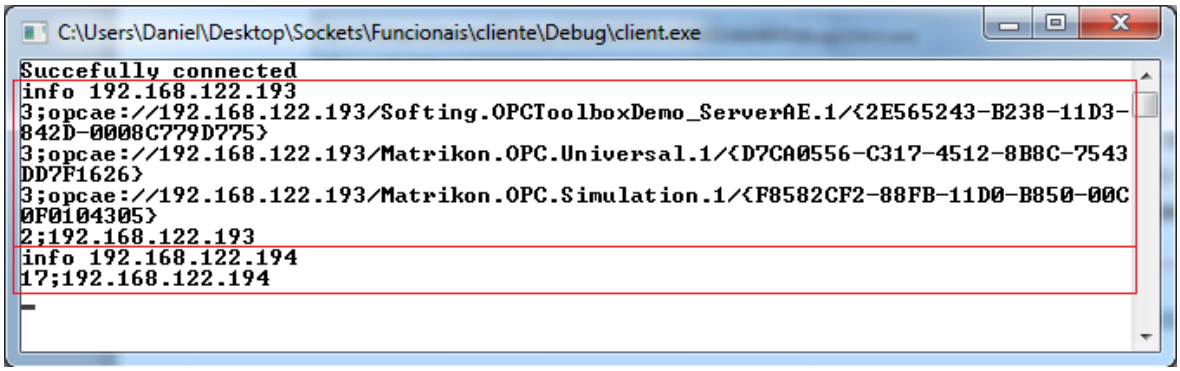


Figura 40 Retorno do comando "info" do serviço AE

4.4. MESTRE IEC 60870-5-104

O mestre IEC 60870-5-104 foi desenvolvido com o intuito de ser instalado no sistema de supervisão (Linux) tendo sido utilizada a biblioteca MRTS-NG [35]. Esta biblioteca é livre, usa a linguagem em C e é compatível com sistemas Linux. Disponibiliza funções para o estabelecimento de ligação com os escravos, inicialização de troca de dados, terminação de ligação com os escravos, entre outras. Para a realização de testes foi usado o simulador "DemoWinPP104" [36], que permite simular um escravo IEC 60870-5-104 (não implementa todas as situações existentes do protocolo).

O mestre IEC 60870-5-104 foi desenvolvido de modo a imprimir no *stdout* a informação proveniente dos escravos e imprimir no *stderr* os comandos de sucesso ou erro. Desta forma, será possível ao sistema de supervisão e de controlo atualmente implementado pela EFACEC retirar a informação gerada pelo *software* mestre desenvolvido. Para comandos de sucesso e erros foram definidos os seguintes:

0 – sucesso;

1 – argumentos de entrada incorretos;

2 – erro na ligação com o escravo;

3 – timeout ou resposta recebida incorreta.

O tipo de pedido a realizar pelo mestre é definido pelos comandos do campo "identificador de tipo" da trama IEC 60870-5-104. Como comandos de leitura optou-se por

implementar os comandos listados na Tabela 13 e como comandos de escrita os listados na Tabela 14.

As respostas esperadas aos pedidos efetuados encontram-se representadas na Tabela 15. Estas respostas serão tratadas pelo mestre, que implementa mecanismos de conversão de números negativos, conversão dos "elementos de informação" de dois ou mais bytes para um número decimal, entre outros. As restantes respostas implementadas pelo protocolo e que não se encontram listadas na tabela serão impressas sem qualquer tipo de formatação, por outras palavras, serão impressos todos os bytes do(s) o(s) campo(s) "Objeto de informação".

Para cada comando e para cada resposta é associada uma causa de transmissão. A causa de transmissão é usada para identificar o motivo da transmissão de dados (por exemplo: confirmação de uma ativação). Na Tabela 16 encontram-se listadas algumas das causas de transmissão definidas pelo protocolo.

Tabela 13 Identificadores de tipo para leitura de dados

Código	Comando	Descrição
100	Comando de interrogação	Pedido de dados

Tabela 14 Identificadores de tipo para escrita de dados

Código	Comando	Descrição
45	Comando simples	Escrita de 1 bit (0- OFF; 1- ON).
46	Comando duplo	Escrita de 2 bits (1- OFF; 2- ON; 0,3- Não usados).
58	Comando simples (CP56Time2a)	Comando simples com *tag* de tempo (data e hora).
59	Comando duplo (CP56Time2a)	Comando duplo com *tag* de tempo (data e hora).

Tabela 15 Identificadores de tipo para as respostas aos pedidos de dados

Código	Resposta	Código do Pedido
1	Informação de um ponto simples	45 ou 100
3	Informação de um ponto duplo	46 ou 100
5	Informação da posição	100
7	*Bit string* - 32 bits	100
9	Valor medido, valor normalizado	100
11	Valor medido, valor em escala	100
30	Informação de um ponto simples (CP56Time2a)	58 ou 100
31	Informação de um ponto duplo (CP56Time2a)	59 ou 100
32	Informação da posição (CP56Time2a)	100
33	*Bit string* - 32 bits (CP56Time2a)	100
34	Valor medido, valor normalizado (CP56Time2a)	100
35	Valor medido, valor em escala (CP56Time2a)	100

Tabela 16 Algumas das causas de transmissão

Código	Causa da Transmissão	Código	Causa da Transmissão
1	Periódica, ciclica	11	*feedback*, causado por comando à distância
2	*Background interrogation*	12	*feedback*, causado por comando local
3	Espontânea	13	Transmissão de dados
4	Inicializado	20	Interrogado por interrogação geral
5	Interrogação ou interrogado	21	Interrogadopor interrogação do grupo 1
6	Ativação	22	Interrogadopor interrogação do grupo 2
7	Confirmação da ativação	44	identificador de tipo desconhecido
8	Desativação	45	causa desconhecida
9	Confirmação da desativação	46	Endereço de ASDU desconhecido
10	Terminação da ativação	47	Endereço do objeto de informação desconhecido

O conteúdo das respostas recebidas encontra-se no interior do(s) campo(s) “Objeto de Informação” da trama, ou seja, os dados recebidos serão os “elementos da informação”. O protocolo IEC 60870-5-104 define ainda descritores de qualidade para as mensagens recebidas que é representado pelo formato QDS (*Quality Descriptor*), como demonstra a Tabela 17. Os descritores de qualidade são os seguintes:

- **OV (*No Overflow / Overflow*)** – O valor do objeto de informação excede os limites predefinidos (1) ou não (0).
- **BL (*Not Blocked / Blocked*)** − O valor do objeto de informação encontra-se bloqueado (1) ou não (0) para a transmissão. Caso, se encontre bloqueado, o valor não é alterado.
- **SB (*Not Substituted / Substituted*)** − O valor do objeto de informação é proveniente do utilizador ou de uma fonte automática.
- **NT (*Topical / Not Topical*)** – O valor é "*topical*" se foi atualizado com sucesso (0) ou "*not topical*" caso não seja atualizado com sucesso durante um intervalo de tempo especificado ou não se encontre disponível (1).
- **IV (*Valid / Invalid*)** – O valor é válido se foi adquirido corretamente (0). Para situações em que o valor, por alguma razão, seja adquirido incorretamente, a *flag* IV será colocada a "1".

Tabela 17 Formato QDS

bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1
IV	NT	SB	BL	0	0	0	OV

O mestre IEC 60870-5-104 terá que receber os seguintes parâmetros de entrada (Figura 41):

- IP do escravo IEC 60870-5-104;
- Comando pretendido (leitura "r" ou escrita "w");
- Endereço comum da ASDU;
- Endereço do objeto de informação;
- Valor a escrever (apenas para comandos de escrita);
- Tipo de comando de escrita (identificador de tipo para o caso de escrita).

```
<Ip do dispositivo> <r/w> <Endereço comum ASDU> <endereço do objeto de
   informação> <(w) Valor a escrever> <(w) comando 45, 46, 58, 59>
```

Figura 41 Argumentos de entrada

Depois de enviar os dados (trama) ao escravo, aguarda por uma ou mais respostas provenientes do escravo, com o seguinte formato: `<identificador de tipo>;<SQ + número de objetos>;<T + P/N + causa de transmissão>;<endereço do objeto de informação>;<elementos de informação>.`

Dependendo do valor recebido do parâmetro SQ, a mensagem pode consistir num ou mais "objetos de informação" (`<endereço do objeto de informação 1>;<elementos de informação 1>;<endereço do objeto de informação n>;<elementos de informação n>`), ou num ou mais conjuntos de "elementos de informação" (`<endereço do objeto de informação>;<elementos de informação 1>;<elementos de informação n>`). Apenas foi possível testar a recepção de um único "objeto de informação", com um SQ = "0".

Na Figura 42 apresenta-se um exemplo de respostas a um "comando simples" para a escrita de dados, gerado pelo simulador escravo (Figura 43). No lado do simulador (escravo) foi parametrizado uma resposta automática para um pedido de "comando simples". No simulador foi definido que sempre que receber um pedido da estação "2" (endereço comum da ASDU), com um endereço do objeto de informação "10", com um descritor de qualidade "0" e um pedido de "comando simples" (*R. Typ* define o código de resposta, em que "1" é uma resposta ao comando "45"), irá gerar uma resposta automática de confirmação ao pedido realizado (ativação ou desativação) e uma resposta com a informação atual do escravo.

Figura 42 Exemplo de respostas a um "comando simples"

Ino.	Station	Object address	QU	Con	Ret	Term	R.Object address	R.Typ
1	2	10	0	☑	☑	☐	10	1

Figura 43 Parâmetros usados no simulador do escravo IEC 60870-5-104

Para realizar o pedido de escrita da Figura 42, foi enviado a ASDU ilustrada na Tabela 18 para o escravo. No campo dos elementos da informação é definido um octeto (Tabela 19) para o pedido de dados, que contém a seguinte informação:

- **S/E (*Select* / *Execute*)** – Para executar o pedido de escrita (0-Executar; 1-Selecionar).
- **CT (*Command Type*)** – Define o tipo de ativação:
  - ✓ 0 – não contém definição adicional;
  - ✓ 1 – impulso de curta duração;
  - ✓ 2 – impulso de longa duração;
  - ✓ 3 – saída constante;
  - ✓ 4 a 31 – reservados.
- **Estado** – Valor a escrever:
  - ✓ 0 – OFF;
  - ✓ 1 – ON.

Tabela 18 Exemplo de uma ASDU para um "comando simples"

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
Identificador de unidade de dados	45								0x2D
	0	1							0x01
	0	0	6						0x06
	0								0x00
	2								0x02
	0								0x00
Informação do objecto 1	10								0x0A
	0								0x00
	0								0x00
	1								0x01

Tabela 19 Elementos da informação de um "comando simples"

bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1
S/E	CT	CT	CT	CT	CT	CT	Estado

Após o envio do "comando simples" ao escravo, representado na Tabela 18, este retornou as respostas ilustradas na Figura 42. Pela análise da Figura 42, verifica-se que retornou uma mensagem de confirmação de ativação ao "comando simples" através do campo "causa de transmissão", com o estado "1" (ON). Depois, o simulador retornou uma segunda mensagem a informar que o endereço do objeto de informação foi colocado a "1" (Tabela 20).

Tabela 20 Elementos da informação da resposta a um "comando simples"

bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1
IV	NT	SB	BL	0	0	0	Estado

Os restantes testes e resultados para a escrita de dados encontram-se representados no Anexo I. O comando de interrogação, utilizado para a aquisição de dados, não foi possível testar. Contudo, foi implementado o tratamento de dados para os comandos apresentados na Tabela 15. Os restantes códigos serão impressos sem qualquer tipo de tratamento, ou seja, serão impressos todos os bytes do(s) o(s) campo(s) "Objeto de informação".

Para a resposta da "Informação da posição" a especificação prevê a recepção de dois octetos nos elementos de informação. O primeiro octeto retorna o estado atual do equipamento com uma gama de valores entre os -64 a 63. O segundo octeto retorna os descritores de qualidade QDS. O sistema desenvolvido trata de imprimir o valor em formato decimal (caso o valor seja negativo, é aplicado o complemento para 2, resultando num valor decimal de -64 a 64) e o octeto correspondente ao QDS: `<identificador de tipo>;<SQ + número de objetos>;<T + P/N + causa de transmissão>;<endereço do objeto de informação>;<valor de -64 a 64>;<QDS>`. Na Tabela 21 encontram-se as restantes respostas que são tratadas pelo mestre IEC 60870-5-104 desenvolvido.

Tabela 21 Elementos da informação tratados pelo mestre IEC 60870-5-104

Resposta	Octetos	Elementos de Informação
Informação da posição	Octeto 1	-64 a 63
	Octeto 2	QDS
Valor medido, valor normalizado	Octeto 1 (LSB)	-1 a $(1-2^{-15})$
	Octeto 2 (MSB)	
	Octeto 3	QDS
Valor medido, valor em escala	Octeto 1 (LSB)	-2^{15} a $(2^{15}-1)$
	Octeto 2 (MSB)	
	Octeto 3	QDS
Bit string - 32 bits	Octeto 1	Caracter 1
	Octeto 2	Caracter 2
	Octeto 3	Caracter 3
	Octeto 4	Caracter 4
	Octeto 5 (para CP56Time2a)	QDS

As respostas CP56Time2a contêm os mesmos "elementos de informação" que as outras respostas sem a *tag* de tempo CP56Time2a, com a diferença de receber mais essa *tag* de tempo (hora + data). A hora retornada será no formato `hh:mm:ss:ms` e a data no formato `DD:MM:YY`, para as respostas apresentadas na Tabela 15. Por exemplo, num comando simples CP56Time2a será recebido: `<identificador de tipo>;<SQ + número de objetos>;<T + P/N + causa de transmissão>;<endereço do objeto de informação>;<QDS + Estado>;<hh:mm:ss:ms>;<DD:MM:YY>`.

Nas restantes respostas serão impressos todos os bytes do(s) o(s) campo(s) "Objeto de informação". O sistema responsável por evocar o módulo desenvolvido terá que interpretar os dados recebidos. De mencionar que o "endereço do objeto" será impresso em três octetos e para identificadores do tipo CP56Time2a, a *tag* de tempo será impressa em sete octetos com o formato apresentado na Figura 44. O `R` significa que são bits reservados, o `S` representa o *summer time* e o `DS` representa os dias da semana (1-Segunda … 7-Domingo).

bit	8	7	6	5	4	3	2	1
octeto 1	ms (LSB)							
octeto 2	ms (MSB)							
octeto 3	IV	R	mm	mm	mm	mm	mm	mm
octeto 4	S	R	R	hh	hh	hh	hh	hh
octeto 5	DS	DS	DS	DD	DD	DD	DD	DD
octeto 6	R	R	R	R	MM	MM	MM	MM
octeto 7	R	YY	YY	YY	YY	YY	YY	YY

Figura 44 Octetos de uma *tag* CP56Time2a

5. CONCLUSÕES

A implementação de módulos protocolares em sistemas de supervisão e de controlo são fundamentais para permitir a comunicação com diversos dispositivos na rede, não ficando estes sistemas restritos à tecnologia implementada por cada um dos equipamentos. Nesta dissertação optou-se pela implementação de soluções baseadas em *fieldbus* e OPC.

A especificação OPC define um conjunto de objetos, interfaces e métodos a ser utilizados no controlo de processo e sistemas de automação de forma a facilitar a interligação entre plataformas que usam diferentes protocolos. A implementação dos módulos protocolares OPC-DA e OPC-AE permite aos sistemas de supervisão e de controlo comunicar com os autómatos por via OPC. Assim, através dos dois módulos desenvolvidos é possível recolher a informação e os eventos e alarmes gerados pelos autómatos. Os módulos desenvolvidos permitem que vários utilizadores se liguem a estes independentemente do sistema operativo utilizado e suportam a funcionalidade de múltiplos pedidos ao servidor OPC.

O protocolo IEC 60870-5-104 define uma trama robusta implementando mecanismos de controlo de erros e de duplicados, possibilitando a sua correção. A trama contém um campo para informar a causa da transmissão e contém descritores de qualidade nos dados recebidos que informam o utilizador de possíveis avisos e erros durante a leitura de dados

no dispositivo. Para o módulo protocolar desenvolvido apenas foram implementados os comandos (identificadores de tipo) mais utilizados pelo protocolo. No desenvolvimento e implementação deste módulo não foi possível testar todas as funcionalidades implementas pelo facto de não dispormos de um módulo escravo.

Como trabalho futuro seria vantajoso acrescentar funcionalidades extra aos clientes OPC-DA (por exemplo, pedido de propriedades dos itens DA) e OPC-AE (por exemplo, melhorar o sistema de remoção de filtros na monitorização de eventos) e ao mestre IEC 60870-5-104 (por exemplo, implementação do tratamento de tramas de mais identificadores de tipo). Os testes dos módulos desenvolvidos foram realizados utilizando simuladores. Seria adequado efetuar também testes com equipamentos reais simulando situações também reais.

Referências Documentais

[1] R. Pantoni; "Desenvolvimento e implementação de uma descrição de dispositivos aberta e não-proprietária para equipamentos Foundation Fieldbus baseada em XML", Dissertação de Mestrado, Universidade de São Paulo, 2006

[2] F. Borges; "Redes de comunicação Industrial", Documento técnico nº 2 Edição de Setembro de 2007, Schneider Electric

[3] T. Gonçalves; "Projecto e implementação de uma aplicação de pequena rede industrial para controlo de ETAR", Dissertação de Mestrado, Universidade do Minho, Junho de 2009

[4] T. Nogueira; "Redes de comunicação para sistemas de automação industrial", Monografia de graduação em Engenharia de controlo e automação, Ouro Preto, 2009

[5] IEC; "Telecontrol equipment and systems – Part 5-101: Transmission protocols – Companion standard for basic telecontrol tasks", 2ª Edição, Internacional Standard IEC 60870-5-101, Fevereiro de 2003

[6] E. Bertazini; "Análise de funções de um conversor de protocolos de comunicação para automação elétrica, baseado na utilização da linguagem de modelagem unificada", Dissertação apresentada à Escola Politécnica da Universidade de São Paulo, 2006

[7] IEC; "Telecontrol equipment and systems – Part 5-104: Transmission protocols – Network access for IEC 60870-5-101 using standard transport profiles", 2ª Edição, Internacional Standard IEC 60870-5-104, Junho de 2006

[8] MAYR Software, "Manual LIAN 98 – IEC 60870-5-104: Telegram structure", disponível em: <http://manuals.lian98.biz/doc.en/html/u_iec104_struct.htm>

[9] F. Foundation; "Fieldbus Foundation", disponível em: <http://www.fieldbus.org>

[10] SMAR; "Maunual dos procedimentos de instalação, operação e manutenção Foundation Fieldbus", Versão 3, Fevereiro de 2008

[11] F. Foundation; "Foundation H1", 2006, disponível em: <http://www.fieldbus.org/index.php?option=com_content&task=view&id=137&Itemid=313>

[12] F. Foundation; "Foundation HSE", 2006, disponível em: <http://www.fieldbus.org/index.php?option=com_content&task=view&id=138&Itemid=314>

[13] PROFIBUS; "Profibus & Profinet",disponível em: <http://www.profibus.com>

[14] HSM; “PROcess FIeld BUS”, HSM Industrial Networks, 2012, disponível em: <http://www.hms.se/technologies/profibus.shtml>

[15] Modbus.org; “Modbus”, disponível em: <http://www.modbus.org>

[16] Modbus.org; “MODBUS over Serial Line Specification and Implementation Guide”, v.1.02, Dezembro de 2006

[17] ODVA; “DeviceNet”, disponível em: <http://www.odva.org/Home/ODVATECHNOLOGIES/DeviceNet/tabid/66/lng/en-US/Default.aspx>

[18] AS-Interface; “Automation is easy – with AS-Interface”, disponível em: <http://www.as-interface.net>

[19] CANopen; “Welcome to CANopen.us”, disponível em: <http://www.canopen.us>

[20] OPC Foundation; “What is OPC?”, 2013, disponível em: <http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp?MID=AboutOPC>

[21] D. Ribeiro; Leandro; R. Anjos; “Utilização do drive OPC na integração de sistemas de automação industrial”, 3º Seminário Nacional de Sistemas Industriais e Automação, Belo Horizonte, Outubro de 2008

[22] W. Mahnke; S. Leitner; M. Damm; “OPC Unified Architecture”, Alemanha, Springer-Verlag Berlin Heidelberg, 2009

[23] OPC Foundation; “OPC Overview”, Versão 1.0, Outubro de 1998, disponível em: < http://www.opcfoundation.org/Archive/ad5c2ed9-ad93-419d-8e99-0bd006a411a9/General/OPC%20Overview%201.00.pdf>

[24] OPC Foundation; “What is OPC?”, 2013, disponível em: <http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp?MID=AboutOPC>

[25] J. Neto; “Desenvolvimento da capacidade de comunicação segundo o protocolo OPC (OLE for Process Control) para o sistema supervisório SISAL (Sistema Supervisório de Automação de Elevação)”, Universidade Federal do Rio Grande do Norte, Julho de 2010

[26] OPC for Linux, “What is OPC”, disponível em: <http://linuxopc.info/>

[27] SourceForge, Free DCE and DCOM, disponível em: <http://sourceforge.net/projects/freedce/?source=dlp>

[28] SourceForge, “PyOPC”, disponível em: <http://pyopc.sourceforge.net>

[29] SourceForge, “OpenOPC”, disponível em: <http://openopc.sourceforge.net>

[30] OpenSCADA, “UTGARD”, disponível em: <http://openscada.org/projects/utgard>

[31] Softing, “OPC Classic Toolkits”, disponível em: <http://industrial.softing.com/en/products/functionality/software/opc-development-

toolkits/opc-classic-dcom-based/windows/opc-classic-da-client-development-toolkit.html>

[32] SourceForge, “JEasyOPC”, disponível em: <http://jeasyopc.sourceforge.net>

[33] Softing, “OPC Easy Connect Suite”, disponível em: <http://industrial.softing.com/en/products/functionality/software/opc-middleware/opc-communication-optimization-software-opc-easy-connect-suite.html>

[34] Jellycan Code | SimpleIni, Fevereiro de 2012, disponível em: <http://code.jellycan.com/simpleini/>

[35] MRTS-NG, “MRTS-NG – open source implementation of IEC 60870-5-104 protocols”, disponível em: <http://code.google.com/p/mrts-ng>

[36] REINHARD FINK; “Products – Windows Programs”, disponível em: <http://www.ppfink.de>

Anexo A. Lista de bibliotecas para implementar um cliente OPC

Nome (Link)	Comercial	Livre	Suporte	Cliente	Servidor	DA2	DA3	AE	HDA	XML-DA	UA	Linguagens
UTGARD	☐	☑	☐	☑	☐	☑	☐	☐	☐	☐	☐	Java
JEasyOPC Client	☐	☑	☑	☑	☐	☑	☑	☐	☐	☐	☐	Delphi, Java
LightOPC	☐	☑	☐	☐	☑	☑	☐	☐	☐	☐	☐	C/C++
beharrell - OPC DA Client SDK	☐	☑	☐	☑	☐	☑	☐	☐	☐	☐	☐	C++
PyOPC	☐	☑	☐	☑	☐	☐	☐	☐	☐	☑	☐	Python
OpenOPC	☐	☑	☐	☑	☐	☑	☐	☐	☐	☐	☐	Python
OPC UA Linux Client Development Toolkit	☑	☐	☑	☑	☐	☑	☑	☑	☐	☐	☑	C++
dOPC Client Toolkits	☑	☐	☑	☑	☐	☑	☑	☑	☐	☑	☐	Delphi, C++
OPC UA Windows C++ Server	☑	☐	☑	☐	☑	☑	☑	☑	☐	☐	☑	C++
OPC UA SDK 1.01	☑	☐	☑	☑	☑	☑	☐	☑	☑	☐	☑	.NET, C/C++
OPC .NET API 2.00	☑	☐	☑	☑	☑	☑	☑	☑	☑	☐	☐	.NET
QuickOPC.NET	☑	☐	☑	☑	☐	☑	☑	☑	☐	☐	☐	.NET
OPC Classic Toolkits	☑	☐	☑	☑	☑	☑	☑	☑	☐	☑	☐	C++, C#, VB.NET
.NET OPC - Advosol	☑	☐	☑	☑	☑	☑	☑	☑	☑	☑	☑	.NET
MatrikonOPC Server for OMRON PLCs	☐	☑	☐									**Simulador**
Softing OPC Classic Demo Server	☐	☑	☐									**Simulador**

Anexo B. Lista de bibliotecas para implementar um *master* 60870-5-104

IEC 60870-5-104				
Nome (Link)	**Comercial**	**Livre**	**Suporte**	**Linguagens**
Lib870-5	☐	☑	☑	C/C++
SIM-104	☐	☑	☐	JAVA
MRTS-NG	☐	☑	☐	C/C++
Triangle Microworks	☑	☐	☑	VB, C#, J#, C++, …
IEC 60870-5-104 .NET Library	☑	☐	☑	C++

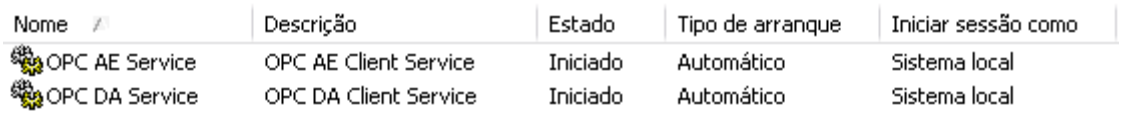
Anexo C. Lista de valores de qualidade OPC

Membros	Valor	Descrição
GOOD	192	O valor é válido.
GOOD_LOCAL_OVERRIDE	216	O valor foi sobrescrito.
BAD	0	Dado inválido por motivo desconhecido.
BAD_CONFIG_ERROR	4	Existe um problema no servidor com a configuração.
BAD_NOT_CONNECTED	8	A ligação da entrada a um dispositivo não foi estabelecida.
BAD_DEVICE_FAILURE	12	Falha no dispositivo.
BAD_SENSOR_FAILURE	16	Falha no sensor.
BAD_LAST_KNOWN	20	Comunicação falhou. O último valor está disponível.
BAD_COMM_FAILURE	24	Comunicação falhou. O último valor está indisponível.
BAD_OUT_OF_SERVICE	28	Fora de serviço.
BAD_WAITING_FOR_INITIAL_DATA	32	Servidor espera por dados iniciais.
UNCERTAIN	64	Não há razão específica para o qual o valor é incerto.
UNCERTAIN_LAST_USABLE	68	Mostra o último valor utilizável.
UNCERTAIN_SENSOR_CAL	80	Valor está nos limites ou fora dos limites do sensor.
UNCERTAIN_EGU_EXCEEDED	84	O valor retornado está fora dos limites deste parâmetro.
UNCERTAIN_SUB_NORMAL	88	Valor originado de várias fontes, mas nem todas são boas.
QUALITY_NOT_SET	65535	A qualidade não está definida.

Anexo D. Instalação dos serviços Windows OPC

Os programas foram desenvolvidos para serem executados como serviços Windows ("Iniciar" → "Executar…" → "services.msc").

Para que os diversos clientes (máquinas com sistemas Linux e Windows) consigam aceder às aplicações desenvolvidas, os serviços terão que se encontrar instalados numa máquina Windows e iniciados, como demonstra a Figura 45.



Nome	Descrição	Estado	Tipo de arranque	Iniciar sessão como
OPC AE Service	OPC AE Client Service	Iniciado	Automático	Sistema local
OPC DA Service	OPC DA Client Service	Iniciado	Automático	Sistema local

Figura 45 Serviços Windows OPC

Para a instalação dos serviços Windows foi criado um ficheiro *batch* (.bat). O código na Figura 46 é referente ao ficheiro *batch* com a instalação do serviço DA, em que o código do serviço AE é de certa forma similar alterando apenas os nomes de "DA" para "AE".

```
sc create "OPC DA Service" binPath= "%~dp0DAConsole Service.exe" DisplayName= "OPC DA Service" depend= RpcSs start= auto
sc description "OPC DA Service" "OPC DA Client Service"
sc failure "OPC DA Service" reset= 86400 actions= restart/1000/restart/1000//
NET start "OPC DA Service"
```

Figura 46 Ficheiro *batch* respetivo à instalação do serviço Windows OPC-DA

```
sc create <nome do serviço> binPath= <caminho\programa.exe> DisplayName= <nome a apresentar> depend= <dependências> start= <tipo de arranque>
```

- A primeira linha corresponde à criação do serviço Windows.
- Resumidamente, o comando inserido irá criar o serviço com o nome de "OPC DA Service", cujo caminho do executável "binPath" se encontra na pasta do ficheiro *batch* e denominado por "DAConsole_Service.exe".

- O nome que será apresentado na janela de serviços será “OPC DA Service”, com um tipo de arranque automático “start=auto”, isto para permitir que o serviço seja automaticamente iniciado após a inicialização do Windows.
- Para que o serviço funcione corretamente, o serviço com o nome de “RpcSs” conhecido por “Chamada de Procedimento Remoto (RPC)” necessita de se encontrar inicializado, razão pela qual foi adicionado como uma dependência deste serviço “depend=RpcSs”.

```
sc description <nome do serviço> <descrição>
```

- Na segunda linha irá atribuir uma descrição ao serviço.

```
sc failure <nome do serviço> reset= <tempo de reinicialização da contagem de falhas> actions= <1ª falha>/<ms>/<2ª falha>/<ms>/<falhas seguintes>/<ms>
```

- A terceira linha permite definir os parâmetros de “Recuperação” do serviço Windows.
- Neste caso atribuiu-se que após 86400 segundos, que representa 1 dia, a contagem de falhas será reinicializada.
- Na primeira e na segunda falha, o serviço será reinicializado, enquanto que nas falhas seguintes o serviço não irá realizar nenhuma ação (Figura 47).

```
NET start <nome do serviço>
```

- A quarta linha irá iniciar o serviço e aguardar até à sua inicialização.

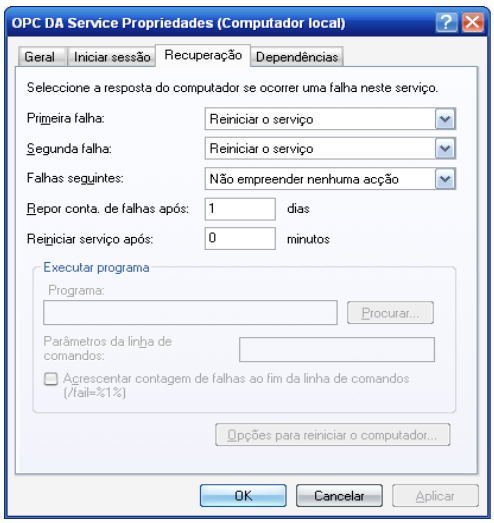


Figura 47 Separador de “Recuperação” das propriedades do Serviço Windows OPC-DA

Para desinstalar os serviços, também foram criados ficheiros *batch*. O código contido neste ficheiro para o exemplo do DA (para o AE altera apenas os nomes dos serviços) é o representado na Figura 48.

```
NET stop "OPC DA Service"
sc delete "OPC DA Service"
```

Figura 48 Ficheiro *batch* respetivo à desinstalação do serviço Windows OPC-DA

`NET stop <nome do serviço>`

- O primeiro comando irá parar o serviço.

`sc delete <nome do serviço>`

- O segundo irá eliminá-lo do sistema.

Para facilitar a instalação dos serviços para qualquer máquina em Windows foram criados dois executáveis. O executável permite ao utilizador instalar o serviço Windows OPC desejado num destino à sua escolha e executa automaticamente o ficheiro *batch* respetivo à criação do serviço (Figura 49). O utilizador ao desinstalar o serviço ("Iniciar" → "Todos os programas" → "OPC DA Service" → "Unninstall") irá remover os ficheiros criados pela instalação e executar o ficheiro *batch* referente à desinstalação (Figura 50).

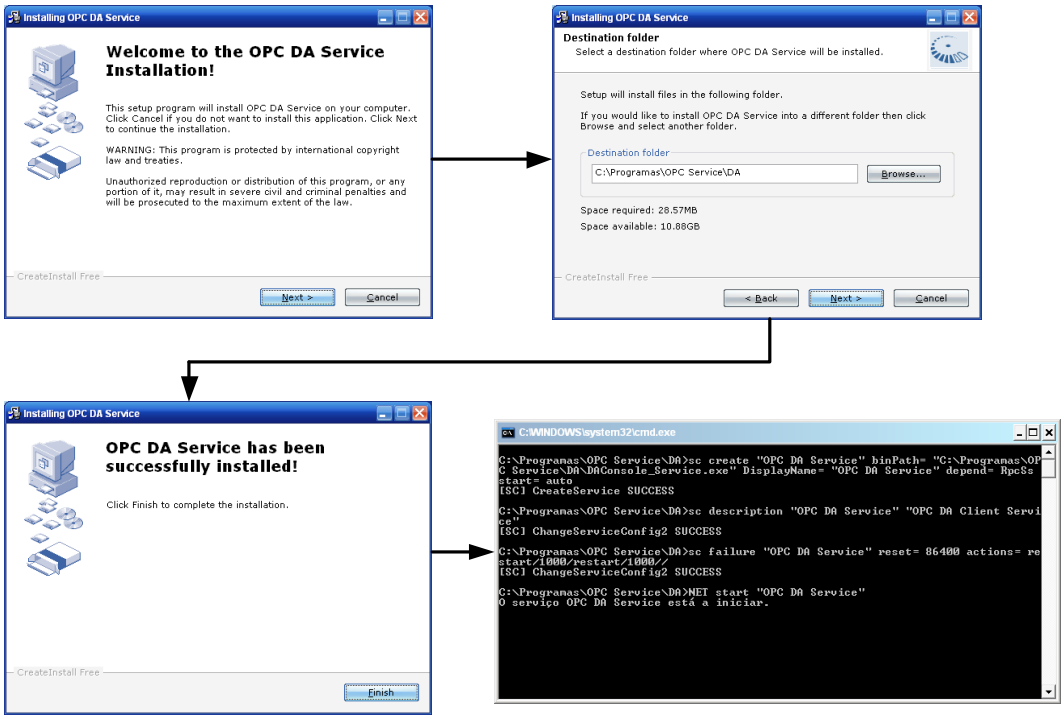


Figura 49 Instalação do Serviço Windows OPC-DA

Figura 50 Desinstalação do Serviço Windows OPC-DA

Anexo E. Métodos e Classes do serviço Windows OPC-DA

- **Classes e métodos do SDK OPC-DA da Softing:**

A classe *Application* disponibiliza os métodos para inicializar e terminar o cliente, detecção e tratamento de objetos *OPCServer* para a comunicação com o servidor. A Tabela 22 contém os principais métodos da classe *Application*. As características dos itens são obtidas através da classe *ValueQT* (Tabela 23). O valor (*value*) do item é do tipo criado pela classe *VARIANT*, a qualidade do valor (*quality*) é dado pela enumeração *EnumQuality* e o *time stamp* é dado pelo tipo criado pela classe *DateTime*. A classe *ValueQT* é a responsável por definir o formato dos itens que são trocados entre o cliente OPC e o servidor OPC.

Tabela 22 Principais métodos da classe *Application*

Classe *Application*	
Métodos	**Descrição**
Initialize	Inicializa a aplicação.
ProcessCommandLine	Responsável pelo registo da aplicação como um serviço.
ReleaseApplication	Liberta todos os recursos alocados pela aplicação.
Terminate	Termina a aplicação.

Tabela 23 Principais métodos da classe *ValueQT*

Classe *ValueQT*	
Métodos	**Descrição**
ValueQT	Cria uma nova instância da classe.
getData	Retorna o valor do item (*value*).
getQuality	Retorna a qualidade do valor (*quality*).
getTimeStamp	Retorna o tempo ao qual o valor foi obtido (*time stamp*).
toString	Retorna característica em forma de *string*.

Para a listagem de servidores OPC num dispositivo são usadas as classes *ServerBrowser* e *ServerBrowserData*. A classe *ServerBrowser* retorna os servidores OPC contidos num dispositivo, consoante a especificação definida por *EnumOPCSpecification*. Os servidores retornados serão guardados na classe *ServerBrowserData*.

Na Tabela 24 encontram-se as classes, métodos e a enumeração necessárias para a listagem de servidores OPC. Para a listagem de itens contidos num determinado servidor OPC é usada a classe *DaAddressSpaceElement* (Tabela 25). A classe retorna os itens de um determinado caminho definido pelo utilizador (expansíveis ou não, especificados pelo *EnumAddressSpaceElementType* selecionado).

Depois da sessão criada e a conexão estabelecida com o servidor OPC pode ser pedido a listagem da informação relativa a esse servidor. As informações do servidor OPC são guardadas na classe *ServerStatus* (Tabela 26).

Tabela 24 Classes e enumerações para a listagem de servidores OPC

Classe *ServerBrowser*	
Métodos	**Descrição**
Browse	Retorna os servidores OPC especificados.
Classe *ServerBrowserData*	
Métodos	**Descrição**
GetUrl	Retorna o URL de um servidor.
Enumeração *EnumOPCSpecification*	
Membros	**Descrição**
EnumOPCSpecification_AE10	Especificação OPC-AE versão 1.0.
EnumOPCSpecification_DA10	Especificação OPC-DA versão 1.0.
EnumOPCSpecification_DA20	Especificação OPC-DA versão 2.0.
EnumOPCSpecification_DA30	Especificação OPC-DA versão 3.0.

Tabela 25 Classe e enumeração para a listagem de itens num servidor OPC

Classe *DaAddressSpaceElement*	
Métodos	**Descrição**
DaAddressSpaceElement	Cria uma nova instância da classe.
Browse	Acede ao *namespace* do caminho definido do servidor OPC-DA.
GetItemId	Retorna o caminho e o nome do item.
GetName	Retorna nome do item.
Enumeração *EnumAddressSpaceElementType*	
Métodos	**Descrição**
EnumAddressSpaceElementType_BRANCH	Itens expansíveis.
EnumAddressSpaceElementType_LEAF	Itens não expansíveis.
EnumAddressSpaceElementType_ALL	Todos os itens.

Tabela 26 Principais métodos da classe *ServerStatus*

Classe *ServerStatus*	
Métodos	**Descrição**
serverStatus	Inicializa uma nova instância da classe.
getBandwidth	Desempenho do processador do servidor.
getCurrentTime	Tempo atual do servidor.
getGroupCount	Número total de subscrições criadas para aceder ao servidor.
getLastUpdateTime	Tempo do último envio de dados para o cliente.
getProductVersion	Versão de *software* do servidor.
getStartTime	Tempo da inicialização do servidor.
getState	Estado atual do servidor (por exemplo: iniciado).
getStatusInfo	Descrição textual do estado atual do servidor.
getSupportedLcIds	Lista de linguagens suportadas pelo servidor.
getVendorInfo	Informações adicionais do servidor.
toString	Retorna um elemento em forma de *string.*

- **Classes e métodos desenvolvidos/aproveitados para o serviço:**

Para o ficheiro de inicialização (*Settings.ini*) foi usada uma classe *CSimpleIniA* [34], que permite o tratamento da informação do ficheiro (criação, leitura, escrita, entre outros). Na Tabela 27 encontram-se os principais métodos desta classe.

Tabela 27 Principais métodos da classe *CSimpleIniA*

Classe *CSimpleIniA*	
Métodos	**Descrição**
GetAllKeys	Retorna todos os campos contidos numa secção.
GetAllSections	Retorna todas as secções.
GetLongValue	Lê um valor do tipo *long* de um determinado campo.
LoadFile	Abre/Cria o ficheiro de inicialização.
SaveFile	Guarda definições do ficheiro de inicialização.
SetLongValue	Adiciona/Modifica um valor do tipo *long* a um determinado campo.

Para a criação de uma sessão com o servidor OPC é usada a classe *MyDaSession* (Tabela 28), para criar uma subscrição é usada a classe *MyDaSubscription* (Tabela 29) e para a inicialização dos itens a serem requisitados é usada a classe *MyDaItem* (Tabela 30). Como já referido, foram usados os métodos da *MyDaSubscription* para a leitura e escrita de itens.

Tabela 28 Principais métodos da classe *MyDaSession*

Classe *MyDaSession*	
Métodos	**Descrição**
MyDaSession	Inicializa uma nova instância da classe.
Connect	Estabelece ligação com um servidor.
Disconnect	Desconecta a ligação com o servidor.
GetCurrentState	Estado atual da ligação (por exemplo: conectado).
GetStatus	Informações sobre o servidor (Tabela 26).
GetUrl	Retorna URL do servidor.
Read	Leitura de um determinado item.
RemoveDaSubscription	Remove uma subscrição da sessão atual.
Write	Escrita de um determinado item.

Tabela 29 Principais métodos da classe *MyDaSubscription*

Classe *MyDaSubscription*	
Métodos	**Descrição**
MyDaSubscription	Inicializa uma nova instância da classe.
connect	Estabelece ligação com um servidor (usado para ativar a subscrição, monitorizando os itens inicializados).
disconnect	Desconecta a ligação com o servidor.
getCurrentState	Estado atual da subscrição (por exemplo: ativado).
read	Leitura de um determinado item.
removeDaItem	Remove um item da subscrição.
SendToClient	Envia uma mensagem para o utilizador.
write	Escrita de um determinado item.
handleDataChanged	Método utilizado quando ocorre uma notificação de mudança de estado na monitorização de itens.
handleReadCompleted	Método utilizado quando ocorre uma recepção dos itens de uma leitura assíncrona.
handleWriteCompleted	Método utilizado quando ocorre uma recepção dos itens de uma escrita assíncrona.

Tabela 30 Principais métodos da classe *MyDaItem*

Classe *MyDaItem*	
Métodos	**Descrição**
MyDaItem	Inicializa uma nova instância da classe. Depois de criada é atribuída à subscrição. Responsável pela inicialização dos itens.
connect	Estabelece ligação com um servidor.
disconnect	Desconecta a ligação com o servidor.
read	Leitura de um determinado item.
write	Escrita de um determinado item.

No programa foram criados três tipos de threads para a implementação de mecanismos multi-clientes e multi-pedidos (Tabela 31). Para a leitura, escrita, monitorização de itens foi criada a classe *OpcClient* (Tabela 32). As listagens de servidores e de itens encontram-se no grupo (*namespace*) denominado por *OPCServers* (Tabela 33).

Tabela 31 Threads do serviço Windows

Threads	
Métodos	**Descrição**
ThreadSocketClient	Thread criada para cliente.
ThreadProcess	Thread criada para processamento de pedidos OPC ao servidor.
ThreadInfoServers	Thread criada para listagem de servidores OPC num dispositivo.

Tabela 32 Principais métodos da classe *OpcClient*

Classe *OpcClient*	
Métodos	**Descrição**
activateAsyncSubscription	Ativa a subscrição assíncrona (monitorização de itens).
activateSyncSubscription	Ativa a subscrição síncrona (monitorização de itens).
getSyncServerStatus	Obtém as informações relativas ao servidor.
initiateActiveVariables	Inicializa os itens a monitorizar.
initiateReadVariables	Inicializa os itens a ler.
initiateWriteVariables	Inicializa os itens a escrever.
readAsyncSubscription	Leitura assíncrona dos itens.
readSyncSubscription	Leitura síncrona dos itens.
terminateOpcClient	Terminar ligação com o servidor.
writeAsyncSubscription	Escrita assíncrona dos itens.
writeSyncSubscription	Escrita síncrona dos itens.

Tabela 33 Métodos principais desenvolvidos para o serviço OPC-DA

Namespace::Método	Descrição
sockmodule	Grupo relativo à criação de *sockets*, envio e recepção de dados.
::GetErrorStringMessage	Permite obter o erro do sistema gerado pela função WSAGetLastError() em forma de *string*.
::LogFile	Permite criar um ficheiro de histórico de dados, denominado por "LogFile.log". Este ficheiro englobará: data, hora, tipo de mensagem e mensagem.
::Receive	Responsável pela recepção da mensagem do cliente.
::recvTimeOutTCP	Criação do mecanismo select(). Espera por ligações TCP.
::Send	Responsável pelo envio da mensagem para o cliente.

::StartServer	Criação dos mecanismos *sockets*: socket(), bind(), listen().
sockclients	Grupo responsável por aceitar pedidos de clientes e criação de *threads*.
::Accept	Função que irá aceitar pedidos de ligações TCP.
::reserve_memory_threads	Reserva uma quantidade de *threads* "MAX_THREADS".
::ThreadCreateClient	Criação de uma *thread* para ser usada pelo cliente. Executa o método denominado por "ThreadSocketClient".
::ThreadCreateInfoServers	Criação de uma *thread* para listar os servidores OPC DA 2.0 num dispositivo. Executa o método denominado por "ThreadInfoServers".
::ThreadCreateProcess	Criação de uma *thread* para a execução de pedidos OPC ao servidor. Executa o método denominado por "ThreadProcess".
::ThreadsStart	Verifica se alguma *thread* se encontra livre.
Commands	Grupo que engloba os métodos mais genéricos do programa.
::CleanActiveItems	Limpa as variáveis associadas à monitorização de eventos da estrutura de dados.
::IsIPv4	Verifica se o endereço IP inserido se encontra correto (IPv4).
::IsOPCDA	Verifica se o endereço OPC inserido se encontra correto
::ItemReceived	Retorna o item introduzido pelo cliente (com ou sem aspas).
::ItemsActiveReceived	Acrescenta o item recebido à monitorização de itens.
::ItemsRemoveReceived	Remove o item recebido da monitorização de itens.
::Lowercase	Converte uma *string* em letras minúsculas.
::NumberReceived	Retorna a quantidade de itens introduzida pelo cliente.
::ReceiveCommandClient	Implementa mecanismos de *timeout* com o valor de "TIMEOUT RECEIVE" e verificação de conectividade com o cliente para a recepção da mensagem. Usa o método "sockmodule::Receive".
::SendCommandClient	Implementa mecanismos de *timeout* com o valor de "TIMEOUT SEND" e verificação de conectividade com o cliente para o envio da mensagem. Usa o método "sockmodule::Send".
::StringCommand	Separação da primeira palavra encontrada (até ao primeiro espaço) na *string*. Retorna a palavra encontrada, ou então "NULL" no caso de haver apenas uma palavra na *string*.
::StringToWrite	Retira as aspas de uma *string*.
::TerminateThreadClient	Limpa as variáveis da estrutura de dados da *thread* do cliente, libertando-a e terminando-a.
::ThreadServerValues	Copia alguns dos dados da estrutura de dados da *thread* do cliente para a nova *thread*.
::ThreadValues	Copia alguns dos dados da estrutura de dados da *thread* do cliente para a nova *thread*.
::ValueToWriteReceived	Retorna o valor a ser escrito, introduzido pelo cliente.
OPCServers	Grupo que engloba métodos para o tratamento dos pedidos OPC-AE.

::ActivateAsyncSubscription	A função trata dos argumentos recebidos pelo cliente (item, e quantidade de itens) a monitorizar, acrescentando-os ao vetor atual. Recorre ao método activateAsyncSubscription() da classe OpcClient.
::ActivateSyncSubscription	A função trata dos argumentos recebidos pelo cliente (item, e quantidade de itens) a monitorizar, acrescentando-os ao vetor atual. Recorre ao método activateSyncSubscription() da classe OpcClient.
::OPCEnumerateItems	Leitura dos itens contidos num determinado caminho do servidor OPC-DA. Envia resultado ao cliente.
::OPCEnumerateServers	Leitura dos servidores OPC-DA 2.0 contidos num dispositivo. Envia resultado ao cliente.
::ReadAsyncItems	A função trata dos argumentos recebidos pelo cliente (item e quantidade de itens) a ler. Responsável pelo envio de mensagem, no caso, de erro de leitura ao cliente. Recorre ao método readAsyncSubscription() da classe OpcClient.
::ReadSyncItems	A função trata dos argumentos recebidos pelo cliente (item e quantidade de itens) a ler. Responsável pelo envio dos argumentos lidos do servidor OPC-DA para o cliente. Recorre ao método readSyncSubscription() da classe OpcClient.
::RemoveItensAsyncSubscription	Remove um item da monitorização assíncrona.
::RemoveItensSyncSubscription	Remove um item da monitorização síncrona.
::WriteAsyncItems	A função trata dos argumentos recebidos pelo cliente (item, valor e quantidade de itens) a escrever. Responsável pelo envio de mensagem, no caso, de erro de escrita ao cliente. Recorre ao método writeAsyncSubscription() da classe OpcClient.
::WriteSyncItems	A função trata dos argumentos recebidos pelo cliente (item, valor e quantidade de itens) a escrever. Responsável pelo envio dos argumentos escritos no servidor OPC-DA para o cliente. Recorre ao método writeSyncSubscription() da classe OpcClient.

Anexo F. Testes e Resultados do serviço Windows OPC-DA

Na Figura 52 demonstra os tipos de mensagens retornadas para os comandos "opcda" (URL do servidor) da Figura 51. Os comandos representados na Figura 51 têm como objetivo testar as mensagens retornadas pelo serviço Windows para as possíveis situações listadas na Tabela 8.

```
opcda://192.168.122.193/Softing.OPCToolboxDemo_ServerDA.1/{2E565242-B238-11D3-842D-
0008C779D775
opcda:/192.168.122.193/Softing.OPCToolboxDemo ServerDA.1/{2E565242-B238-11D3-842D-
0008C779D775}
opcda://192.168.122/Softing.OPCToolboxDemo_ServerDA.1/{2E565242-B238-11D3-842D-
0008C779D775}
opcda://192.168.122Softing.OPCToolboxDemo_ServerDA.1/{2E565242-B238-11D3-842D-
0008C779D775}
opcda://192.168.122.193/{Softing.OPCToolboxDemo_ServerDA.1/2E565242-B238-11D3-842D-
0008C779D775}
opcda://192.168.122.198/Softing.OPCToolboxDemo_ServerDA.1/{2E565242-B238-11D3-842D-
0008C779D775}
opcda://192.168.122.193/Softing.OPCToolboxDemo_ServerDA.1/{2E565242-B238-11D3-842D-
0008C779D775}
```

Figura 51 Teste de comandos para URL do servidor

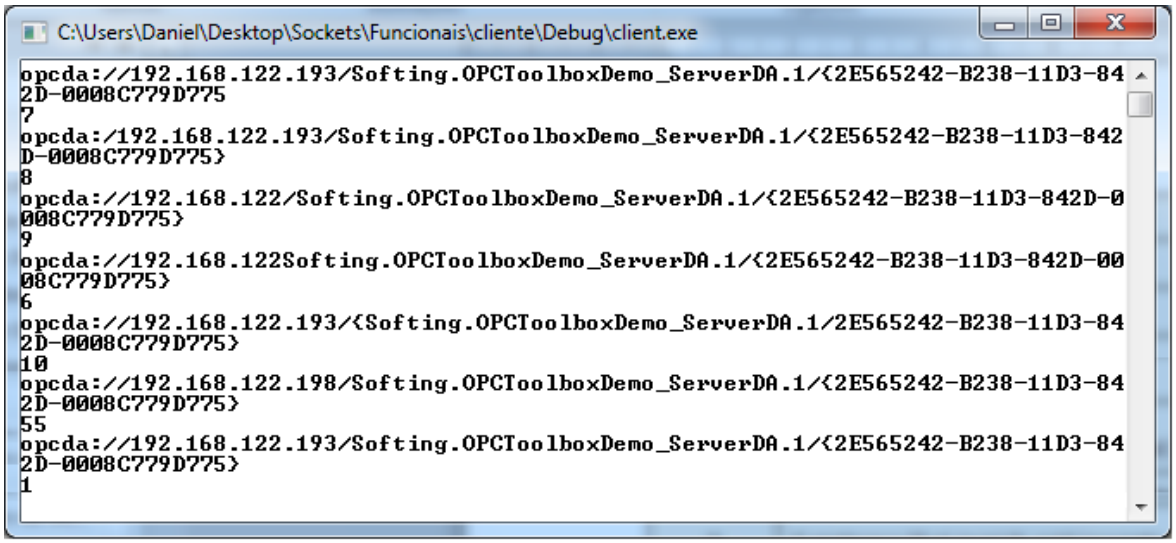


Figura 52 Comandos para URL do servidor

Na Figura 54, encontra-se o comando "r IOP.static.I1 4" que corresponde à leitura síncrona de itens, nomeadamente aos itens I1 a I4. Pela análise da figura, verifica-se que os itens I1, I2 e I4 foram lidos com sucesso (código 6) e com valor "0". A razão pela qual, o item I3 não ser lido é devido ao facto do servidor em questão não conter esse item, motivo pelo qual, o código 7 ser retornado. O campo a seguir, com valor 192, corresponde ao valor da qualidade. O campo seguinte corresponde ao *time stamp*. O comando para a leitura assíncrona funciona de uma forma semelhante. Neste caso, foi realizado uma leitura assíncrona aos itens I1 e I2 com sucesso, ou seja, código 11. Os restantes campos retornados são iguais aos da leitura síncrona.

```
r IOP.static.I1 4
ra IOP.static.I1 2
```

Figura 53 Teste para comandos de leitura síncrona e assíncrona

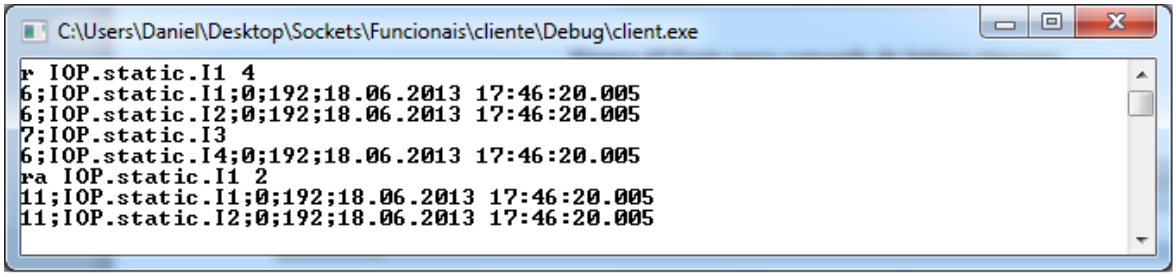


Figura 54 Comandos de leitura síncrona e assíncrona

Na Figura 56 encontra-se um exemplo de comandos de escrita síncrona e assíncrona de itens. Na escrita síncrona, os itens I1, I2 e I4 foram escritos com o valor "1" com sucesso (código 16). A razão pela qual, o item I3 não ser escrito é devido ao facto do servidor em questão não conter esse item, motivo pelo qual, o código 17 ser retornado. A razão pela qual os campos da qualidade (com o valor "65535") e o *time stamp* estarem com os valores apresentados, deve-se ao facto do servidor não considerar relevante estes campos para a escrita de dados. O comando de escrita assíncrona funciona de forma semelhante. Neste comando apenas são retornadas as mensagens de sucesso ou de erro dos itens escritos.

```
w IOP.static.I1 1 4
wa IOP.static.I1 0 2
```

Figura 55 Teste para comandos de escrita síncrona e assíncrona

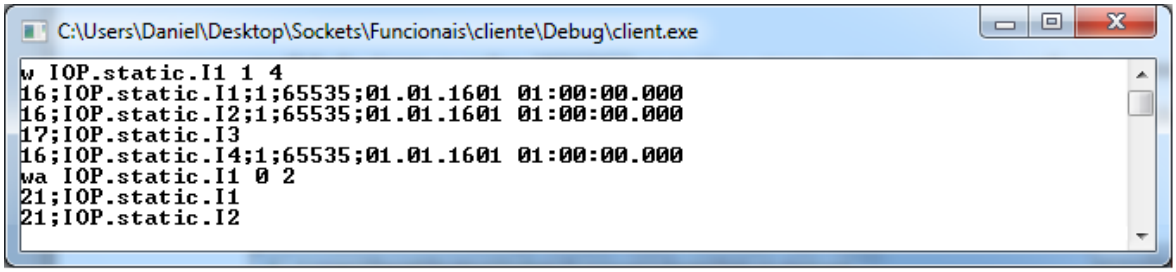


Figura 56 Comandos de escrita síncrona e assíncrona

Na Figura 58 mostra o funcionamento do comando de monitorização síncrona de itens "m". Na primeira situação foram adicionados os itens I1 e I2 ao vetor de monitorização, retornando os códigos 11 e 28, correspondendo à leitura assíncrona dos itens com sucesso e à modificação de itens, respetivamente. A seguir foi adicionado o item UI1 à monitorização. Depois foi realizada uma escrita síncrona aos itens I1 e I2 com o valor 0. Como a variável modificou o estado, para além de receber o código de sucesso da escrita síncrona dos itens I1 e I2 (código 16) foi também recebida a notificação de mudança de estado das variáveis I1 e I2 (código 28). Após isso cancelou-se a monitorização de itens "mq", recebendo o código 32 e retornou-se ao primeiro menu "end", recebendo o código 0.

Na Figura 60 demonstra uma listagem de itens no simulador do servidor OPC. A Figura 62 demonstra um exemplo de uma remoção de um item da monitorização. No caso apresentado foi eliminado o item "I2" da monitorização com o código de sucesso 35. Na Figura 64 demonstra um pedido de informações ao servidor OPC.

```
m IOP.static.I1 2
m IOP.static.UI1
w IOP.static.I1 0 2
mq
end
```

Figura 57 Teste para monitorização de itens

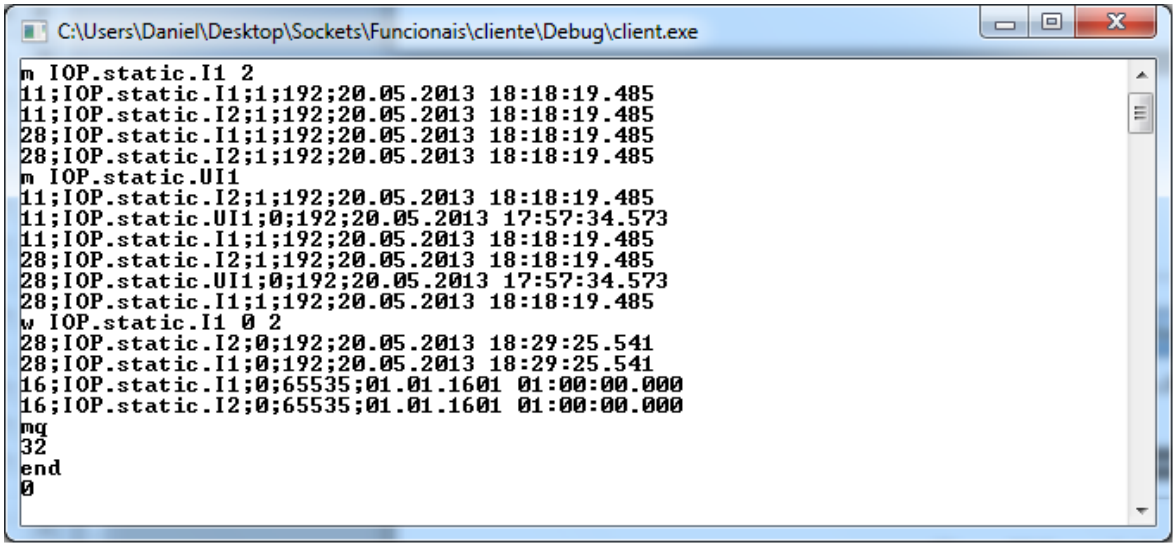


Figura 58 Comandos de monitorização e escrita de itens, cancelamento de monitorização e retorno ao primeiro menu

```
opcda://192.168.122.193/Softing.OPCToolboxDemo ServerDA.1/{2E565242-B238-11D3-842D-0008C779D775}

l

l "watch"

l "watch.device 1"
```

Figura 59 Teste listagem de itens no servidor OPC

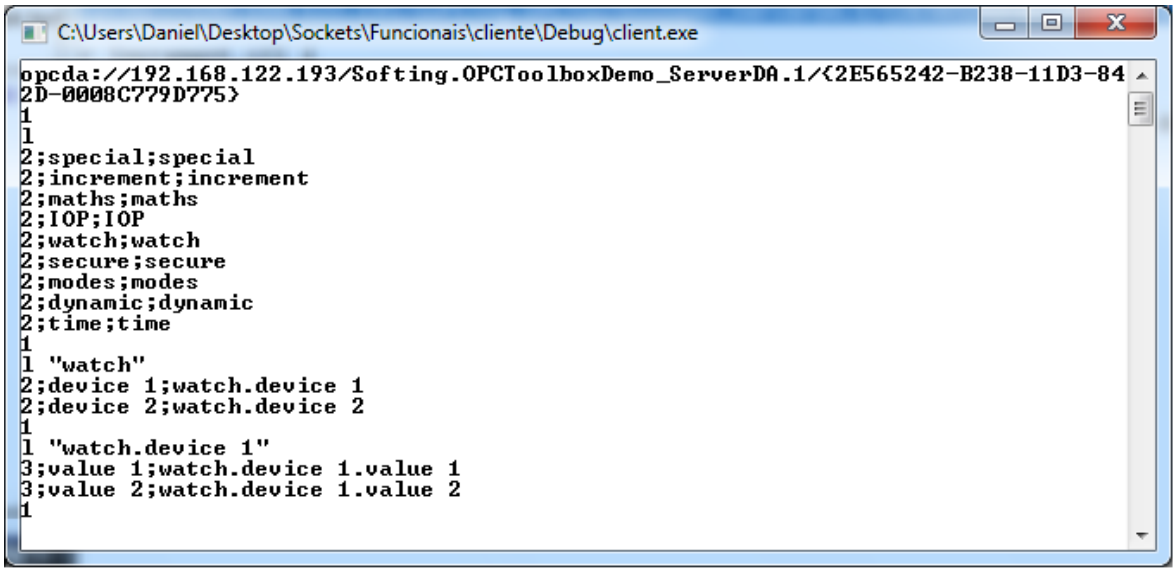


Figura 60 Comandos para listagem de itens do servidor OPC

```
m IOP.static.I1 2
m IOP.static.UI1 2
mr IOP.static.I2
```

Figura 61 Teste para remover item da monitorização

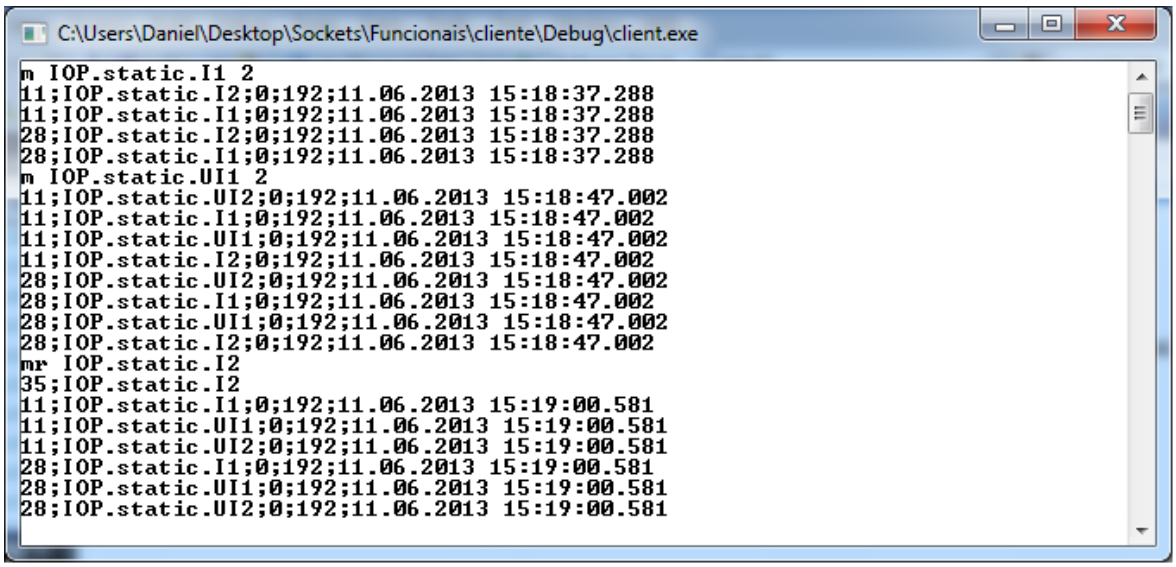


Figura 62 Comando para a remoção de um item no servidor OPC-DA

```
s
```

Figura 63 Teste para listar informações do servidor OPC-DA

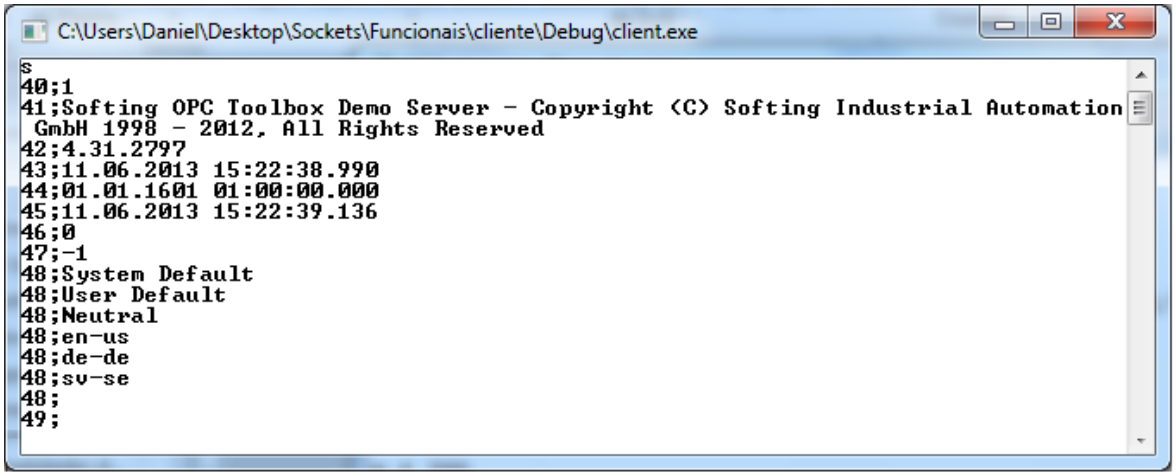


Figura 64 Comando para a listagem de informações do servidor OPC-DA

Na Figura 66 inseriu-se um comando inválido “xpto” que corresponde ao código 60 e o comando para terminar e sair “q” que retorna o código 0.

```
xpto
q
```

Figura 65 Teste para comando inválido “xpto” e comando sair “q”

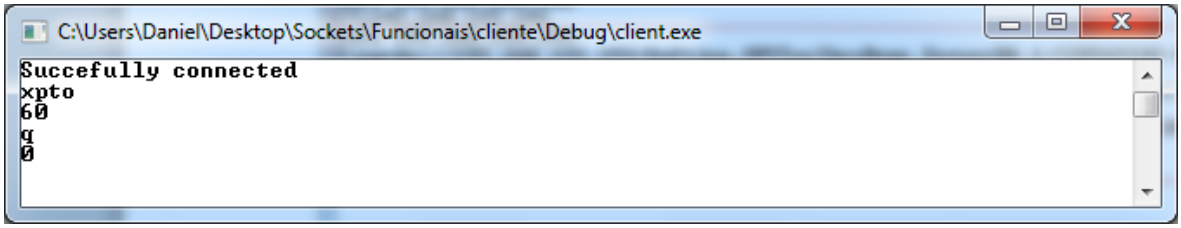


Figura 66 Comando inválido “xpto” e comando sair “q”

Anexo G. Métodos e Classes do serviço Windows OPC-AE

- **Classes e métodos do SDK OPC-AE da Softing:**

O cliente OPC-AE utiliza as classes *Application* (para disponibilizar os métodos para inicializar e terminar o cliente, e métodos para a comunicação com o servidor), *ServerStatus* (para a listagem das informações relativas ao servidor), *ServerBrowser* e *ServerBrowserData* (para a listagem de servidores OPC-AE) do serviço Windows que implementa o cliente OPC-DA. Para a listagem de itens contidos no *namespace* da *Event Area* são usadas as classes *AddressSpaceElement* (Tabela 34) e os métodos *browse* e *queryAeSourceConditions* da *MyAeSession* (Tabela 37). Para listar o esquema de eventos (*Filter Space*) do servidor são usadas as classes *AeCategory* e *AeAttribute* (Tabela 35). A classe *AeEvent* é usada para os atributos retornados pelos eventos (Tabela 36).

Tabela 34 Classe e enumeração para a listagem de itens da *Event Area*

Classe *AddressSpaceElement*	
Métodos	**Descrição**
DaAddressSpaceElement	Cria uma nova instância da classe.
GetName	Retorna nome do elemento.
GetQualifiedName	Retorna o caminho + nome do elemento.
Enumeração *EnumAddressSpaceElementType*	
Métodos	**Descrição**
EnumAddressSpaceElementType_BRANCH	Elementos expansíveis.
EnumAddressSpaceElementType_LEAF	Elementos não expansíveis.
EnumAddressSpaceElementType_ALL	Todos os elementos.

Tabela 35 Principais métodos das classes *AeCategory* e *AeAttribute*

Classe *AeCategory*	
Métodos	**Descrição**
getDescription	Retorna a descrição da categoria.
queryAeAttributes	Retorna os atributos de uma categoria.
queryAeConditionNames	Retorna as condições associadas a uma categoria.
Classe *AeAttribute*	
Métodos	**Descrição**
getDescription	Retorna a descrição do atributo.

Tabela 36 Principais métodos da classe *AeEvent*

Classe *AeEvent*	
Métodos	**Descrição**
getAckRequired	Retorna se condição necessita de reconhecimento (acknowledgment) deste evento.
getActiveTime	Retorna o tempo que a condição ficou ativa (Active) ou o tempo da transição da atual subcondição.
getActorId	Retorna o utilizador que gerou o evento.
getCategory	Retorna a categoria do evento.
getConditionName	Retorna o nome da condição.
getEventType	Retorna o tipo de evento.
getMessage	Retorna a descrição do evento.
getOcurrenceTime	Retorna o tempo ao qual foi gerado o evento.
getQuality	Retorna a qualidade da condição.
getSeverity	Retorna o valor de severidade do evento.
getSourcePath	Retorna o item ao qual o evento foi gerado.
getState	Retorna o estado da condição (ACT, ENA, ACK, DIS).
getSubConditionName	Retorna o nome da subcondição.
toString	Retorna uma variável em forma de *string*.
Enumeração *EnumEventType*	
Métodos	**Descrição**
EnumEventType_CONDITION	Evento do tipo condição.
EnumEventType_SIMPLE	Evento do tipo simples.
EnumEventType_TRACKING	Evento do tipo tracking.

- **Classes e métodos desenvolvidos/aproveitados para o serviço:**

Para o ficheiro de inicialização (*Settings.ini*) foi usada a mesma classe que no serviço OPC-DA, ou seja, a classe *CSimpleIniA*. Para a criação de uma sessão com o servidor OPC e para realizar determinadas *querys* ao servidor é usada a classe *MyAeSession* (Tabela 37), e para a criação de uma subscrição é usada a classe *MyAeSubscription* (Tabela 38). Sempre que é recebido um evento será tratado no método *handleAeEventsReceived* da classe *MyAeSubscription.*

No programa foram criados três tipos de threads para a implementação de mecanismos multi-clientes e multi-pedidos, tal como no serviço OPC-DA. Para a realização de *querys* das condições associadas a um item e monitorização de eventos foi criada a classe *OpcClient* (Tabela 39). As listagens de servidores, da *event area*, da *filter space* e informações relativas ao servidor encontram-se no grupo (*namespace*) denominado por *OPCServers*. Na Tabela 40 encontram-se os grupos desenvolvidos para o OPC-AE. Os grupos *sockmodule* e *sockclients* não são representados devido ao facto de serem iguais ao serviço OPC-DA.

Tabela 37 Principais métodos da classe *MyAeSession*

Classe *MyAeSession*	
Métodos	**Descrição**
MyAeSession	Inicializa uma nova instância da classe.
Browse	Acede ao namespace da Event Area.
Connect	Estabelece ligação com um servidor.
Disconnect	Desconecta a ligação com o servidor.
GetCurrentState	Estado atual da ligação (por exemplo: conectado).
GetStatus	Informações sobre o servidor (Tabela 26).
GetUrl	Retorna URL do servidor.
QueryAeCategories	Retorna as categorias dos eventos do servidor.
QueryAeSourceConditions	Retorna as condições associadas a um item.
RemoveAeSubscription	Remove uma subscrição da sessão atual.

Tabela 38 Principais métodos da classe *MyAeSubscription*

Classe *MyAeSubscription*	
Métodos	**Descrição**
MyAeSubscription	Inicializa uma nova instância da classe.
connect	Estabelece ligação com um servidor (usado para ativar a subscrição, monitorizando os eventos).
disconnect	Desconecta a ligação com o servidor.
eventTypeToString	Conversão do valor do tipo de evento para string.
getCurrentState	Estado atual da subscrição (por exemplo: ativado).
SendToClient	Envia uma mensagem para o utilizador.
setFilterAreas	Filtro para áreas.
setFilterCategories	Filtro para categorias.
setFilterEventTypes	Filtro para tipo de eventos.
setFilterSeverityHigh	Filtro para severidade máxima.
setFilterSeverityLow	Filtro para severidade mínima.
setFilterSources	Filtro para fontes (itens).
handleAeEventsReceived	Método utilizado quando ocorre uma notificação gerada pelo servidor.

Tabela 39 Principais métodos da classe *OpcClient*

Classe *OpcClient*	
Métodos	**Descrição**
::AeQueryConditions	Listagem de condições de um determinado item.
::AllEventsSubscription	Iniciar a monitorização de todos os eventos.
::FilterEventsSyncSubscription	Iniciar a monitorização com os filtros definidos pelo cliente (filtros de área, categoria, tipo, severidade mínima, severidade máxima, fontes).
::getStateToString	Usado para converter o valor correspondente aos estados do evento para uma *string* (ACT ENA ACK DIS).
::getSyncServerStatus	Obter informações do servidor.
::InitiateSession	Iniciar uma sessão com o servidor.
::TerminateSession	Terminar a sessão com o servidor.

Tabela 40 Métodos principais desenvolvidos para o serviço OPC-AE

Namespace::Método	Descrição
Commands	Grupo que engloba os métodos mais genéricos do programa.
::CleanActiveItems	Limpa as variáveis associadas à monitorização de eventos da estrutura de dados.
::EventTypeToLong	Interpretação dos tipos de eventos (c: condição, s: simples, t: tracking) inseridos pelo cliente na monitorização.
::FilterReceived	Verifica o tipo de filtro inserido para a monitorização e preenche a respetiva variável da estrutura de dados do cliente.
::FilterRemoveReceived	Remoção de um elemento contido num filtro (vetor) definido pelo cliente (área, categoria, fonte).
::IsIPv4	Verifica se o endereço IP inserido se encontra correto (IPv4).
::IsOPCAE	Verifica se o endereço OPC inserido se encontra correto
::ItemReceived	Retorna o item introduzido pelo cliente (com ou sem aspas).
::Lowercase	Converte uma *string* em letras minúsculas.
::NumberReceived	Retorna a quantidade de itens introduzida pelo cliente.
::ReceiveCommandClient	Implementa mecanismos de *timeout* com o valor de "TIMEOUT RECEIVE" e verificação de conectividade com o cliente para a recepção da mensagem. Usa o método "sockmodule::Receive".
::SendCommandClient	Implementa mecanismos de *timeout* com o valor de "TIMEOUT SEND" e verificação de conectividade com o cliente para o envio da mensagem. Usa o método "sockmodule::Send".
::StringCommand	Separação da primeira palavra encontrada (até ao primeiro espaço) na *string*. Retorna a palavra encontrada, ou então "NULL" no caso de haver apenas uma palavra na *string*.
::StringToWrite	Retira as aspas de uma *string*.
::TerminateThreadClient	Limpa as variáveis da estrutura de dados da *thread* do cliente, libertando-a e terminando-a.
::ThreadServerValues	Copia alguns dos dados da estrutura de dados da *thread* do cliente para a nova *thread*.
::ThreadValues	Copia alguns dos dados da estrutura de dados da *thread* do cliente para a nova *thread*.
OPCServers	Grupo que engloba métodos para o tratamento dos pedidos OPC-AE.
::AllEvents	Executa o método "AllEventsSubscription" da classe OpcClient, passando os devidos parâmetros.
::FiltersSyncSubscription	Executa o método "FilterEventsSyncSubscription" da classe OpcClient, passando os devidos parâmetros.
::OPCEnumerateEvents	Envia ao cliente um esquema dos eventos contidos num determinado servidor OPC-AE (filter space).
::OPCEnumerateItems	Envia ao cliente os itens contidos num determinado caminho do servidor OPC-AE (event area). Os itens podem ser expansíveis ou não.
::OPCEnumerateServers	Envia ao cliente os servidores OPC-AE contidos num determinado dispositivo.

::OPCServerStaus	Envia ao cliente informações relativas ao servidor OPC-AE.
::QueryConditions	Executa o método “AeQueryConditions” da classe OpcClient, passando os devidos parâmetros.

Anexo H. Testes e Resultados do serviço Windows OPC-AE

Na Figura 68 demonstra os tipos de mensagens retornadas para o comando "opcae" (URL do servidor) da Figura 67. Os comandos representados na Figura 67 têm como objetivo testar as mensagens retornadas pelo serviço Windows para as possíveis situações listadas na Tabela 11. Na Figura 70 demonstra a listagem de itens contidos na *event area* e na Figura 72 demonstra a listagem do *filtre space* do simulador do servidor OPC-AE da Softing. A Figura 74 mostra as informações do servidor OPC-AE e a Figura 76 representa uma *query* de condições associadas ao item "`computer.clock.time slot 1`". Na Figura 78 demonstra um exemplo da monitorização de eventos com a aplicação de filtros, em que se aplicam os filtros para o item "`computer.clock.time slot 1`" com uma severidade mínima de "300", desta forma apenas se irá receber os eventos gerados por esse item com uma severidade acima de 300. A Figura 80 demonstra a remoção de itens da monitorização.

```
opcae://192.168.122.193/Softing.OPCToolboxDemo_ServerAE.1{2E565243-B238-11D3-842D-
0008C779D775}
opcae://192.168.122.193/Softing.OPCToolboxDemo ServerAE.1/{2E565243-B238-11D3-842D-
0008C779D775
opcae:/192.168.122.193/Softing.OPCToolboxDemo_ServerAE.1/{2E565243-B238-11D3-842D-
0008C779D775}
opcae://192.168.122/Softing.OPCToolboxDemo_ServerAE.1/{2E565243-B238-11D3-842D-
0008C779D775}
opcae://192.168.122.193/Softing.OPCToolboxDemo ServerAE.1}/{2E565243-B238-11D3-842D-
0008C779D775
opcae://192.168.122.198/Softing.OPCToolboxDemo_ServerAE.1/{2E565243-B238-11D3-842D-
0008C779D775}
opcae://192.168.122.193/Softing.OPCToolboxDemo ServerAE.1/{2E565243-B238-11D3-842D-
0008C779D775}
```

Figura 67 Teste de comandos para URL do servidor AE

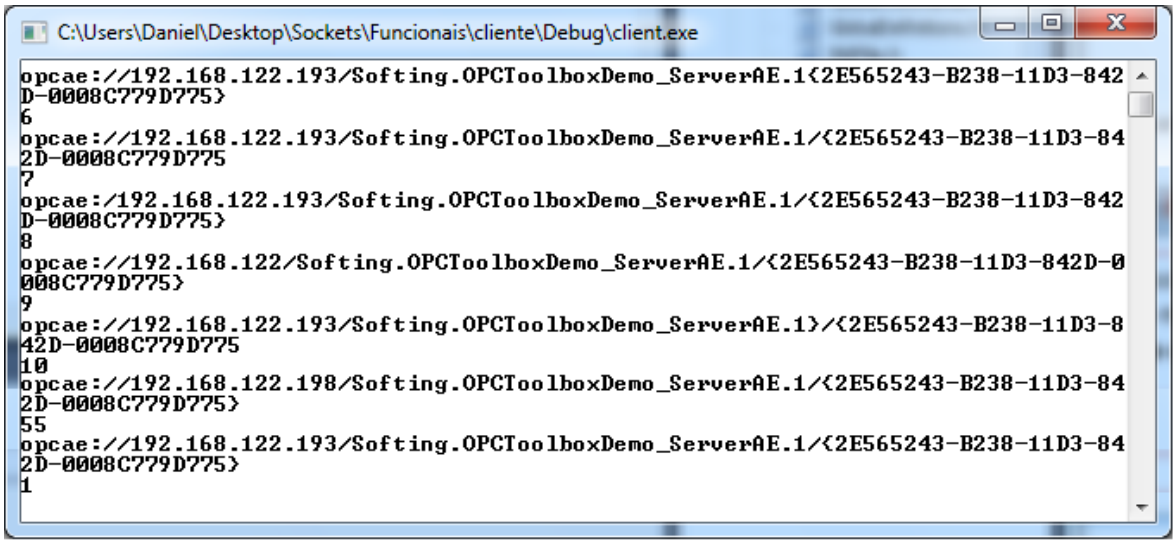


Figura 68 Comandos para URL do servidor AE

```
l

l computer

l computer.clock
```

Figura 69 Teste de comandos "L" do servidor AE

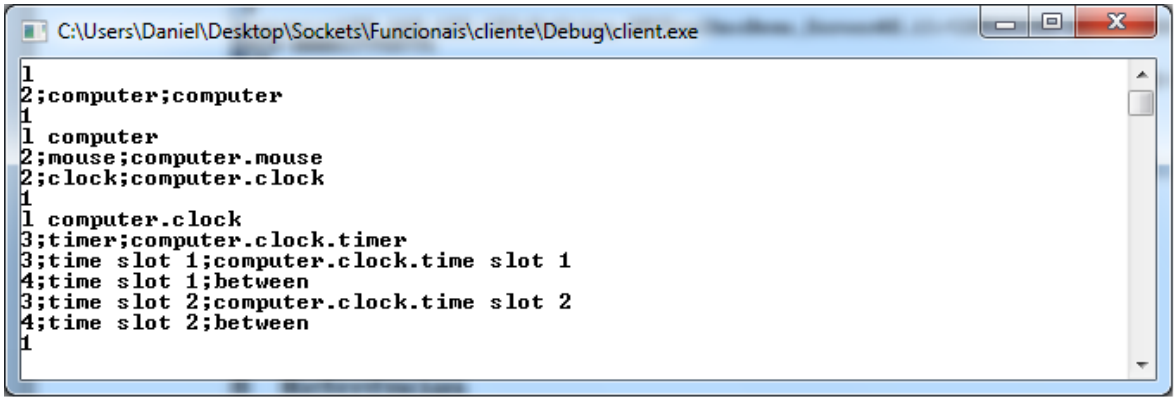


Figura 70 Retorno do comando para a listagem de itens "L" do serviço AE

```
ls
```

Figura 71 Teste de comando “LS” do servidor AE

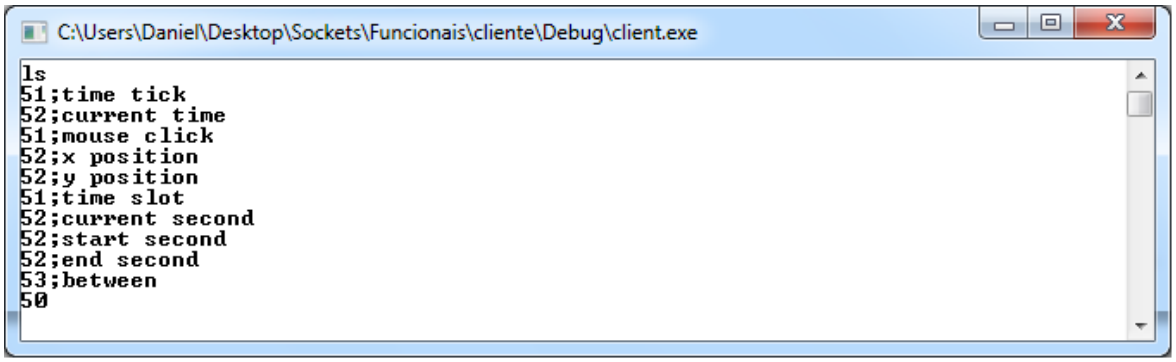


Figura 72 Comando para a listar o esquema de eventos do servidor “LS” do serviço AE

```
s
```

Figura 73 Teste de comando “S” do servidor AE

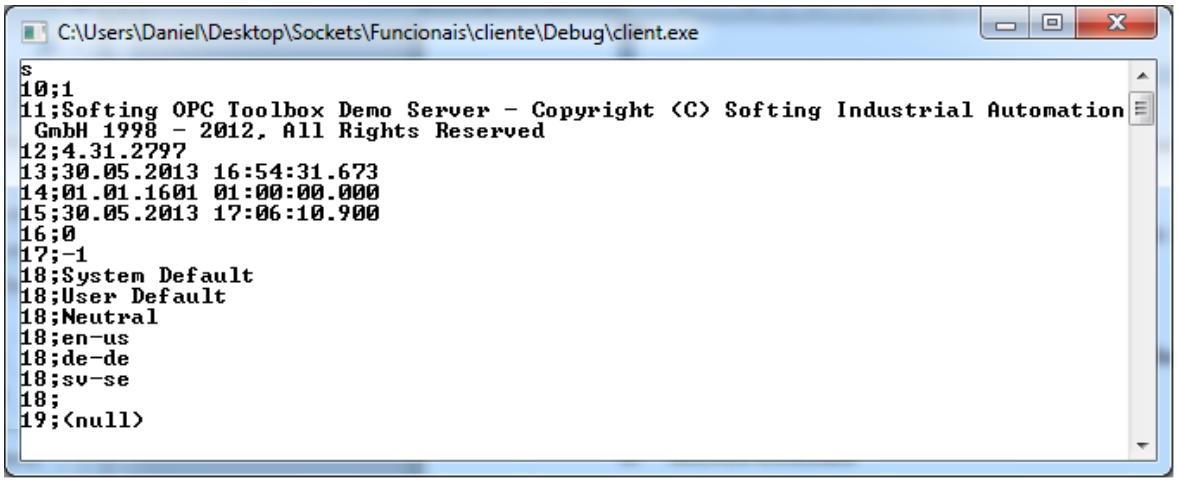


Figura 74 Comando para a listagem de informações do servidor “S” do serviço AE

```
c "computer.clock.time slot 1"
c "computer.clock"
```

Figura 75 Teste de comandos "C" do servidor AE

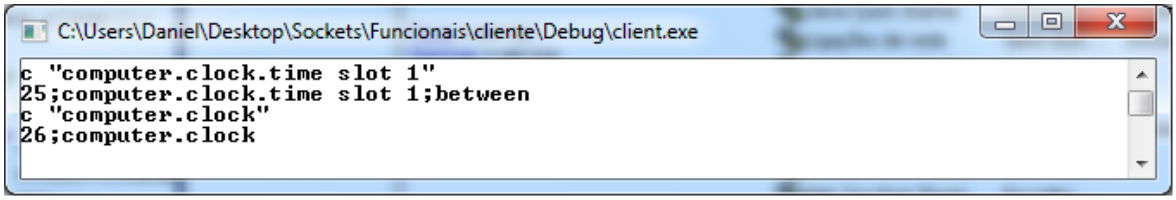


Figura 76 Comando para a listagem de condições de um item "C" do serviço AE

```
m 6 "computer.clock.time slot 1"
m 4 300
mq
```

Figura 77 Teste de comandos "M" e "MQ" do servidor AE

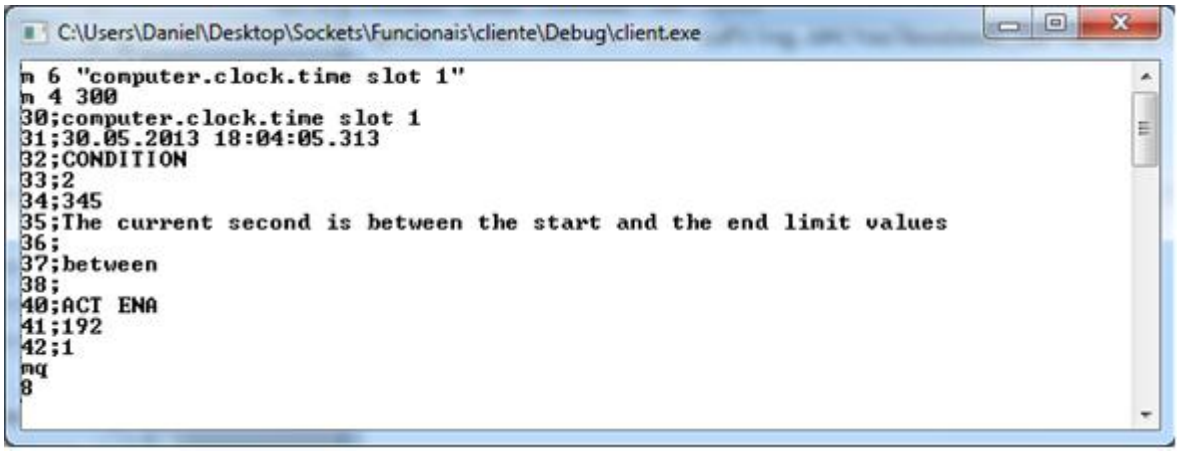


Figura 78 Comando para a monitorização de eventos "M" do serviço AE

```
m 6 “computer.clock.time slot 1”
mr 6 “computer.clock.time slot 1”
mr 2 2
mr 2
```

Figura 79 Teste de comandos “MR” do servidor AE

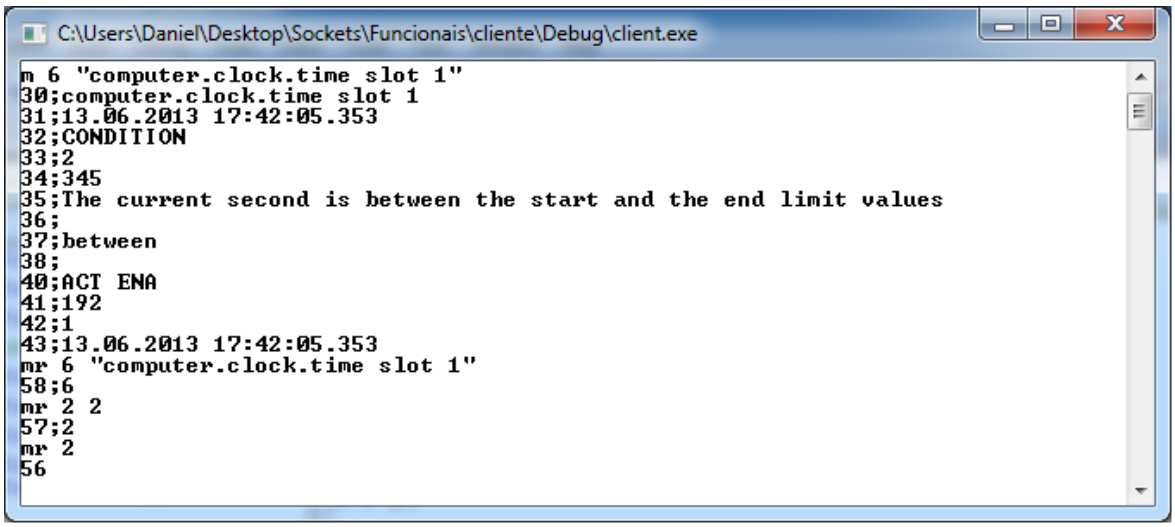


Figura 80 Comando para remover elementos da monitorização “MR” do serviço AE

```
m 6 “computer.clock.time slot 1”
mr 6 “computer.clock.time slot 1”
mr 2 2
mr 2
```

Figura 81 Teste de comandos “E” do servidor AE

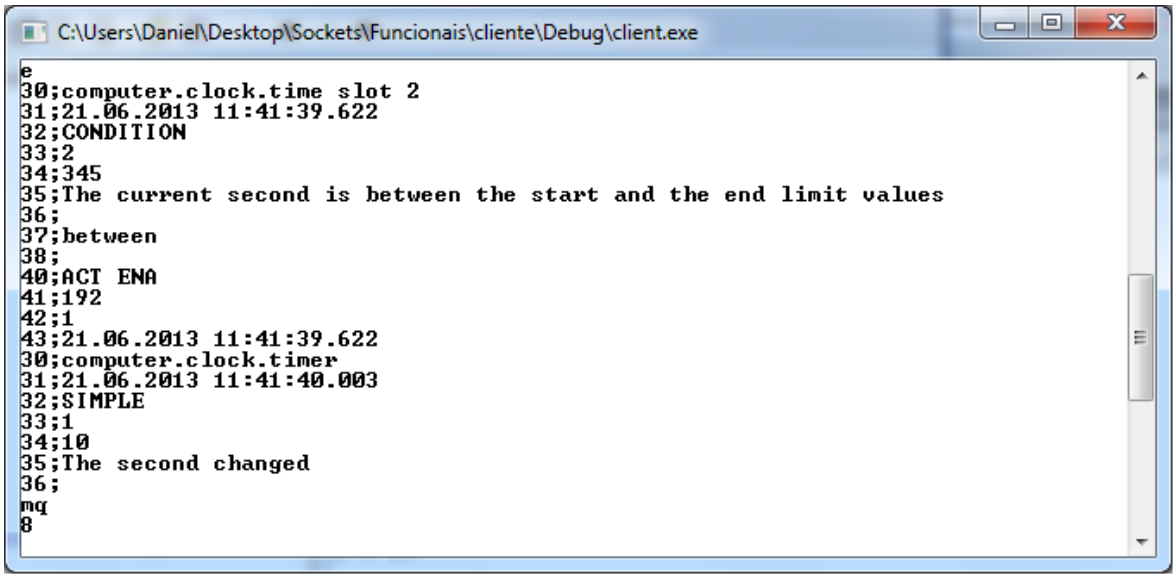


Figura 82 Comando para a monitorização de todos os eventos "E" do serviço AE

Anexo I. Testes e Resultados do mestre IEC 60870-5-104

Apenas foi possível testar os comandos de escrita com o simulador, com um SQ=0 e um retorno de dados de um objeto de informação e de um elemento de informação. Por isso serão apresentados todos os testes que foram possíveis realizar.

- **Escrita com um comando duplo**

As respostas aos comandos duplos (código 3) contêm os "elementos de informação" descritos na Tabela 41. Na Figura 83 é possível observar um exemplo de um comando duplo e respetivas respostas retornadas pelo simulador.

Tabela 41 Elementos da informação da resposta a um "comando duplo"

bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1
IV	NT	SB	BL	0	0	Estado (MSB)	Estado (LSB)

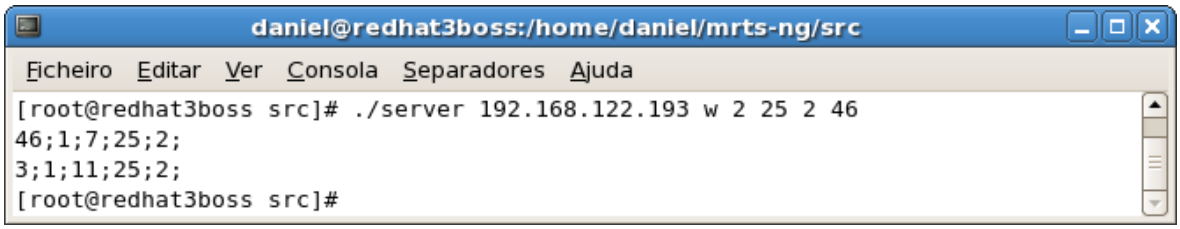


Figura 83 Exemplo de respostas a um "comando duplo"

- **Escrita com um comando simples (CP56Time2a)**

As respostas aos comandos simples CP56Time2a (código 30) contêm os mesmos "elementos de informação" que um comando simples, com a diferença de receber mais sete octetos correspondentes à *tag* de tempo (hora + data). Na Figura 84 demonstra um exemplo de um comando simples CP56Time2a e respetivas respostas retornadas pelo simulador. A hora retornada será no formato `hh:mm:ss:ms` e a data no formato `DD:MM:YY`.

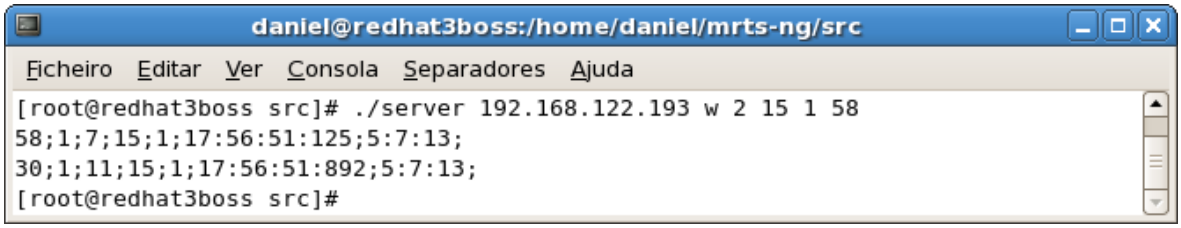


Figura 84 Exemplo de respostas a um "comando simples CP56Time2a"

- **Escrita com um comando duplo (CP56Time2a)**

As respostas aos comandos duplos CP56Time2a (código 31) contêm os mesmos "elementos de informação" que um comando duplo, com a diferença de receber mais sete octetos correspondentes à *tag* de tempo (hora + data). Na Figura 85 demonstra um exemplo de um comando duplo CP56Time2a e respetivas respostas retornadas pelo simulador. A hora retornada será no formato `hh:mm:ss:ms` e a data no formato `DD:MM:YY`.

Figura 85 Exemplo de respostas a um "comando duplo CP56Time2a"