Faculdade de Engenharia da Universidade do Porto

Avaliação de plataformas de software para um veículo eléctrico

António Manuel Ribeiro Patrício

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

Orientador: Prof. Doutor Paulo Portugal

Março de 2011

Resumo

Nos últimos anos tem-se assistido a um aumento gradual do número de veículos eléctricos em circulação. Por utilizarem uma forma de energia menos poluente, estes veículos representam cada vez mais uma alternativa aos veículos com motor de combustão. Por outro lado, o crescente desempenho e confiabilidade dos componentes de hardware e a evolução das tecnologias de software, possibilitaram a implementação de funções complexas que melhoram o conforto e a segurança dos ocupantes do veículo. Sistemas que outrora foram puramente mecânicos, têm vindo a ser substituídos de forma progressiva por sistemas electrónicos. Os veículos actuais incorporam já uma série de sistemas electrónicos embarcados responsáveis por um conjunto vasto funcionalidades.

No âmbito desta dissertação foi realizado um levantamento e respectiva descrição de alguns dos sistemas embarcados presentes nos veículos actuais e que possam ser implementados num veículo motorizado (moto 4x4). O veículo em questão irá ser adquirido pelo Departamento de Engenharia Electrotécnica e Computadores (DEEC) da Faculdade de Engenharia da Universidade do Porto e será posteriormente transformado num veículo eléctrico. Pretende-se que o veículo sirva de plataforma de suporte aos trabalhos de várias disciplinas leccionadas no departamento e possibilite aos alunos a aquisição de conhecimentos relativamente ao tipo de sistemas e tecnologias usadas nos veículos actuais.

Depois de realizado o levantamento dos sistemas existentes, efectuou-se um estudo detalhado sobre os tipos de plataformas de software existentes, passíveis de ser utilizadas como suporte ao desenvolvimento dos sistemas anteriormente enumerados. Daqui, surgiu uma lista de requisitos que o sistema deverá comportar, e em função da mesma foram propostos sistemas operativos e *middlewares* que possibilitem o desenvolvimento, implementação e integração das diferentes aplicações distribuídas pelo veículo.

Abstract

During the last years the number of electric vehicles in circulation has gradually increased. Due to the fact these vehicles use a cheaper and less pollutant form of energy ones represent more and more an alternative to traditional combustion engine vehicles. On the other hand, the higher performance and reliability of hardware components and the software technology evolution allowed the development of complex functions which improved the vehicles users' suitability and security. Systems that once were purely mechanical have been considerably changed by electronic embedded systems. Nowadays, vehicles already embody a set of electronic embedded systems responsible by several distinct functionalities.

In the context of this thesis, a research and respective description about embedded systems in actual vehicles capable of being implemented in a motor vehicle (4x4 motorcycle) has taken part. The target vehicle will be acquired by the Department of Electrical and Computer Engineering (DEEC) of the Faculty of Engineering of the University of Porto and modified afterwards into an electrical vehicle. The vehicle should work as a support platform to several courses running in the department as well as to provide the students means to develop their knowledge about types of systems and technologies used on current vehicles.

After researching the existing systems, a detailed study on different software platform types capable of being used as support on the aforementioned systems was performed. Hereof, a requirement list regarding the system behavior was built and according to it operative systems and middleware's providing the means to develop, to implement and to integrate different applications were proposed.

Agradecimentos

Após a conclusão deste trabalho gostaria de expressar aqui o mais profundo agradecimento a todos aqueles que tornaram a realização deste trabalho possível.

Ao meu orientador Prof. Doutor Paulo Portugal, agradeço pelas palavras de incentivo, apoio e cooperação no desenvolvimento desta dissertação.

Aos meus colegas e amigos de curso pelo bom ambiente proporcionado ao longo destes últimos anos.

Ao Celso Lopes e ao André Cordeiro, que mesmo não estando directamente envolvidos na realização deste trabalho a sua contribuição foi decisiva.

A minha irmã por todas as conversas e palavras de incentivo.

E por fim, mas nem por isso menos importante, um agradecimento especial aos meus pais pelo constante apoio e carinho.

Índice

Lista de Figuras

Lista de Tabelas

Abreviaturas

Lista de abreviaturas (ordenadas por ordem alfabética)

ABS	Anti-lock Braking System
ACC	Adaptive Cruise Control
API	Application Programming Interfaces
AUTOSAR	Automotive Open System Architecture
CAN	Controller Area Network
CC	Cruise Control
CO2	Dióxido de Carbono
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA/CD	Carrier Sense Multiple Access/Collision Detection
DEEC	Departamento de Engenharia Electrotécnica e de Computadores
EBA	Electronic Brake Assist
EBD	Electronic Brake force Distribution
ECOS	Embedded Configurable Operating System
ECU	Electronic Control Unit
EDF	Early Deadline First
EEPROM	Electrically Erasable Programmable Read Only Memory
ESP	Electronic Stability Program
FEUP	Faculdade de Engenharia da Universidade do Porto
FIFO	First In First Out
GCC	GNU Compiler Collection
HMI	Human Machine Interface
I^2c	Inter Intergrated Circuit
ID	Identificador
ISO	International Standardization Organization
LIN	Local Interconnect Network
LLC	Local Link Control
MAC	Medium Access Control

MOM	Message Oriented Middleware
MOST	Media Oriented Systems Transport
MUP	Multiuniprocessor
NDA	Non-Destructive Arbitration
ORB	Object Request Broker
RMI	Remote Method Invocation
RPC	Remote Procedure Calls
RR	Round Roubin
RTAI	RealTime Application Interface for Linux
SAE	Society of Automotive Engineers
SMP	Symetric Multi Processors
SPI	Serial Peripheral Interface
TCP/IP	Transmission Control Protocol/Internet Protocol
TCS	Traction Control System
TDMA	Time Division Multiple Access
TT-CAN	The Time Triggered Controller Area Network
TTP	Time Triggered Protocol
UART	Universal Asynchronous Receiver Transmitter
UP	Uniprocessor

Capítulo 1

Introdução

1.1 - Motivação

Actualmente vivemos um momento de mudança devido à necessidade de responder aos desafios criados pelas alterações climáticas e de reduzir a dependência de combustíveis fosseis. Apesar da evolução a nível tecnológico, económico e social estar relacionado com a utilização destes combustíveis, o seu uso de forma indiscriminada, nomeadamente no sector dos transportes, tem originado uma crescente degradação ambiental no planeta. Além disso, questões sobre a sustentabilidade têm sido colocadas devido a estes recursos serem não renováveis.

O sector dos transportes é fortemente dependente de recursos energéticos não renováveis, nomeadamente dos produtos petrolíferos, sendo por isso um dos principais responsáveis pelas emissões de gases com efeito de estufa.

Como ilustrado no gráfico da Figura 1.1, em 2008 os sectores de transporte e da electricidade e calor, eram responsáveis por dois terços das emissões globais de C02.[1]

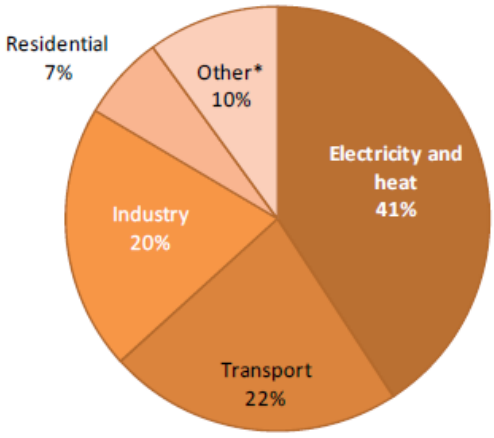

Figura 1.1 - Emissões mundiais por sector de CO2 [1]

É neste contexto que os veículos eléctricos se apresentam como uma alternativa eficaz para de forma urgente reverter a actual situação em que se encontra o nosso planeta. Por utilizarem uma forma de energia menos poluente, estes veículos representam cada vez mais uma alternativa aos veículos com motor de combustão.

Por outro lado, o crescente desempenho e confiabilidade dos componentes de hardware e a evolução das tecnologias de software, possibilitaram a implementação de funções complexas que melhoram o desempenho do veículo, o conforto e a segurança dos ocupantes. Sistemas que outrora foram puramente mecânicos, têm vindo a ser substituídos de forma progressiva por sistemas electrónicos. Os veículos actuais incorporam já uma série de sistemas electrónicos embarcados responsáveis por um conjunto vasto de funcionalidades.

1.2 - Contexto e objectivos

O Departamento de Engenharia Electrotécnica e de Computadores (DEEC) da Faculdade de Engenharia da Universidade do Porto (FEUP) criou recentemente um laboratório de electrónica automóvel. O objectivo é possibilitar aos alunos o desenvolvimento de projectos de sistemas electrónicos vocacionados essencialmente para veículos eléctricos, facultando a aquisição de conhecimentos sobre o tipo de sistemas e tecnologias usadas nos veículos actuais.

Neste contexto foi adquirido um veículo motorizado, HONDA SPORTRAX 250EX, que será posteriormente transformado num veículo eléctrico e constituíra uma plataforma onde os alunos realizarão o desenvolvimento e integração de diferentes trabalhos. Na Figura 1.2 encontra-se ilustrado o veículo que foi adquirido e que será usado.

Figura 1.2 – Veículo motorizado que será usado [2]

No âmbito desta dissertação pretende-se realizar um levantamento e respectiva descrição de alguns dos sistemas embarcados presentes nos veículos actuais e que possam ser implementados no veículo em questão. A implementação destes sistemas requer a existência no veículo de plataformas computacionais que permitam aos alunos realizar os trabalhos e integra-los facilmente na arquitectura de controlo do veículo. Assim, constituem também objectivos deste trabalho efectuar um estudo sobre os tipos de plataformas de software existentes, nomeadamente sistemas operativos e *middlewares*, passíveis de ser utilizadas no desenvolvimento dos diversos sistemas.

Importa referir que não se pretende neste trabalho desenvolver ou implementar sistemas no veículo mas sim referir sistemas que possam ser implementados pelos alunos, propondo sistemas operativos e *middlewares* que possam ser utilizadas durante o desenvolvimento e implementação. Foi tido em consideração as competências dos alunos do 4º e 5º ano, de forma a não serem propostas plataformas demasiado complexas e que exigissem dos alunos um grande investimento de tempo para a sua compreensão.

O documento elaborado no âmbito deste trabalho pretende servir de apoio aos alunos aquando do desenvolvimento de sistemas no veículo.

1.3 - Organização do documento

Nesta secção é apresentada a estrutura e organização deste documento, bem como o conteúdo apresentado em cada um dos capítulos que o constituem.

Estruturalmente, a dissertação está dividida em seis capítulos.

O capítulo 1 no qual se insere esta secção, apresenta a motivação e enquadramento deste trabalho.

No capítulo 2 é realizado um levantamento e descrição de alguns dos sistemas electrónicos presentes nos veículos. São mencionadas as principias redes usadas para interligar os diferentes sistemas incorporados nos automóveis. É também abordada uma arquitectura de software standard denominada AUTOSAR, criada como o objectivo tornar o desenvolvimento do software para aplicações automóveis mais fácil.

Nos capítulos 3 são abordados os principias conceitos referentes aos sistemas operativos. Efectuou-se um estudo detalhado sobre os sistemas operativos existentes passíveis de ser utilizados como suporte ao desenvolvimento dos sistemas enumerados no capítulo 2. Daqui, surgiu uma lista de requisitos que o sistema deverá comportar e em função da mesma foram propostos sistemas operativos que possibilitem o desenvolvimento, implementação e integração das diferentes aplicações distribuídas pelo veículo.

O capítulo 4 trata das plataformas de *middleware*. São mencionadas as principais características e modos de comunicação destas, bem como as vantagens da sua utilização. É efectuado um levantamento de plataformas de *middleware* existentes.

No capítulo 5 são propostas diferentes arquitecturas que permitam a implementação de alguns dos sistemas descritos no capítulo 2. Serão descritas diferentes abordagens que permitem tirar partido da utilização de sistemas operativos e *middlewares* no desenvolvimento das diferentes aplicações. Será abordado o sistema de travagem.

No sexto e último capítulo, são apresentadas as conclusões referentes ao trabalho realizado.

Capítulo 2

Sistemas embarcados

2.1 - Introdução

Os sistemas electrónicos embarcados são responsáveis pela implementação de uma grande variedade de funcionalidades incorporadas nos veículos. Cada um destas funcionalidades possui diferentes necessidades de desempenho e/ou segurança. Assim, com base nas propriedades correspondentes tais como arquitecturas, serviços e restrições, os sistemas electrónicos embarcados no veículo são divididos em quatro diferentes domínios funcionais[3]:

1. Chassis
2. Drive train
3. Body
4. Interface homem-máquina, telemática e multimédia

Outras divisões podem ser consideradas, sendo aquela que foi apresentada a mais genérica. Nas próximas secções serão abordados cada um dos domínios, sendo feita uma descrição de alguns dos sistemas que os constituem.

2.1.1 - Chassis

O domínio Chassis inclui todos os sistemas que controlam a interacção dos diversos componentes do chassis do veículo (rodas, suspensão, etc) com a estrada de acordo com o pedido do condutor (de direcção, travagem ou aceleração), o perfil da estrada, e as condições ambientais. Sistemas de segurança activa, incluindo funções condução dinâmica e de assistência ao condutor estão incluídos neste domínio. Por estarem directamente relacionados com a segurança dos ocupantes, os sistemas presentes neste domínio exigem o controlo de diversas variáveis com requisitos temporais e soluções técnicas que assegurem que os mesmos são confiáveis. De seguida são enumeradas as características e requisitos relevantes referentes a este domínio [3]:

- Leis de controlo multivariavel em malha fechada;
- Alto poder computacional: sistemas de vírgula flutuante;

- Restrições temporais: cerca de 10ms;
- Frequência de aquisição: até 25 vezes por segundo;
- Sistemas multi-tarefa;
- Sistemas discretos e contínuos;
- Interação com outros domínios;

Os principais sistemas deste domínio são o sistema de travagem e o sistema de direcção, ambos cruciais para o controlo do veículo e segurança dos ocupantes. Em seguida serão descritos alguns destes subsistemas electrónicos pertencentes ao mesmo.

2.1.1.1 - Anti-lock braking system

O *Anti-lock braking system* (ABS) é um sistema electrónico controlado que tem como função impedir o bloqueio das rodas aquando de travagens bruscas, possibilitando desta forma a continuidade de controlo direccional do veículo. Geralmente numa situação de emergência, o pedal do travão é pisado fortemente, o que pode bloquear as rodas do veículo, fazendo com que este deixe de reagir aos movimentos do volante. Esta situação resulta na perda efectiva do controlo direccional do veículo. O sistema ABS controla o deslizamento de cada uma das rodas de forma a assegurar a operação numa zona de funcionamento que não resulte no bloqueio das mesmas. Como ilustrado na Figura 2.1, o sistema de travagem antibloqueio é constituído essencialmente por:

- Sensor do pedal do travão, responsável por detectar uma situação de travagem;
- Sensores de velocidade, responsáveis pela medição em tempo real da velocidade de cada roda do veículo;
- Unidade de controlo electrónico responsável pelos cálculos para a geração do sinal de controlo. Esta é geralmente conhecida pelas abreviaturas ECU (do inglês *Electronic Control Unit)*. Ao longo deste documento este termo será frequentemente usado.
- Modelador hidráulico, responsável pela modelização da pressão aplicar ao travão de cada uma das rodas;

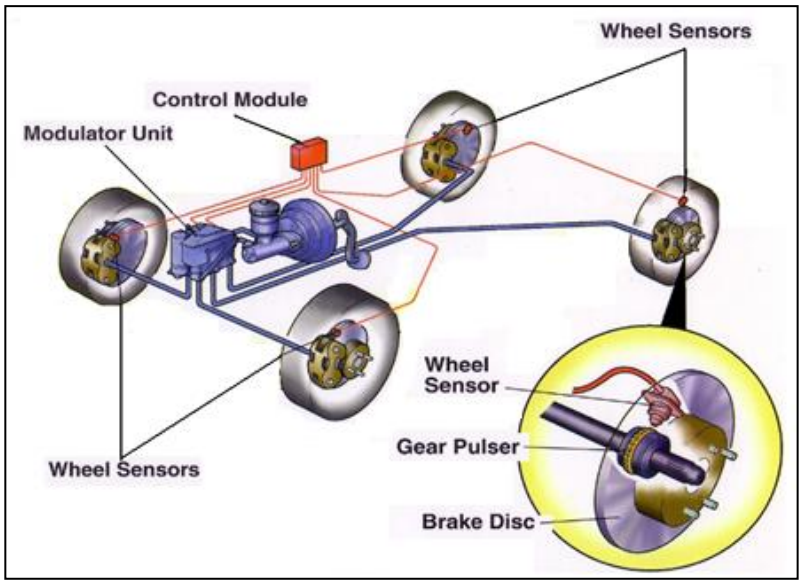

Figura 2.1- Componentes do sistema de ABS (adaptado de [4])

No que respeita ao funcionamento, podemos generalizar o funcionamento do sistema da seguinte forma:

1. Os sinais provenientes do sensor do pedal do travão permitem detectar que o veiculo se encontra numa situação de travagem.
2. Detectada uma situação de travagem o sistema de ABS entra em funcionamento. A velocidade angular de cada roda é medida através dos sensores de velocidade. Estes sensores enviam impulsos eléctricos para a unidade de comando electrónico, numa taxa proporcional a velocidade do veículo.
3. Através da velocidade de cada roda a unidade de controlo determina a velocidade de referência do veículo. Aquando de uma desaceleração brusca da velocidade de uma das rodas e consequente afastamento da velocidade desta relativamente a velocidade de referência do veículo, um estado de iminente bloqueio é detectado por parte da ECU.
4. Quando um estado de iminente bloqueio da roda é detectado, a ECU envia sinais eléctricos para o modelador hidráulico no sentido de reduzir a pressão aplicada ao travão da respectiva roda, evitando desta forma o bloqueio da mesma.

Os passos descritos anteriormente são efectuados enquanto o veículo se encontra em travagem no sentido de evitar o bloqueio das rodas. O sistema ABS termina o seu funcionamento caso o condutor liberte o pedal do travão ou caso o veiculo fique imóvel.

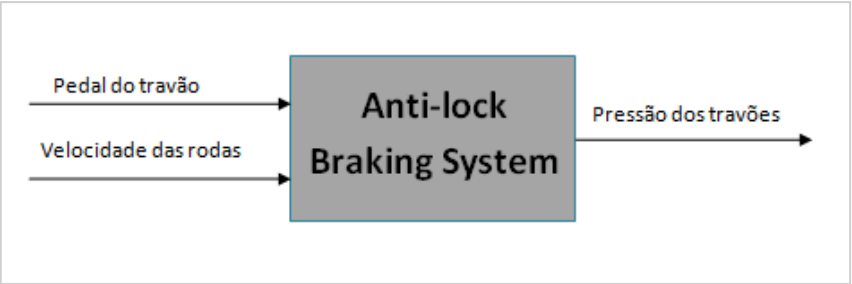

Figura 2.2 – Variáveis adquiridas e controladas pelo sistema ABS

2.1.1.2 - Electronic Brake force distribution

O Electronic Brake Force Distribution (EBD) é um sistema que tem como função garantir a distribuição eficaz da força de travagem pelas diferentes rodas do veiculo de forma a minimizar a distância de travagem. O que determina a eficiência de travagem é a aderência disponível entre as rodas e a superfície da estrada, sendo que, quanto maior a aderência entre a roda e o piso, maior será a força possível de aplicar ao travão sem provocar deslizamento. Por outro lado, factores como a localização do motor e da carga levam a que exista uma distribuição não uniforme do peso pelas diferentes rodas do veículo, originado níveis de aderência diferentes para cada uma das rodas. Assim durante a travagem, este sistema analisa instantaneamente a velocidade de cada uma das rodas de forma a detectar deslizamentos iminentes. É controlada automaticamente a quantidade de força a ser aplicada a cada um dos travões, no sentido de garantir que as rodas com maior eficácia de travagem recebem maior força de travagem, e rodas com menor eficácia recebem menos força de travagem.

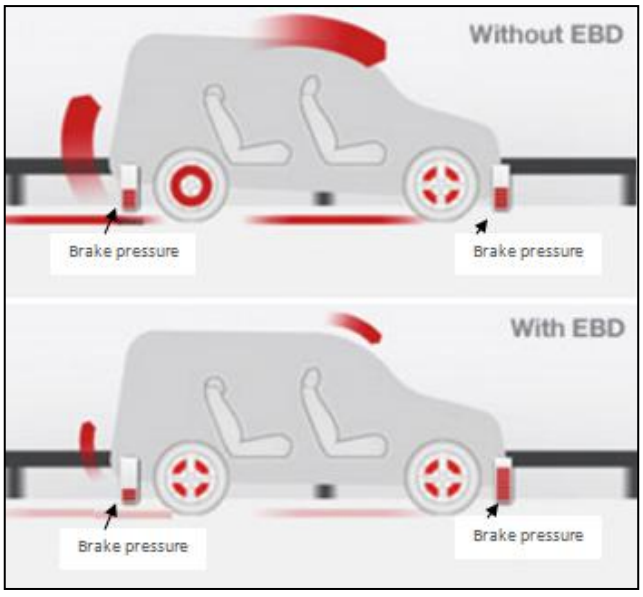

Figura 2.3 - Ilustração do funcionamento do sistema EBD[5]

Como ilustrado na Figura 2.3, aquando de uma travagem brusca a maior quantidade de peso na parte dianteira do veículo devido ao motor, leva o chassis do veículo a inclinar se ligeiramente para a frente, a suspensão dianteira é comprimida, e mais peso é transferido

para os pneus dianteiros. Se igual quantidade de força de travagem for aplicada para as rodas dianteiras e traseiras, as rodas traseiras perdem a aderência em primeiro lugar, potenciando ao bloqueio das mesmas. Maior quantidade de peso dos pneus da frente significa mais aderência disponível, e portanto maior quantidade de travagem poderá ser aplicada sem provocar deslizamentos. Assim, o sistema irá aplicar maior força de travagem aos travões da roda da frente minimizando a distância de paragem. Este sistema revela-se útil em situações de distribuição desigual do veículo, mas também, em situações de travagem em que as diferentes rodas se encontram sobre pisos com aderência diferente. O sistema garante que as rodas com maior aderência recebem maior força de travagem, aumentado assim a eficiência de travagem.

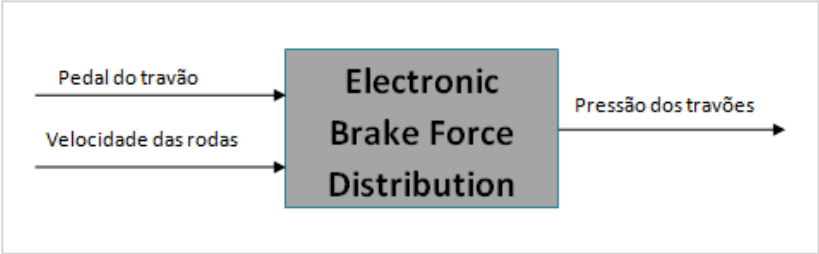

Figura 2.4 - Variáveis adquiridas e controladas pelo sistema EBD

2.1.1.3 - Electronic brake assist

O *electronic brake assist*(EBA) funciona em conjunto com o ABS, e tal como este, tem como objectivo auxiliar o condutor numa travagem de emergência. Aquando de uma travagem de emergência, o condutor poderá reagir rápido o suficiente, não efectuando no entanto a pressão necessária que optimiza uso dos travões. Tal poderá resultar num sub-aproveitamento da capacidade de travagem dos mesmos. Este tipo de situação é detectada pelo sistema EBA, que de seguida efectua o respectivo reforço da pressão nos travões. Para detectar este tipo de situações diferentes algoritmos poderão ser usados. O mais comum consiste na monitorização da velocidade a que pedal do travão é pressionado, que no caso de travagem de emergência é relativamente mais rápido do que numa situação normal.

Figura 2.5 – Variáveis adquiridas e controladas pelo sistema EBA

Assim com base na velocidade em que o pedal é pressionado, é efectuado um reforço da pressão aplicada aos travões de forma assegurar o uso integral das capacidades dos travões.

De referir que este sistema funciona em conjunto com o ABS e EBD, podendo ser considerado uma actualização ou característica adicional do sistema ABS.

2.1.1.4 - Traction Control System

O *Traction Control System* (TCS) é um sistema que tem como finalidade a manutenção permanente da tracção das rodas motrizes, garantindo a direcção e estabilidade do veículo. Durante o movimento normal de aceleração, a eficiência com que as forças possam ser transferidas para a estrada depende da tracção disponível entre as rodas e a superfície da estrada. Devido à falta de aderência entre as rodas motrizes e a superfície da estrada, o binário aplicado a cada umas das rodas motrizes num determinado instante poderá causar deslizamentos excessivos, o que resulta em derrapagens e consequentemente perda de tracção da respectiva roda. Esta situação ganha especial relevância quando o veiculo ou uma das rodas deste se desloca sobre superfícies com pouca aderência, como pisos molhados, gelo, etc. Perante as situações enumeradas, o uso de controlo de tracção é usado para limitar o deslizamento das rodas motrizes e desta forma garantir a continuidade de tracção.

Este sistema tem um funcionamento semelhante ao sistema ABS, diferenciando-se deste pelo facto de ser usado durante o movimento de aceleração, enquanto o ABS é usado durante a travagem. O TCS é controlado pela mesma ECU e utiliza os mesmos sensores e actuadores, dos sistemas anteriormente enumerados (ABS, EBD, EBA).

Genericamente o funcionamento deste sistema pode ser definido da seguinte forma:

1. Um sensor colocado no acelerador é responsável por detectar situações em que o veiculo se encontra em aceleração, ou seja, situações em que o motor esteja aplicar binário às rodas.
2. Sensores de velocidade medem continuamente a velocidade de cada uma das rodas do veículo, gerando um sinal que é proporcional à velocidade da roda.
3. A velocidade de cada uma das rodas é monitorizada de forma a detectar se estas rodam a velocidade diferente. Quando uma roda motriz possui uma velocidade relativamente superior as restantes, significa que o veiculo se encontra perante uma situação de perda de tracção.
4. Quando esta situação é detectada, um sinal é enviado para aumentar a pressão do travão da roda que se encontra sem tracção de forma a reduzir a aceleração da mesma. Como consequência desta acção, aquecimentos excessivos poderão ocorrer nos travões, pelo que paralelamente um sinal é enviado para a ECU do motor no sentido de reduzir o binário aplicado às rodas.

Os passos anteriores são executados sempre que o controlo de tracção se encontra em funcionamento. Quando o pedal do acelerador é largado o sistema deixa de estar em funcionamento.

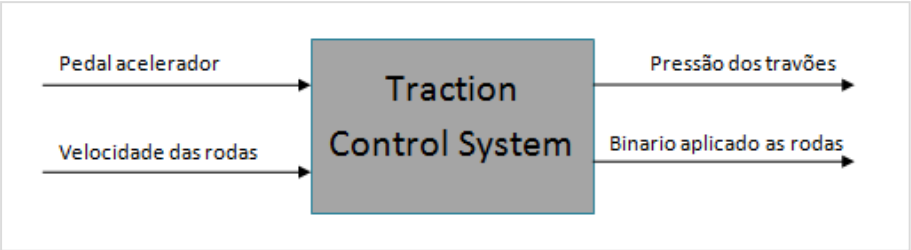

Figura 2.6 – Variáveis adquiridas e controladas pelo TCS

2.1.1.5 - Electronic Stability Program

O *Electronic Stability Program* (ESP) é um sistema que monitoriza continuamente a forma como o veículo responde á direcção pretendida pelo condutor. O objectivo é assegurar que a direcção real do veículo corresponda à direcção pretendida pelo condutor. Este sistema revela-se particularmente útil em situações de mudança brusca de direcção ou curvatura a velocidades excessivas, que poderão resultar em situações de subviragem (situação em que o veículo numa curva tende a sair de frente) e sobreviragem (situação em que o veículo tende a deslizar de traseira). O ESP detecta este tipo de situações e perante as mesmas efectua a devida correcção no sentido de repor o veículo na linha de viragem pretendida.

Os diversos sensores medem a velocidade e ângulo de viragem das rodas, a rotação do veículo sobre seu eixo vertical, e a aceleração lateral do veículo. O ESP monitoriza e compara instantaneamente a direcção pretendida pelo condutor (determinada através do ângulo medido volante) com a direcção real do veículo (determinada através da medição da aceleração lateral, da rotação do veículo, e da velocidades individual de cada roda), para assim confirmar que este se encontra a efectuar a linha de viragem pretendida. Caso contrário, o ESP envia sinais para os travões de forma aplicar pressão de travagem na roda adequada e efectuar ajustes na aceleração do veículo até a estabilidade direccional do veículo estar garantida. No sentido de optimizar o funcionamento do sistema geralmente são também enviados comandos para a ECU do motor. Na Figura 2.7(b) encontra-se ilustrado o funcionamento do ESP.

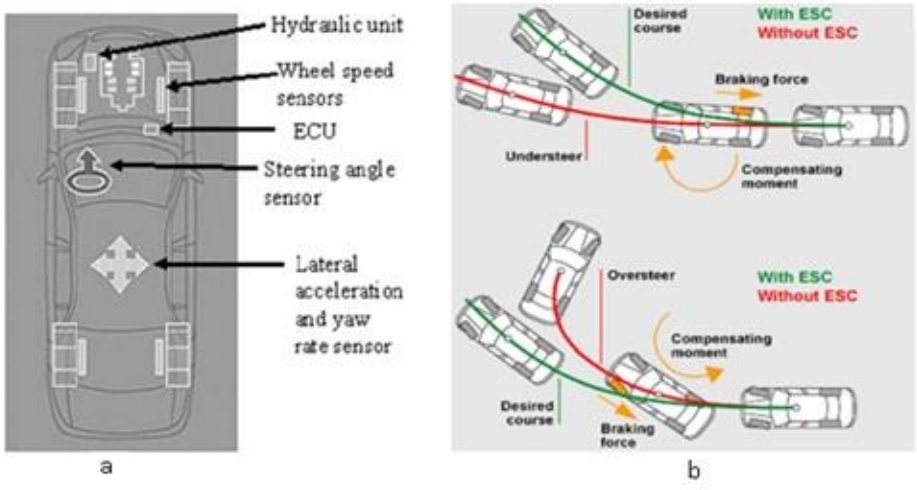

Figura 2.7 - (a) Componentes do ESP; (b) Funcionamento do ESP [6]

Perante uma situação de subviragem o ESP acciona automaticamente o travão da roda traseira interior à curva. Em caso de sobreviragem, o ESP acciona momentaneamente o travão da roda dianteira exterior à curva. Em ambas as situações a intervenção dos travões poderá também ser acompanhada por ajustes no binário aplicado às rodas, através do envio de sinais para a ECU do motor. De referir que o funcionamento do ESP não depende do condutor, pelo que quando este entra em funcionamento um sinal é enviado para a ECU do painel de instrumentos com a finalidade de gerar um aviso ao condutor relativamente à sua activação.

O ESP encontra-se geralmente incorporado na mesma ECU dos sistemas ABS e TCS.

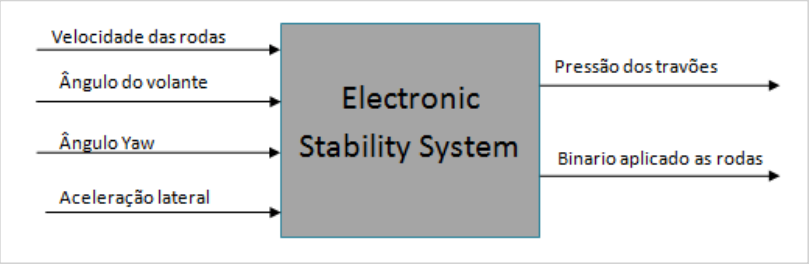

Figura 2.8 – Variáveis adquiridas e controladas pelo ESP

2.1.1.6 - Sistema de direcção assistida electrónico

O sistema de direcção assistida tem como função fornecer assistência ao condutor durante acções de viragem de forma a reduzir a força necessária a aplicar pelo condutor durante esta operação. Este faz parte do sistema direcção constituído também pelo volante, pela coluna da direcção e pelas barras transversais que se encontram ligadas nas suas extremidades às rodas.

O sistema de direcção assistida electrónico inclui um sensor de binário, uma unidade de controlo electrónica e um motor eléctrico acoplado a coluna de direcção, que em conjunto com os restantes componentes mecânicos do sistema de direcção, fornecem ao condutor a referida direcção assistida.

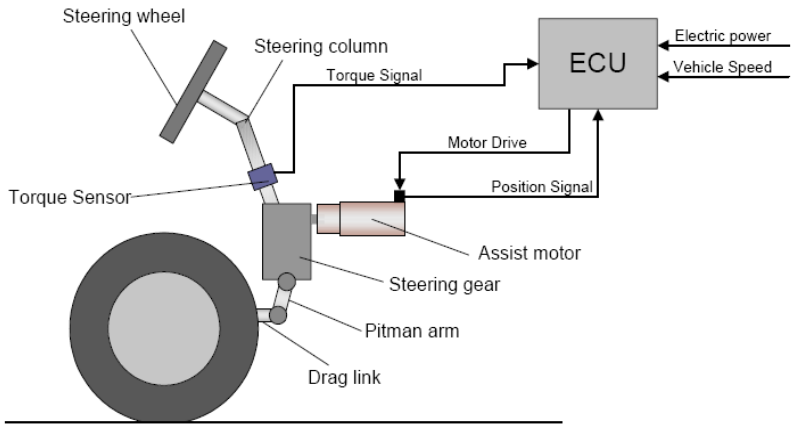

Figura 2.9 - Componentes do sistema de direcção assistida [6]

O sensor de binário detecta os movimentos realizados pelo condutor, medindo a amplitude e a direcção do binário aplicado ao volante, sendo os valores medidos por este enviados para a unidade de controlo, que os monitoriza. Em função dos valores recebidos, um sinal de comando é gerado por parte da unidade de controlo no sentido de definir a assistência adequada a ser dada a cada momento, sendo este valor modelado em função da velocidade actual do veículo. Tal facto advém da força necessária para mover o volante estar directamente relacionada com a velocidade actual do veículo. Assim, para cada valor de binário e velocidade, é determinada a corrente a aplicar ao motor eléctrico de forma a fazê-lo rodar e consequentemente mover a direcção, reduzindo a força necessária aplicar por parte do condutor para a realização desta tarefa.

No sentido de optimizar o funcionamento deste sistema, outras variáveis podem também ser adquiridas tais como a posição do motor eléctrico. De referir que o motor eléctrico é alimentado por uma bateria, permitindo assim uma total independência em relação ao motor do veiculo do ponto de vista funcional. Desta forma, o motor eléctrico da direcção só entra em funcionamento quando necessário e permite assistência de direcção mesmo com o veiculo desligado.

2.1.1.7 - Adaptive Cruise control

Adaptive Cruise Control (ACC) é um sistema que controla automaticamente a velocidade do veículo em função do valor pretendido pelo condutor e, quando aplicável, efectua ajustes nesta de forma a manter uma distância predefinida em relação ao veículo da frente.

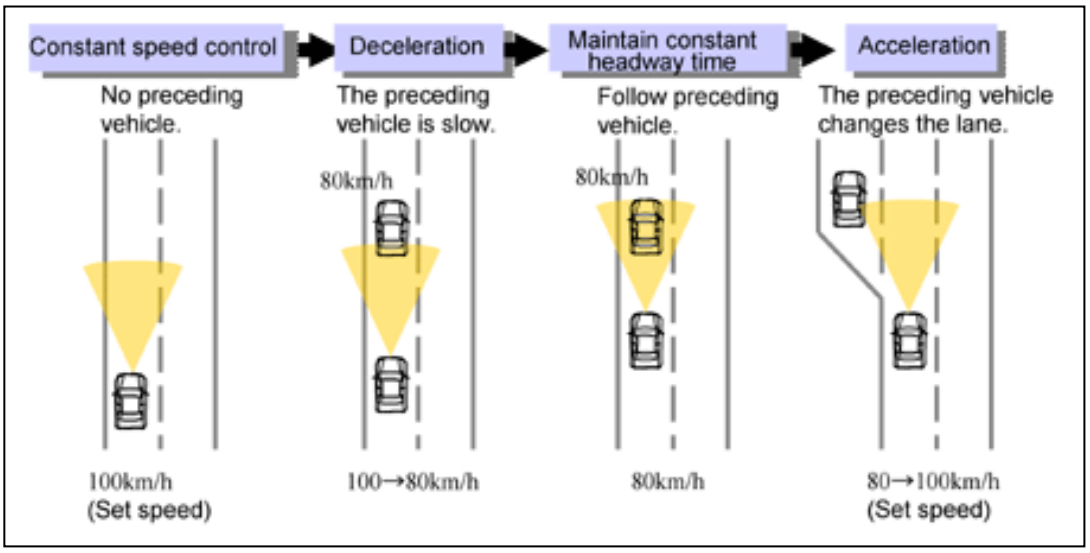

Figura 2.10 - Funcionamento do sistema ACC[7]

O funcionamento deste sistema assenta em três etapas:

1. Quando o condutor coloca o veículo em modo *adaptive cruise control*, insere o valor da velocidade desejada e da distância pretendida em relação ao veiculo da frente. Estes valores ficam guardados na memória da unidade de controlo electrónico do *Adaptive Cruise Control*.

2. A unidade de controlo recebe continuamente os impulsos dos sensores de velocidade colocados nas rodas. Através do uso de sensores de radar é também fornecida a distância para veículo da frente, caso este se encontre no campo de visão do veículo e na mesma faixa de rodagem.

3. A ECU do ACC compara os valores recebidos com os valores guardados em memória. Em função da comparação efectuada, envia sinais para a ECU do motor no sentido de definir o binário que deve ser aplicado as rodas de forma a manter a velocidade deste o mais próximo do pretendido. Quando um veículo mais lento é detectado na mesma faixa de rodagem, a ECU do ACC envia sinais para a ECU do motor e para a ECU dos travões no sentido de efectuar ajustes na velocidade para a manutenção de uma distância em relação ao veículo da frente em torno do valor definido.

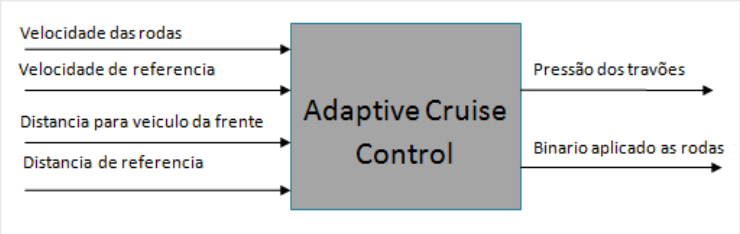

Figura 2.11 – Variáveis adquiridas e controladas pelo sistema ACC

O sistema ACC constitui uma evolução do sistema *Cruise Control* (CC). Este último apenas controla automaticamente a velocidade do veículo em função da velocidade indicada e pretendida pelo condutor.

2.1.2 - Drive train

Do domínio *drive train* fazem parte os subsistemas responsáveis pela propulsão do veículo, nomeadamente os sistemas que participam na geração de energia no motor e na sua transmissão para o eixo motriz e para as rodas. Tais funções são realizadas recorrendo a diferentes componentes mecânicos e eléctricos. Convencionalmente esses componentes incluem uma fonte de energia, um motor eléctrico, uma caixa de velocidades e um diferencial. Configurações mais recentes permitem abdicar de alguns componentes mecânicos tornando o sistema geral mais compacto e leve.

O domínio *drive train*, no caso de veículos eléctricos, pode ser dividido em três grandes subsistemas[8]:

- Sistema de propulsão eléctrico: inclui o motor eléctrico, os drivers de potência, a unidade de controlo electrónico, a transmissão e o diferencial. É responsável pela transmissão de energia mecânica as rodas e constitui a par dos sistemas de travagem e de direcção, um dos sistemas fundamentais para o devido funcionamento do veículo
- Sistema de gestão e fornecimento de energia: inclui as baterias que fornecem energia eléctrica ao motor eléctrico de tracção, e uma unidade de controlo de

energia responsável por interagir com a unidade de controlo do motor no sentido de assegurar reabastecimento de energia aquando de travagens regenerativas.

- Sistema auxiliar de energia: responsável pelo fornecimento de energia para os restantes componentes electrónicos que não associados ao funcionamento do motor. Este sistema poderá estar incluído no sistema de gestão e fornecimento de energia.

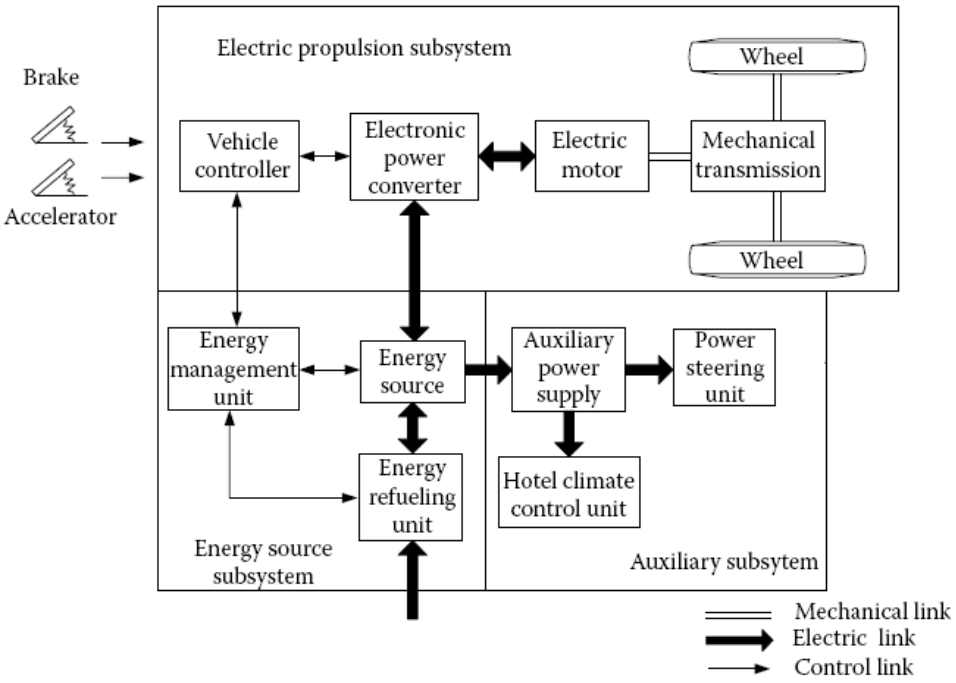

Figura 2.12 - *Drive train* do veículo eléctrico [8]

Este domínio, tal como o *Chassis*, requer o controlo em malha fechada e rigorosas restrições temporais de diversas variáveis, uma vez que é responsável pelo controlo do motor de acordo com as ordens do condutor do veículo, e de acordo com as ordens recebidas de outros sistemas que requerem o uso do motor. Os sistemas presentes neste domínio realizam uma forte interacção e troca de dados com os domínios do *Body* e de *Chassis*. De seguida são enumeradas as características e requisitos relevantes referentes a este domínio [3]:

- Leis de controlo em malha fechada multivariável;
- Alto poder computacional: sistemas de vírgula flutuante;
- Rigorosas restrições temporais: cerca de 10ms;
- Períodos de aquisição entre 0.1 a 5ms;
- Sistemas multi-tarefa;
- Sistemas discretos e contínuos;
- Interacção com outros domínios;

2.1.2.1 - Unidade de controlo do motor

A unidade de controlo electrónico do motor é das unidades com maior importância presente no veículo. Para se entender as funcionalidades desta unidade de controlo, é necessário entender o funcionamento de todo o sistema de propulsão. Tal advêm do facto da unidade de controlo do motor interagir com todos os componentes deste sistema, como ilustrado na Figura 2.13. Para além da unidade de controlo do motor e do motor propriamente dito, o sistema de propulsão inclui geralmente um conversor electrónico, uma transmissão e um diferencial.

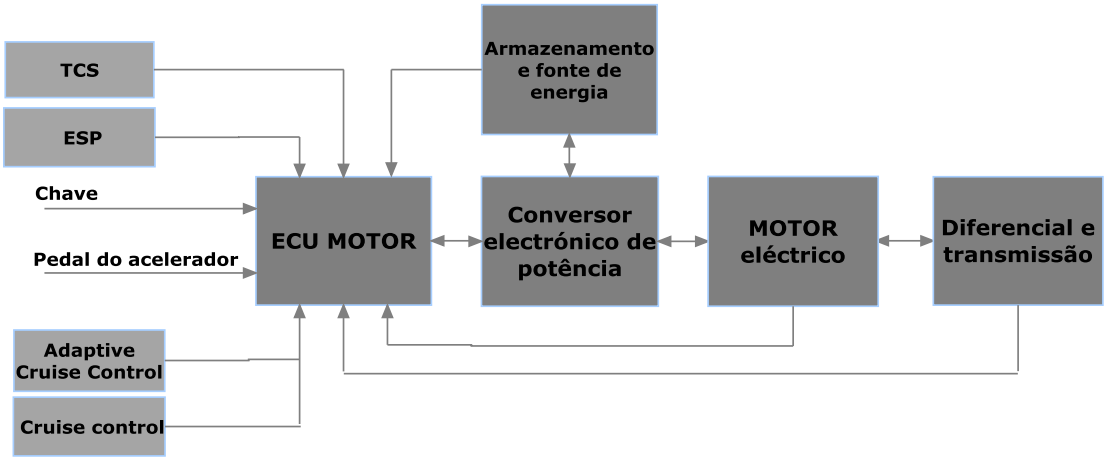

Figura 2.13 - Sistema de propulsão eléctrico [8]

A unidade de controlo electrónico constitui o ”cérebro” do sistema. É responsável por monitorizar os sinais provenientes dos pedais e dos diferentes sistemas que requerem o controlo da potência instantâneo do motor (*Cruise control*, *Adaptive Cruise Control*, TCS, ESC). Em função dos diferentes sinais recebidos, um valor de referência de binário e/ou velocidade é calculado. Em função dos valores de referência, a unidade de controlo gera os sinais apropriados para o conversor de potência do motor. Dependendo da topologia do conversor usado e do tipo de controlo implementado, as variáveis de saída da unidade de controlo podem ser desde correntes, tensões, ângulos de disparo [8],[9]. De referir, que a unidade de controlo recebe também variáveis provenientes do motor, nomeadamente a corrente, a tensão e temperatura deste. A medição destes valores permitem conhecer o estado actual do motor, de que forma evolui e reage aos comandos recebidos.

O conversor de potência do motor recebe os sinais de comando provenientes da unidade de controlo, sendo responsável por alimentar o motor eléctrico com valores de tensão e corrente adequados para assegurar o binário pretendido.

O motor eléctrico é o responsável directo pela movimentação do veículo. Este converte energia eléctrica em energia mecânica, sendo esta última responsável pelo movimento do veículo. No caso de travagem regenerativa, este funciona com gerador, possibilitando o armazenamento de energia eléctrica nas baterias, gerada a partir da energia da mecânica das rodas. Existem diferentes tipos de motores no mercado e que podem ser usados para este tipo de aplicação. Visto que o motor é o responsável directo pela aplicação de binário as rodas, a

performance do veículo depende directamente da escolha do motor. Os critérios essências na escolha de um motor podem ser encontrados em [9].

Por fim, a transmissão e o diferencial, são os componentes mecânicos convencionalmente usados para aplicar o binário gerado pelo motor as diferentes rodas do veículo.

2.1.3 - Body

O domínio *Body* incorpora os sistemas responsáveis pela implementação de funções de conforto e que não estão relacionados com o controle da dinâmica do veículo. Ao contrário dos domínios de *Chassis* e *Drive train*, as funções incluídas neste domínio, não exigem por norma restrições de desempenho rigorosas e, do ponto de vista de segurança, não representam uma parte crítica do sistema global do veículo. Como exemplo, de sistemas pertencentes a este domínio, temos o controlo dos faróis, dos espelhos e dos assentos.

2.1.3.1 - Sistemas de iluminação exterior

O sistema de iluminação exterior consiste de um conjunto de faróis inseridos no exterior do veículo. É o sistema responsável por permitir ao condutor a visualização do ambiente envolvente em condições de fraca visibilidade. Por outro lado permite sinalizar a presença do veículo e algumas das intenções do condutor, tais como acções de travagem e mudança de direcção. Estudos [10] demonstram que a maior incidência de acidentes rodoviários ocorre durante a noite, muitos dos quais relacionados com a falta de visibilidade. Com instituto de resolver esta questão, diferentes propostas tem vindo a ser a desenvolvidos quer a nível da implementação de lâmpadas mais eficientes, quer ao nível do desenvolvimento de sistemas de iluminação inteligentes.

Geralmente a activação de um determinado farol ou conjunto de faróis é realizada como resposta a uma acção efectuada pelo condutor. Esta acção poderá ser por exemplo, o premir um botão para ligar um determinado conjunto de faróis ou pressionar o pedal do travão, que automaticamente activa a luz vermelha da traseira do veículo. Consoante a acção realizada pelo condutor, um comando específico é enviado para a unidade de controlo eléctrico deste sistema. Após identificar o comando recebido, a unidade de controlo gera um sinal no sentido de activar o respectivo interruptor que irá permitir a alimentação do farol. A unidade de controlo electrónico poderá também efectuar a monitorização da tensão e corrente de cada uma das luzes no sentido de detectar possíveis anomalias que serão posteriormente reportadas ao condutor através do painel de instrumentos.

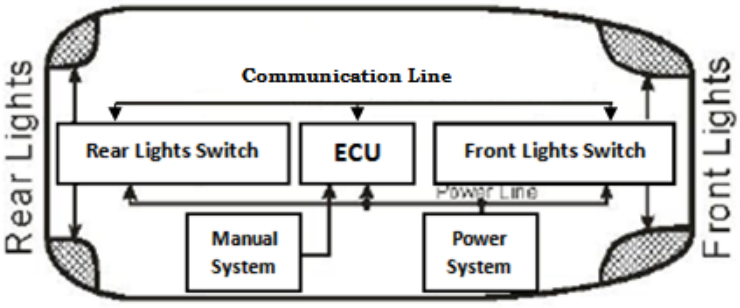

Figura 2.14 - Componentes do sistema de iluminação (adaptado de [11])

Sistemas de iluminação mais recentes permitem o controlo da intensidade luminosa dos faróis dianteiros e traseiros em função das condições de luminosidade envolventes, evitando desta forma gastos de energia desnecessários. O interesse deste sistema reside no facto de que, em determinadas circunstâncias é necessário ou mesmo obrigatório o uso do sistema de iluminação, não sendo no entanto necessário o emprego de toda a intensidade luminosa. Concretamente, se o veiculo se desloca de dia ou de noite, ou sobre condições climatéricas adversas como nevoeiro e chuva, diferentes necessidades de iluminação são esperadas. Desta forma este sistema utiliza um sensor de luminosidade responsável por fornecer informações sobre este parâmetro. Em função da luminosidade actual, um valor de referência é gerado por parte da unidade de controlo no sentido de definir a intensidade luminosa necessária para as condições actuais. Diversas técnicas e algoritmos podem ser usados para efectuar referido controlo da intensidade luminosa [11]. Quando comparado com os sistemas convencionais, este sistema permite uma redução considerável da energia dispensada.

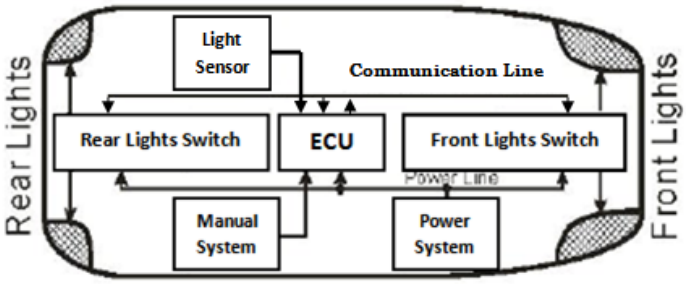

Figura 2.15 - Sistema de iluminação com controlo da intensidade luminosa [11]

Um outro sistema de iluminação inteligente faz uso de faróis direccionais/giratórios de forma efectuar ajustes na sua posição vertical e horizontal destes, permitindo um controlo dinâmico da direcção da luz emitida.

O controlo da posição vertical do farol é útil em situações de distribuição desigual de carga no veículo, nomeadamente quando a maior quantidade de peso presente na parte traseira causa um levantamento da parte dianteira do veículo. Nesta situação os feixes de luz não incidem directamente sobre a superfície da estrada, resultando numa diminuição da visibilidade assegurada e num possível encadeamento dos condutores que se encontram em sentido contrário. Este sistema requer acréscimo de dois sensores de pressão e dois motores eléctricos em relação aos sistemas de iluminação convencionais. Os sensores são colocados na suspensão traseira e dianteira do veículo, de forma a detectar situações de distribuição de carga desigual. A unidade de controlo monitoriza os valores de cada sensor e quando necessário acciona o sinal de comando dos motores eléctricos de modo efectuar a correcção da posição vertical do respectivo farol.

Por outro lado, o controlo da posição horizontal do farol permite uma maior eficiência no uso do sistema de iluminação em situações de viragem. A medição do ângulo de viragem do volante e do momento angular do veículo, permitem a unidade de controlo electrónico conhecer a direcção actual. Em função desta, a ECU envia comandos aos motores eléctricos para efectuar ajustes na posição dos faróis, garantindo que a direcção dos feixes de luz por estes emitidos coincide com a direcção actual do veículo.

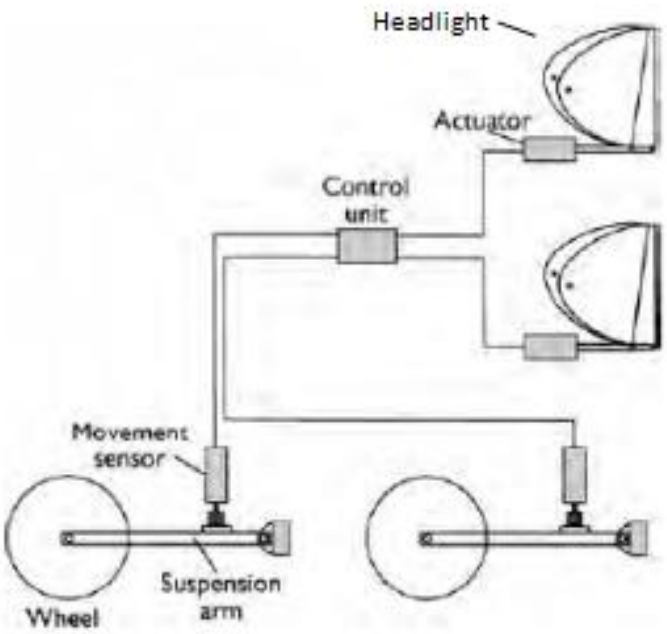

Figura 2.16 - Sistema de faróis direccionais[8]

2.1.4 - Interface homem-máquina, Telemática e multimédia

O domínio Multimédia, Telemática e interface homem-máquina inclui de forma genérica funções de interface, bem como funções de informação e entretenimento. Os sistemas presentes neste domínio são responsáveis pela interacção entre o condutor com inúmeras funções incorporadas no veículo, fornecendo informações sobre o estado do veículo, sobre os pedidos efectuados pelos ocupantes e sobre o estado dos diversos dispositivos multimédia e telemáticos incorporados. Os sistemas aqui presentes permitem não só a interacção com outros sistemas no interior do veículo mas também a troca de informações com o mundo externo através de sistemas como o GPS, sistemas de navegação, rádio, etc. O que mais caracteriza este domínio é enorme quantidade de dados transmitidos para os outros domínios e para o mundo exterior, o que se reflecte em fortes exigências de largura de banda. A forte interacção dos sistemas aqui presentes com o mundo exterior, traduz-se também em fortes requisitos de segurança no sentido de ser garantida a integridade e confiabilidade dos dados.

2.1.4.1 - Painel de instrumentos

O painel de instrumentos tem como propósito fornecer ao condutor informações pertinentes necessárias para operar o veículo com êxito. Este consiste num conjuntos de indicadores que registam quantidades e valores relevantes para o condutor, e num conjunto de luzes de sinalização que alertam o condutor para situações anómalas ou sobre a activação e desactivação de algum dispositivo do veículo.

As principais variáveis e indicadores que constam do painel de instrumentos são:

- Velocidade instantânea do veículo;
- Temperatura do motor;
- Rotação instantânea do motor;

- Indicador de luzes ligadas;
- Indicador da carga da bateria;
- Indicador de entrada em funcionamento do ABS, TCS e ESP, entre outros;

Para obter as variáveis anteriormente enumeradas, este sistema interage com diversos sensores distribuídos e com outros sistemas incorporados no veículo.

2.1.4.2 - Sistema de monitorização da pressão dos pneus

O sistema de monitorização da pressão dos pneus permite o controlo da pressão dos pneus, mediante o envio de alertas para o condutor quando a pressão de um dado pneu se encontra abaixo do tolerável. Os pneus são projectados para operar dentro de uma determinada faixa de pressão, sendo esta geralmente definida no manual do veículo e a que permite a melhor combinação de conforto, capacidade de carga e resistência do pneu. Genericamente e do ponto de vista operacional, existem dois métodos diferentes de realizar o referido controlo de pressão: métodos directos e métodos indirectos.

Nos métodos directos a pressão é medida directamente no pneu através do uso de sensores colocados no seu interior. Estes sensores enviam instantaneamente o valor da pressão via rádio frequência para uma antena colocada próximo das rodas, que posteriormente encaminha os valores medidos para a unidade de controlo do sistema de monitorização. Quando a unidade de controlo detecta que a pressão de um dado pneu se encontra abaixo do tolerável, envia um sinal para a unidade de controlo do painel de instrumentos ou para um painel de visualização independente específico para a monitorização da pressão, gerando um alerta visível ao condutor. Existem outras soluções do ponto de vista da implementação no entanto o princípio de funcionamento é idêntico.

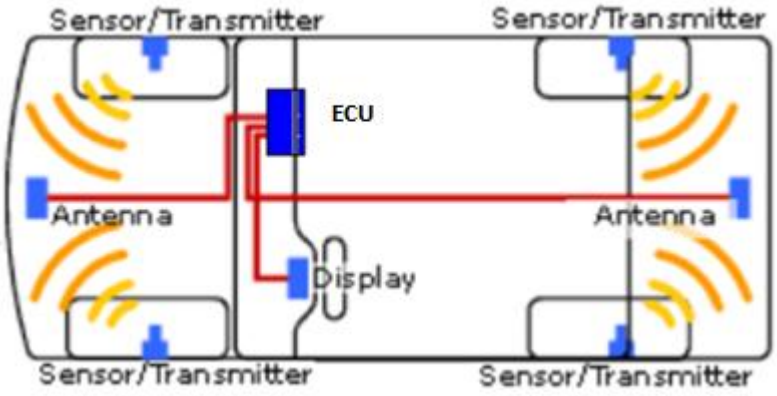

Figura 2.17 - Funcionamento do sistema de monitorização da pressão dos pneus (adaptado de [12])

Os métodos indirectos não usam qualquer tipo de sensores de pressão. Usam o facto de que, um pneu com pressão insuficiente tem um diâmetro ligeiramente menor do que um pneu com pressão adequada, logo, tem que rodar a uma velocidade angular maior para cobrir a mesma distância. Comparando então a velocidade de cada uma das rodas é possível detectar quando uma determinada roda se encontra com a pressão inferior, sem a necessidade do uso de sensores de pressão. Quando é detectado que uma das rodas se encontra a rodar a uma

velocidade ligeiramente inferior, é gerado um alerta para que o condutor confirme a pressão dos pneus.

2.2 - Requisitos de comunicação e redes

Os sistemas embebidos são responsáveis pela implementação de uma série de funções presentes nos veículos. A realização destas funções requer a troca de dados e sinais entre diversos sensores, actuadores e unidades de controlo, distribuídos pelo veículo. Neste sentido é necessário assegurar a efectiva interligação entre os diversos dispositivos. Inicialmente, nos veículos mais antigos, este tipo de interligação era assegurada maioritariamente por ligações ponto a ponto. No entanto, o crescente aparecimento de novos sistemas electrónicos nos automóveis, bem como o aumento da complexidade destes, resultou num aumento exponencial do número de cabos necessários para interligar os diversos dispositivos. No sentido de reduzir o número de cabos necessários, bem com os custos associados a esta tarefa, foi introduzido o conceito de redes de comunicação. Ao contrário de uma conexão ponto-a-ponto , uma rede de campo permite ligar vários periféricos e controladores sobre o mesmo conjunto de fios.

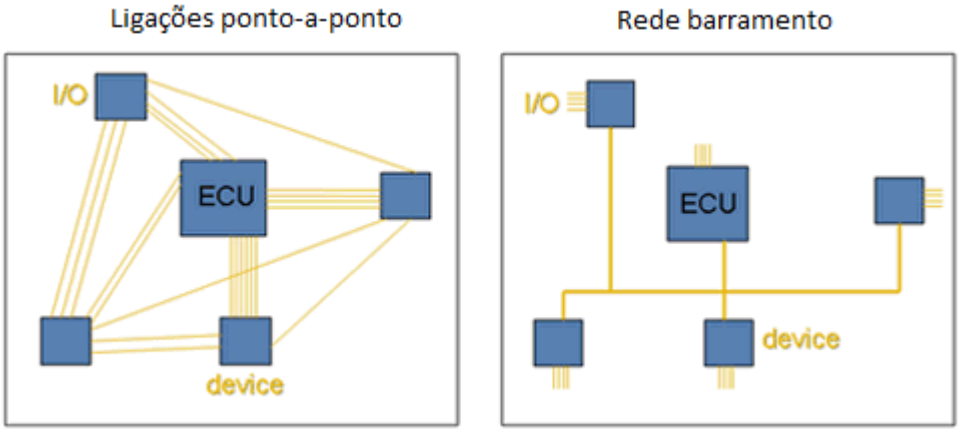

Figura 2.18 – Ligações ponto-a-ponto versus rede barramento (adaptado de [13])

A utilização de uma rede de comunicações permite um fácil acréscimo de novos dispositivos ao barramento e no caso de falha de um deles, os restantes não são afectados.

Do ponto de vista técnico diferentes requisitos de comunicação são considerados, sendo que estes são mais ou menos relevantes dependendo dos domínios ou sistemas em questão. Geralmente os requisitos considerados são [14]:

- **Determinismo**
  Um sistema de comunicação determinístico permite saber o tempo máximo de transmissão de uma mensagem. Este requisito é importante em sistemas *hard* e *soft real time*, onde o devido funcionamento do sistema exige o cumprimento de *deadlines* na execução das tarefas.

- **Tolerância a falhas**
  Quando o sistema não se comporta de acordo com as suas especificações, o seu comportamento incorrecto poderá ser causador de falhas, tais como omissão e duplicação de mensagens, falha na linha de comunicação, etc. Um sistema de comunicação tolerante a falhas utiliza redundância de hardware e arquitecturas de software que lhe permitem tolerar falhas.
- **Largura de banda**
  A largura de banda refere–se à quantidade de bits/s que a rede de comunicação suporta e é capaz transmitir. Este requisito é essencial para alguns tipos de sistemas devido a grande quantidade de dados que estes trocam com outros sistemas. A *ASE* (*Society of Automotive Engineers)* [15] agrupou as diferentes redes automóveis em diferentes classes comunicação, em função da sua capacidade de largura de banda: classe A (menos de 10 kb / s), classe B (entre 10 - 125 kb/s), classe C (entre 125 Kb/s e 1 Mb/s), e classe D (superior a 1Mb/s).
- **Flexibilidade**
  Este requisito define a capacidade do sistema de comunicação em lidar e se adaptar a diferentes condições da rede (por exemplo diferentes cargas na rede, falhas esporádicas).
- **Segurança**
  Este requisito refere-se ao controlo do acesso aos dados por terceiros. É importante no sentido de assegurar a integridade e confiabilidade dos dados transmitidos.

A importância de cada um dos requesitos enumerados varia consoante o domínio ou sistema em questão. Concretamente, tolerância a falhas, determinismo temporal e a necessidade de elevada largura de banda são requisitos cruciais dos domínios *Chassis* e *Power Train*. Tal deve–se ao facto de muitos dos sistemas presentes nestes domínios estarem relacionados com o controlo da dinâmica do veiculo e com a segurança dos ocupantes, pelo que, exigem o cumprimento das tarefas em instantes de tempo concretos e precisos.

O domínio Body caracteriza-se pela necessidade de largura de banda (Class B) e flexibilidade, devido a grande quantidade de dispositivos presentes neste domínio.

Por fim o domínio de interface Homem-Máquina, Telemática e Multimédia, requer elevada largura de banda devido a enorme troca de dados que os sistemas presentes neste domínio realizam com os diferentes sistemas do veículo.

Na Tabela 2.1 encontra-se ilustrado os requisitos de comunicação de cada um dos domínios abordados.

Tabela 2.1- Requisitos de comunicação dos diferentes domínios[14]

Domínio	*Requisitos de comunicação*				
	Tolerância a Falhas	**Determinismo**	**Largura de banda**	**Flexibilidade**	**Segurança**
Power train	Algum	Sim	Sim	Sim	Não
Chassis	Sim	Sim	Algum	Não	Não
Body	Não	Algum	Algum	Sim	Não
HMI, Telemática e Multimédia	Não	Algum	Sim	Sim	Algum

Actualmente existe uma grande variedade de redes de comunicação, desde redes barramento e wireless, e que são utilizadas nos diferentes domínios. Na tabela 2.2 encontra-se ilustrado as principais redes usadas nos veículos e a sua utilização nos diferentes domínios. Quando é utilizado o símbolo "++" significa que a rede é largamente utilizada nesse domínio, "+-" significa que a rede é utilizada mas não é predominante nesse domínio e "--" a rede não é utilizada no domínio em questão.

Tabela 2.2 - Principais redes de comunicação usadas em veículos[14]

Domínio	***Redes barramento actuais***			***Redes barramento futuras***			***Multimédia***	***Wireless***	
	LIN	CAN	Byteflight	TTP	TT-CAN	FlexRay	Most	Bluetooth	ZigBee
Chassis	--	++	--	+-	++	--	--	--	--
Powertrain	--	++	++	+-	++	+-	--	--	--
Body	++	++	--	--	--	--	--	--	++
HMI, telemática e multimédia	--	--	--	--	--	--	++	++	++

As redes CAN e TT-CAN são utilizadas principalmente nos sistemas pertencentes aos domínios com fortes restrições de desempenho como os domínios *Chassis* e *Power train*. A rede LIN é essencialmente usada no domínio Body e as redes MOST, *Bluetooth* e *ZigBee* no domínio interface homem-máquina, telemática e multimédia. A rede FlexRay, embora seja usada nos domínios *Chassis* e *Power train*, é direccionada para aplicações x-by-wire, um domínio que não foi considerado neste documento. Em seguida serão descritas três das redes mencionadas anteriormente.

2.2.1 - Controller Area Network

Controller Area Network (CAN) é uma rede de comunicações de barramento série desenvolvido pela Bosch no início de 1980 [16]. Surgiu com o intuito de reduzir as ligações

eléctricas entre as diferentes unidades de controlo presentes nos veículos. Actualmente constitui um padrão para a transmissão de dados em aplicações embarcadas nos veículos devido ao seu baixo custo, robustez, tolerância a falhas e isolamento de erros. Estas características fazem com que seja hoje usada nas mais diversas áreas da automação e controlo. A sua regulamentação específica para as diversas aplicações em automóveis foi padronizada pela SAE. É usada como rede de rede de classe C para o controle em tempo real nos domínios de *Drive train* e *Chassis*, e como rede de classe B no domínio *Body*.

A rede CAN foi dividida em duas camadas diferentes, obedecendo ao modelo OSI: camada de ligação de dados e camada física. A camada física do barramento CAN é formada por um barramento formado por um par de fios trançados. Cada ponta terminal terá uma resistência representativa da impedância característica da rede. Os fios deste barramento são denominados CAN_H (*High*) e CAN_L (*Low*) e os sinais eléctricos são representados pelo nível recessivo (valor lógico 1) e dominante (valor lógico 0). Sempre que ambos estão presentes no meio, é sempre o bit dominante que prevalece. Se todos os nós transmitirem bits recessivos, diz-se que o barramento está no estado recessivo. Assim que um dos nós transmitir um bit dominante o barramento passa a considerar-se no estado dominante.

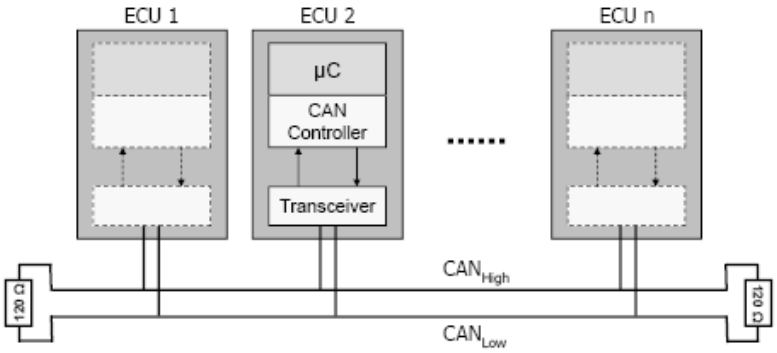

Figura 2.19 – Rede CAN interligando diferentes ECUs [17]

Um ponto forte do CAN reside no uso de sinais diferenciais balanceados, pois o ruído é reduzido o que permite altas taxas de comunicação. Balanceado significa que o fluxo de corrente é igual em cada linha porém oposta em direcção, resultando em um cancelamento do efeito de campo que é a chave para baixas emissões de ruído.

A rede CAN utiliza uma abordagem multi-mestre onde todos os módulos podem ser mestres num dado instante. Entenda-se por mestre o modulo/nó que num dado instante esta aceder ao meio de transmissão e a transmitir uma mensagem, sendo que neste instante os restantes módulos são escravos. Esta abordagem permite assegurar uma maior disponibilidade da rede visto que uma possível falha num dos módulos não invalida que os restantes continuem a transmitir as suas mensagens.

As mensagens são enviadas em regime *broadcast*, onde todas as mensagens são enviadas para todos os módulos ligados ao barramento e ao mesmo tempo, o que permite assegurar uma grande consistência dos dados. Todos os nós da rede monitorizam continuamente o barramento e todas as mensagens que nele circulam de forma a identificar as mensagens que a eles se destinam. Concretamente, os diferentes nós poderão possuir filtros de mensagens que separam individualmente em cada nó as mensagens que lhes interessam das que não lhes

interessam. Assim fica assegurada uma optimização na tarefa de processamento de mensagens pelo processador do nó, pois apenas é interrompido quando recebe uma mensagem que de facto lhe é importante.

Um aspecto importante do CAN é ser um protocolo orientado a mensagem pelo que não há definição de nós nem de endereços, mas sim de mensagens. Estas mensagens são identificadas pelo uso de um identificador de mensagem (ID), o qual será único em toda a rede. Este aspecto permite uma grande flexibilidade do sistema: novos nós poderão ser adicionados ao sistema sem mudanças significativas de hardware /software dos outros nós. As mensagens são enviadas em tramas de dados e são geradas pelo nó que pretende transmitir os dados. Estas tramas são constituídas por diferentes campos com determinado número de bits como ilustra a Figura 2.20.

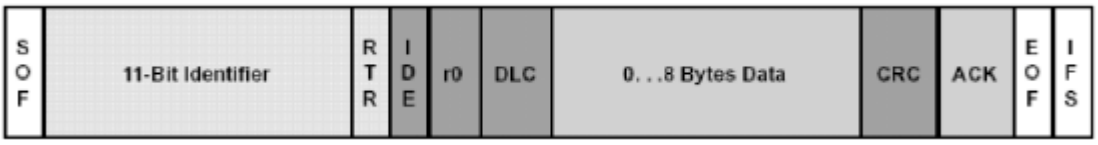

Figura 2.20 – Trama de dados do CAN [18]

A transferência de mensagens é feita através de diferentes tipos de tramas: tramas de dados, trama de pedido remoto, trama de erro e trama de sobrecarga (*overload*). Actualmente existem dois tipos de formato de trama de mensagens suportados pelo protocolo CAN: o CAN 2.0A com o campo identificador composto por 11 bits, e o CAN 2.0B com identificador composto por dois campos, um com 11 bits e outro com 18 bits (total de 29 bits).

Como mencionado anteriormente todos os módulos num dado instante podem ser mestres e enviar suas mensagens. Neste sentido é necessário assegurar mecanismos de arbitragem no excesso ao barramento para que apenas um modulo se encontre a transmitir e evitar possíveis colisões. Tais mecanismos são baseados no conceito CSMA/CD com NBDA (*Carrier Sense Multiple Access /Collision Detection).* Cada nó monitoriza o estado do barramento antes de transmitir uma determinada mensagem. Ao verificarem que o barramento se encontra livre todos os módulos poderão aceder ao meio de transmissão, sendo que o módulo com a mensagem de maior prioridade iniciará imediatamente a transmissão e os demais irão esperar até que o barramento fique livre novamente. O mesmo acontece numa situação em que dois módulos comecem a transmitir dados simultaneamente: o módulo com a mensagem de menor prioridade cessa sua transmissão e o módulo com a mensagem de maior prioridade continua enviando sua mensagem sem quaisquer perdas de tempo nem de dados. Consideremos o exemplo ilustrado na Figura 2.21.

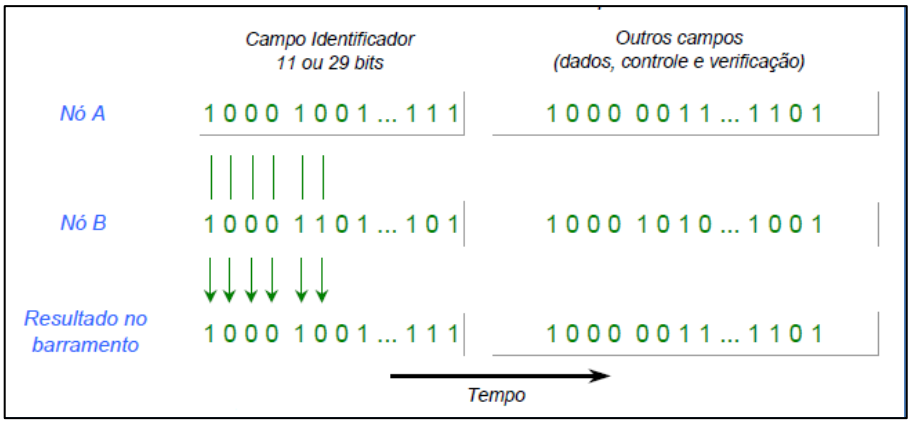

Figura 2.21 – Transmissão de mensagens no CAN

Os nós A e B começam a transmitir as suas mensagens em simultâneo. Aquando da transmissão do sexto bit do campo identificador, o nó A transmite um bit dominante e o nó B um bit recessivo. Como o bit dominante prevalece sobre o bit recessivo, o nó B ira detectar um bit dominante no barramento quando havia transmitido um bit recessivo. Então o nó B deixa de transmitir e torna-se nó receptor, enquanto o nó A continua normalmente com a sua transmissão. Este conceito permite que as mensagens sejam enviadas em função da sua prioridade independentemente do nó que as produziu.

2.2.2 - Local Interconnect Network

Local Area Network (LIN) é uma rede de comunicação série de baixo custo direccionada para interligar sistemas electrónicos distribuídos nos veículos. Foi especialmente projectada para funções com reduzidas necessidades de larga de banda (até 20Kbit/s). É considerada apropriada para redes de veículos automóveis Classe A e B pela SAE.

O barramento LIN é geralmente conectado entre o sensor ou actuador e uma unidade de controlo electrónico. Utiliza uma configuração de um único mestre e um ou mais escravos, sendo que o número máximo de escravos recomendado é de dezasseis [19]. O nó mestre é responsável pelo controlo e monitorização do barramento, incluindo a definição da velocidade de transmissão e funções de sincronização. Visto que a transferência de mensagens é sempre iniciativa do mestre não é necessário o emprego de mecanismos de arbitragem ou contenção no acesso ao barramento.

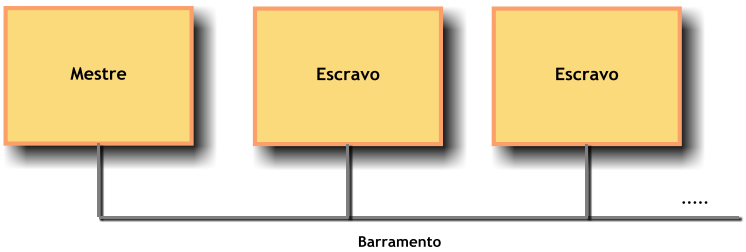

Figura 2.22 – Interligação de nós usando o barramento LIN

A implementação do barramento LIN é realizado recorrendo apenas a um único fio, o que se traduz em significativas emissões de ruído quando comparado com o par traçado usado no

barramento CAN. Para assegurar a imunidade a ruídos suficientes, a tensão de alimentação e o neutro da ECU são utilizadas como tensões de referência para o nível do barramento. Um nível de pelo menos 40% abaixo da tensão de alimentação é interpretada pelo receptor como um resultado lógico "0", enquanto um nível de pelo menos 60% acima da tensão de alimentação são interpretados como um valor lógico "1" [20].

A trama consiste num cabeçalho, um segmento de resposta e um campo referente ao Checksum. O cabeçalho é de comprimento fixo enquanto a parte da resposta consiste de 0-8 bytes de dados. Tal como ilustrado na Figura 2.23, o cabeçalho é dividido em três campos: quebra de sincronização, campo de sincronização e identificador.

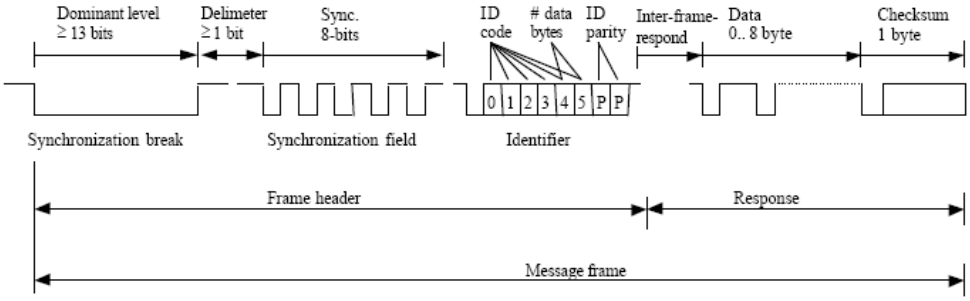

Figura 2.23 – Formato de trama LIN[3]

O campo de quebra de sincronização consiste de pelo menos 13 bis de zeros necessários para informar todos os nós escravos do inicio de uma transmissão no barramento. O campo de sincronização como o próprio nome indica, é usado pelos diferentes nós para efectuarem a sincronização dos seus relógios sempre que uma mensagem é recebida. Por fim, o identificador indica o conteúdo da parte de dados da mensagem, não fazendo qualquer tipo de referência ao destinatário.

A ordem de envio das tramas é definida num cronograma que é gerado off-line e é administrado on-line pelo mestre. Aquando da necessidade de enviar uma mensagem o nó mestre envia para o barramento uma trama com o referido cabeçalho. Ao ler o cabeçalho recebido, os nós escravos verificam o identificador e decidem se devem enviar ou receber alguma mensagem de resposta.

Em função da actividade no barramento dois estados são definidos para os nós do sistema: *sleep mode* e *active mode*. Quando se encontram dados a circular no barramento, todos os nós são requisitados para estarem no estado activo. Depois de um *timeout* especificado sem dados no barramento, os nós entram em *sleep mode* e só serão novamente colocados no estado activo quando receberam uma trama específica denominada *WAKE_UP*. Esta trama pode ser enviada por qualquer nó requisitando a utilização do barramento.

2.2.3 - FlexRay

FlexRay é uma rede de comunicação focada para os veículos automóveis desenvolvida por um consórcio fundado pela BMW, Bosch, DaimlerChrysler e Philips em 2000 [21]. Surgiu da necessidade de existir um protocolo de comunicação standard para todos os veículos automóveis. Foi desenvolvida e especificada para proporcionar uma comunicação

determinística, sincronizada, com elevada largura de banda e tolerante a falhas, requisitos importantes na nova geração de veículos com diversas aplicações x-by-wire.

FlexRay pode ser implementada sobre uma topologia do tipo barramento, estrela ou híbrida com recurso a um ou dois canais, como ilustrado na Figura 2.24. Quando o uso de dois canais é considerado, estes poderão funcionar paralelamente e/ou de forma redundante. Se cada um dos nós ligar apenas a um dos canais permite uma velocidade de comunicação de 20 MB/s. Por outro lado, se cada um dos nós for conectado a ambos os canais, a entrega de mensagens críticas é assegurada mesmo que um dos canais falhe.

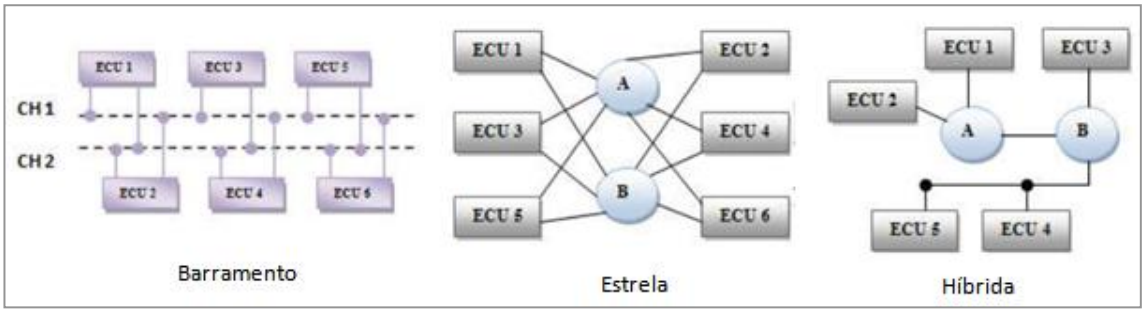

Figura 2.24 - Topologias da rede FlexRay [22]

O controlo de acesso ao meio é construído sobre uma base de tempo global sincronizada e a comunicação é organizada em ciclos de comunicação. Estes ciclos são repetidos periodicamente e têm uma duração predefinida que depende do tipo da aplicação. Durante cada ciclo de comunicação são atribuídos períodos de tempo específicos denominados slots, para cada nó aceder ao barramento e desta forma enviar ou receber dados. Apenas um nó pode comunicar de cada vez, por isso teremos um nó emissor e um outro nó receptor, identificados por um identificador (ID).

Um ciclo de comunicação consiste essencialmente de quatro segmentos distintos: um segmento estático, um segmento dinâmico, um segmento denominado *Symbol Window* e um segmento associado ao período de inactividade no barramento denominado *Network Idle Time (NIT*). Na Figura 2.25 encontra-se ilustrado os diferentes segmentos de um ciclo de comunicação FlexRay.

Figura 2.25 - Ciclo de comunicação FlexRay [23]

O segmento estático funciona de acordo com o princípio do *Time Division Multiple Access* (TDMA) e é dedicado a transmissão síncrona. Este segmento contém um número configurável de *slots* estáticos com duração idêntica que possibilitam aos nós enviarem as suas mensagem. Cada nó tem um *slot* atribuído e só poderá aceder ao meio nesse *slot*, pelo que se uma mensagem não estiver pronta no início de seu *slot*, este permanece vazio e nenhuma outra mensagem poderá ser enviada. Desta forma cada nó ocupa o meio de transmissão durante um

intervalo de tempo específico e conhecido, o que garante determinismo e impede problemas de interferência entre os diversos nós.

O segmento dinâmico é dedicado a transmissão assíncrona e permite transferir mensagens adicionais ou de menor importância geradas por eventos (*Event-Triggered*). Tal como o segmento estático, contém um determinado número de *slots* contudo de tamanho menor (*minislots*). Estes são atribuídos de forma flexível e dinâmica em função das necessidades de comunicação em cada ciclo. Neste caso o acesso ao meio de transmissão é feito em função da prioridade do nó emissor definida no campo ID da trama. A transmissão de uma determinada mensagem poderá ocupar vários *minislots* pelo que mensagens correspondentes a nós com menor prioridade poderão ter que esperar ate ao próximo ciclo de comunicação para serem enviadas. O segmento Symbol Window é um segmento opcional e quando considerado é usado para testes de execução.

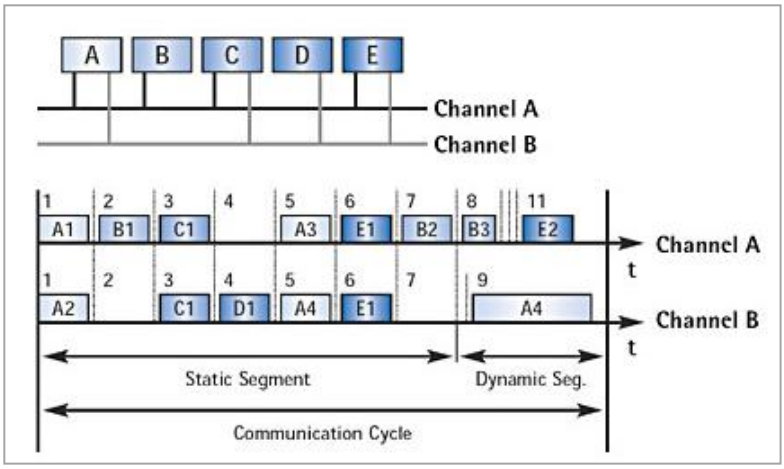

Figura 2.26 – Transmissão de dados Flexray [24]

A trama no protocolo FlexRay é composta por três segmentos, como é possível visualizar na Figura 2.27.

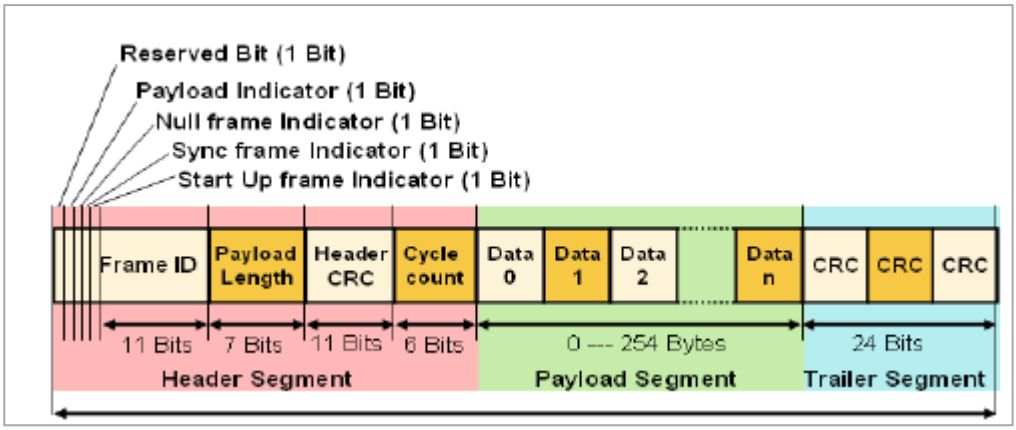

Figura 2.27 – Formato da trama de dados Flexray [25]

O segmento *Header* começa com 5 bits indicadores responsáveis pela definição de características básicas da trama e pela sua identificação. O segmento Payload contém os dados da trama e pode ter um comprimento variável entre 0 e 254 bytes. O segmento *Trailer* contém 24 bits de CRC calculado para os segmentos anteriores e permite a detecção de erros na mensagem.

Os nós são constituídos por várias partes funcionais: um host, um controlador de comunicações, dois controladores de barramento e dois condutores de barramento.

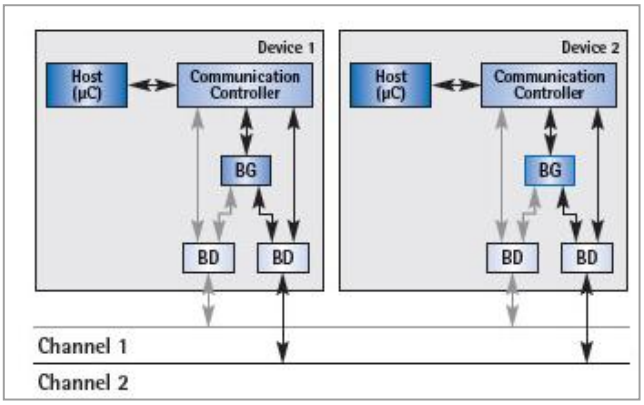

Figura 2.28 – Partes funcionais de um nó FlexRay[24]

O *host* é o microcontrolador que executa a aplicação, processando e fornecendo dados para o controlador de comunicações. O controlador de comunicações é responsável por implementar os aspectos do protocolo de comunicação FlayRay, enviando os dados recebidos do *host* para o barramento através dos condutores de barramento. Os condutores de barramento consistem de um transmissor e um receptor que são responsáveis por conectar o controlador de comunicação para um canal de comunicação. Cada canal de comunicação tem um condutor de barramento para ligar o nó para o canal. O controlador do barramento é responsável pela detecção de erros no barramento gerando interrupções e bloqueio do barramento aquando da ocorrência de problemas críticos no barramento.

Como já foi referido, FlexRay é implementada sobre uma base de tempo global sincronizada, sendo que cada nó possui um intervalo de tempo específico para aceder ao barramento e enviar as suas mensagens. Assim, antes de os diferentes nós começarem a comunicar e a enviar dados para o barramento é necessário garantir a devida sincronização dos relógios dos diversos nós da rede. Neste sentido dois processos são considerados:

- Activação: aviso e preparação de todos os nós para o inicio da comunicação;
- Inicialização: sincronização dos diversos nós e o inicio da comunicação. Neste processo dois tipos de nós são considerados: nós *coldstart* e nós *noncoldstart*.Um nó *coldstart* tem a capacidade de iniciar o procedimento de inicialização através do envio de tramas especiais de inicialização. Estas tramas são usadas pelos restantes nós para efectuarem a sincronização dos seus relógios.

2.3 - Automotive Open System Architecture

Automotive Open System Architecture (AUTOSAR) é uma arquitectura de software aberta

e padronizada, desenvolvida conjuntamente pelos construtores e fornecedores de automóveis. É resultado de um consórcio internacional fundado em 2003 com o intuito de resolver um serie de questões. Por um lado as constantes e crescentes necessidades de funcionalidades relacionadas com segurança, economia e conforto nos automóveis, provocaram um aumento significativo no número de unidades de controlo electrónico presentes nos veículos e na complexidade do software das mesmas. Por outro lado, as unidades de controlo electrónico encontram-se agrupadas em vários domínios. Cada um destes domínios evoluiu individualmente e muitas vezes de forma divergente, sendo que as redes de interligação das unidades de controlo electrónico não foram padronizadas de acordo com as interfaces fronteira entre esses domínios. Além disso o software, as aplicações e o hardware são muitas vezes desenvolvidos por diferentes fabricantes, e para diferentes clientes. Todos estes factos revelaram a necessidade de criação de uma arquitectura standard com o objectivo de proporcionar uma infra-estrutura comum para ajudar no desenvolvimento de software, de interfaces com o utilizador e na gestão todos os domínios de aplicação.

De uma forma genérica os objectivos gerais da AUTOSAR são :

- Desenvolvimento de software independente do hardware;
- Escalabilidade para veículos diferentes e respectivas plataformas;
- Transmissibilidade de funções em toda a rede;
- Integração de módulos funcionais de vários fabricantes;
- Análise dos requisitos de disponibilidade e segurança;
- Possibilidade de manutenção durante todo o "*Product Life Cycle*"
- Actualizações, upgrades e reutilização de software ao longo da vida do veículo

A arquitectura AUTOSAR, tal como ilustrado na Figura 2.29, é constituída por quatro camadas principais: microcontrolador, software básico, ambiente de execução e por fim a camada de aplicação.

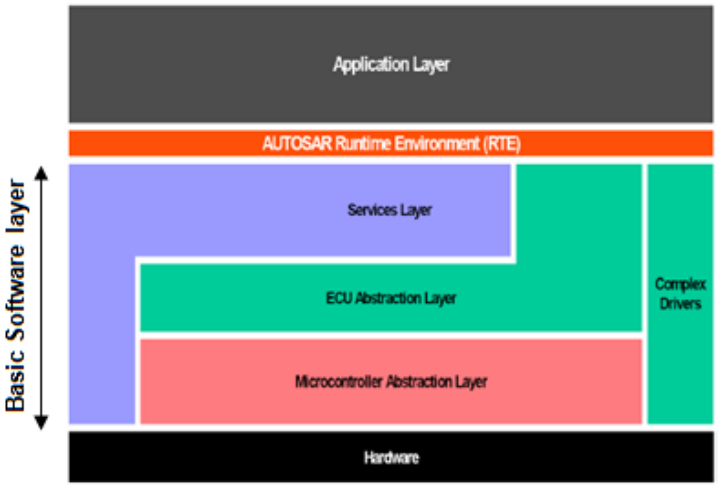

Figura 2.29 – Modelo em camadas do AUTOSAR[26]

Cada camada é responsável por uma função específica. A camada microcontrolador contém o Hardware da unidade de controlo electrónico. A camada *de* software básico é

responsável juntamente com a camada de ambiente de execução pela abstracção entre o hardware e a camada de aplicação, podendo ser dividida em três subcamadas:

- Camada de abstracção do microcontrolador: contém um conjunto de drivers, nomeadamente drivers do microcontrolador, drivers de comunicação e de memória necessários para controlar e aceder aos periféricos internos do microcontrolador. Todo o acesso ao hardware é encaminhado e gerado por esta camada, evitando desta forma o acesso directo ao microcontrolador pelas camadas de nível superior. Implementa mecanismos de notificação para apoiar a distribuição dos comandos, respostas e informações para diferentes processos: entradas e saídas digitas, conversor analógicos, EEPROM, etc .
- Camada de abstracção da ECU: para além de realizar a interface com a camada de abstracção do microcontrolador, esta camada permite aceder a periféricos e dispositivos externos independentemente da sua localização ou da forma como estão conectados ao microcontrolador. Fornece uma interface de software para qualquer ECU especifica de forma a oferecer o acesso ao sinais de Entrada/Saida do microcontrolador.
- Camada de Serviços: responsável por fornecimento de serviços básicos tais como funcionalidades do sistema operativo, gestão de memória, serviços de rede e comunicação, serviços de diagnóstico, etc.

A camada de software básico pode ser dividida em diferentes pilhas funcionais. Esta divisão é realizada no sentido de permitir visualizar a funcionalidade dos diversos módulos de software contidos na camada de software básico.

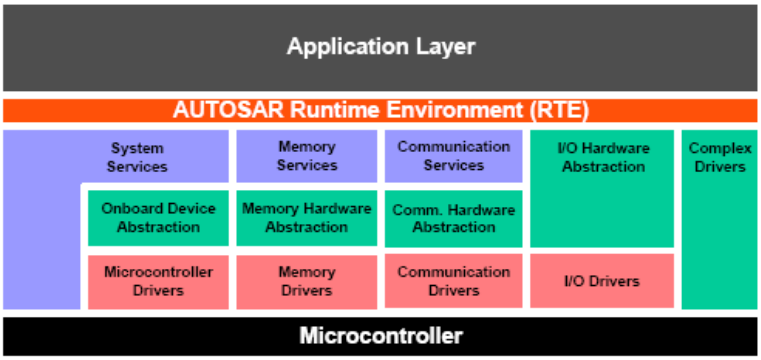

Figura 2.30 - Diferentes pilhas funcionais da camada de software básico[26]

A pilha de sistema inclui o módulo de drivers do microncontrolador, o modulo de serviços e um modulo referente a abstracção de dispositivo. Este conjunto de módulos fornece serviços básicos para aplicação e para os restantes módulos da camada de software básico, como funcionalidade de sistema operativo, gestão de erros, etc.

A pilha de gestão de memória inclui os módulos de drivers de memória, o módulo de abstracção de hardware e o módulo de serviços de memória. Estes fornecem acesso padronizado para memória não volátil. Assim cada componente da aplicação de software pode aceder a memória independentemente se esta se encontra externamente ou internamente localizada na ECU. Desta forma, o componente de software apenas necessita de solicitar memoria através da interface padronizada

A pilha de comunicação inclui os módulos de drivers de comunicação, o módulo de abstracção de Hardware e de serviços de comunicação e fornece um acesso padronizado para o sistema de rede do veículo. A pilha de comunicação em conjunto com a camada de ambiente de execução permite que os componentes de software comuniquem entre si mesmo estando em ECU diferentes.

A pilha input/output inclui um módulo de drivers e de abstracção de hardware que permitem fornecer um acesso padronizado para sensores, actuadores e diferentes periféricos no interior da ECU. Essa pilha não tem uma camada de serviço pois não há em geral interface para todos os sensores e actuadores.

A camada de ambiente de execução fornece a ligação entre a camada de software básico e a camada de aplicação, permitindo que a camada de aplicação seja completamente independente da ECU em questão. Esta camada é também responsável por prestar serviços de comunicação de forma que quando um componente da aplicação necessita de comunicar com outros componentes ou outros serviços faz–o através desta camada.

Como visível na figura Figura 2.30, o conjunto de drivers complexos não é mapeado em nenhuma camada específica. Tal advém do facto deste conjunto de drivers ser responsável pela implementação de tarefas com estreitas restrições de tempo que necessitam de acesso directo ao hardware da ECU. Este acesso é realizado através de interrupções e periféricos específicos.

Por fim, a camada aplicação possui a aplicação de software em si. Encontra–se organizada em componentes e módulos de software. Um componente de software é uma peça de software que implementa uma parte da aplicação, sendo que, geralmente uma aplicação é constituída por diferentes componentes de software que em conjunto com o hardware realizam uma determinada função ou serviço. Os componentes de software encontram –se interligados por portos que permitem e asseguram a comunicação entre os diferentes componentes e módulos de software.

Assim a camada de software básico é necessária para executar a parte funcional do software, e em conjunto com a camada de ambiente de execução permitem uma separação clara entre aplicação e a infra-estrutura. Estas duas camadas fornecem a camada de aplicação uma abstracção em relação a hardware, prestando os serviços de software funcionais necessários como gestão de recursos, comunicação, meios de diagnóstico e ambiente de execução. Estes serviços podem ser realizados recorrendo a sistemas operativos e a *middlewares*.

Capítulo 3

Sistemas operativos

3.1 - Introdução

Um sistema operativo é um conjunto de programas e rotinas computacionais que tem como objectivo criar uma camada de abstracção entre o utilizador e o hardware propriamente dito. Entenda-se por utilizador todo e qualquer objecto que necessite de aceder ao sistema computacional, seja ele um utilizador real ou uma aplicação. Por outro lado o sistema operativo controla e coordenada a utilização dos recursos de hardware e software durante a execução das varias aplicações, de forma a garantir que o sistema global opera de forma eficiente e confiável.

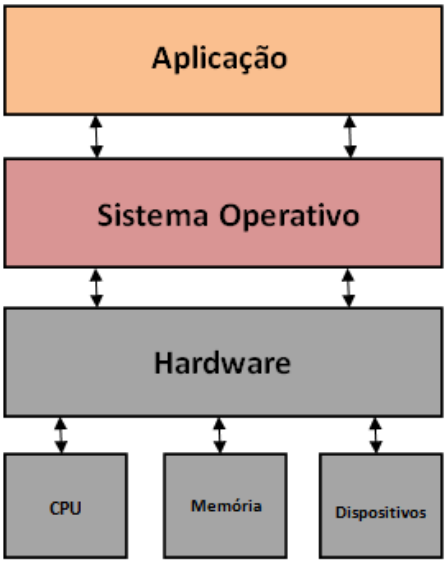

Figura 3.1 - Componentes de software e hardware de um sistema computacional

Para se entender as funcionalidades de um sistema operativo importa primeiro abordar três conceitos importantes: o conceito de programa, processo e thread. O programa é uma entidade estática, que consiste numa sequência de instruções referentes a uma ou mais aplicações escritas numa linguagem de programação compreensível pelo sistema. Um processo representa um programa em execução. É uma entidade activa e é criado pelo sistema operativo para encapsular toda a informação que esta envolvida na execução do programa: variáveis, dados, espaço de endereçamento, utilização de CPU, utilização de dispositivos E/S, etc. O termo processo aparece muitas vezes na literatura associado ao conceito de tarefa, especialmente no âmbito de sistema operativos embarcados. Neste documento considera-se que ambos representam e possuem a mesma funcionalidade, ou seja, a execução de um dado programa.

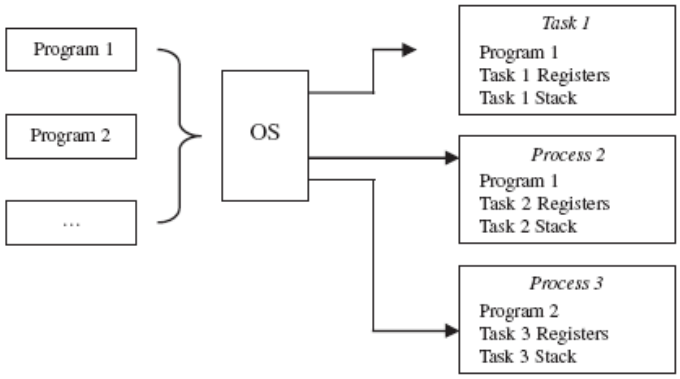

Figura 3.2 – Programas e processos num sistema operativo [27]

Outro conceito importante é o conceito de *thread*. Esta representa uma sequência de execução dentro de um processo, utilizando os recursos desse processo. Um processo pode ter várias threads que permitem ao processo dividir-se a si em duas ou mais sequências de execução diferentes. A grande vantagem do uso de *threads* reside capacidade de resposta pois o uso de várias threads independentes permite que um processo continue a sua execução mesmo que uma parte desse processo esteja bloqueada, aumentando assim a capacidade de resposta. Por outro lado o uso de threads permite uma economia de recursos, visto que threads de um mesmo processo partilham os mesmos recursos.

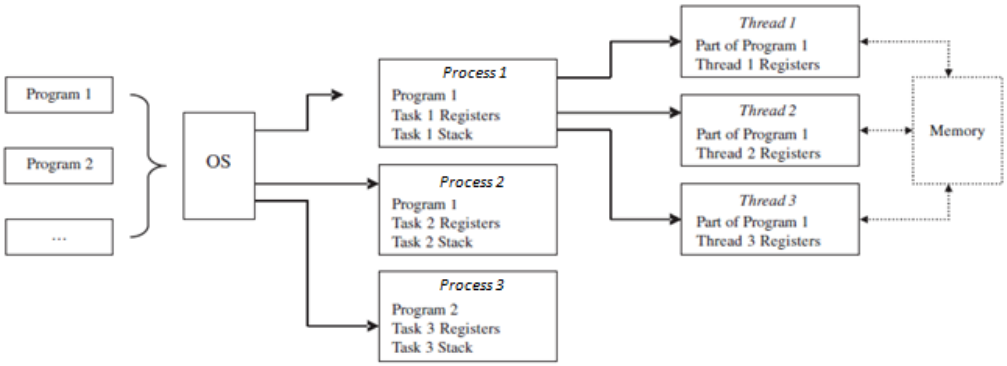

Figura 3.3 – Programas, processos e threads num sistema operativo [27]

Assim, a execução dos diferentes programas aplicativos é realizada pelo sistema operativo recorrendo a processos e a threads. O sistema operativo é assim responsável pela execução dos programas e por todas as actividades associadas e necessárias a sua execução:

- Gestão de processos: o sistema operativo disponibiliza sobre os processos as seguintes funções básicas: criação, eliminação, suspensão, activação, sincronização e comunicação entre processos. O sistema operativo garante também uma partilha correcta do tempo de CPU entre os vários processos e disponibiliza mecanismos de protecção para controlar os acessos dos processos aos diversos recursos;
- Gestão de memória: os processos e o próprio sistema operativo encontram-se armazenados no dispositivo de memória. O sistema operativo garante a reserva, a libertação e monitorização do espaço em memória;
- Gestão de I/O (Entrada/Saída): o sistema operativo dispõe de módulos que lidam especificamente com cada tipo de dispositivo (*device drivers*) de forma a realizar operações de entrada e saída;
- Chamadas de sistema: A interface entre os processos e o sistema operativo é realizada através de comandos de chamada do sistema (*systemcalls*). Estas são funções disponibilizadas através de bibliotecas (APIs-*Application Programming Interfaces)* pelo sistema operativo e podem ser utilizadas em qualquer programa;

No que respeita a execução dos programas os sistemas operativos podem ser classificados em duas categorias: sistemas monoprogramaveis (ou monotarefa) e sistemas multiprogramaveis (ou multitarefa). Esta classificação diz respeito ao número de tarefas que o sistema é capaz de executar a cada instante. Num sistema operativo monotarefa apenas uma tarefa pode estar a ser executada num dado momento, sendo que todos os recursos (processador, memória e periféricos) ficam exclusivamente a ela dedicados. Por outro lado em sistemas multitarefa, várias tarefas residem em memória e competem entre si pelo uso do processador que apenas pode ser acedido por uma tarefa em cada instante. A utilização do processador é repartida entre as diversas tarefas. O processador vai alternado a execução tarefas criando desta forma a ilusão que todas estão a ser executados simultaneamente. No sentido de evitar conflitos no acesso ao CPU, este tipo de sistemas exige o devido escalonamento, sendo este da responsabilidade do sistema operativo. Este escalonamento é realizado de forma cooperativa ou preemptiva recorrendo a diversos algoritmos.

Num sistema multitarefa cooperativo quando uma determinada tarefa acede ao processador, esta liberta-o voluntariamente somente quando termina a sua execução ou quando não consegue prosseguir com a sua execução. Assim, uma vez iniciada a execução duma tarefa esta não pode ser interrompida, pelo que, as restantes tarefas terão que esperar pela libertação do processador para iniciarem a sua execução. Este algoritmo é simples de ser implementado mas as interacções entre as tarefas devem ser bem elaboradas para não causar latência na execução e um possível *crash* do sistema.

Num sistema multitarefa preemptivo, o tempo de utilização do processador é gerido de forma inteligente através da atribuição de tempos específicos de acesso ao processador para

cada tarefa, e através do uso diferentes níveis de prioridades para as tarefas. Neste caso, uma tarefa em execução será interrompida e perderá o acesso ao processador caso termine o tempo a esta concedido para aceder ao processador, ou caso uma tarefa de maior prioridade entretanto activa necessite de aceder ao mesmo.

Para além do devido escalonamento os sistemas multitarefa necessitam da existência de mecanismos que permitam a devida comunicação e sincronização de tarefas. Esta necessidade surge nas mais variadas situações: uma tarefa deseja passar informações para outra, duas ou mais tarefas querem utilizar o mesmo recurso ou uma tarefa depende do resultado produzido por outra tarefa para prosseguir e ser executada. Estes mecanismos são também fornecidos pelo sistema operativo, desde filas de mensagens e semáforos.

O escalonamento preemptivo, a sincronização e comunicação de tarefas, constitui um aspecto especialmente importante em sistemas operativos de tempo real, usados em sistemas críticos que requerem determinismo temporal. Estes sistemas operativos fazem uso da atribuição de diferentes níveis de prioridades para garantir o atendimento e execução das tarefas num prazo bem definido e de acordo com a sua importância.

No que respeita a sua estrutura, isto é, ao modo como o código do sistema operativo é organizado e ao inter-relacionamento entre os seus diversos módulos de software, três modelos são geralmente considerados [27]: monolítico, em camadas e cliente-servidor. Antes de mais importa referir a existência de dois espaços/modos distintos: o modo aplicativo (ou utilizador) e o modo supervisor(ou kernel). Aos programas aplicativos é reservado o modo utilizador com menos privilégio e menor prioridade, e ás rotinas do sistema operativo é reservado o modo supervisor. Desta forma é garantido que a que um programa incorrecto não causa danos à execução de outros programas.

Na estrutura monolítica o sistema operativo é escrito como um programa único composto por uma colecção de subrotinas que se chamam umas as outras sempre que necessário. Os serviços prestados pelo sistema operativo (gestão de processos, gestão de memoria, gestão de dispositivos, etc) encontram-se compactados num único ficheiro executável. Todos componentes têm contacto directo com o hardware ou que controlam os recursos lógicos do sistema, estão integrados no núcleo e correm em modo Kernel. Embora forneçam grande desempenho caracterizam-se por possuírem um tamanho considerável e são difíceis de modificar: uma adição de uma nova funcionalidade poderá requerer a recompilação de todo o código. Na Figura 3.4 encontra-se ilustrado a estrutura de um sistema operativo monolítico.

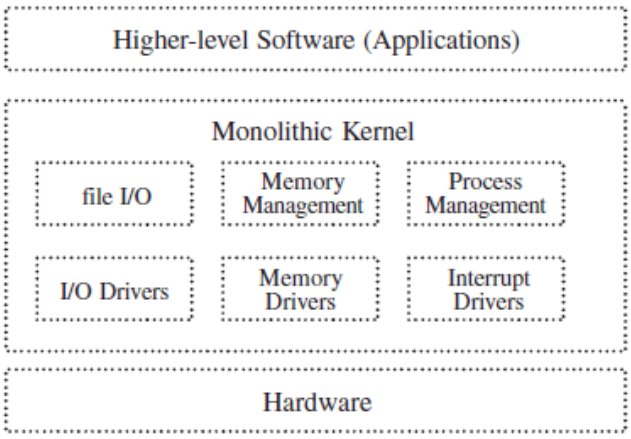

Figura 3.4 – Estrutura de um sistema operativo monolítico[27]

O sistema operativo organizado em camadas é dividido em diferentes camadas hierárquicas. Cada camada constitui um módulo responsável por oferecer um determinado serviço à camada superior. Tal como ilustrado na Figura 3.5A camada mais inferior é a que tem acesso aos dispositivos de hardware e a camada mais externa é a que realiza a interface com a aplicação. Esta abordagem fornece um sistema modular mais fácil de desenvolver, no entanto as APIs fornecidas em cada camada poderão criar uma sobrecarga adicional que pode afectar o tamanho e desempenho (sistema global mais lento).

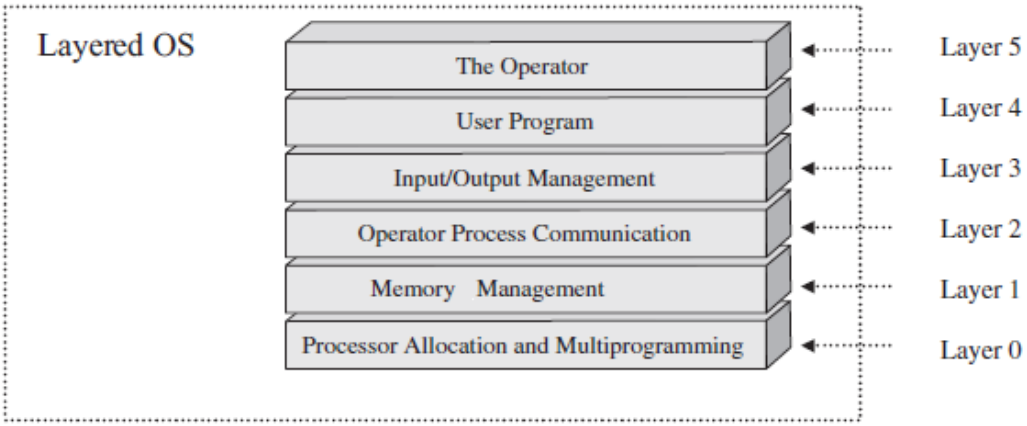

Figura 3.5 – Sistema operativo em camadas [27]

Na estrutura cliente-servidor, o tamanho e as funcionalidades do núcleo do sistema operativo são reduzidas ao máximo. Aqui apenas correm em modo supervisor um conjunto restrito de componentes que providenciam funcionalidades básicas como gestão da comunicação e sincronização entre processos. Todos os outros componentes são implementados como serviços, ou seja, processos que correm fora do núcleo. O programa aplicativo, agora chamado de cliente, para requisitar a execução de um serviço envia uma mensagem ao processo servidor que realiza a tarefa e envia de volta a resposta ao cliente. Esta abordagem permite adição de novos componentes dinamicamente sem a necessidade de recompilação de todo o código e é facilmente adaptável para uso em sistemas distribuídos.

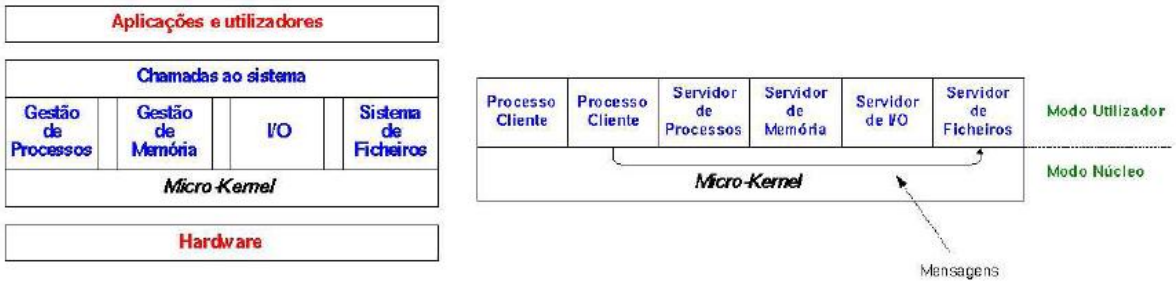

Figura 3.6 – Estrutura do sistema operativo cliente-servidor

3.2 - Requisitos

No contexto deste trabalho os sistemas operativos irão servir de suporte ao desenvolvimento de diversas aplicações. Estas aplicações irão estar embebidas em diversos sistemas distribuídos num veículo eléctrico e desenvolvidas por diferentes grupos de alunos. Neste sentido requisitos especiais são esperados dos sistemas operativos que irão ser utilizados, nomeadamente:

I. Reduzida necessidades de memória e de recursos energéticos: uma característica presente em sistemas embarcados reside na reduzida capacidade de memória, de processamento e de recursos energéticos. Assim o sistema operativo deve ser simples, ter um tamanho reduzido, necessitar de poucos recursos de memória para a sua execução e optimizar o uso do hardware no sentido de minimizar os gastos energéticos.

II. Portabilidade e interface com redes de comunicação: o sistema operativo deve ser portável para diversas plataformas de hardware e possuir se possível, interface para as redes de comunicação normalmente usadas nos veículos.

III. Sistema operativo multitarefa: muitos dos sistemas que irão ser desenvolvidos como ABS, controlo de tracção, controlo de estabilidade, entre outros, são responsáveis pela execução de diferentes tarefas em simultâneo. Desta forma o sistema operativo deve ser multitarefa e garantir o devido escalonamento das mesmas.

IV. Determinismo temporal: muitas das aplicações que irão ser desenvolvidas exigem as execução das suas tarefas em intervalos de tempo bem definidos, pois representam aplicações críticas e de tempo real. Desta forma, espera-se que o sistema operativo seja um sistema operativo de tempo real com todas as características presentes neste tipo de sistemas: sistema multitarefa, escalonamento preemptivo, diferentes níveis de prioridades, sincronização de tarefas, etc.

V. Licença livre de custos: as diversas aplicações irão ser desenvolvidas em meio académico e por diferentes grupos de alunos. Assim, seria óptimo que o sistema operativo fosse de licença livre de custos e de código aberto. Desta forma os custos seriam nulos e os alunos poderiam aceder ao código do sistema operativo de forma altera-lo sempre que necessário e adapta-lo as diversas aplicações.

VI. Linguagem de programação: é importante que o sistema suporte linguagens de desenvolvimento que sejam já do conhecimento dos alunos como C, C++, e Java, de forma a possibilitar uma rápida implementação. Neste contexto é também importante que o sistema operativo disponibilize um conjunto de funções básicas já definidas que permitam a fácil gestão do sistema, a gestão de tarefas, gestão

de memória e dispositivos, etc. Desta forma o desenvolvimento da aplicação será muito mais fácil e rápido.

VII. Ferramentas de desenvolvimento: no sentido facilitar o desenvolvimento de aplicações um requisito a ter em conta diz respeito as ferramentas de *bebug* e de ambiente integrado de desenvolvimento que o sistema operativo oferece ou suporta.

VIII. Suporte documental: é importante que exista uma grande variedade de informação credível referente a especificação do sistema operativo. A existência de documentos e tutoriais oficiais que permitam ajudar os alunos a compreender o funcionamento do sistema operativo deve ser levado em conta.

Os requisitos aqui enumerados e definidos constituem características gerais que os sistemas operativos deverão possuir no contexto deste trabalho. Contudo alguns dos requisitos terão maior ou menor importância dependendo da aplicação em si. Concretamente, poderá não ser exigível a utilização de um sistema operativo de tempo real visto que tal requisito está directamente relacionado com aplicação a que se destina.

3.3 - Alguns sistemas operativos

Nesta secção serão descritos alguns dos sistemas operativos que poderão ser considerados mediante os requisitos anteriormente enumerados. São descritos seis sistemas operativos: três com características e requisitos de tempo real vocacionados para plataformas baseadas em microcontroladores e aplicações distribuídas, dois sistemas operativos baseados em eventos e um sistema operativo mais vocacionado para plataformas baseadas em computador. Importa referir que os sistemas que aqui serão apresentados constituem exemplos de uma grande variedade de sistemas operativos existentes direccionados para sistemas embebidos. Isto significa que outros sistemas operativos aqui não mencionados poderão eventualmente ser adequados e utilizados no âmbito deste trabalho.

3.3.1 - FreeRTOS

O sistema operativo *free Real Time Operating System* (freeRTOS) [28] é um sistema operativo desenvolvido e gerido por Richard Barry & FreeRTOS Team. Possui uma estrutura *microkernel* e uma das suas características chave é a sua elevada portabilidade: necessita de reduzidos recursos de memória e pode ser utilizado em diversas plataformas de hardware. O seu código é livre e aberto podendo ser adquirido de forma gratuita em [28]. Juntamente com o código é também possível adquirir gratuitamente documentação referente a especificação do sistema operativo e uma demo contendo aplicações exemplo para cada uma das plataformas de hardware suportadas.

3.3.1.1 - Características

O freeRTOS é um sistema multitarefa e *multithread*. Não define um número máximo de tarefas suportadas em simultâneo nem um número máximo de níveis de prioridade, sendo que este limite esta apenas condicionado pelos recursos de hardware disponíveis. Uma tarefa no freeRTOS possui as seguintes características [28] :

- Um estado que demonstra a actual situação da tarefa;
- Uma prioridade que varia de zero até uma constante máxima definida pelo programador;
- Uma pilha onde é armazenada o ambiente de execução da tarefa quando esta é interrompida:

O freeRTOS fornece um conjunto de bibliotecas e funções para a gestão de tarefas: Criação, remoção, suspensão atribuição de prioridade, são algumas das operaçoes que o sistema permite realizar. Um conceito importante neste sistema operativo é o de *idle task*. Esta tarefa é executada quando nenhuma tarefa esta em execução e tem como finalidade excluir da memória tarefas que não serão mais usadas pelo sistema. Desta forma quando uma aplicação informa o sistema que uma tarefa não será mais utilizada essa tarefa só será excluída quando a tarefa ociosa entrar em execução.

No que respeita ao escalonamento, este sistema fornece um conjunto de bibliotecas que permitem o escalonamento preemptivo (através da atribuição de prioridades) e também escalonamento cooperativo. Como já aqui foi referido, não há limites para o numero de níveis de prioridade implementados, sendo possível que duas ou mais tarefas possuam a mesma prioridade de execução. Nesta situação específica cada tarefa possui um intervalo de tempo específico (igual para todas as tarefas) para utilizar o CPU. Ultrapassado este tempo o CPU é alocado a outra tarefa com a mesma prioridade. São também disponibilizadas funções que que permitem permitir a comunicação e a sincronização de tarefas, nomeadamente a criação de caixas de mensagens e de semáforos.

Como foi referido anteriormente o freeRTOS foi pensado e desenvolvido para ser utilizado em sistemas embarcados. Concretamente a sua imagem binária ocupa entre 4kB e 9kB. O código é constituído por quatro arquivos escritos maioritariamente em C com pequenas partes escritas em *Assembly*.

Este sistema operativo serve de suporte a diversas plataformas de hardware desde pequenos microcontroladores de 8bits até plataformas de 32 bits: Altera, Atmel, Cortus, Energy Micro, Freescale, Fujitsu, Texas Instruments, Microchips, Nec, NXP, Renesas, ST, TI, Xilinx e X86. Para cada plataforma de hardware, o freeRTOS suporta e é compatível com diversas ferramentas de desenvolvimento existentes. De uma forma genérica as ferramentas suportadas são: GCC, Rowley CrossWorks, IAR, Keil, Red Suite. Uma analise mais detalhada dos processadores suportados e das ferramentas oferecidas encontra-se em [28].

3.3.2 - eCos

O Embedded Configurable Operating System (eCos) [29] é um sistema operativo de tempo real desenvolvido pela Cygnus Solutions. Flexibilidade, configurabilidade e excelente desempenho de tempo real são as suas principais características. Apresenta-se como um

sistema operativo livre de custos e de código aberto. Junto com o seu código são fornecidas algumas ferramentas de desenvolvimento e configuração também estas gratuitas. Possui um bom suporte documental, desde documentação oficial online (incluindo um livro), tutoriais e fóruns de discussão. Fornece também apoio técnico através de uma ferramenta online que permite reportar eventuais problemas.

3.3.2.1 - Características

eCos é um sistema *multithread* e utiliza um escalonamento preemptivo realizado por dois escalonadores: *Multi-Level Queue Scheduler* e o *Bitmap Scheduler*. O *Multi-Level Queue Scheduler* permite a execução de múltiplas *threads* em cada um dos seus níveis de prioridade, sendo disponibilizados de 1 até 32 níveis configurados pelo programador. Para cada um dos níveis de prioridade, este escalonador permite a definição pelo programador de um intervalo de tempo para a execução das diferentes *threads*. Será este intervalo de tempo que irá fornecer a desejada preempção entre tarefas do mesmo nível, visto que finito o intervalo de tempo para uma tarefa executar esta será interrompida. O *Bitmap Scheduler* tem um funcionamento mais simples: a cada um dos 32 níveis de prioridade apenas pode estar atribuída uma tarefa. Dependendo do tipo de aplicação, caberá ao programador seleccionar o tipo de escalonador pretendido.

Este sistema operativo apresenta-se como um modelo organizado em camadas, sendo cada camada construída por diversos componentes ou módulos, como se encontra ilustrado na Figura 3.7.

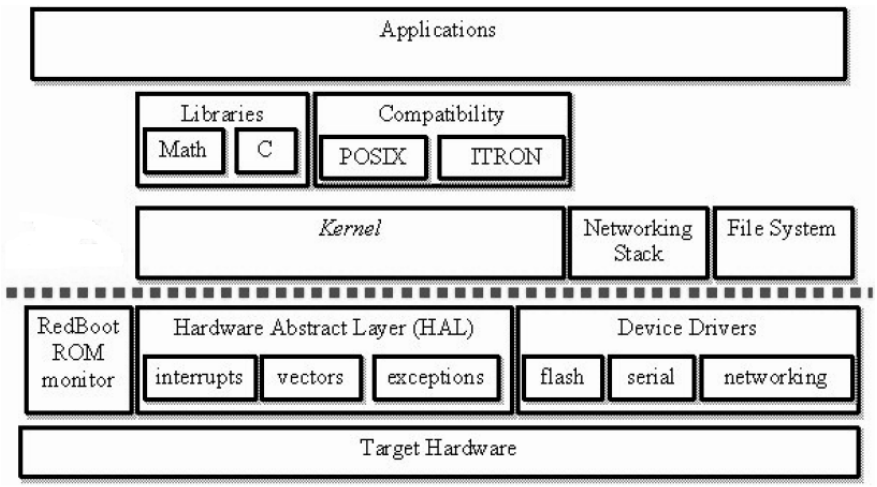

Figura 3.7 - Arquitectura do eCos [30]

A camada de abstracção de hardware permite que todos os componentes da estrutura consigam realizar operações específicas de hardware de forma transparente. Destaque para a grande variedade de gestores de dispositivos incluindo a padronização série, Ethernet, Flash, CAN, SPI e I^2C. São também fornecidas bibliotecas de funções de matemática, tratamento de interrupções, gestão de memória dinâmica, implementação de semáforos e caixas de mensagens, contadores e alarmes. É ainda fornecido um conjunto de bibliotecas que servem de suporte ao protocolo TCP/IP.

Um aspecto interessante e bastante útil neste sistema operativo é que este é configurável, permitindo a parametrização dos seus recursos para satisfazer requisitos específicos de uma determinada aplicação. Através de uma ferramenta gráfica denominada *eCos configuration tool*, é permitido ao programador seleccionar os componentes que satisfaçam as necessidades da aplicação e configurar reservadamente cada componente. Desta forma é evitado o desperdício de serviços não usados o que se traduz numa rentabilização da memória usada. Neste sentido, dependendo da configuração efectuada e dos serviços necessários, o tamanho do eCos varia podendo ir de algumas dezenas de kilobytes até algumas dezenas de megabytes.

O sistema operativo eCos encontra-se completamente escrito em C permitindo também a programação em C++, o que o torna portável para inúmeras arquitecturas e plataformas nomeadamente arquitecturas de 16, 32 e 64 bits. Actualmente suporta as seguintes arquitecturas: 68K/ColdFire, ARM,Hitachi H8300, Intel x86,MIPS,Matsushita AM3x, Fujitsu FR-V e FR30 , PowerPC, SuperH , SPARC e NEC V8xx.

Relativamente a compiladores e ferramentas de desenvolvimento, juntamente com o código são fornecidas as seguintes ferramentas:

- GCC - compilador ANSI-C
- GDB - depurador de código
- Cygwin™ – ambiente Unix para Windows
- Insight – Interface gráfica para o GDB
- Source Navigator – ferramenta de compressão de codigode paradigma de operação

A distribuição eCos está disponível tanto em versões Linux e Windows (Windows 2000 Professional, Windows XP e Windows Vista).

3.3.3 - MicroC/OS- II

MicroC/OS-II [31] também designado por µC/OS-II, é um sistema operativo desenvolvido por Jean J. Labrosse e actualmente distribuído pela Micrium [32]. Foi concebido especificamente para sistemas embarcados usados em aplicações críticas com fortes exigências de determinismo temporal. É um sistema operativo de código aberto e livre para aplicações educacionais, no entanto exige o registo no site oficial do mesmo. Destaca-se pelo grande número de plataformas de hardware suportadas e pelas inúmeras bibliotecas e aplicações adicionais desenvolvidas pela mesma empresa e que é possível adicionar ao seu núcleo. É fornecido online um livro com a especificação do sistema operativo.

3.3.3.1 - Características

O µC/OS-II é um sistema multitarefa preemptivo executando sempre a tarefa com maior prioridade. Se uma interrupção suspender a execução de uma determinada tarefa e se outra tarefa de maior prioridade ficar ”pronta” como efeito da interrupção, a mesma irá executar assim que o tratamento da interrupção terminar. A última versão permite a execução de 250 tarefas. São considerados 250 níveis de prioridade, sendo que cada tarefa possui uma prioridade única e exclusiva no sistema. O tempo de execução para a maioria dos serviços

prestados pela µC/OS-II é constante e determinístico não dependendo do número de tarefas em execução na aplicação.

A gestão de memória é realizada recorrendo a partições de memória, sendo que cada partição de memória consiste em vários blocos de tamanhos fixos. Uma tarefa pode criar e utilizar partições múltiplas de memória, de forma a usar blocos de memória de tamanhos diferentes. Alocação e desalocação de blocos de memória de tamanho fixo são feitas em tempo constante e determinista.

O uC/OS-II apresenta uma arquitectura monolítica, sendo implementada por diferentes arquivos. A estrutura de arquivos do µC/OS-II e sua relação com o hardware está apresentada na Figura 3.8.

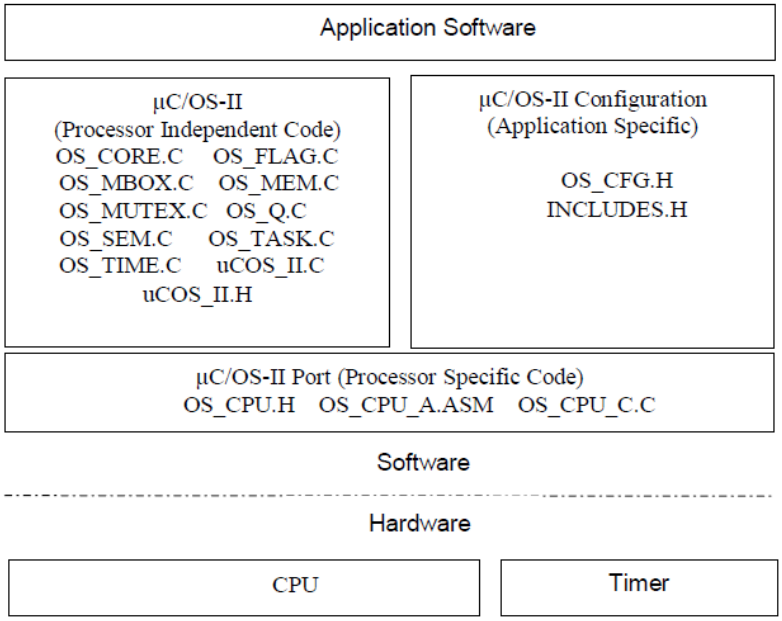

Figura 3.8 - Arquitectura do µC/OS-II [33]

O conjunto de arquivos independentes do processador contém os códigos dos serviços e das chamadas do kernel que permitem a gestão de tarefas da aplicação. Por sua vez, o conjunto de arquivos referente a configuração contém o código específico para aplicação, sendo possível ao programador seleccionar os arquivos necessários de acordo com os serviços pretendidos, permitindo desta forma a rentabilização da memória usada. Dependendo dos arquivos e serviços usados, o sistema operativo utiliza entre 5 Kbytes e 24 Kbytes de memória. O conjunto de arquivos denominado *Port* contém a parte do código dependente e especifica para cada microprocessador, tendo esta de ser codificada em função do processador usado. Este último conjunto de arquivos encontra-se escrito em *Assembly*, sendo que o restante sistema operativo está escrito em C.

Como já foi referido, a empresa distribuidora do µC/OS-II fornece uma série de módulos e aplicações adicionais que permitem aumentar a funcionalidade deste sistema operativo. Neste campo, destaque para o µC/CAN, o µC/TCP-IP e µC/Modbus, que permitem uma fácil implementação dos protocolos CAN, TCP/IP, e Modbus.

O sistema pode ser utilizado em diversas plataformas de software com se encontra ilustrado na tabela 3.1.

Tabela 3.1 - Plataformas suportadas pelo µC/OS-II([31]

Marca	*Arquitectura*
Actel	Cortex –M1
Altera	Nios II, Cortex-M1
Analog Devices	AduC7xxx, ADSP-21xx, Blackfin 5xx, SHARC
ARM	ARM7, ARM9, AMR11
Atmel	SAM7, SAM9, AVR, AVR32
Freescale	9S08, 9S12, Coldfire, PowerPC, i.MX
Fujitsu	Fr50
Infineon	Tricore, 80C16x
Intel	80x86
Lattice	Micro32
Microchip	PIC24, dsPIC33, PIC32
MIPS	R3000, R4000
NEC	78Kx, V850
Ti	MSP430, TMS320, TMS470
Xilinx	MicroBlaze, PowerPC
Zilog	Z80, eZ80

No que respeita as ferramentas de desenvolvimento o µC/OS-II possui a sua própria aplicação para esse fim, o µC/Probe. Como complemento é também fornecida a ferramenta IAR Kickstart Kit™.

3.3.4 - RTAI Linux

O RTAI (Real-Time Application Interface) [34] surgiu de um projecto desenvolvido pelo Departamento de Engenharia Aeroespacial de Milão. Foi criado com o intuito de criar uma interface entre o hardware e o kernel do Linux [35] [36] de forma a possibilitar o uso do deste em aplicações com características de tempo real. O kernel Linux não foi inicialmente desenvolvido nem pensado para aplicações embarcadas e de tempo real. Contudo a sua elevada configurabilidade, flexibilidade e escalabilidade, tornaram-no cada vez mais popular e usado neste tipo de aplicações. Além disso, o facto de o Linux ser um kernel livre e de código aberto possibilitou o desenvolvimento de diversas modificações o que levou ao surgimento de novas versões e novos núcleos baseados em Linux para as mais variadas aplicações. O RTAI e o Linux são facilmente descarregados online e encontram-se devidamente documentados.

3.3.4.1 - Características

RTAI disponibiliza um núcleo preemptivo que controla o hardware e que permite o escalonamento de tarefas baseado em prioridades. Este encontra-se situado entre o hardware e o kernel Linux criando desta forma uma camada de abstracção ao sistema Linux. O kernel Linux é executado sobre a forma de uma tarefa de menor prioridade, sendo os restantes níveis de prioridade atribuídos às tarefas de tempo real. Desta forma o Linux não interfere com as tarefas de tempo real uma vez que só é executado quando nenhuma tarefa de maior

prioridade se encontra em execução. Na Figura 3.9 encontra-se ilustrado o funcionamento e a arquitectura do sistema operativo RTAI.

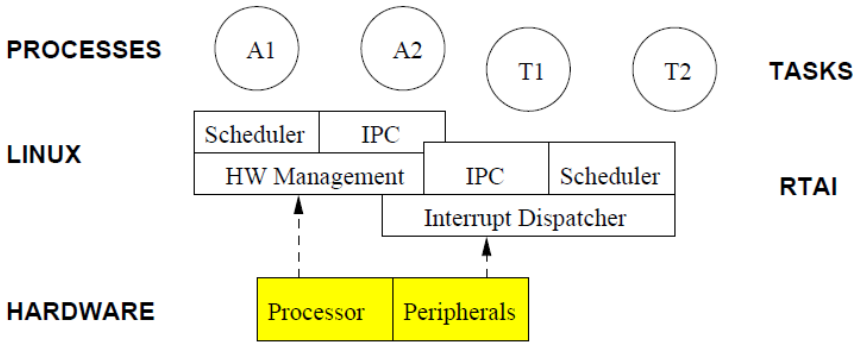

Figura 3.9 - Arquitectura do RTAI [37]

O RTAI é responsável por interceptar as interrupções dos periféricos e sempre que necessário encaminha-las para o *kernel* Linux quando não há tarefas em tempo real activas. Como é visível na Figura 3.9, os mecanismos de comunicação entre processos e o escalonamento são implementados separadamente para o Linux e para o RTAI.

O RTAI fornece mecanismos como caixa de mensagens, semáforos e memória partilhada que permitem a comunicação e sincronização de processos. O sistema dispõe de três escalonadores: o *Uniprocessor* (UP), o *Symetric Multi Processors* (SMP) e o *Multiuniprocessor* (MUP). A diferença entre eles reside basicamente no tipo de arquitecturas e plataformas a que se destinam e na forma como as tarefas são alocadas aos processadores. Concretamente o *Uniprocessor* (UP) é optimizado para arquitecturas monoprocessador. O *Symetric Multi Processors* é usado em plataformas multiprocessador e fornece uma interface para as aplicações para seleccionar o processador ou o conjunto de processadores em que uma dada tarefa é executada. O *Multiuniprocessor* pode ser utilizado tanto em plataformas multiprocessador como monoprocessador e as tarefas as alocadas a um determinado processador no momento da sua criação.

Encontram-se disponíveis diversas politicas de escalonamento, a destacar:

- *First in Fisrt Out* (FIFO) totalmente preemptivel para escalonamento cooperativo: quando uma tarefa acede ao CPU, este permanece alocado a tarefa até esta terminar a execução ou até uma tarefa de maior prioridade fique entretanto pronta;
- *Round Roubin* (RR),em que cada tarefa possui um tempo especifico para aceder ao CPU. Terminado este tempo o CPU é alocado a outra tarefa com a mesma prioridade;
- *Early Deadline First* (EDF), que permite a atribuição de prioridades dinamicamente de acordo com o prazo de execução de cada tarefa

Nas versões iniciais do RTAI a memória tinha que ser alocada estaticamente. No entanto as versões actuais incluem já um módulo de gestão de memória que permitem a

alocação de forma dinâmica. O RTAI utiliza cerca de 630KB de memória não incluindo o kernel Linux. O tamanho do Linux dependera da plataforma alvo e dos serviços pretendidos, sendo possível a adição ou remoção dos módulos consoante as necessidades da aplicação.

RTAI encontra-se escrito em C o que permite a sua utilização em diversas plataformas de software [34]: x86, x86_64, PowerPC,ARM (StrongARM; ARM7) e m68k.

3.3.5 - TinyOS

O sistema operativo TinyOS [38] é um sistema operativo desenvolvido pela Universidade de Berkeley. Flexibilidade e reduzida necessidade de memória e energia são os seus pontos fortes, sendo actualmente dos sistemas operativos mais usados em redes sensores sem fios. É um sistema de código aberto e livre com uma grande variedade de documentação disponível. A página na Web disponibiliza toda a informação para descarregar o software, documentação através de uma wiki, um livro de Philip Lewis "TinyOS programming Manual" e tutoriais. Além disso, visto que o TinyOS é muito usado em aplicações e trabalhos académicos, uma grande variedade de artigos estão disponíveis e que poderão de alguma forma servir de suporte.

3.3.5.1 - Características

O TinyOS apresenta um conjunto de características que o diferenciam dos sistemas operativos até agora mencionados:

- As aplicações são um conjunto de componentes interligados entre si;
- Modelo de execução baseado em eventos que desencadeiam a execução de tarefas;
- Operações divididas em fases;

O TinyOS disponibiliza um conjunto de componentes /módulos de software. Estes constituem pequenos pedaços de código reutilizáveis e independentes entre si. São responsáveis pela implementação de determinados serviços e devidamente interligados formam a aplicação. Cada componente é composto por quatro partes inter-relacionadas: uma frame, uma lista de comandos, uma lista de eventos e um conjunto de tarefas.

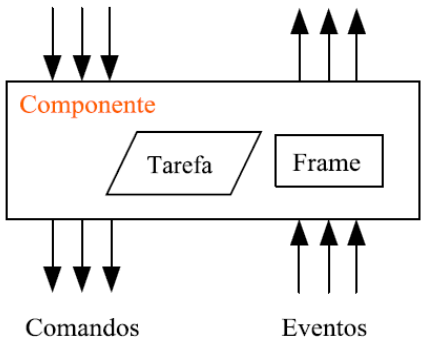

Figura 3.10 - Componentes, eventos e comandos no TinyOS

Comandos e eventos representam mecanismos de comunicação entre componentes, enquanto as tarefas são usadas para expressar concorrência intra-componente. Um comando normalmente é um pedido a um componente para executar alguns serviços, tais como iniciar uma leitura do sensor, enquanto um evento sinaliza a conclusão desse serviço ou a ocorrência de um evento de hardware. Por fim as tarefas constituem unidades básicas de execução executadas como resposta a um determinado comando ou evento, sendo responsáveis pela realização dos serviços associados a cada componente. Assim execução de uma tarefa ou serviço é dividida em diferentes fases, visto que existe uma clara distinção entre a requisição e a conclusão de uma tarefa. As tarefas são requisitadas por comandos e a sua execução é sinalizada para a aplicação através de um evento. A arquitectura baseada em componentes permite a construção de um sistema modular e flexível visto que cada componente é apenas responsável pelos serviços que oferece e desta forma apenas os componentes necessários á aplicação são considerados.

O TinyOS é composto por dois níveis de escalonamento. O primeiro nível é responsável por gerir os comandos e eventos. O segundo nível e responsável pelo controlo das tarefas. Assim, quando uma tarefa é chamada dentro de um evento ou comando, esta é colocada numa fila de espera do tipo FIFO (*First In First Out*), sendo que a execução das tarefas é apenas realizada quando o processador não esta a executar qualquer evento ou comando. Inicialmente nas primeiras versões deste sistema operativo a execução das diferentes tarefas não interferiam entre si, isto é, não era permitida preempção de tarefas. Esta abordagem dificultava a concorrência entre tarefas e o uso em aplicações de tempo real. Assim, a partir da versão 2.0 foi adicionado ao sistema operativo um novo escalonamento capaz de interromper uma tarefa considerada não crítica e executar outra tarefa de maior prioridade. Quando a fila de mensagens se encontra vazia, o processador e colocado em modo *sleep*, enquanto os periféricos continuam operacionais de modo a permitirem que o sistema possa ficar activo novamente aquando da ocorrência de interrupcão por hardware.

O sistema operativo, as suas bibliotecas, e aplicações estão escritas em nesC [39], linguagem que é uma extensão do C, sendo semelhante a esta no que a sintaxe diz respeito. NesC foi criada para incorporar os conceitos e o modelo de execução do TinyOS. Desta forma uma aplicação no nesC consiste de um ou mais componentes ligados entre si e que disponibilizam ou usam interfaces. Estas interfaces constituem o único ponto de acesso aos componentes e são responsáveis por declarar um conjunto de funções, comandos e eventos. As interfaces são direccionais, ou seja, certos componentes implementam a interface, enquanto outros as utilizam. Na Figura 3.11 encontra-se ilustrado um exemplo de uma aplicação: as caixas representam componentes e as ligações representam as interfaces utilizadas e fornecidas pelos componentes.

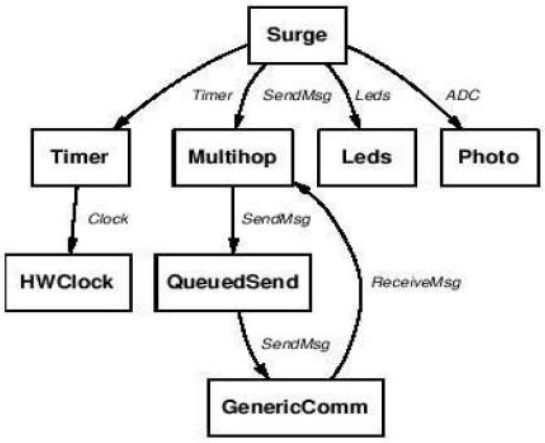

Figura 3.11 – Exemplo de uma aplicação no TinyOS [40]

O TinyOS suporta uma grande variedade de plataformas: BTnode, EyesIF X v1, EyesIF X v2, IMote, IMote 1.0, IMote 2.0, Iris, KMote, Mica, Mica2, MicaZ, Rene, SenseNode, TelosB, T-Mote Sky. Pode ser instalado numa plataforma Linux, ou Windows 2000/XP recorrendo ao programa Cygwin. Juntamente com o código é fornecida o compilador Yet para nesc e a ferramenta de desenvolvimento Eclipse.

3.3.6 - Contiki

O Contiki é um sistema operativo desenvolvido por investigadores do Instituto de Ciências da Computação da Suécia destinado a aplicações com escassos recursos de memória e processamento. Caracteriza-se pela sua elevada configurabilidade e pela reduzida quantidade de memória necessária para a sua execução. É usado principalmente em redes de sistemas embarcados e redes de sensores sem fio. É um sistema livre e de código aberto. Na página Web oficial do sistema é possível descarregar o software e aceder a uma grande variedade de informação: documentação relativa a especificação do sistema, tutoriais e notícias recentes sobre o software. Instruções relativas a instalação e compilação do sistema estão também disponíveis online [41].

3.3.6.1 - Características

O Contiki é um sistema multitarefa implementado através de um núcleo baseado em eventos, o que significa que a execução dos diferentes processos surge no contexto da ocorrência de um determinado evento. Na execução dos diferentes processos o Contiki utiliza protothreads. Estes são semelhantes as threads contudo são invocados em resposta a eventos. A comunicação entre processos é realizada através da passagem de mensagens entre processos. O sistema fornece também o suporte a *multithreads* e a respectiva preempção, através de uma biblioteca opcional que é ligada à imagem final do sistema quando esta opção é necessária.

O kernel do Contiki é apenas responsável pela gestão de eventos, sendo as restantes funcionalidades implementadas sobre a forma de bibliotecas do sistema que são ligadas ao kernel consoante a necessidade de utilização. O kernel, os *drivers* de comunicação e bibliotecas estão sempre presentes em memória. Os programas aplicativos e alguns dos

serviços são carregados em r*untime* permitindo desta forma a execução dinâmica de programas e serviços. Como se encontra ilustrado na Figura 3.12, serviços no Contiki constituem processos que implementam determinadas funcionalidades que podem ser usadas por outros processos, nomeadamente os processos da aplicação.

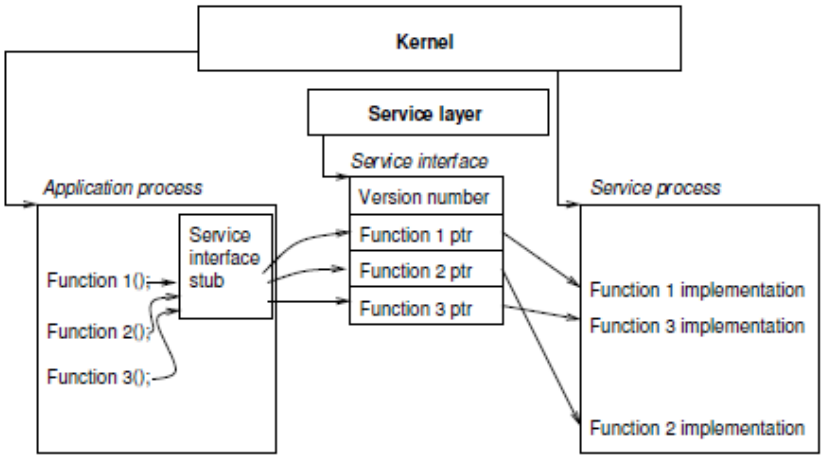

Figura 3.12 - Serviços sobre a forma de processos no Contiki[42]

A camada de serviços permite que os diferentes serviços sejam compartilhados pelos diferentes processos e é responsável pela geração e eliminação dos serviços de forma dinâmica em run time.

O sistema operativo e todos os programas são escritos na linguagem de programação C. É fornecido um arquivo denominado *Instant Contiki*, que contém todas as ferramentas de desenvolvimento e compiladores necessários para o desenvolvimento de aplicações. Como já foi referido, programas da aplicação podem ser dinamicamente carregados e descarregados em tempo de execução. Uma configuração tipica utiliza 2 KB de memória RAM e 40 KB de memória ROM. O sistema fornece duas stacks de comunicação que permitem a implementação do protocolo TCP/IP.

O Contiki poderá ser usado numa grande variedade de plataformas, nomedamente: T-Mote Sky,TelosB, AVR, ARM 7,MSP430, x86, 6502.

3.4 - Comparação dos sistemas apresentados

Na tabela 3.2 encontram-se algumas das principais características dos sistemas operativos anteriormente descritos. Os sistemas aqui propostos são livres e de fácil aquisição, , bastando aceder ao site do mesmo e fazer o respectivo download. Excepção ar regra é o Mc/OS-II, que embora seja livre exige o registo no *Website* para posterior download. Todos fornecem ferramentas de desenvolvimento no momento do download, e uma grande variedade de documentação, desde livros, tutoriais e aplicações exemplo. Destaque para o *freeRTOS* que disponibiliza uma demo com aplicações exemplo e para *eCos* e o *TinyOS* que oferecem uma grande variedade de tutoriais.

Tabela 3.2 – Comparação dos sistemas operativos mencionados

Sistema operativo	Licença	Suporte documental	Ferramentas de desenvolvimento	Plataformas suportadas
freeRTOS	Livre (download directo)	Especificação online, e demo como exemplos	GCC, Rowley CrossWorks, IAR,Keil, Red Suite	Altera, Atmel, Cortus, Energy Micro, Freescale, Fujitsu, Texas Instruments, Microchips, Nec, NXP, Renesas, ST, TI, Xilinx e X86
eCos	Livre (download directo)	Livro, tutoriais, fóruns de discussão	Gcc, IAR	68K/ColdFire, ARM,Hitachi H8300,Intelx86,MIPS,Matsushita AM3x, Fujitsu (FR-V,FR30) , PowerPC, SuperH , SPARC ,NEC V8xx
Mc/OS-II	Livre (requer registo no site)	Livro, diversa documentação online	µC/Probe, IAR	Actel, Altera, Analog Devices, ARM, Atmel,Freescale,Fujitsu, Infineon, 80x86, PIC24, dsPIC33, PIC32,MIPS, NEC, MicroBlaze, PowerPC
RTAI	Livre (download directo)	Documentação online	Gcc	x86, x86_64, PowerPC,ARM (StrongARM; ARM7) e m68k
TinyOS	Livre (download directo)	Livro, tutoriais	Eclipse, Yet	BTnode, EyesIF X v1, EyesIF X v2, IMote, IMote 1.0, IMote 2.0, Iris, KMote, Mica, Mica2, MicaZ, Rene, SenseNode, TelosB, T-Mote Sky
Contiki	Livre (download directo)	Documentação online	Eclipse, Gcc	T-Mote Sky, TelosB, AVR,ARM7,MSP430,x86,6502

Os sistemas apresentados são multitrarefa e suportam o escalonamento preemptivo e/ou escalonamento cooperativo. Uma característica geralmente presente nos sistemas operativos apresentados é a reduzida memória utilizada (a exepção do RTAI Linux) e a sua configurabilidade: é permitido ao programador seleccionar os módulos do sistema operativo de acordo com os serviços necessários. Desta forma é rentabilizada a memória utilizada. A linguagem de programação C é usada na grande maioria dos sistemas propostos. Excepção é o TinyOS que é escrito numa linguagem própria, o nesC.

Na Tabela 3.3 encontra-se ilustrado as principais características técnicas dos sistemas operativos mencionados.

Tabela 3.3 – Características dos sistemas operativos mencionados

Sistema operativo	*Escalonamento*	*Modelo de execução*	*Determinismo temporal*	*Linguagem suportada*	*Memoria utilizada*
freeRTOS	Cooperativo, preemptivo	Thread	Sim	C,assembly	4Kb- 9Kb
eCOS	Preemptivo	Thread	Sim	C, C++	Configurável
Mc/OS-II	Preemptivo	Thread	Sim	C,assembly	5Kb-24Kb
RTAI	Preemptivo	Thread	Sim	C	630Kb
TinyOS	Cooperativo	Eventos	Não	nesC	Configurável
Contiki	Cooperativo, preemptivo	Eventos	Não	C	2Kb RAM,40Kb de ROM

Capítulo 4

Middleware

4.1 - Introdução

Uma característica presente em aplicações distribuídas diz respeito ao seu ambiente heterogéneo: as aplicações encontram-se distribuídas em diferentes plataformas de hardware, muitas das vezes com diferentes sistemas operativos e que comunicam recorrendo a diversos protocolos de comunicação. Estas características fazem com que o desenvolvimento de aplicações seja complexo para os programadores. Neste contexto é geralmente introduzido o conceito de *middleware* como camada de abstracção entre as diferentes aplicações. O uso de um middleware possibilita a comunicação entre as diversas aplicações de forma transparente, fornecendo um conjunto de serviços que possibilitam a integração de aplicações e respectiva interoperabilidade entre as mesmas. O middleware realiza assim a mediação entre o software distribuído, ocultando do programador as diferenças de protocolos de comunicação, plataformas e sistemas operativos.

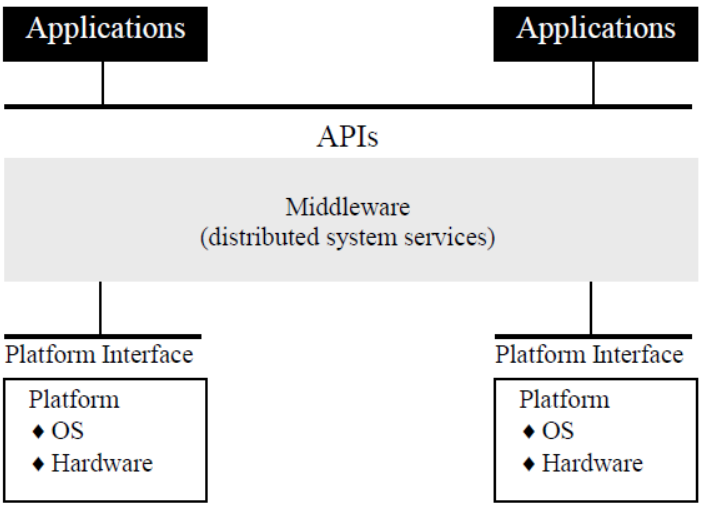

Figura 4.1 – *Middleware* num sistema distribuído[43]

Os serviços prestados pelo *middleware* são de propósito geral, o que significa que os serviços não são dependentes da plataforma ou aplicação. Os serviços oferecidos atendem geralmente as necessidades de uma gama de aplicações de um determinado domínio, não se focando apenas nos serviços necessários para uma aplicação específica. Para além de serviços de comunicação (identificação, autenticação, autorização, encaminhamento de dados, etc), o *middleware* inclui também funcionalidades relacionadas com segurança , gestão e energia da rede. Todos estes serviços são implementados sobre a formas de APIs(*Application Programming Interface*) fornecidas pelo *middleware*. Estas APIs constituem geralmente o único conhecimento que os programadores possuem do *middleware* e ao fornecerem um conjunto de funções e interfaces padrão, permitem assim uma abstracção de alto nível do sistema global distribuído. O sistema passa a ser visto pelas diversas aplicações e pelo programador com um sistema homogéneo uma vez que heterogeneidade é tratada pela camada de *middleware*. O desenvolvimento das aplicações é assim bem mais simples, visto que o programador só terá de se concentrar com os detalhes e funcionalidades das aplicações.

De acordo com modelo usado para estabelecer a comunicação entre aplicações, os *middleware* podem ser classificados em quatro diferentes categorias [44]:

- *Message Oriented Middleware* (MOM)
- *Remote Procedure Calls* (RPC)
- *Object Oriented Middleware* (ORB)
- *Transaction Processing Monitors* (TPMON)

O *Middleware* orientado a mensagens usa o paradigma de comunicação por mensagens para promover a comunicação entre aplicações distribuídas. Genericamente este *middleware* estabelece dois tipos de comunicação: directa e indirecta. Quando a comunicação é realizada de forma directa a mensagem é enviada através do *middleware* de forma directa entre aplicações, sendo necessário o estabelecimento de uma conexão lógica entre as referidas aplicações. Este tipo de comunicação fornece um modelo de comunicação síncrono entre aplicações. Na comunicação indirecta são usadas filas de mensagens para promover armazenamento temporário de mensagens e a comunicação assíncrona entre aplicações. Neste caso as mensagens são armazenadas a mediada que vão sendo enviadas e tratadas pelo receptor de acordo com a posição na fila e/ou ordem de importância. As mensagens contêm um assunto/tópico que permite aos clientes efectuar a respectiva filtragem de mensagens. Este facto, permite que não seja necessário que o emissor e o receptor se encontrem activos ou sincronizados durante a transmissão da mensagem, nem exigido o conhecimento da localização dos diversos clientes. Desta forma, a remoção ou adição de um ou mais componentes do sistema não prejudica o funcionamento global do mesmo. O *middleware* orientado a mensagem constitui uma boa opção para a implementação de arquitecturas *publish-subscribe* e serviços de eventos.

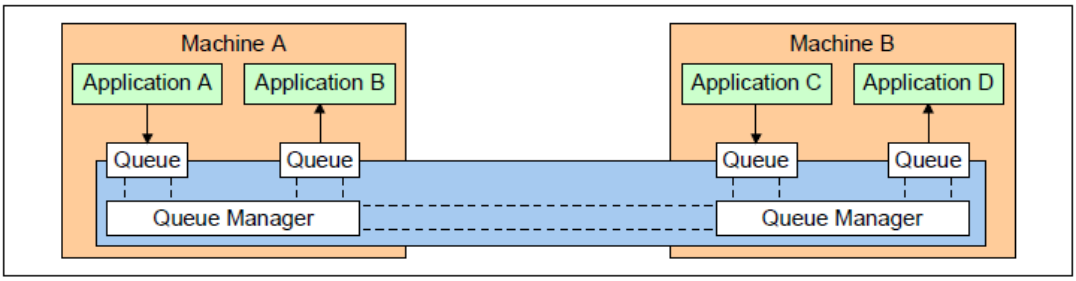

Figura 4.2 - Middleware orientado a mensagem[45]

O *middleware* do tipo RPC apresenta uma arquitectura cliente-servidor para permitir a interacção entre dois processos que executam em plataformas distintas como se estivessem a executar localmente. A comunicação é realizada sobre a forma de procedimentos ou chamadas entre aplicações. O cliente constitui o elemento que requer o serviço, sendo ele que localiza e efectua a conexão ao servidor, efectuando um pedido através da passagem de um serie de parâmetros. O servidor, entidade responsável pela prestação de um serviço requerido, avalia os parâmetros e retorna um resultado, que é transferido para o cliente. Todo o procedimento de comunicação é baseado num modelo de interacção síncrono. Assim ao realizar uma chamada, um determinado processo permanecera em estado de espera até receber resposta referente a chamada executada. Quer o cliente quer e o servidor possuem interfaces lógicas localmente denominadas *stub*. O *stub* actua como tradutor da informação passada entre o cliente e o servidor, sendo responsável por estabelecer um protocolo de comunicação comum entre os aplicativos servidor e cliente. Os dois *stubs* regulam todos os detalhes das comunicações, o que permite que os programas construídos usem as chamadas a procedimentos como se fossem locais.

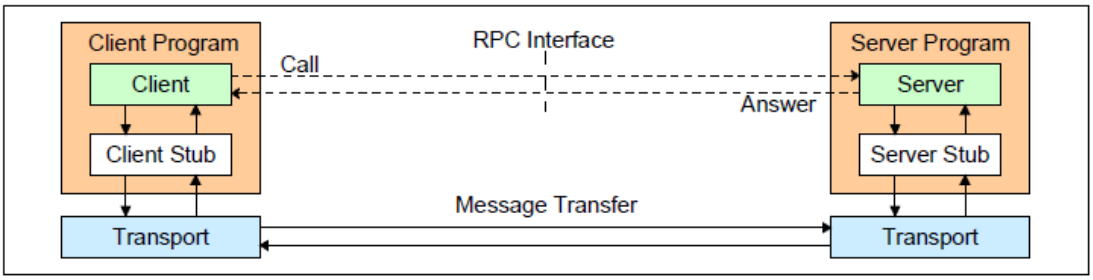

Figura 4.3 - Middleware RPC[45]

Um *middleware* orientado a objectos, também conhecido por *Object Request Broker*, apresenta um funcionamento semelhante ao *middleware* RPC, usando o paradigma de procedimentos ou chamadas remotas sobre uma estrutura distribuída orientada a objectos. O *middleware* fornece mecanismos que possibilitam a troca de parâmetros entre objectos independentemente da sua localização e independentemente do protocolo de comunicação. É assim possível objectos clientes solicitam a execução de uma operação em um objecto servidor. A comunicação é tipicamente síncrona e portanto o objecto cliente que requisitou o serviço permanece bloqueado enquanto não obtiver a resposta. Contudo existem middlewares orientados a objectos que fornecem mecanismos de suporte a comunicação assíncrona entre

objectos. Como é visível na figura 4.4, o middleware é responsável por todos os mecanismos necessários para assegurar a comunicação entre objectos: localização, conexão e activação.

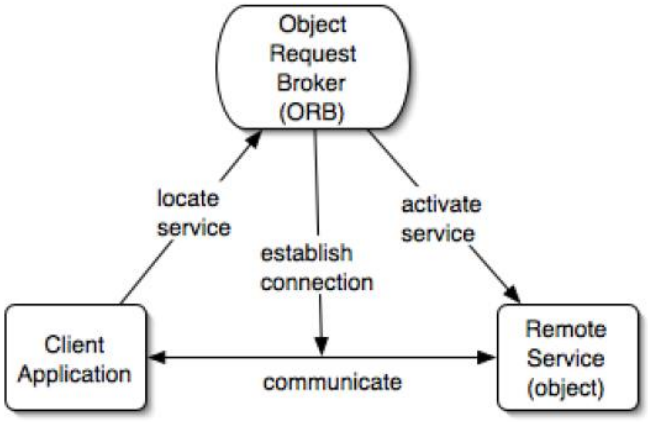

Figura 4.4 - Objectos interagindo através do middleware [46]

O modelo de *middleware* transacional é principalmente utilizado em arquiteturas em que os componentes são aplicações de base de dados. Esses sistemas de *middleware* suportam transações que envolvem componente executando em diferentes *hosts ou* plataformas: um componente cliente agrupa várias operações em uma transação que o *middleware* então transmite pela rede para os componentes servidores de maneira transparente

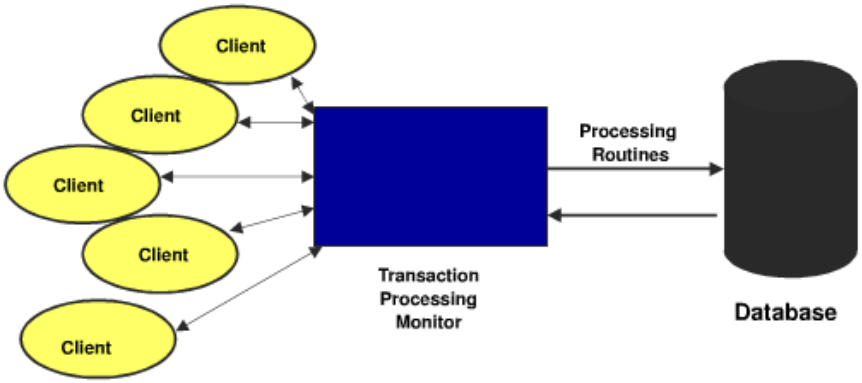

Figura 4.5 - Middleware transacional

4.2 - Requisitos

O *middleware* é responsável por coordenar a interacção entre aplicações distribuídas e isolar as aplicações das características de infra-estrutura e dos protocolos de comunicação subjacentes. Como já foi referido, este deve fornecer serviços de âmbito geral focando-se nas necessidades de um grande número de aplicações, independentemente das plataformas e sistemas operativos usados. Além disso o *middleware* deve oferecer confiabilidade e escalabilidade. Estas constituem as características gerais geralmente associadas a um middleware.

No âmbito deste trabalho características adicionais são exigíveis da plataforma de *middleware*, nomeadamente:

- Reduzida utilização de memória: as aplicações serão desenvolvidas em sistemas embebidos. O *Middleware* deverá ser de tamanho reduzido e permitir utilização eficiente da memória disponível. O middleware deverá ser configurável no sentido de permitir utilizar apenas os serviços necessários e assim rentabilizar a memória utilizada;
- Linguagem de programação: a linguagem de programação suportada pelo *middleware* deve ser do conhecimento dos alunos (C,C++,Java);
- *Open-source*: se possível o *middleware* deverá possuir uma licença livre de custos
- Determinismo temporal: o *middleware* deverá fornecer mecanismos que possibilitem a sua utilização em sistemas com requisitos temporais. Neste sentido deve minimizar o *overhead* e permitir o estabelecimento de prioridades entre serviços;
- Suporte documental: o *middleware* deve estar devidamente documentado. A existência de informação referente a especificação do *middleware* que possibilite uma fácil aprendizagem do funcionamento deve ser levada em conta.

4.3 - Alguns exemplos de plataformas de middleware

Actualmente existe uma grande variedade de plataformas de *middleware* disponíveis vocacionadas para os mais variados tipos de aplicações. Alguns exemplos de *middlewares* existentes são:

- *Middleware* de uso geral: CORBA[47], TAO[48], CIAO[49], DCOM[50], Java RMI [51], OpenCOM[52]
- *Middleware* vocaciona para redes sem fio e dispositivos móveis: MiLAN[53], Mires[54], Impala[55], Cougar [56], Miro [57].
- *Middleware* para sistemas embarcados: RTZen[58], e* ORB[59], ROFES[60], MidART[61], OSA+[62].

Apesar da grande variedade de soluções disponíveis, foram sentidas dificuldades em encontrar *middlewares* adequados no âmbito deste trabalho. Por um lado, a informação referente a grande parte dos *middleware* é escassa e muitos deles representam produtos comerciais com custos associados. Além disso muitos dos *middleware* apresentados são complexos, exigindo do utilizador alguma experiencia neste tipo de plataformas e a sua

implementação exigiria aos alunos um investimento de tempo maior que o aceitável, não praticável durante um semestre.

Assim uma alternativa passa pela utilização de sistemas operativos que forneçam alguns dos serviços que são implementados pelas plataformas de middleware. Embora a pesquisa efectuada relativamente aos sistemas operativos não tenha sido focada neste tipo de serviços, alguns dos sistemas que foram propostas apresentam características que devem ser consideras. Concretamente o sistema operativo eCos, o µC/OS-II e TinyOS. Este sistemas operativos oferecem um conjunto de pacotes de software adicionais que podem ser adquiridos aquando da aquisição do sistema e que possibilitam troca de mensagens síncrona e assíncrona (no caso do TinyOs) entre diferentes aplicações e diferentes nós distribuídos.

Na próxima secção será descrito um dos *middleware* abordados ao longo da pesquisa efectuada.

4.3.1 - OSA+

O OSA+ é um *middleware* escalonável vocacionado para sistemas embutidos desenvolvido por investigadores da universidade de Karlsruhe. Foi projectado para funcionar em ambientes heterogéneos e facilitar o desenvolvimento de aplicações distribuídas de tempo real. Este *middleware* é baseado numa arquitectura microkernel e caracteriza-se pela reduzida quantidade de memória que ocupa. Concretamente o tamanho do *middleware* varia entre 29 KB e 75KB. Esta variação deve-se ao facto do *middleware* ser altamente escalonável possibilitando a adição e remoção de funcionalidades e serviços em função das aplicações a que se destina.

O funcionamento do OSA é orientado a serviços. As aplicações são vistas como fornecedores de serviços que são partilhados pelas diversas aplicações em ambiente de execução através do *middleware*. Associado aos serviços aparece o conceito de *Job* . *Job* é a entidade responsável pelo fornecimento dos serviços entre aplicações, consistindo de uma ordem e de um resultado correspondente. A ordem representa um pedido de um serviço enquanto o resultado se refere a conclusão da ordem efectuada.

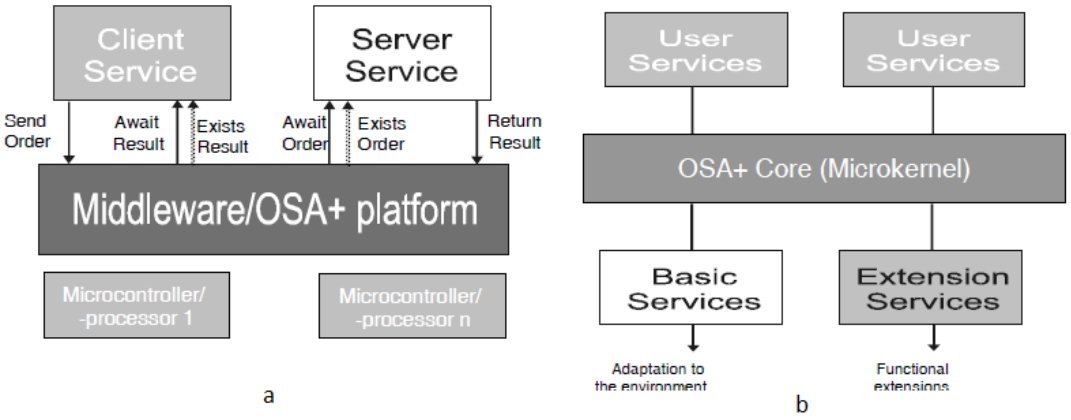

Figura 4.6 – (a)Arquitectura baseada em serviços[62]; (b) Arquitectura microkernel OSA+ [62]

Seguindo a abordagem microkernel a plataforma OSA+ consiste de uma plataforma central de pequeno porte que oferece as funcionalidades básicas para se adaptar às exigências das aplicações e do ambiente de execução. Estas funcionalidades incluem[62] :

- Gestão de serviços: a plataforma oferece mecanismos para instalar, remover e localizar serviços, bem como para oferecer Jobs. Os serviços podem ser locais ou globais, sendo que cada serviço tem um nome único no sistema global, e pode ser dinamicamente conectado em tempo de execução;
- Gestão de *Jobs*: *Job* é a entidade responsável pelo fornecimento dos serviços entre aplicações. O *middleware* é responsável pela parametrização, criação e destruição de Jobs. Tal é conseguido através de seis funções fornecidas pelo middleware: *sendOrder(), awaitOrder(), existsOrder(), returnResult(), awaitResult()* e *existsResult()*. A comunicação entre serviços poderá ser assíncrona (através do uso das funções *existOrder()* e *existResult()*) ou síncrona (usando as funções *awaitOrder()* and *awaitResult()*).
- Suporte básico de qualidade de serviço: é possível solicitar e especificar parâmetros como: prazos, prioridades, largura de banda de rede, etc.
- Interface com o utilizador: é fornecido um conjunto de APIs simples e consistentes que permitem ao programador lidar com serviços e *Jobs*.

A plataforma central é completamente independente do hardware e do sistema operacional usado. Tal é conseguido através de serviços denominados serviços básicos que são utilizados para a adaptação a um hardware e sistema operativo específicos. De referir a existência também de serviços de extensão utilizados para estender funcionalidade da plataforma e normalmente não dependem directamente do sistema operacional ou do hardware. Exemplos típicos desses serviços são os serviços de registo e os serviços de segurança. O *middleware* suporta a linguagem C e Java.

Capítulo 5

Arquitecturas de software

Neste capítulo serão propostas diferentes arquitecturas que permitam a implementação de alguns dos sistemas descritos no capítulo 2. Serão descritas diferentes abordagens que permitem tirar partido da utilização de sistemas operativos e *middleware* no desenvolvimento de diferentes sistemas. Será abordado apenas o sistema de travagem, pois este capítulo pretende apenas exemplificar a forma como a implementação e desenvolvimento dos sistemas poderá ser realizado. No entanto, muitos dos conceitos e arquitecturas aqui abordadas poderão ser aplicados aos restantes sistemas.

5.1 - Introdução

O sistema de travagem constitui um dos sistemas com maior importância presente no veiculo pois esta directamente relacionado com o controlo do mesmo. Este sistema inclui um conjunto de subsistemas: ABS, TCS, ESP, EBA e EBD. Todos estes subsistemas fazem uso dos travões para fornecer ao condutor funções de assistência nas mais variadas situações e encontram-se geralmente incorporados na mesma ECU.

Uma característica comum dos diferentes subsistemas aqui apresentados diz respeito ao facto de todos eles efectuarem a monitorização de sinais provenientes dos diversos sensores e em função dos valores provenientes destes actuar de acordo com especificação.

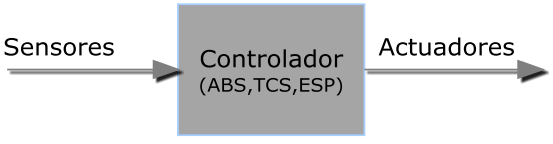

Figura 5.1 – Sistema computacional

Apesar de integrados na mesma ECU estes subsistemas apresentam especificações distintas e actuam em contextos diferentes:

- Durante a travagem: ABS, EBD, EBA, ESP
- Durante a aceleração: TCS, ESP

As actividades associadas a cada um dos sistemas são implementadas através de tarefas de software. A implementação destas tarefas poderá seguir diferentes abordagens. Nas secções seguintes serão abordadas diferentes arquitecturas de software que possibilitam a implementação dos sistemas aqui mencionados. Serão descritas algumas arquitecturas, desde uma arquitectura monotarefa, multitarefa e tolerantes a falhas.

5.1.1 - Proposta 1

Uma abordagem possível consiste em realizar todas as actividades referentes aos sistemas numa única tarefa periódica, tal como se encontra ilustrado na figura 5.2.

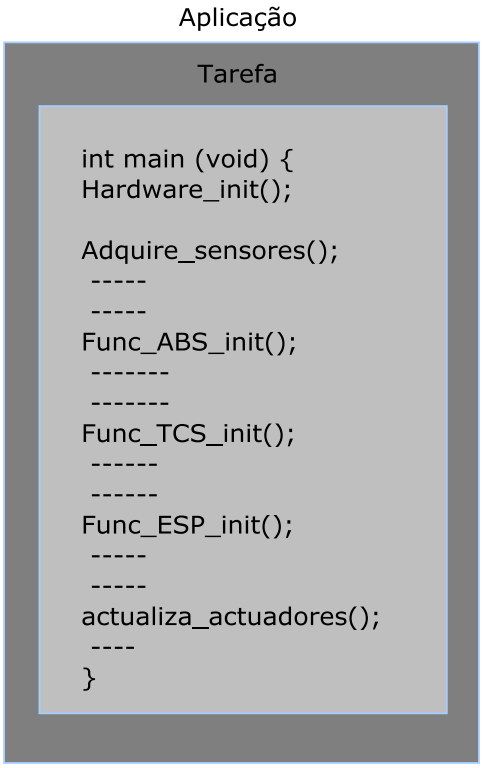

Figura 5.2 - Tarefa única executa os diferentes sistemas

O uso de uma tarefa periódica garante uma certa previsibilidade ao sistema pois é garantido que as actividades referentes aos sistemas são executadas em períodos fixos. A tarefa em questão é constituída por um conjunto de funções que implementam as actividades necessárias para a implementação dos diferentes sistemas: aquisição dos diversos sensores (pedal do travão, velocidades das rodas, ângulo do volante, etc.), algoritmos referentes ao ABS, TCS e ESP, e a actuação nos travões. Esta abordagem é simples de implementar, no entanto apresenta alguns inconvenientes. O facto de todos os subsistemas serem executados no contexto da mesma tarefa obriga a que sejam executados com a mesma periodicidade, o que pode não ser o pretendido. Por outro lado a dimensão do código da tarefa poderá resultar

no incumprimento de *deadlines* por incapacidade do hardware usado. Além disso qualquer bloqueio numa das secções de código resultará na falha de todos os subsistemas.

Esta abordagem poderá ser implementada sem o uso de um sistema operativo, mas tal opção implica que a gestão de interrupções de hardware e a interface de rede seja desenvolvida e implementada pelo programador.

5.1.2 - Proposta 2

Atendendo ao facto de que os subsistemas apresentam especificações diferentes, cada um dos sistemas poderá constituir uma tarefa que é responsável pela realização das actividades específicas para cada sistema. Desta forma teremos várias tarefas distintas referentes aos principais sistemas que executam em paralelo e de forma concorrente.

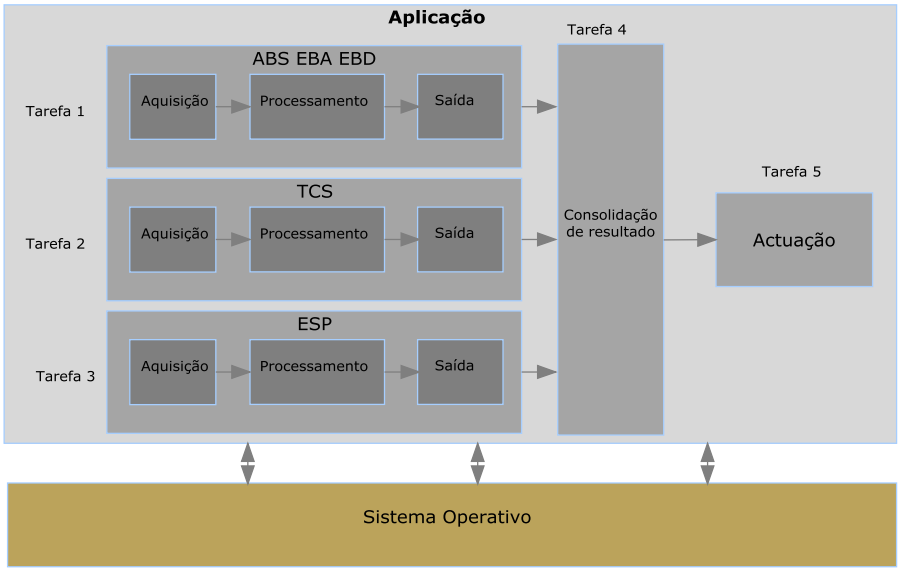

Figura 5.3 – Uma tarefa por sistema

Desta forma teremos três tarefas específicas referentes aos sistemas ABS (EBD, EBA), TCS e ESP. Os sistemas ABS, EBD e EBA são aqui tratados pela mesma tarefa pois estes sistemas executam em conjunto e no mesmo contexto. Cada uma destas tarefas realiza as actividades referentes ao respectivo sistema: aquisição dos dados dos sensores, processamento e geração dos dados de saída. A utilização de uma tarefa específica para cada subsistema, permite que estes sejam executados com períodos diferentes e permite especificar *deadlines* específicas para cada um dos subsistemas.

A tarefa 4 é responsável por gerir os comandos provenientes das restantes tarefas no sentido de assegurar a existência de consensos e de definir qual a ordem que deve ser executada pelos actuadores num dado instante. Tal necessidade advém do facto de todos os sistemas aqui presentes actuarem sobre os mesmos actuadores. Concretamente numa situação em que o ABS e o ESP enviem comandos no sentido de definir a pressão a ser aplicada a cada uma das rodas, estes podem diferir ou ser mesmo contraditórios. É necessário a implementação de um algoritmo que tendo em conta a dinâmica do veículo, a segurança e as possíveis situações que poderão ocorrer, defina qual a ordem que deve ser enviada aos

actuadores. Atendendo a criticidade de cada sistema, a prioridade de envio poderá ser a seguinte:

1. ESP
2. ABS ou TCS

Sendo o ESP o sistema que de forma mais directa interfere com a estabilidade do veiculo, as ordens enviadas por este sistema devem prevalecer sobre os demais. O ABS e o TCS executam em contextos diferentes pelo que em nenhuma situação existiram comandos dos dois sistemas em simultâneo.

Visto que as diversas tarefas executam em paralelo e de forma concorrente, e atendendo a necessidade de determinismo temporal das mesmas, é conveniente o uso de uma sistema operativo de tempo real multitarefa que forneça algoritmos de escalonamento de tarefas. Assim os sistemas operativos freeRTOS, µC/OS-II, e eCos poderão ser aqui utilizados, pois possuem as características necessárias.

5.1.3 - Proposta 3

Cada subsistema realiza uma sequência de actividades específicas: aquisição, processamento e actuação. Desta forma poderemos considerar cada uma destas actividades como uma tarefa periódica tal como ilustrado na Figura 5.4.

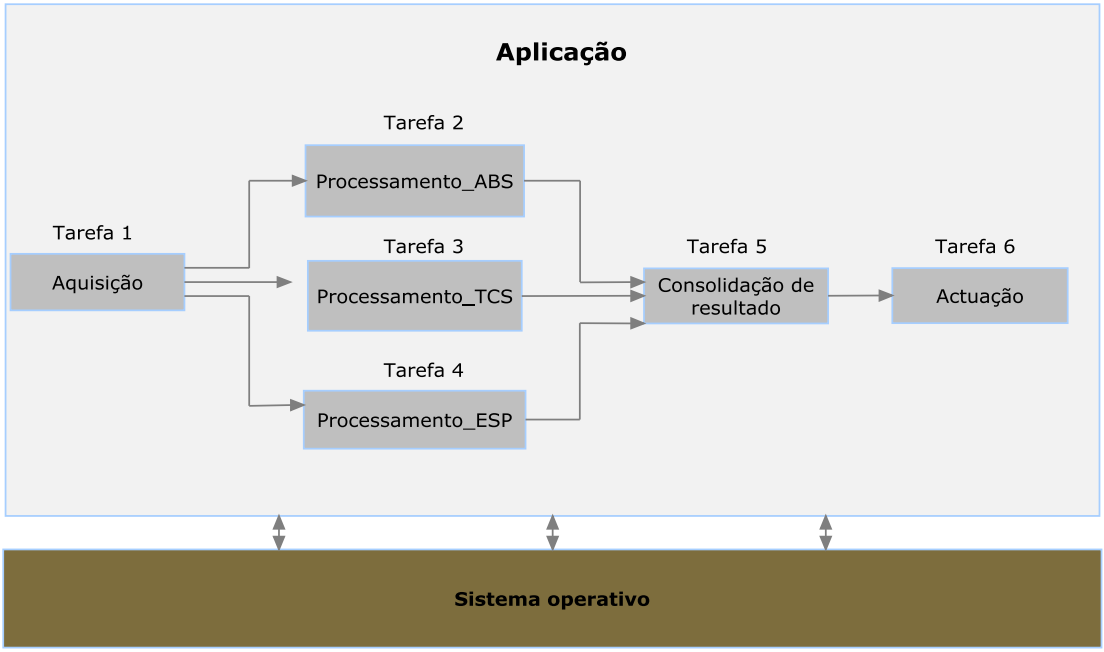

Figura 5.4 – Diferentes tarefas para o mesmo sistema

A tarefa 1 é referente a aquisição dos dados dos diferentes sensores. Desta forma a aquisição dos diversos sensores é realizada em simultâneo e como o mesmo período de aquisição. As tarefas 2,3 e 4 são responsáveis pela implementação dos algoritmos e geração de dados de saída referentes aos diferentes sistemas. As tarefas 5 e 6 definem os comandos a serem enviados para os actuadores.

As diferentes tarefas possuem requisitos de procedência. Por exemplo, a execução das tarefas 2,3 e 4 exige que a tarefa 1 tenha já concluído a sua execução. Desta forma, esta abordagem requer o uso de um sistema operativo de tempo real multitarefa que forneça mecanismos de escalonamento de tarefas e a respectiva comunicação e sincronização entre as mesmas.

5.1.4 - Proposta 4

Os sistemas aqui abordados constituem sistemas críticos pois uma falha ou erro na especificação poderão ser prejudiciais para os ocupantes. Consideremos o caso do ESP. Este sistema efectua monitorização de diversos sensores que lhe permitem conhecer a direcção actual do veículo e a direcção pretendida pelo condutor. Em caso de discrepância o ESP efectua a força de travagem numa das rodas para efectuar a devida correcção. Supondo agora que devido a uma falha de software ou hardware, o sistema ESP envia comandos errados ou fora do contexto, tal situação poderá resultar num acidente grave.

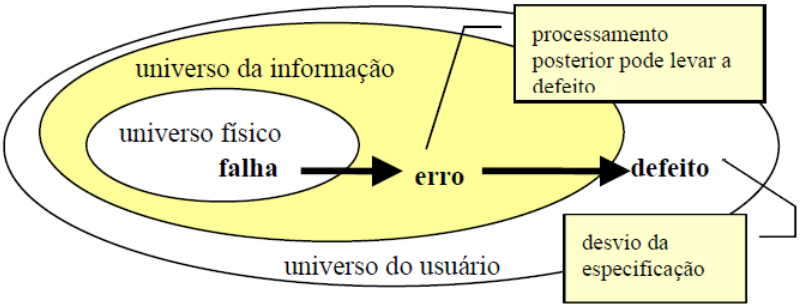

Figura 5.5 – Falha, erro e defeito de um sistema [63]

Em sistemas críticos é comum a introdução de mecanismos que permitam a tolerância a falhas. Tais mecanismos incidem geralmente na introdução de redundância.

No contexto deste trabalho, uma abordagem comum consiste em utilizar componentes de software redundantes com concepções e implementações distintas. Estes componentes denominados variantes, consistem em várias versões de softwares que geralmente seguem as seguintes metodologias:

- Uso de linguagens diferentes para cada versão
- Utilizar, se possível, diferentes abordagens para o mesmo algoritmo
- O desenvolvimento de cada versão realizado por diferentes programadores

Desta forma é possível tolerar possíveis erros de software que sejam cometidos na fase de concepção e não detectados. Consideremos uma versão simplificada da proposta 3 apresentada na secção anterior.

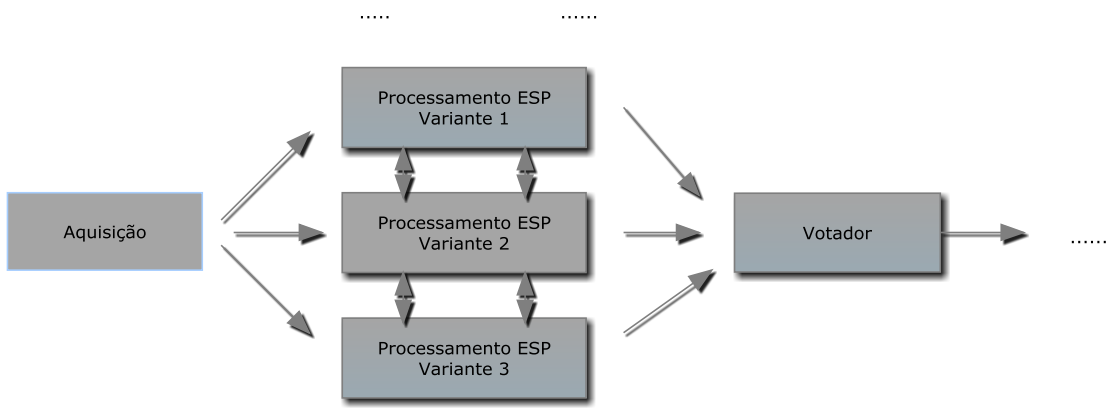

Figura 5.6 – Replicação de tarefas

A tarefa referente ao processamento do ESP foi replicada. Existem agora três variantes que executam as suas actividades paralelamente e de forma sincronizada. A necessidade de sincronismo entre variantes é essencial pois é necessário garantir que estas estão a tratar entradas adquiridas no mesmo instante ou suficientemente próximos.

Os resultados provenientes das variantes são em seguida transmitidos a um votador.Este representa aqui o elemento decisor responsável por determinar um resultado de acordo com algum critério. A estratégia adoptada no critério de decisão poderá seguir diferentes abordagens: maioria (2 em 1), mediana, média, maior ou menor dos valores processados. Se o votador conseguir determinar um resultado, é sinalizada a tarefa seguinte desta ocorrência. No entanto, neste contexto duas situações poderão ocorrer: os resultados provenientes das três variantes são demasiados diferentes e como tal não é permitido ao votador determinar um resultado válido, ou uma das variantes não devolve o resultado no tempo exigido. Em ambas as situações é gerada uma excepção.

A tarefa de aquisição poderá também utilizar mecanismos de detecção de falhas. Atendendo a dinâmica do veículo não serão de espera mudanças buscas nas variáveis adquiridas entre duas aquisições consecutivas. Consideremos que a tarefa aquisição é executada com um período de 10ms e o caso concreto da aquisição da velocidade das rodas.

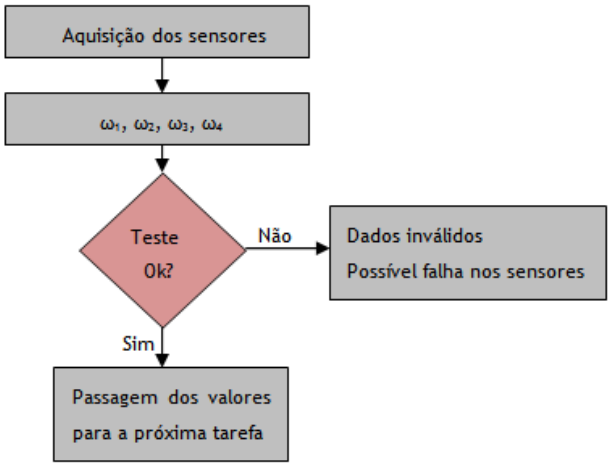

Figura 5.7 – Teste dos valores da velocidade adquiridos

Após a aquisição da velocidade das rodas alguns testes poderão ser realizados no sentido de garantir a confiabilidade dos dados, nomeadamente:

- Testes derivativos: a análise da variação da velocidade de cada uma das rodas (ω_1, ω_2, ω_3, ω_4) permitirá detectar possíveis falhas nos sensores. Entre aquisições consecutivas e atendendo os períodos de aquisição uma variação brusca ou superior a um determinado valor, significará uma possível falha ocorrida no processo de aquisição (ou no sensor);
- Para cada aquisição calcular a média da velocidade dos valores adquiridos. Se uma das rodas possuir uma velocidade superior ou inferior a um valor pré-definido (*threshold*), tal poderá significar um defeito no sensor;
- Análise da gama de valores. Atendendo a aplicação que se trata é de esperar que os valores se encontrem dentro de uma determinada gama. Valores negativos poderão também ser significar falhas nos sensores;

Os testes referidos podem também ser aplicados as restantes variáveis adquiridas. Esta arquitectura requer também a utilização de um sistema operativo, devido a execução de múltiplas tarefas em execução e a necessidade de estabelecer a devida comunicação e sincronização entre as mesmas. Atendendo as características de tempo real da aplicação e a necessidade de determinismo temporal das tarefas aqui presentes, os sistemas operativos freeRTOS, µC/OS-II, e ecos poderão ser aqui utilizados. Comparativamente com as propostas anteriores, esta abordagem permite introduzir tolerância a falhas, no entanto é mais trabalhosa e complexa de implementar: utilização de diferentes linguagens e diferentes abordagens para o mesmo algoritmo e necessidade de sincronização entre variantes.

5.1.5 - Proposta 5

As falhas poderão ocorrer devido a anomalias no software mas também no Hardware. As falhas de hardware são geralmente classificadas em relação à sua duração:

- Falhas permanentes: resultam de um defeito físico irreversível e permanecem indefinidamente até serem reparadas;
- Falhas intermitentes: falhas temporárias que ocorrem repetidamente;
- Falhas transitórias: falhas temporárias que ocorrem ocasionalmente num muito curto espaço de tempo. São as mais frequentes e mais difíceis de detectar, pois podem ser causadas por oscilações na corrente eléctrica, interferências electromagnéticas ou radiação;

Uma abordagem possível para detectar falhas transitórias consiste no uso de redundância temporal, que consiste em executar uma determinada acção ou tarefa em instantes de tempo diferentes (mas próximos) usando o mesmo software e os mesmos recursos de hardware. Consideremos uma versão simplificada da proposta 3, tal como ilustrado na Figura 5.8.

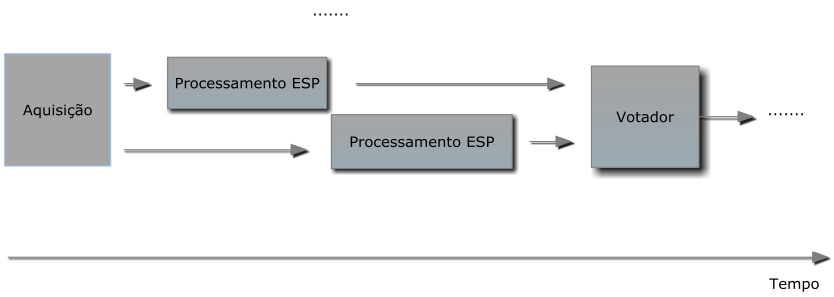

Figura 5.8 - Tarefa ESP com redundância temporal

A tarefa de processamento do ESP é executada duas vezes em instantes de tempo diferentes, mas usando os mesmos dados de entrada. O votador compara os dados provenientes das duas tarefas de processamento no sentido de detectar possíveis falhas transitórias. Concretamente, resultados diferentes são uma forte indicação da ocorrência de uma falha transitória.

Comparativamente com a proposta anterior, redundância temporal não necessita de software ou hardware redundantes o que se traduz numa redução dos custos de implementação, contudo exige maior tempo disponível para a execução das tarefas pois executa a mesma tarefa duas vezes.

5.1.6 - Proposta 6

No funcionamento dos sistemas ESP e ABS, o envio de comandos para os travões é geralmente acompanhado do envio de comandos para o motor com finalidade de reduzir o binário aplicado as rodas. Esta operação exige assim a troca de dados entre a ECU dos travões e a ECU do motor. Por outro lado, os sensores e actuadores encontram-se distribuídos pelo veículo uma vez que existe a necessidade de estes se encontram próximos da sua área de operação. Quer isto dizer que estamos perante uma arquitectura distribuída de implementação complexa.

Dado que a implementação será feita por alunos no âmbito das unidades curriculares do curso, o desenvolvimento deste sistema poderá ser demasiado complexo ou não exequível para um único grupo de trabalho. Desta forma poderá ser realizada uma divisão em diferentes módulos/aplicações, de modo a que o sistema possa ser implementado paralelamente por diferentes grupos de alunos.

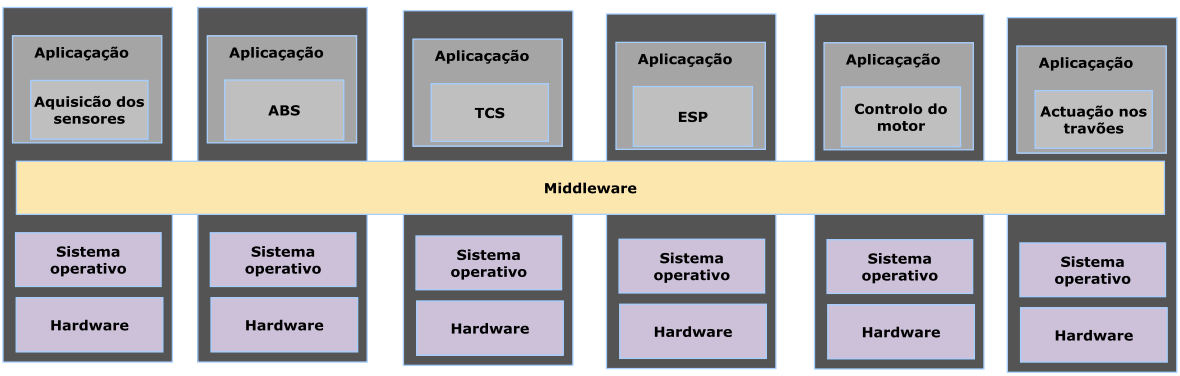

Figura 5.9 – Arquitectura distribuída

O módulo de aquisição é responsável pela aquisição dos dados dos sensores e respectivo processamento. Depois de processadas as variáveis provenientes dos sensores são enviadas para a rede de forma a serem utilizadas pelos módulos que necessitem.

Os diferentes módulos ABS, ESP, TCS executam um conjunto de tarefas responsáveis por implementar os algoritmos referentes a cada um dos sistemas. O módulo de actuação é responsável por actuar nos travões em função dos dados recebidos pelas restantes módulos (a excepção do modulo de aquisição). Esta abordagem permite a cada grupo de alunos se concentre numa tarefa específica. Qualquer um dos módulos poderá implementar as suas actividades seguindo um das abordagens descritas nas secções anteriores: implementação de uma única tarefa, tarefas replicadas, redundância temporal, etc.

A necessidade de troca de dados entre diferentes plataformas requer o uso de uma rede que possibilite essa operação. A rede CAN poderá ser uma boa aposta dado que esta é estudada no âmbito do curso, e como tal é já do conhecimento dos alunos. Além disso CAN fornece mecanismo de tolerância a falha, detecção de erros e o envio de mensagens com diferentes prioridades.

Dado que cada uma das aplicações será desenvolvida por grupos de trabalho diferentes, estes poderão optar por usar hardware e software diferentes. Além disso, um dado microcontrolador poderá ser o mais adequado para usar na aplicação de aquisição, mas não cumprir os requisitos essências para executar o algoritmo do TCS. Esta situação resulta numa arquitectura heterogénea. O uso de um *middleware* possibilita a comunicação entre as diversas aplicações de forma transparente, fornecendo um conjunto de serviços que possibilitam a integração de aplicações e respectiva interoperabilidade entre as mesmas. Neste caso concreto o uso de um *middleware* orientado a mensagens, seguindo uma arquitectura do tipo *publish-subscribe* seria o mais adequado. Consideremos o exemplo da plataforma da aquisição dos sensores, tal como ilustrado na Figura 5.10.

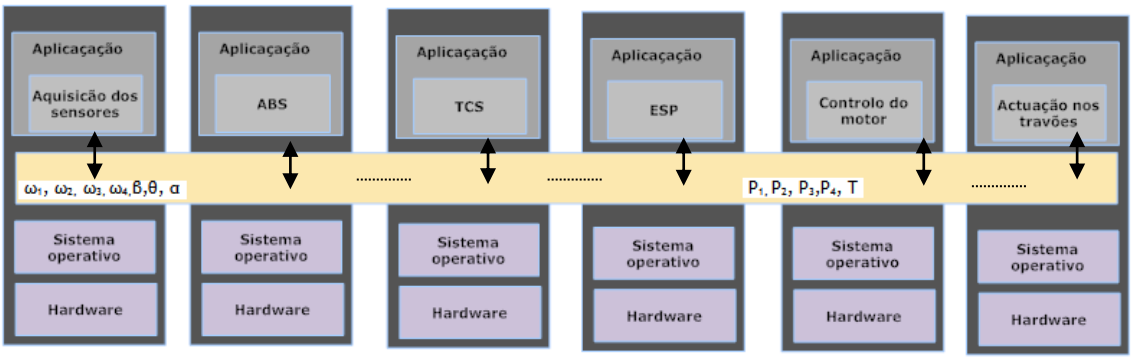

Figura 5.10 – Troca de dados entre aplicações

A plataforma de aquisição adquire a diferentes variáveis necessárias: velocidade das rodas (ω_1, ω_2, ω_3, ω_4), ângulo do volante (θ), aceleração lateral (α) e momento angular do veículo (β). Quando esta plataforma termina a sua tarefa, os dados resultantes da sua execução são 'publicados' através do *middleware*. As restantes aplicações consultam o *middleware* e subscrevem as variáveis caso estas lhes sejam importantes no contexto da sua execução. O mesmo acontece para as plataformas de processamento do ABS, TCS e ESP. Atendendo ao tipo de aplicações aqui abordadas o *middleware* deverá fornecer mecanismos de tolerância a falhas, detecção de erros e o uso de prioridades.

Capítulo 6

Conclusão

Este trabalho foi elaborado em de três fases distintas: estudo dos sistemas electrónicos embarcados nos veículos, levantamento de plataformas de software e por fim foram propostas arquitecturas de software.

A primeira fase incidiu sobre os sistemas electrónicos embarcados nos veículos actuais. Foi realizado um levantamento dos diversos sistemas, seguida da descrição do seu princípio de funcionamento: hardware necessário, variáveis envolvidas e controladas, etc. A informação recolhida foi obtida essencialmente através da consulta de material fornecido pelas principais marcas de automóveis e na consulta de artigos. Atendendo ao facto de que os sistemas se encontram distribuídos pelo veículo com necessidade de trocas de dados entre si, foram também abordadas as principais redes de comunicação usadas nos veículos. Importa referir que muitas das redes mencionadas poderão não ser praticáveis e adequadas no contexto deste trabalho, no entanto estas foram aqui abordadas para fins ilustrativos.

Na segunda fase foi realizado um estudo relativo as plataformas de software existentes passíveis de ser utilizadas no desenvolvimento e implementação dos sistemas anteriormente enumerados. A pesquisa foi focada em sistemas operativos e plataformas de *middleware* para sistemas embarcados. Em primeiro lugar foi realizado um levantamento relativo aos sistemas operativos. Atendendo a grande variedade de soluções disponíveis, elaborou-se uma lista de requisitos que estas plataformas deveriam cumprir. Foi especialmente difícil encontrar sistemas operativos cuja especificação se encontra-se devidamente documentada, o que constituía um requisito essencial no contexto deste trabalho. No final desta fase foram propostos seis sistemas operativos e comparadas as principais características.

Seguidamente foram abordadas as plataformas de *middleware*. Aqui, foram sentidas grandes dificuldades em encontrar plataformas adequadas ao contexto deste trabalho. Concretamente, a grande maioria dos *middlewares* encontrados possuíam custos associados e não se encontravam devidamente documentados. Além disso muitos deles apresentam uma complexidade elevada e a sua implementação exigiria aos alunos uma investimento de tempo maior que o aceitável, não praticável durante um semestre.

Na terceira e última fase do trabalho foram propostas arquitecturas de software para o veículo eléctrico. Estas constituem diferentes abordagens que poderão ser seguidas na implementação dos diferentes sistemas. Embora tenha sido tratado apenas o sistema de travagem, os conceitos abordados são aplicados aos restantes sistemas presentes nos veículos.

Por fim, importa referir que este documento pretende servir de apoio aos alunos aquando do desenvolvimento de sistemas no veículo. São descritos genericamente o funcionamento dos sistemas ficando a cargo dos alunos a implementação dos algoritmos necessários. A implementação de tais algoritmos requer o estudo aprofundado do funcionamento e da dinâmica do veículo, nomeadamente das variáveis mecânicas e físicas envolvidas. Tal facto advém do controlo efectuado por alguns dos sistemas que foram abordados estar directamente relacionado com a dinâmica do veículo. A escolha de um sistema operativo e/ou *middleware* apropriado esta directamente relacionado com aplicação a que se destinada, cabendo também aos alunos analisar qual deles cumpre os requisitos para uma aplicação específica.

Referências

[1] "CO2 EMISSIONS FROM FUEL COMBUSTION," International Energy Agency2010. Disponível em http://www.iea.org/co2highlights/co2highlights.pdf. Acedido em Dezembro 2010.

[2] *Honda Portugal, S.A.* Disponível em www.honda.pt. Acedido em Dezembro 2010.

[3] N. Navet and F. Simonot-Lion, *Automotive Embedded Systems Handbook*. CRC Press, 2009.

[4] *ABS - Anti-lock Braking System*. Disponível em http://www.three2tango.com/car-and-bike-corner/abs-anti-lock-braking-system.html/. Acedido em Dezembro 2010.

[5] *Toyota Safety,Electronic Brake-Force Distribution (EBD).Disponível em* http://www.toyota.com/safety/star-safety-system/electronic-brake-force-distribution.html. Acedido em Dezembro 2010.

[6] F. Roos, "Design and theoretical evaluation of electric power steering in heavy vehicles," Royal Institute of Technology, Stockholm2005.

[7] *DENSO,Adaptive Cruise Control System*. Disponível em www.globaldensoproducts.com/dcs/accs/. Acedido em Dezembro 2010.

[8] M. Ehsani, Y. Gao, and A. Emadi, *Modern Electric, Hybrid Electric, and Fuel Cell Vehicles: Fundamentals, Theory, and Design*, Second edition ed.: CRC Press, 2010.

[9] R. Krishnan, *Electric Motor Drives: Modeling, Analysis, and Control*: Prentice Hall, 2001.

[10] Christian Wegwerth, et al., *Active Safety Light*, V.I.S.I.O.N proceedings ref 2008-02, France: Versailles Satory, 2008.

[11] Z.Hocenski, T.Keser, and K.Nenadic, "Distributed intelligent control of car lighting system with fault detection," in *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, 2007.

[12] *Tire-Pressure-Monitors,Wireless Digital Tire Pressure Monitor Systems*. Disponível em http://www.tire-pressure-monitors.com/. Acedido em Dezembro 2010.

[13] *Controller Area Network (CAN) Overview*. Disponível em http://zone.ni.com/devzone/cda/tut/p/id/2732. Acedido em Dezembro 2010.

[14] T.Nolte, H.Hansson, and L.L.Bello, "Automotive communications-past, current and future," in *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*, 2005.

[15] *SAE International, The Society of Automotive Engineers*. Disponível em http://www.sae.org/. Acedido em Dezembro 2010.

[16] N. Navet, "Controller area network [automotive applications]," *Potentials, IEEE,* vol. 17, 1998.

[17] T. Herpel, "Performance Evaluation of Time-Critical Data Transmission in Automotive Communication Systems," Universidade Erlangen-Nürnberg, 2009.

[18] V. A. Souza, "Introdução ao CAN," Cerne Tecnologia. Disponível em http://www.artigos.com/components/com_mtree/attachment.php?link_id=6695&cf_id=24. Acedido em Dezembro 2010.

[19] J. Wang, X. Wang, X. Zhai, and H. Wang, "CAN/LIN Hybrid Network for Automobile," Dept. of Automation, Shanghai Jiaotong University, , Shanghai, China.

[20] S. REY, "Introduction to LIN" 2003. Disponível em http://rs-rey.pagesperso-orange.fr/electronic_ressources/Ressources/Networks/LIN/LIN_bus.pdf. Acedido em Dezembro 2010.

[21] *Flexray - The communication system for advanced automotive control applications*. Disponível em http://www.flexray.com/. Acedido em Dezembro 2010.

[22] N. Sousa and N. Fonseca, "FlexRay – Futuro na Comunicação Automóvel," Departamento de Engenharia Electrotécnica, ISEP,, Porto.

[23] F. C. Carvalho, "Uma extensão do protocolo CAN para aplicações críticas em sistemas distribuídos," Mestrado, Instituto de Informática, Universidade Federal do Rio Grande do Sul, 2006.

[24] *IXXAT – a reliable and strong partner for advanced data communication*. Disponível em http://www.ixxat.com/introduction_flexray_en.html. Acedido em Dezembro 2010

[25] J. Y. Hande, M. Khanapurkar, and P. Bajaj, "Approach for VHDL and FPGA Implementation of Communication Controller of Flex-Ray Controller," *Second International Conference on Emerging Trends in Engineering and Technology*. Disponível em http://portal.acm.org/citation.cfm?id=1726549. Acedido em Dezembro 2010.

[26] *Homepage of the development partnership Automotive Open System Architecture (AUTOSAR)*. Disponível em http://www.autosar.org/. Acedido em Janeiro 2011.

[27] T. Noergaard, *Embedded Systems Architecture:A Comprehensive Guide for Engineers and Programmers* em Embedded operating systems. Newnes,Elsevier, 2005.

[28] *freeRTOS*. Disponível em www.freertos.org/. Acedido em Janeiro 2011.

[29] *eCos*. Disponível em http://ecos.sourceware.org/. Acedido em Janeiro 2011.

[30] A. J. Massa, *Embedded Software Development with ecos*: PRENTICE HALL, 2003.

[31] *MicroC/OS- II*. Disponível em http://micrium.com/page/products/rtos/os-ii. Acedido em Janeiro 2011.

[32] *Micrium*. Disponível em http://micrium.com/page/home. Acedido em Janeiro 2011.

[33] L. Shaofen, Z. Shengbin, and J. Xiaoxia, "The Porting of Real-Time Operating System uC/OS-II on MCS-51 Series of MCU," presented at the 2009 International Conference on Measuring Technology and Mechatronics Automation, 2009.

[34] *RTAI - the RealTime Application Interface for Linux*. Disponível em http://www.rtai.org/. Acedido em Janeiro 2011.

[35] *Linux Online*. Disponível em http://www.linux.org/. Acedido em 2011.

[36] *The Linux Kernel Archives*. Disponível em http://www.kernel.org/. Acedido em Janeiro 2011.

[37] P. Sarolahti, "Real-Time Application Interface," 26th February 2001. Disponível em http://www.cse.iitb.ac.in/~cs684/RTATutors/rtai.pdf. Acedido em Janeiro 2011.

[38] *TinyOS*. Disponível em http://www.tinyos.net. Acedido em Janeiro 2011.

[39] *nesC: A Programming Language for Deeply Networked Systems*. Disponível em http://nescc.sourceforge.net/. Acedido em Janeiro 2011.

[40] D. Gay, P.Levis, and R. Behren.The nesC Language: A Holistic Approach to Networked Embedded Systems.Disponível em www.cs.binghamton.edu/~kang/teaching/cs580s/nesc.ppt. Acedido em Janeiro 2011.

[41] *The Contiki Operating System*. Disponível em http://www.sics.se/contiki/. Acedido em Janeiro 2011.

[42] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors.

[43] P. A. Bernstein. Middleware: a model for distributed system services. *Communications of the ACM*. Fevereiro 1996. Disponível em http://portal.acm.org/citation.cfm?id=230809. Acedido em Janeiro 2011.

[44] L. JingYong, Z. LiChen, Z. Yong, and C. Yong, "Middleware-based Distributed Systems Software Process," *International Journal of Advanced Science and Technology*, vol. 13, December 2009.

[45] A. Davy, J. Finnegan, R. Carroll, and S. Cummins, "State of the Art: Middleware in Smart Space Management," Cork Institute of Technology Waterford Institute of TechnologyMay 2003.

[46] *Software Engineering Institute Home page*. Disponível em http://www.sei.cmu.edu/. Acedido em Janeiro 2011.

[47] *Object Management Group: The Common Object Request Broker: Architecture and Specification*. Disponível em www.omg.org. Acedido em Janeiro 2011.

[48] *Overview of the ACE+TAO Project*. Disponível em http://www.cs.wustl.edu/~schmidt/TAO-intro.html. Acedido em Janeiro 2011.

[49] *Real-time CCM with CIAO(Component Integrated ACE ORB)*. Disponível em http://www.cs.wustl.edu/~schmidt/CIAO.html. Acedido em Janeiro 2011.

[50] *Microsoft: DCOM Architecture*. Disponível em http://microsoft.com/com/wpaper/#DCOMpapers. Acedido em Janeiro 2011.

[51] *Remote Method Invocation(RMI)*. Disponível em http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html. Acedido em Janeiro 2011.

[52] C. M. Blair, G. Coulson, and G. P. N., "An efficient component model for the construction of adaptive middleware.," presented at the IFIP / ACM International Conference on Distributed Systems Platforms. Disponível em http://portal.acm.org/citation.cfm?id=646591.697779. Acedido em Janeiro 2011.

[53] *MiLAN Project*. Disponível em http://www.futurehealth.rochester.edu/milan. Acedido em Janeiro 2011.

[54] E. Souto, G. Guimaraes, G. Vasconcelos, M. Vieira, N. Rosa, and C. Ferraz, "A message-oriented middleware for sensor networks," presented at the Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-hoc Computing, 2004. Disponível em http://portal.acm.org/citation.cfm?id=1028514. Acedido em Janeiro 2011.

[55] T. Liu and M. Martonosi, "Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems," presented at the Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming, 2003.

[56] *Cougar Project*. Disponível em http://www.cs.cornell.edu/bigreddata/cougar/index.php. Acedido em Janeiro 2011.

[57] H. Utz, S. Sablatng, S. Enderle, and G. Kraetzschmar, "Miro – Middleware for Mobile Robot Applications,," presented at the IEEE Transactions on Robotics and Automation. Disponível em http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01044362. Acedido em Janeiro 2011.

[58] K. Raman, Y. Zhang, M. Panahi, J. A. Colmenares, R. Klefstad, and T. Harmon, "RTZen: Highly Predictable, Real-Time Java Middleware for Distributed and Embedded Systems," Department of Electrical Engineering and Computer Science,University of California, Irvine.

[59] *OpenFusion e*ORB*. Disponível em http://www.prismtechnologies.com/section-item.asp?snum=5&sid=6. Acedido em Janeiro 2011.

[60] *ROFES: Real-Time CORBA for embedded systems*. Disponível em http://www.lfbs.rwthaachen.de/content/250?PHPSESSID=98d3e2b8a1316d3cb0539f7640d409b4. Acedido em Janeiro 2011.

[61] I. Mizunuma, S. Horiike, and I. Hiroshima, "MidART. Middleware for Real-Time Distributed Systems for Industrial Applications.," *IEICE TRANSACTIONS on Electronics*, vol. E84-D, 2001/04/01. Disponível em http://sciencelinks.jp/j-east/article/200117/000020011701A0411546.php. Acedido em Janeiro 2011.

[62] U. Brinkschulte, A. Bechina, F. Picioroaga, and E. Schneider, "Distributed Real-Time Computing for Microcontrollers - the OSA+ Approach," IPR, Institute for Process Control,University of Karlsruhe,Germany.

[63] WEBER, Taisy Silva. Um roteiro para exploracao dos conceitos basicos de tolerancia a falhas. 2002. Disponível em www.inf.ufrgs.br/~taisy/disciplinas/textos/Dependabilidade.pdf. Acedido em Janeiro 2011.