UAlg ISE
UNIVERSIDADE DO ALGARVE
INSTITUTO SUPERIOR DE ENGENHARIA

UNIVERSIDADE DO ALGARVE

INTERFACE EM SMARTPHONE PARA SUPERVISÃO DE UM SISTEMA DE CONTROLO DE UMA INSTALAÇÃO SOLAR TÉRMICA

Rui Guerreiro Sales Brito Palma

Projeto para obtenção do grau de Mestre em Engenharia Elétrica e Eletrónica
Área de Especialização: Sistemas de Energia e Controlo

Trabalho efetuado sob orientação de:
Professor Roberto Lam e Professor Raul Lana

Julho, 2014

UNIVERSIDADE DO ALGARVE
INSTITUTO SUPERIOR DE ENGENHARIA

UNIVERSIDADE DO ALGARVE

INTERFACE EM SMARTPHONE PARA SUPERVISÃO DE UM SISTEMA DE CONTROLO DE UMA INSTALAÇÃO SOLAR TÉRMICA

Rui Guerreiro Sales Brito Palma

Projeto para obtenção do grau de Mestre em Engenharia Elétrica e Eletrónica
Área de Especialização: Sistemas de Energia e Controlo

Trabalho efetuado sob orientação de:
Professor Roberto Lam e Professor Raul Lana

Julho, 2014

INTERFACE EM SMARTPHONE PARA SUPERVISÃO DE UM SISTEMA DE CONTROLO DE UMA INSTALAÇÃO SOLAR TÉRMICA

Declaração de autoria de trabalho

Declaro ser o(a) autor(a) deste trabalho, que é original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências incluída.

(Rui Guerreiro Sales Brito Palma)

Agradecimentos

À minha esposa Vera e à minha filha Carolina pela paciência e pelo tempo que me dispensaram para todas as fases do desenvolvimento do presente projeto.
Aos meus pais, Hélder e Isabel, pela minha formação, o melhor presente que os pais podem dar aos filhos.
Ao professor Roberto Lam e ao professor Raul Lana pelo apoio, orientação e confiança que depositaram em mim.

Resumo

A grande penetração dos dispositivos móveis, normalmente designados por *smarphones* e *tablets*, em todos os setores da sociedade, a sua capacidade para ligações de dados e um cada vez maior poder de processamento fazem deles plataformas com grande potencial para aplicações de aquisição e supervisão de dados de sistemas de automação e controlo.

Este relatório descreve o projeto e desenvolvimento de uma solução para monitorização do controlador de sistema solar térmico DeltaSol M da marca RESOL. A solução desenvolvida opera numa arquitetura servidor/cliente em que o servidor é composto por uma aplicação para computador com arquitetura x86 e sistema operativo Windows e o cliente por uma aplicação para dispositivo móvel com sistema operativo Android.

Nesta solução o servidor atua como interface, sendo responsável pela comunicação com o controlador solar através do protocolo VBus e pela posterior transmissão dos dados à aplicação cliente instalada no dispositivo móvel. Nela são disponibilizados os dados do controlador DeltaSol M adquiridos em tempo real, bem como o acesso de leitura e escrita aos seus parâmetros de configuração.

São descritos os vários estágios para concretização da solução, nomeadamente o desenvolvimento: do protocolo de comunicação com o controlador; da transmissão de dados em tempo real da aplicação servidor para a aplicação cliente; do método de apresentação dos dados na aplicação cliente; do sistema de menus que permite alterar e consultar remotamente parâmetros do controlador e da apresentação de dados de forma gráfica através de um sinóptico do sistema.

PALAVRAS-CHAVE: Aquisição e supervisão de dados; Android HMI; Resol DeltaSol M; Protocolo VBus; Controlador de sistema solar térmico; Telegestão.

Abstract

The high penetration of mobile devices commonly referred to as smartphones and tablets, in all society sectors, its capacities for data connections and increasing processing power, makes them platforms with great potential for supervisory control and data acquisition applications.
This report describes the project and development of a data acquisition solution for the solar thermal system controller DeltaSol M of RESOL brand. This solution operates in a server / client architecture where the server is an application for an x86 architecture computer with Windows operating system and the client is an application for a mobile device with Android operating system.
In this solution, the server acts as a gateway, requesting data to the controller through VBus protocol and transmitting data to the client application installed on the remote mobile device. All the data from the controller DeltaSol M is acquired in real time as well as the read and write access to its configuration parameters.
The several stages for the solution implementation are described, including the development of: the communication protocol with the controller; the real-time data transmitting by the server to the client application; the data presentation method in the client application; the menu system that allows the remote read and write of the configuration parameters and the graphic display of data through a system synoptic.

KEYWORDS: Supervisory control and data aquisition; Android HMI; Resol DeltaSol M; VBus protocol; Solar thermal system controler; Supervision.

Índice

Índice de figuras

Índice de tabelas

1. Introdução

1.1. Enquadramento

Este relatório descreve o projeto e desenvolvimento de uma solução de supervisão de controlador de sistema solar térmico através de um dispositivo móvel Android e foi desenvolvido no ano letivo de 2013/14, no âmbito da cadeira de "Projeto" do Mestrado em Engenharia Elétrica e Eletrónica, área de especialização de Sistemas de Energia e Controlo, do Instituto Superior de Engenharia da Universidade do Algarve.

1.2. Objetivo do projeto

Este projeto tem como objetivo o desenvolvimento de uma aplicação para dispositivo móvel Android que permita visualizar remotamente os dados do controlador de sistemas solares térmicos, modelo DeltaSol M da marca RESOL. Esta aplicação deve permitir:

- Visualizar os dados mais relevantes do controlador em tempo real, por exemplo: temperaturas e estado dos equipamentos;
- Alterar parâmetros de configuração do controlador, por exemplo: objetivos de temperatura e temporizações;
- Visualizar os principais dados do sistema solar térmico na forma gráfica através de um sinóptico, por exemplo: temperaturas e estado dos equipamentos;

1.3. Motivações

É inevitável constatar a grande proliferação de dispositivos móveis equipados com sistema operativo Android em todos os estratos da sociedade, como se pode observar no gráfico da Figura 1.1, no ano de 2013 foram vendidos 800 milhões destes dispositivos, o que corresponde a 80% das vendas globais [1]. Estes números mostram o crescimento da base de potenciais utilizadores de aplicações móveis. Este mercado global e em crescimento, é por isso, um espaço para novas oportunidades de negócio, ideias e produtos.

Pelas razões expostas, o desenvolvimento deste projeto teve como principais motivações a aquisição de competências na programação de aplicações Android e também a oportunidade de explorar esta plataforma no suporte a aplicações de supervisão de sistemas de controlo.

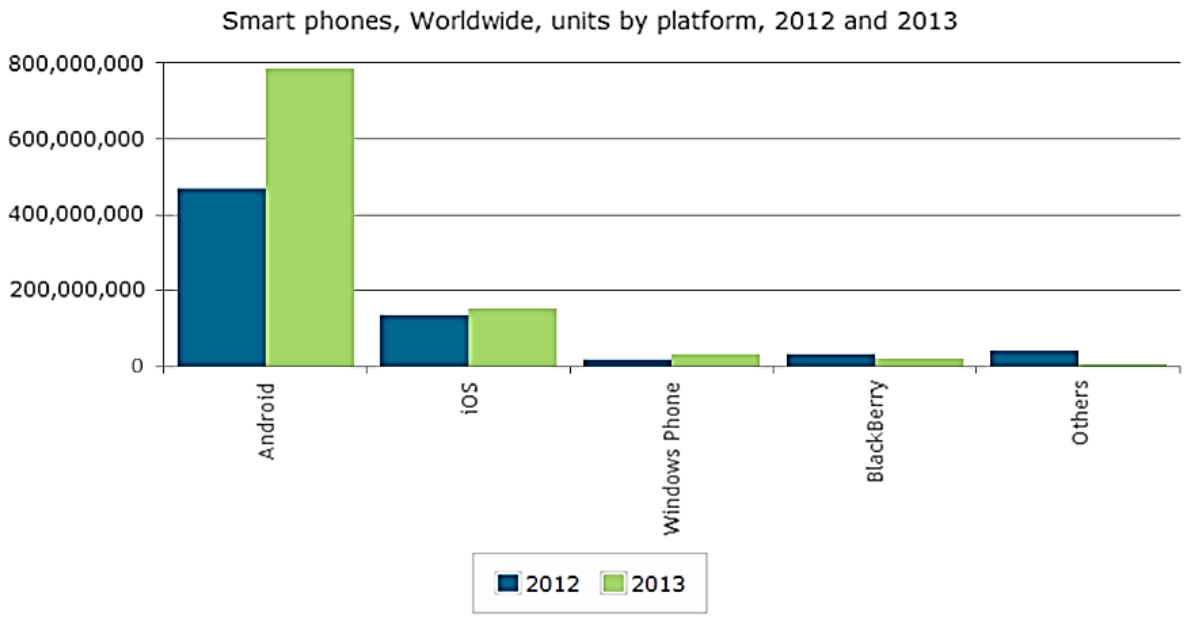

Figura 1.1 - Distribuição dos sistemas operativos nos *smartphones* vendidos em 2012/13 [1]

1.4. Estrutura do relatório

No presente capítulo é feito o enquadramento do projeto, definidos os seus objetivos e as motivações para a sua execução. É também descrita a plataforma Android e são analisadas duas aplicações de supervisão para dispositivo móvel Android desenvolvidas por marcas de grande reputação no mercado.

No capítulo 2, é dimensionada a arquitetura do projeto, identificados os seus componentes, descrita a estratégia de comunicação entre a aplicação servidor e a aplicação cliente e definidos os estágio necessários para o desenvolvimento do sistema.

Pela sua importância, no capítulo 3 são analisadas em grande detalhe as características do protocolo VBus, este capítulo serve de suporte a outras partes do relatório que remetem para os dados analisados.

Nos capítulos 4 e 5, são descritas respetivamente a aplicação servidor e cliente, as suas funcionalidades e os pontos mais relevantes do seu desenvolvimento.

No capítulo 6 são analisados e discutidos os resultados do projeto bem como as propostas para melhoria do mesmo.

1.5. A plataforma Android

A plataforma Android é um conjunto de *software* para dispositivos móveis que disponibiliza todas as ferramentas necessárias ao funcionamento de telefones móveis ou outros dispositivos móveis como *tablets*.

O Android está disponível sob a licença Apache V2 [2], ou seja, é um *software open-source*. A plataforma tem na sua base a *kernel* Linux 2.6 e foi desenvolvida para permitir que os programadores possam tirar partido de todas as capacidades dos dispositivos móveis.

1.5.1. Breve descrição histórica

Em 2005 a Goggle adquiriu a *Android, Inc*, uma empresa fundada em 2003 por Andy Rubin, Rich Miner, Nick Sears, e Chris White. Em Novembro de 2007 foi revelado pela primeira vez o *Android* marcando o início da penetração da Goggle no setor móvel, aproximadamente um ano depois, em Outubro de 2008 foi lançado o primeiro *smartphone Android*, o HTC *Dream*.

A primeira versão do *Android* tinha funcionalidades como *pull-down* da barra de notificações, *homescreens* múltiplos, *widgets* e integração com vários produtos Goggle como *Gmail*, *Contacts* e *Calendar*, foi também introduzido o *Android market* actualmente *Goggle Play* que permite aos utilizadores a instalação de aplicações e jogos diretamente para o seu dispositivo.

1.5.2. Arquitetura Android

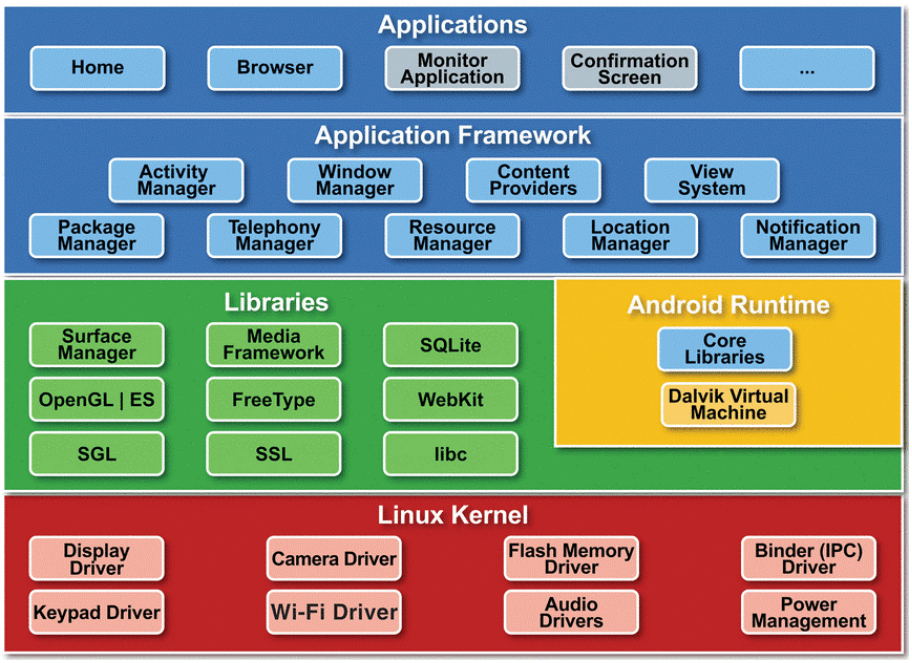

Figura 1.2 - Arquitetura da plataforma Android

A plataforma Android para além do sistema operativo inclui um conjunto de importantes aplicações como por exemplo a calculadora, alarme, lista de contactos, cliente de *e-mail*, marcador telefónico entre outros.
A plataforma permite também a utilização de aplicações criadas por terceiros, oferecendo para isso um conjunto de bibliotecas e ferramentas que permitem o acesso às várias camadas e funções presentes no dispositivo móvel. Como se pode ver na Figura 1.2, a arquitetura Android pode ser dividida em cinco partes:

- **Linux Kernel**

  Na base de todas as camadas da arquitetura Android está a *Kernel* Linux 2.6. Esta camada permite o acesso a funcionalidades de baixo nível como gestão de processos, gestão de memória, gestão de dispositivos de *hardware* como a câmara fotográfica, teclas físicas, ecrã, GPS, etc...A *kernel* é também responsável pela gestão de recursos como a comunicação em rede e suporte para periféricos.

- **Bibliotecas**

  Acima da camada Linux *Kernel* está um conjunto de bibliotecas que disponibilizam funções como a reprodução e gravação de ficheiros multimédia, acesso a bases de dados SQLite, segurança SSL, etc...

- **Android *runtime***

  Esta camada contém o componente central na arquitetura Android, a máquina virtual *Dalvik*, uma máquina virtual de Java especialmente desenhada e otimizada para o Android. Esta máquina permite que cada aplicação Android corra um processo independente, que é uma instância da própria máquina virtual. Nesta camada existem também um conjunto de bibliotecas base que permitem aos programadores escrever aplicações usando linguagem Java.

- ***Application framework***

  A camada *Application framework* é um conjunto de classes Java de alto nível que podem ser usadas pelos programadores nas aplicações *Android*.

- **Aplicações**

  Camada onde são executadas as aplicações que beneficiam da abstração providenciada por todas as outras camadas.

1.5.3. *Activities*

As *activities* são centrais nas aplicações *Android*, uma *activity* representa uma vista única com a qual o utilizador pode interagir. Esta vista, ou interface gráfico é composta por uma hierarquia de *views*, ou seja, objetos derivados da classe *View*, cada *view* controla uma área retangular do ecrã e responde às interações do utilizador.

Uma aplicação pode ter várias *activities* e o utilizador pode navegar entre estas de forma semelhante ao que acontece com as páginas de um *website*, durante esta interação a *activity* transita entre diferentes estados do seu ciclo de vida, nestas transições o sistema invoca de forma automática determinados métodos consoante o estado para o qual a *activity* transita. Um esquema representativo deste ciclo de vida pode ser consultado na Figura 1.3 onde estão representados os vários estados e os respetivos métodos de *callback* que lhes estão associados.

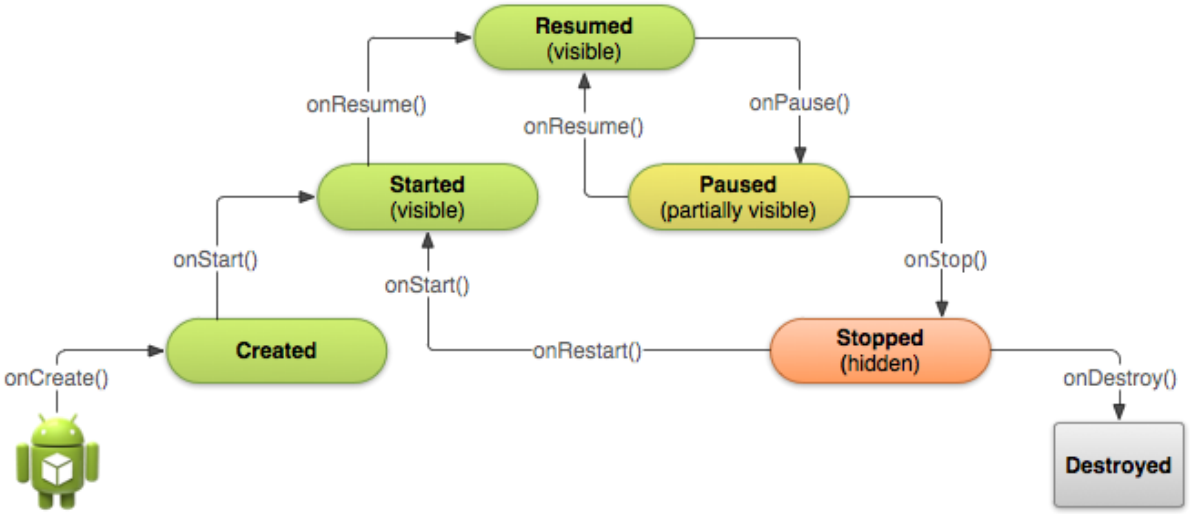

Figura 1.3 - Ciclo de vida das *activities*

Os vários estados representados na Figura 1.3 podem ser caracterizados pela visibilidade ou não visibilidade da *activity* e também pela possibilidade desta executar código ou não, como é descrito de seguida:

- ***Created*** - Primeiro estado que uma *activity* assume quando é criada e não existe previamente em memória. É invocado o *callback* ***onCreate()*** sempre que há uma transição para este estado.
- ***Started*** - Estado intermédio da *activity* que está imediatamente antes do estado *Resumed* e é assumido quando a aplicação transitou do estado *Created* ou *Stopped*. É invocado o *callback* ***onStart()*** sempre que há uma transição para este estado.

- ***Resumed*** - Neste estado a *activity* está visivel, em primeiro plano, e é possível a interação do utilizador. É invocado o *callback* ***onResume()*** sempre que há uma transição para este estado.
- ***Paused*** - Neste estado, a *activity* está em segundo plano devido à sobreposição de outra *activity* que pode ser transparente ou não ocupar todo o ecrã, não é possível a execução de código nem a interação do utilizador. É invocado o *callback* ***onPaus****e()* sempre que há uma transição para este estado.
- ***Stopped*** - Neste estado a *activity* está em segundo plano e não está visível, toda a instância da *activity* é preservada mas não pode ser executado qualquer código. É invocado o *callback* ***onStop()*** sempre que há uma transição para este estado.
- ***Destroyed*** – O callback ***onDestroy()*** é invocado quando a activity é removida da memória pelo sistema, que o pode fazer para libertar recursos.

1.5.4. *Events handlers*

Ao interagir com o interface gráfico o utilizador origina uma série de eventos. Como por exemplo, os toques nas *views* da *activity*, a estes eventos está normalmente associada uma ação, assim é necessário capturar estes eventos para que seja possível à *activity* correr o código associado à ação. A este processo dá-se o nome de *Event Handling* e nele estão envolvidos três conceitos fundamentais:

- ***Event Listeners*** - Interface de uma classe que recebe a notificação quando um evento acontece.
- **Registo dos *Event Listeners*** - Processo pelo qual um *Event Listener* é registado e associado ao respetivo *Event Handler* que é invocado quando o *Event Listener* reage a um evento.
- ***Event Handlers*** - Método associado a um evento através do registo do respetivo *Event Listener*.

A classe *View* é o bloco básico de construção dos componentes gráficos de interação com o utilizador, esta classe implementa os *Event Listeners* e respectivos *Handlers* da Tabela 1.1.

Event Handler	*Event Listener*
onClick()	*OnClickListener()*
onLongClick()	*OnLongClickListener()*
onFocusChange()	*OnFocusChangeListener()*
onKey()	*OnFocusChangeListener()*
onTouch()	*OnTouchListener()*
onMenuItemClick()	*OnMenuItemClickListener()*

Tabela 1.1 - *Event listeners* e *handlers* da classe *View*

1.5.5. *Custom adapters*

Nas aplicações Android uma das formas utilizada para apresentação de dados ao utilizador é a *ListView*. O preenchimento de dados na *ListView* é feito através de um *Adapter*, é exemplo disso a classe *ArrayAdapter* que permite carregar os dados na *ListView* a partir de um *Array*, esta abordagem é satisfatória para aplicações com *ListView* simples, constituídas por linhas com uma *TextView* apenas, no entanto sempre que é necessária uma maior personalização da *ListView*, com imagens, mais informações de texto ou mesmo botões, é necessário definir de que forma os dados serão transferidos da fonte para a *ListView*, para isso cria-se uma classe que estende a classe *ArrayAdapter* e que define no método *getView()* de que forma os dados serão preenchidos nas várias *views* da linha da *ListView*.

Figura 1.4 - Linhas de uma *ListView*

No exemplo da Figura 1.4 estão representadas duas linhas de uma *ListView*, cada uma destas linhas é preenchida com 4 *TextView* em diferentes posições, conforme a informação que representam. As características de uma linha, como o número e o tipo de elementos que a integram, as suas posições, tipos de letra e outras propriedades gráficas são definidos num ficheiro XML com o *layout* de uma única linha, este *layout* será processado pelo *Adapter* através de um *LayoutInflater*.

Figura 1.5 - *ListView* com *Custom Adapter*

1.5.6. *AsyncTasks*

As aplicações Android são executadas por defeito na *thread* principal do UI (*User Interface*). Isto significa que qualquer processo com um tempo de execução elevado pode dar origem a um diálogo ANR (*Application Not Responding*) porque a aplicação não pode lidar com os eventos do UI. Por esse motivo, todos os métodos executados na *thread* principal devem ter execuções rápidas, operações potencialmente demoradas como solicitações a dispositivos remotos pela rede ou cálculos computacionalmente exigentes devem ser feitos numa *thread* independente [3].

A forma mais eficiente de criar uma *thread* para processamento independente é através da classe *AsyncTask* que tem três métodos principais [4]:

- *onPreExecute()* - Executado na *thread* principal antes da tarefa iniciar, deve ser usado, por exemplo, para informar o utilizador do inicio da operação.

- *doInBackground()* - Método efetivamente executado numa *thread* independente, deve ser usado para executar todo o código da operação.

- *onPostExecute()* - Executado na *thread* principal após a conclusão da operação, deve ser usado, por exemplo, para informar o utilizador do fim da operação.

1.5.7. Mensagem do tipo *Toast*

Uma mensagem do tipo *toast* como a representada na Figura 1.6, permite que sejam feitas simples notificações ao utilizador, estas mensagens desaparecem automaticamente após um tempo pré-definido. Uma mensagem *toast* é uma instância da classe Toast que permite a definição do texto e outras propriedades da mensagem.

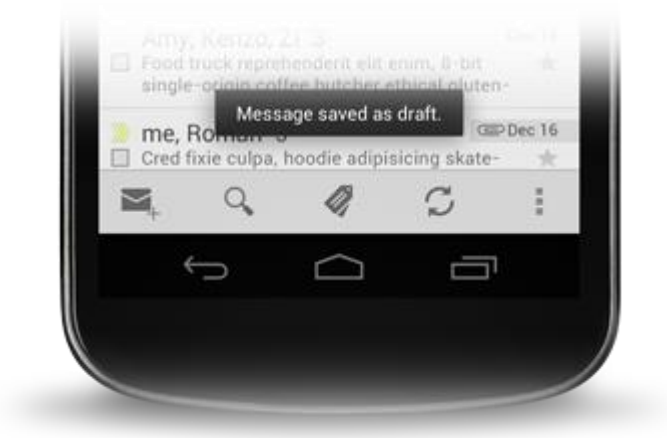

Figura 1.6 - Mensagem do tipo *Toast*

1.6. Aplicações de supervisão em Android

Apesar dos dispositivos móveis serem usados sobretudo como dispositivos de comunicação e entretenimento pessoal, a sua portabilidade e conetividade à internet permitem que os seus utilizadores possam aceder a dados de medição em qualquer lugar. A National Intruments publicou recentemente uma *newsletter* [5] sobre a temática das aplicações HMI (*Human machine interface*) em dispositivos móveis em que destaca as vantagens destas aplicações para os responsáveis de sistemas, que essencialmente podem ter nos seus bolsos um interface para consulta remota de dados. Nesta *newsletter* a empresa promove a sua aplicação de supervisão "LabVIEW app" representada na Figura 1.7. Esta aplicação não solicita os dados diretamente aos equipamentos de aquisição, solicita-os através do protocolo TCP/IP ao *software* "LabVIEW" que atua como interface entre estes e a aplicação. Todo o desenvolvimento das aplicações para a "LabVIEW app" é feito com recurso a uma aplicação de *software* disponibilizada pela empresa. Assim, para a personalização dos ecrãs com os valores e objetos a apresentar não é necessário o Android SDK.

Figura 1.7 - "LabVIEW app" executado no Android

A empresa Schneider Eletric, tem uma abordagem diferente a este mercado. Esta empresa comercializa uma linha de terminais gráficos HMI denominada "Magelis" que têm ferramentas de desenvolvimento próprio. De forma a aproveitar todo o ecossistema de produtos já comercializado e os conhecimentos dos técnicos que os instalam, a empresa optou por desenvolver a aplicação de supervisão "Vijeo design'Air" [6], que acede remotamente aos terminais gráficos "Magelis" e disponibiliza o seu interface no ecrã do dispositivo móvel. Desta forma não é necessário qualquer tipo de desenvolvimento específico para a aplicação móvel já que esta reproduz simplesmente a aplicação já desenvolvida no terminal. A Figura 1.8 mostra o esquema da arquitetura servidor/cliente que esta aplicação utiliza, sendo a aplicação "Vijeo design'Air" o cliente e o terminal gráfico o servidor.

Figura 1.8 - Arquitetura usada pela aplicação "Vijeo design'Air" [6]

Um estudo recente [7] que aponta vários exemplos de aplicações de supervisão aplicadas à indústria refere as vantagens da utilização de dispositivos móveis para a visualização remota dos dados, devido às suas vantagens relativamente a dispositivos dedicados ao mercado da automação industrial, como é o caso dos terminais gráficos "Magelis", que por norma são fabricados em menor número e por isso têm custos mais elevados. Esse mesmo estudo refere que, devido ao posicionamento de vanguarda dos fabricantes de dispositivos móveis na investigação e desenvolvimento de componentes de *hardware*, estes dispositivos representam a melhor escolha em termos de consumo de energia, *performance* e baixo custo.

Estes exemplos, e a movimentação de grandes empresas como a National Instrument ou a Schneider Eletric neste setor, mostram que apesar de relativamente recentes, os dispositivos móveis, em particular os equipados com o sistema operativo Android (instalado em 80% das unidades fabricadas mundialmente em 2013 [1]), são hoje uma escolha viável como plataformas de suporte às aplicações HMI.

2. Dimensionamento do sistema

2.1. Abordagem inicial

O dispositivo móvel escolhido para servir de suporte à aplicação de supervisão foi um Samsung Galaxy S4 com sistema operativo Android 4.4.2 e arquitetura ARM, esta escolha deveu-se à disponibilidade deste dispositivo durante o desenvolvimento do projeto, no entanto qualquer smartphone com sistema operativo igual ou superior ao Android 2.3 poderia ter sido usado.

O equipamento a supervisionar pela aplicação Android é o controlador solar modelo DeltaSol M da marca RESOL instalado numa das bancadas de ensaios do laboratório de Engenharia Mecânica do Instituto Superior de Engenharia da Universidade do Algarve. Esta bancada de ensaios pode ser observada na Figura 2.1 onde estão identificados os equipamentos relevantes para o projeto:

a) Controlador solar RESOL DeltaSol M
b) Adaptador de barramento VBus/USB marca RESOL
c) Computador tipo torre, com processador Pentium 4 a 2,4 GHz, 512 Mb de memória RAM, disco rígido de 100Gb e sistema operativo Windows XP
d) Simulador de sondas de temperatura

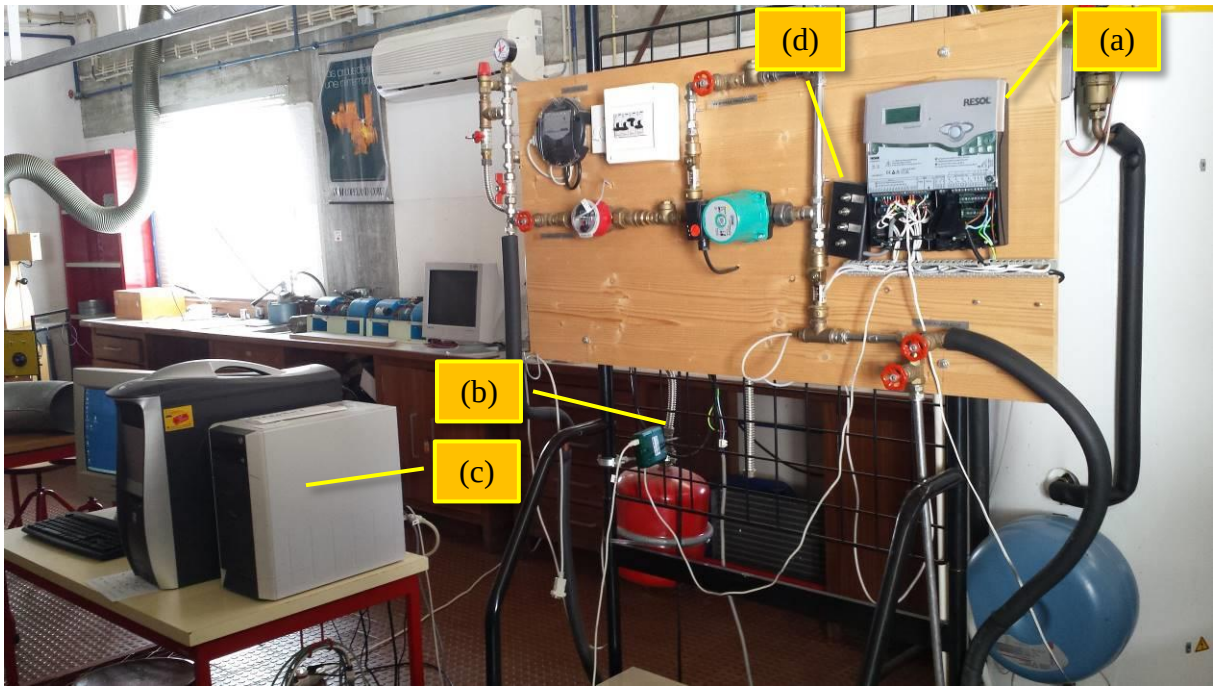

Figura 2.1 - Fotografia da bancada de ensaios

A análise às características dos equipamentos identificados na Figura 2.1 e à documentação dos seus fabricantes permitiu verificar que:

- O controlador solar RESOL DeltaSol M (a), está equipado com um *display* e 3 botões físicos, 9 saídas a relé, 12 entradas para ligação de sensores e suporta uma variedade de funções e opções que lhe permitem adaptar-se a diferentes tipos de sistemas solares térmicos [8]. A configuração do controlador é efetuada através de um menu multi-língua. Nos seus terminais, (ver Figura 2.3) está disponível a ligação ao canal de comunicação VBus desenvolvido pelo fabricante RESOL para comunicação com dispositivos externos. Este barramento usa um protocolo de comunicação aberto, ou seja, as suas especificações [9] são de acesso público.
- O computador (c) tem instalada a aplicação "RESOL *Service Center*" que permite consultar e configurar o controlador DeltaSol M (a), esta aplicação é disponibilizada gratuitamente pelo fabricante RESOL na sua página da internet [10].
- O adaptador de barramento VBus/USB (b) da marca RESOL está equipado com uma porta mini USB ligada ao computador (c), e dois bornes para ligação ao barramento VBus ligados ao controlador DeltaSol M (a). Para este adaptador apenas estão disponíveis drivers para sistema operativo Windows [10], não havendo por isso alternativa à utilização deste sistema operativo no computador (c).
- O computador (c) está ligado à rede *ethernet* interna da UALG e tem acesso à internet, sendo portanto adequado para disponibilizar os dados do controlador a sistemas remotos.
- O simulador de sondas de temperatura (b), está equipado com 4 potenciómetros (ver Figura 2.2) ligados aos terminais "S1", "S2", "S6" e "S9" do controlador (a) de forma a variar as temperaturas associadas a estas entradas.

Figura 2.2 - Pormenor do simulador de sondas

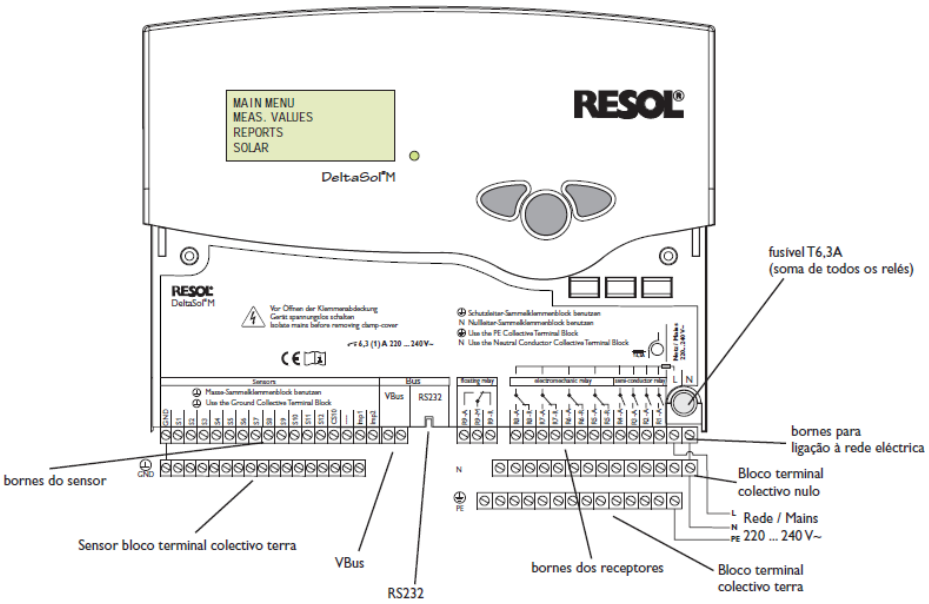

Figura 2.3 - Representação do RESOL DeltaSol M [11]

2.2. Definição da arquitetura

Tipicamente os canais de comunicação disponíveis para acesso à internet nos dispositivos móveis do tipo *tablet* ou *smartphone* são a rede sem fios IEEE 802.11 (*Wi-Fi*) ou as ligações de dados disponibilizadas pelas redes móveis, 2G, 3G, 3,5G ou 4G. Estes canais usam o protocolo TCP/IP para a comunicação com outros dispositivos através da internet, no entanto, dado que este protocolo não é suportado pelo controlador DeltaSol M, é necessária uma aplicação de *software* que converta o protocolo TCP/IP usado pelo dispositivo móvel no protocolo VBus usado pelo controlador. Essa aplicação poderá ser instalada no computador ligado ao controlador pelo adaptador de barramento VBus/USB (ver seção 2.1).

Assim, foram identificados 4 componentes necessários ao projeto, este conjunto de componentes será denominado ao longo deste relatório por SCIS, um acrónimo para Sistema de Controlo de Instalação Solar, constituído por:

1) Aplicação SCIS cliente, instalada num dispositivo móvel com arquitetura ARM e sistema operativo Android 2.3 ou superior. Esta aplicação permite a consulta e configuração do controlador DeltaSol M através do SCIS servidor.
2) Aplicação SCIS servidor, instalada num computador com arquitetura x86 e sistema operativo Windows XP ou superior. Esta aplicação implementa o protocolo VBus para comunicação de leitura e escrita com o controlador DeltaSol M e disponibiliza os dados à aplicação SCIS cliente.
3) Adaptador de barramento VBus para barramento USB.

4) Controlador RESOL DeltaSol M

Na Figura 2.4 está representada de forma esquemática a arquitetura do SCIS com todos os seus componentes.

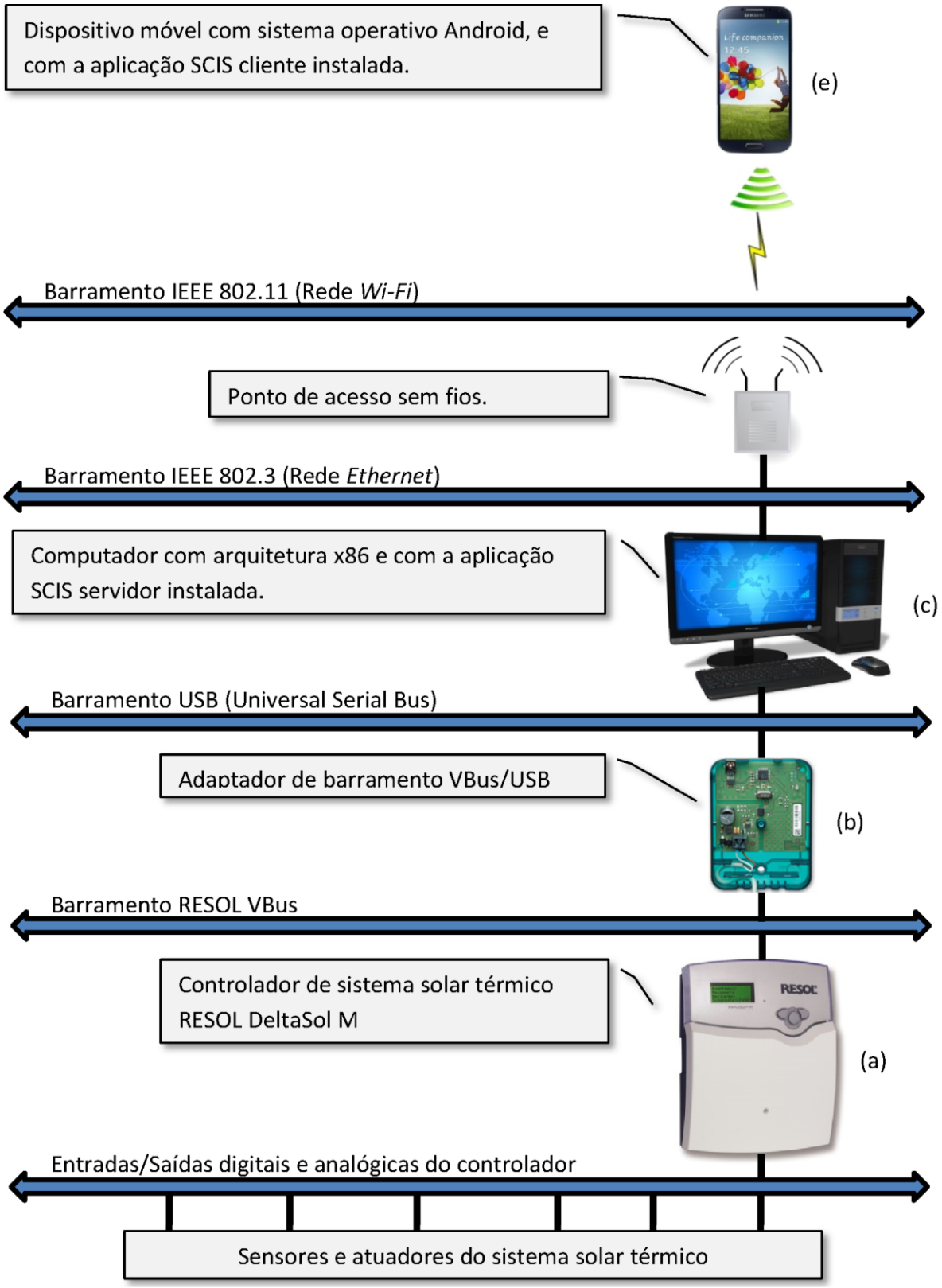

Figura 2.4 - Esquema da arquitetura do sistema

2.3. Plano de desenvolvimento

Após o dimensionamento da arquitetura do sistema, referido na subseção 2.2, planeou-se o desenvolvimento do SCIS, onde se definiram as linguagens de programação, o protocolo a usar na comunicação entre as aplicações cliente e servidor, e a sequência do desenvolvimento das diversas funções do SCIS.

2.3.1. Linguagens de programação das aplicações

Foram escolhidas as seguintes linguagens para a programação das aplicações servidor e cliente:

- **Linguagem de desenvolvimento da aplicação SCIS Cliente**
  A aplicação SCIS cliente será desenvolvida em linguagem Java pois é a usada pelo Android SDK. Esta plataforma disponibilizada pela Google é a mais popular para o desenvolvimento de aplicações Android nativas e disponibiliza as várias bibliotecas que permitem o acesso às funções do dispositivo móvel (ver subseção 1.5.2), desta forma foi uma escolha natural e que não suscitou grandes dúvidas.

- **Linguagem de desenvolvimento da aplicação SCIS servidor**
  Do grande número de linguagens de programação disponíveis para o desenvolvimento da aplicação SCIS servidor, optou-se pela linguagem C#, esta escolha deveu-se sobretudo à experiência prévia do autor do projeto na utilização desta linguagem de alto nível que através do *framework* .NET permite o acesso a um conjunto de bibliotecas com recursos que facilitam o desenvolvimento de aplicações.

2.3.2. Protocolo e formato dos dados para a comunicação entre aplicações

Para o transporte dos dados entre as aplicações cliente e servidor optou-se pelo protocolo HTTP (*Hypertext Transfer Protocol*) [12], pela sua ampla aceitação, e por existirem bibliotecas no Android SDK que implementam as comunicações HTTP de forma nativa. O formato usado para a transmissão dos dados será o JSON (*JavaScript Object Notation*), este formato tem a vantagem de ser suportado nativamente no *framework* .NET e no Android SDK. O formato JSON é em texto e completamente independente de linguagem, pois usa convenções que são similares às linguagens C e familiares, incluindo C++, C#, Java, JavaScript, Perl, Python e muitas outras [13]. Estas propriedades fazem com que JSON seja um formato ideal de troca de dados entre as aplicações SCIS cliente e SCIS servidor.

2.3.3. Estágios do desenvolvimento

Para o desenvolvimento do sistema SCIS foram definidos os seguintes estágios e respetiva sequência:

1º Estágio - Análise do protocolo VBus

Para que sejam adquiridos conhecimentos que permitam implementar as comunicações com o controlador DeltaSol M através o protocolo VBus este deve ser analisado de forma aprofundada.

2º Estágio - Desenvolvimento da comunicação com o controlador através do protocolo VBus na aplicação SCIS servidor

Neste estágio serão programados os algoritmos de comunicação com o controlador DeltaSol M na aplicação SCIS servidor, simultaneamente será criado um interface gráfico na aplicação que permita testar os algoritmos programados e definir os parâmetros de configuração das comunicações.

3º Estágio - Ensaio da aplicação SCIS servidor em ambiente real

Neste estágio serão testadas todas as funções relativas à comunicação com o controlador através do protocolo VBus.

4º Estágio - Desenvolvimento do interface gráfico da aplicação SCIS cliente

Neste estágio será feito o desenvolvimento do interface gráfico da aplicação SCIS cliente, este interface será povoado pelos dados a solicitar ao SCIS servidor.

5º Estágio - Desenvolvimento das comunicações entre as aplicações servidor e cliente

Neste estágio serão desenvolvidas simultaneamente as funções de comunicação entre as aplicações servidor e cliente, usando o protocolo HTTP para transporte dos dados e JSON para a sua formatação.

6º Estágio - Ensaio do SCIS

Neste estágio serão testadas, de modo integrado, todas as funções relativas à comunicação entre o SCIS cliente e controlador DeltaSol M.

3. Protocolo VBus

3.1. Origem

O RESOL VBus é um barramento de comunicação a dois fios desenvolvido pelo fabricante de controladores solares RESOL e foi concebido para permitir a comunicação entre os vários controladores da marca e outros dispositivos.
As especificações do protocolo foram publicadas pela RESOL em 29/08/2003 originalmente em língua alemã no documento intitulado "*RESOL VBUS Protokollspezifikation*" [11]. As especificações em língua inglesa foram publicadas em 27/01/2011 num documento intitulado "*VBus Protocol Specification*" [9].

3.2. Camada física

De acordo com as especificações do protocolo [9] apenas um dos dispositivos é mestre e todos os outros são escravos, estas definem que:

- O dispositivo mestre alimenta o bus com uma corrente constante de valor máximo 35mA ou de 52mA em versões especiais
- A voltagem máxima entre os dois terminais do bus é de 8,2 Volts
- Ao nível lógico "1" corresponde a uma tensão de 0 Volts e ao nível lógico "0" corresponde a tensão máxima no bus, ou seja, 8,2Volts.
- A deteção dos níveis lógicos é feito por um comparador analógico que tem as seguintes tensões limite, ver Tabela 3.1:

	Mestre	**Escravo**
Nível lógico 0	3,25 < V < 8,2	2,25 < V < 8,2
Nível lógico 1	0,0 < V < 3,0	0,0 < V < 3,0

Tabela 3.1 - Níveis de tensão VBus

Nas especificações do protocolo [9] são também disponibilizados 4 circuitos eletrónicos em diferentes configurações para adaptar os níveis de tensão de um UART (*Universal Asynchronous Receiver/Transmitter*) aos níveis do VBus.

3.3. Fluxo de dados

A transmissão e receção de dados através do UART são feitas em modo *half-duplex* a 9600 *bits* por segundo com 1 *start bit* e 1 *stop bit*. Não são usadas paridades nem *handshake*.
Nas especificações [9], o protocolo é definido como tendo uma arquitetura mestre-escravo, no entanto, tanto o mestre como os escravos podem transmitir e solicitar informação, estas são características do tipo *token passing* em que o acesso ao canal de comunicação é multiplexado no tempo e partilhado por todos os dispositivos, sendo no caso do VBus o dispositivo mestre que gere as solicitações de acesso ao canal por parte dos escravos. Esta gestão é feita através da emissão de um datagrama que informa os escravos da disponibilidade do canal, após esta emissão os escravos podem reservar o acesso ao canal com um datagrama num formato específico, no fim da comunicação devem informar o mestre da libertação do canal (ver Tabela 3.10).

3.4. Endereços dos dispositivos VBus

Os dispositivos VBus não são endereçáveis, cada modelo de controlador e dispositivo VBus possui um endereço fixo reservado. Os vários endereços podem ser consultados nas especificações do protocolo [9], para o caso particular deste projeto são relevantes os definidos na Tabela 3.2.

Endereço	Máscara	Destino
0x0000	0xFFFF	*Broadcast*
0x0010	0xFFFF	DFA
0x0020	0xFFFF	PC
0x7311	0xFFFF	DeltaSol M [*Controller*]
0x7312	0xFFFF	DeltaSol M [HK1]
0x7313	0xFFFF	DeltaSol M [HK2]
0x7314	0xFFFF	DeltaSol M [WMZ1]
0x7315	0xFFFF	DeltaSol M [WMZ2]

Tabela 3.2 - Endereços de dispositivos Vbus usados no projeto

O endereço atribuído ao *broadcast* está reservado para pacotes transmitidos na forma de difusão pelos controladores, não se destinam a um só dispositivo escravo mas a todos os que eventualmente possam estar ligados ao bus.

O dispositivo RESOL DFA de endereço 0x0010 representado na Figura 3.1 é um visualizador de dados remotos, os datagramas destinados a este equipamento são transmitidos na forma de difusão pelo controlador DeltaSol M, estes pacotes contêm uma grande quantidade de dados relevantes e são capturados e processados pela aplicação SCIS servidor com o objetivo de os transmitir posteriormente à aplicação SCIS cliente.

O endereço 0x0020 está reservado a dispositivos do tipo PC, e destina-se a ser usado por *software* como é o caso da aplicação SCIS servidor deste projeto.

O endereço 0x7311, 0x7312, 0x7313, 0x7314 e 0x7315, referem-se ao mesmo dispositivo físico, o controlador DeltaSol M, mas dizem respeito a diferentes funções que este implementa, estas funções são respetivamente: funções de controlo do sistema solar; expansão para circuito de calor 1; expansão para circuito de calor 2; calorímetro 1 e calorímetro 2 [8].

Figura 3.1 - Fotografia do RESOL DFA

3.5. Formato dos datagramas

Todos os datagramas VBus iniciam com uma sequência de 6 bytes denominada cabeçalho, o seu formato está representado na Tabela 3.3. O primeiro *byte* da sequência é denominado por *SYNC-BYTE* e tem o valor 0xAA ou 170 decimal. Num datagrama VBus apenas o SYNC-*BYTE* pode ter o bit mais significativo com o valor "1", se algum dos restantes *bytes* apresentar este valor, o datagrama é rejeitado. Na transmissão do datagrama, o bit mais significativo é extraído de todos os bytes do corpo de dados e transferido para um *byte* denominado de *Septed-Byte* que precede cada conjunto de 4 bytes, este byte é usado pelo recetor do datagrama para recuperar os valores originais dos *bytes* do corpo de dados.

Offset	Descrição
0	*SYNC-BYTE* (0xAA)
1	Endereço de destino (LSB)
2	Endereço de destino (MSB)
3	Endereço de origem (LSB)
4	Endereço de origem (MSB)
5	Versão do protocolo

Tabela 3.3 - Cabeçalho do datagrama VBus

Os restantes bytes do datagrama variam de acordo com a versão do protocolo usado. A versão do protocolo é definida pelo valor do *byte* índice 5 segundo a Tabela 3.4.

Valor	Descrição
0x10	Versão 1.0 do protocolo
0x20	Versão 2.0 do protocolo
0x30	Versão 3.0 do protocolo

Tabela 3.4 - Valores do *byte* de versão do protocolo

3.5.1. Versão 1.0 do protocolo

A versão 1.0 do protocolo é usada para transmissão contínua de dados de medição, controlo e balanço entre o dispositivo mestre e os escravos [9]. Como representado na Tabela 3.5, os datagramas da versão 1.0 acrescentam 4 *bytes* ao cabeçalho e nestes é definido o comando e o número variável de *bytes* a ser transmitido no corpo de dados.

Offset	Descrição	Nota
0	SYNC-BYTE (0xAA)	Cabeçalho
1	Endereço de destino (LSB)	
2	Endereço de destino (MSB)	
3	Endereço de origem (LSB)	
4	Endereço de origem (MSB)	
5	Versão do protocolo = 0x10	
6	Comando (LSB)	*Bytes* da Versão 1.0
7	Comando (MSB)	
8	Número de *frames* do corpo de dados	
9	*Checksum* do *offset* 1-8	

Tabela 3.5 - Datagrama do protocolo V1.0

Os comandos a transmitir são definidos pelos valores especificados na Tabela 3.6.

Valor	Descrição
0x0100	O datagrama contém dados para os escravos
0x0200	O datagrama contém dados para os escravos e requer resposta
0x0300	Solicitação a um escravo

Tabela 3.6 - Comandos disponíveis no datagrama versão 1.0

Os *bytes* do corpo de dados são transmitidos em conjuntos de 6 bytes denominados *frames*, o seu formato está definido na Tabela 3.7.

Offset	Descrição
i+0	Primeiro byte do corpo de dados
i+1	Segundo byte do corpo de dados
i+2	Terceiro byte do corpo de dados
i+3	Quarto byte do corpo de dados
i+4	Septed-Byte
i+5	Checksum do offset i+0 até i+4

Tabela 3.7 - Formato de um *frame* do corpo de dados

Nas especificações do protocolo VBus [9] é possível consultar o formato dos pacotes da versão 1.0 do protocolo, para o caso do controlador DeltaSol M, na Tabela 3.8 está especificado o formato do pacote que é transmitido para o dispositivo de visualização remota RESOL DFA, já referido na seção 3.4, o comando do pacote é o 0x0100, que segundo a Tabela 3.6 define que o datagrama contém dados para os dispositivos escravos. Analisando os dados deste pacote verifica-se que contém 67 *bytes* de informação, através do tamanho de cada variável é possível conhecer o seu tipo, assim para os tamanhos 1, 2 e 4 temos respetivamente os tipos *byte*, *int16* e *int32*. Também é especificado o fator multiplicativo que deve ser aplicado ao valor, bem como, o seu nome e unidade. É importante referir que antes destes dados são transmitidos os *bytes* definidos na Tabela 3.5, e que por cada um dos conjuntos de 4 bytes deste pacote são adicionados o respetivo *septed-byte* e o *byte* de CRC conforme especificado na Tabela 3.5 e Tabela 3.7. Assim, serão necessários:

$$10 + 67 + (17 * 2) = 111\ bytes$$

para a transmissão dos dados do pacote definido na Tabela 3.8.

Offset [*bytes*]	**Tamanho [*bytes*]**	**Descrição**	**Fator multiplicativo**	**Unidade**
0	2	*Temperature sensor 1*	0.1	°C
2	2	*Temperature sensor 2*	0.1	°C
4	2	*Temperature sensor 3*	0.1	°C
6	2	*Temperature sensor 4*	0.1	°C
8	2	*Temperature sensor 5*	0.1	°C
10	2	*Temperature sensor 6*	0.1	°C
12	2	*Temperature sensor 7*	0.1	°C
14	2	*Temperature sensor 8*	0.1	°C
16	2	*Temperature sensor 9*	0.1	°C
18	2	*Temperature sensor 10*	0.1	°C
20	2	*Temperature sensor 11*	0.1	°C
22	2	*Temperature sensor 12*	0.1	°C
24	2	*Irradiation*	1	W/m²
28	4	*Impulse input 1*	1	
32	4	*Impulse input 2*	1	
36	2	*Sensor line break mask*	1	
38	2	*Sensor short-circuit mask*	1	
40	2	*Sensor usage mask*	1	
44	1	*Pump speed relay 1*	1	%
45	1	*Pump speed relay 2*	1	%
46	1	*Pump speed relay 3*	1	%
47	1	*Pump speed relay 4*	1	%
48	1	*Pump speed relay 5*	1	%
49	1	*Pump speed relay 6*	1	%
50	1	*Pump speed relay 7*	1	%
51	1	*Pump speed relay 8*	1	%
52	1	*Pump speed relay 9*	1	%
58	2	*Relay usage mask*	1	
60	2	*Error mask*	1	
62	2	*Warning mask*	1	
64	2	*Controller version*	1	
66	2	*System time*	1	

Tabela 3.8 - Dados transmitidos do dispositivo DeltaSol M para o RESOL DFA

3.5.2. Versão 2.0 do protocolo

A versão 2.0 do protocolo é usada para funções de parametrização remota e permite o acesso de leitura e escrita a todos os parâmetros do controlador. Cada datagrama contém um número fixo de 16 *bytes* onde são definidos o endereço do dispositivo de origem e destino, o endereço e o valor do parâmetro, ver Tabela 3.9.

Offset	**Descrição**	**Nota**
0	SYNC-BYTE (0xAA)	Cabeçalho
1	Endereço de destino (LSB)	
2	Endereço de destino (MSB)	
3	Endereço de origem (LSB)	
4	Endereço de origem (MSB)	
5	Versão do protocolo = 0x20	
6	Comando (LSB)	*Bytes* da Versão 2.0
7	Comando (MSB)	
8	Endereço da variável (LSB)	
9	Endereço da variável (MSB)	
10	Valor da variável (*byte* 0)	
11	Valor da variável (*byte* 1)	
12	Valor da variável (*byte* 2)	
13	Valor da variável (*byte* 3)	
14	*Septed-Byte* do *offset* 8-13	
15	*Checksum* do *offset* 1-14	

Tabela 3.9 - Datagrama do protocolo V2.0

Os comandos a transmitir são definidos pelos valores especificados na Tabela 3.10.

Valor	Descrição
0x1000	Resposta do dispositivo com o valor solicitado
0x2000	Escrita de valor, confirmação necessária
0x3000	Leitura de valor, confirmação necessária
0x4000	Escrita de valor, confirmação necessária
0x5000	Libertação do VBus pelo Mestre
0x6000	Libertação do VBus pelo Escravo

Tabela 3.10 - Comandos disponíveis no datagrama versão 2.0

As especificações do protocolo [9] remetem para a aplicação *"RESOL service center"*, como fonte de obtenção endereços dos parâmetros a usar nos datagramas da versão 2.0 do protocolo. Após a instalação desta aplicação, disponível na página web do fabricante RESOL [10], é possível consultar os endereços dos parâmetros de um dispositivo através de uma lista. Na Figura 3.2, podem observar-se parte dos parâmetros disponíveis no controlador DeltaSol M. Nesta lista é possível identificar o endereço de cada parâmetro, o seu nome e a correspondente unidade, no entanto o tipo de variável e o seu fator multiplicativo não estão acessíveis, apesar de serem absolutamente necessários para a comunicação com o controlador.

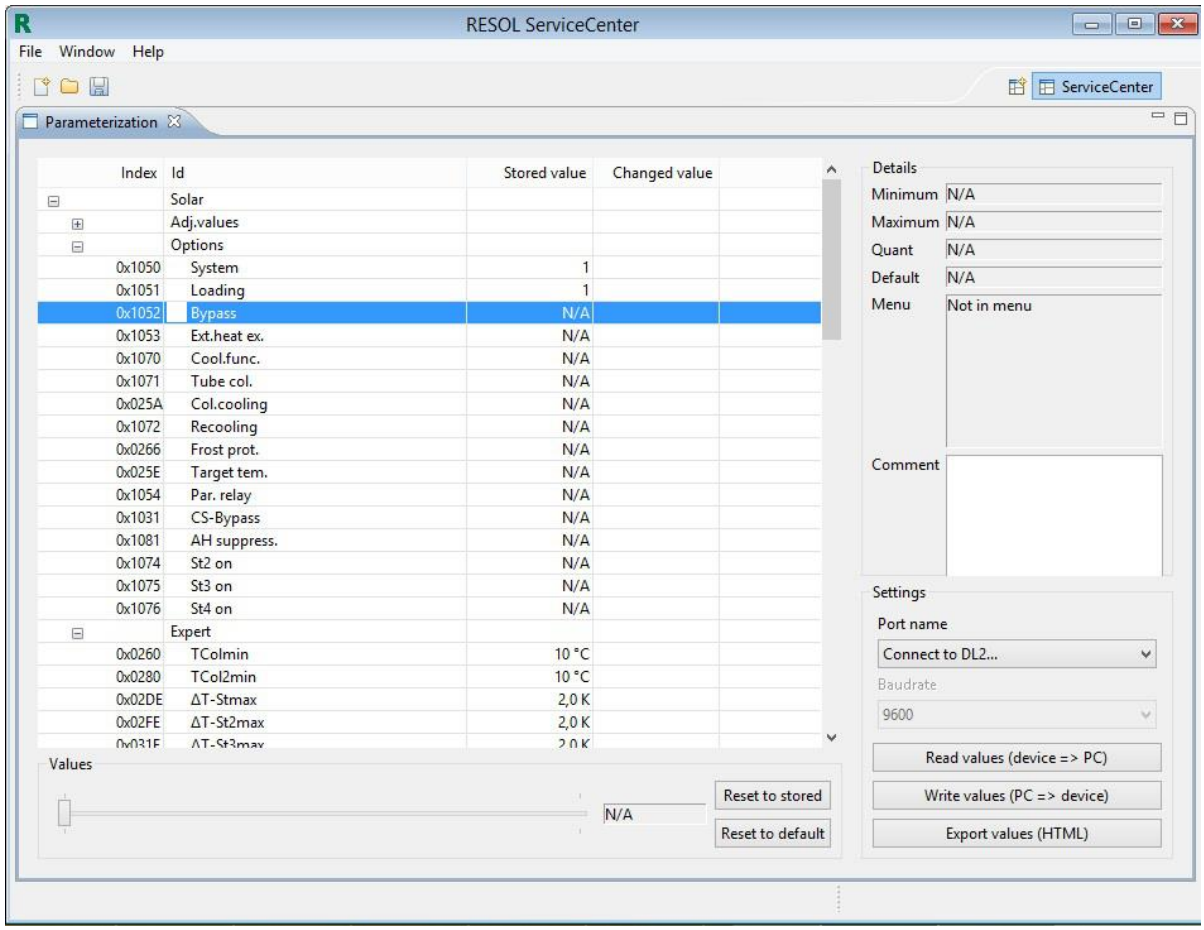

Figura 3.2 - Aplicação *"RESOL Service Center"*

Devido a esta lacuna na informação das propriedades dos parâmetros, foi necessário encontrar uma forma alternativa de obter os dados necessários. Explorando a pasta de instalação da aplicação "*RESOL service center*" verificou-se que todas as configurações relacionadas com os dispositivos VBus estão guardadas em ficheiros no formato XML *(Extensible Markup Language)* [14], onde a informação é mais abrangente que a disponibilizada pela aplicação.

Através destes ficheiros foi possível obter todos os dados necessários para a implementação das comunicações VBus.

3.6. Estrutura dos ficheiros XML do "RESOL *Service Center*"

Como exposto na subseção 3.5.2, parte das informações dos parâmetros de configuração do controlador DeltaSol M, não estão disponíveis na aplicação "RESOL *Service Center*", no entanto, na pasta de instalação desta aplicação é possível aceder a ficheiros no formato XML que contêm as informações em falta, dos vários ficheiros existentes, dois têm informações sobre o controlador DeltaSol M:

- "VbusSpecificationResol.xml"
- "MenuDeltaSolM_1.2.0.xml".

3.6.1. Ficheiro "VbusSpecificationResol.xml"

Este ficheiro contém os endereços de todos os dispositivos VBus da RESOL e a especificação dos pacotes, do protocolo versão 1.0, destes dispositivos. A mesma informação disponível no documento com a especificação do protocolo [9]. Este ficheiro está guardado na pasta: ***"C:\Program Files (x86)\RESOL\ServiceCenterFull\eclipse\plugins\de.resol.servicecenter.vbus.resol_2.0.0\"*** a sua estrutura inicia no nó raiz "*vbusSpecification*" que contém os elementos do tipo "*device*" e "*packet*", que descrevem respetivamente as propriedades dos dispositivos VBus e os pacotes que estes transmitem para o dispositivo de visualização remota RESOL DFA.

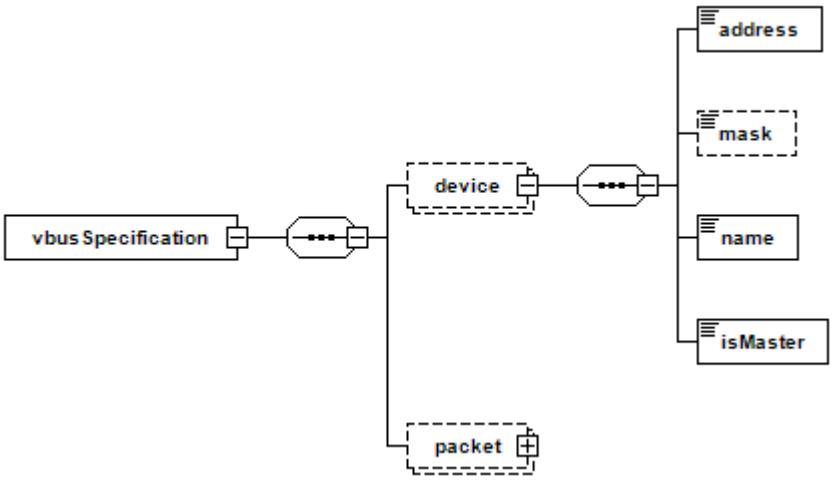

Figura 3.3 - Estrutura do elemento "*device*"

Como se pode observar na Figura 3.3 os elementos do tipo "*device*" têm os seguintes elementos filhos:

- "*Address*" - Endereço do dispositivo VBus
- "*Mask*" - Máscara a aplicar ao endereço
- "*Name*" - Nome do dispositivo
- "*isMaster*" - Tipo de dispositivo, "*true"* para mestre, "*false"* para escravo.

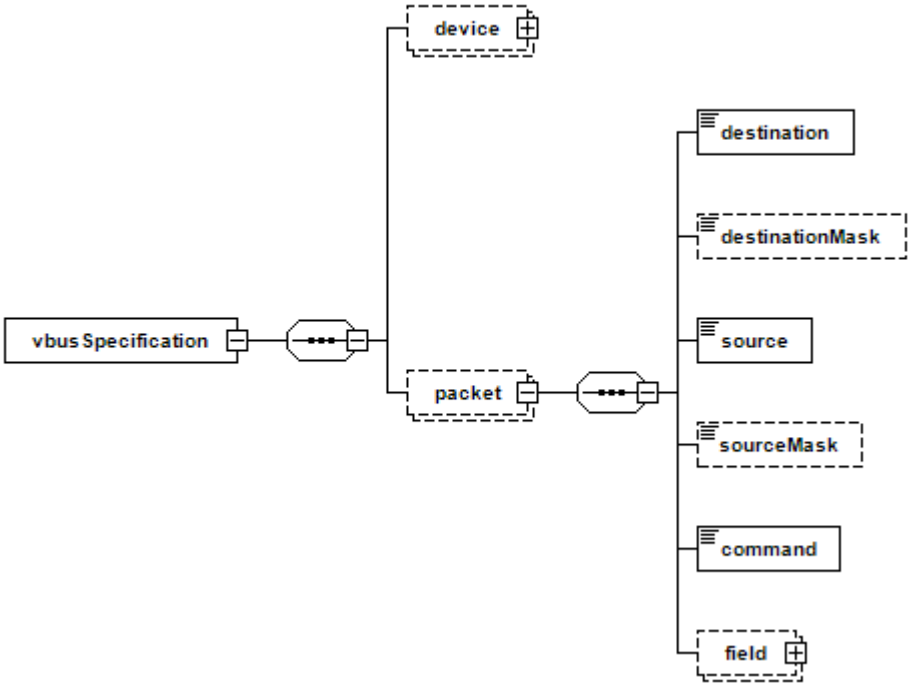

Figura 3.4 - Estrutura do elemento "*vbusSpecification*"

O elemento *"packet"*, ver Figura 3.4, por sua vez contém os seguintes elementos filhos:

- "*destination*" - Endereço de destino do pacote
- "*destinationMask*" - Máscara a aplicar ao endereço de destino
- "*source*" - Endereço de origem do pacote
- "*sourceMask*" - Máscara a aplicar ao endereço de origem
- "*command*" - Comando do pacote

Os elementos do tipo *"field"*, ver Figura 3.5, contém os vários parâmetros transmitidos no pacote. Os elementos filhos deste tipo de elemento são os seguintes:

- "offset" - Offset no pacote, em bytes
- "*name*" - Nome
- "*bitsize*" - Número de *bits* do parâmetro, define o tipo de variável, valores como 15, 16, 31 e 32 representam respetivamente *int16*, *uint16*, *int32*, *uint32*
- "*factor*" - Fator multiplicativo a aplicar ao parâmetro
- "*unit*" - Unidade do parâmetro
- "*format*" - Formato do parâmetro, define a forma de representação do valor, por exemplo valores com formato do tipo "*Time*" são visualizados na forma "Sa 00:00"
- "*bitpos*" - *Offset* em *bits* a partir do *byte* do parâmetro
- "*timeRef*" - Define a origem do tempo para parâmetros do tipo "*Time*"

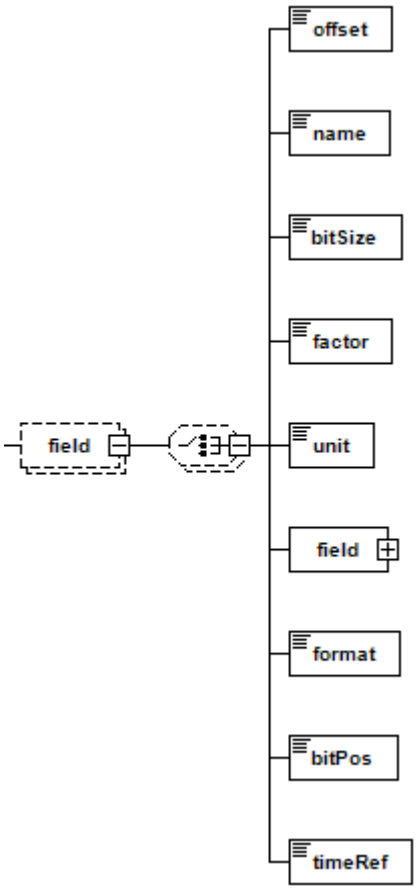

Figura 3.5 - Estrutura do elemento "*field*"

O elemento "*field*" pode conter outros elementos do mesmo tipo, é o caso de parâmetros com contagens em que o seu valor é visualizado em várias parcelas com diferentes escalas.

3.6.2. Ficheiro “MenuDeltaSolM_1.2.0.xml”

Este ficheiro contém as propriedades de todos os parâmetros do controlador DeltaSol M hierarquicamente organizados numa estrutura que representa o menu disponível no display físico do controlador. Este ficheiro está guardado na pasta:

“C:\Program Files(x86)\RESOL\ServiceCenterFull\eclipse\plugins\de.resol.servicecenter.devices.resol_2.0.0”

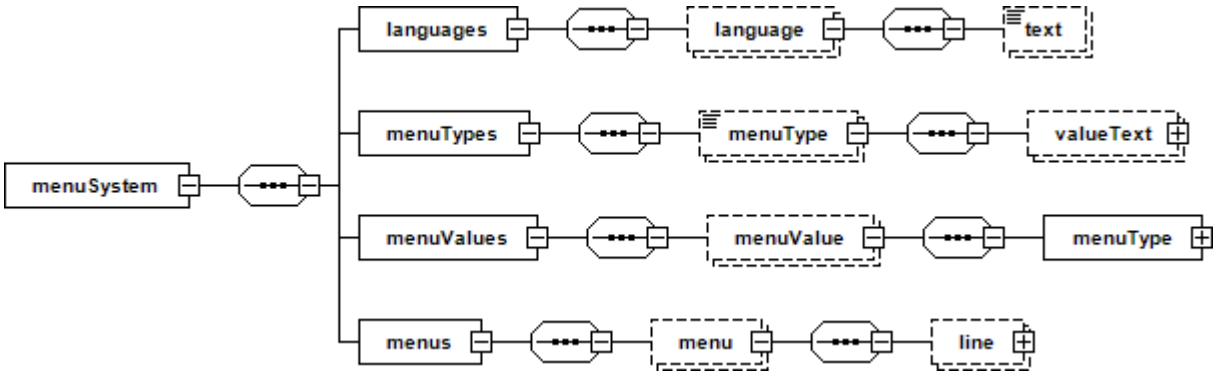

Figura 3.6 - Estrutura do elemento “*menuSystem*”

Na Figura 3.6, pode observar-se a estrutura do ficheiro “MenuDeltaSolM_1.2.0.xml”. Inicia no nó raiz “*menuSystem*” que contém os seguintes elementos filhos:

- “*languages*” - Elemento que contém a definição das várias linguagens que o dispositivo suporta, cada elemento filho “*language*” tem como atributo o “*id”* que define o código da língua.
- “*menuTypes*” - Elemento que contém a definição dos vários tipos de formatos de visualização disponíveis no menu do dispositivo, cada um dos seus elementos filhos “*menuType*” tem um atributo “*id*” que identifica o nome do formato, pode assumir valores como: “*Number*”; “*Time*”; “*Boolean*”, entre outros. Cada elemento filho “*valueText*” tem um atributo “*value*” que define um valor, e nos seus elementos filhos “*menuText*” está definido o texto a apresentar no menu para aquele valor, este texto pode ser apresentado em várias linguagens cujo código é definido no atributo “*lang*” deste último elemento.
- “*menuValues*” - Elemento onde são efetivamente definidos os parâmetros, cada elemento filho “*menuValue*” tem como atributos o “*id*” que define o nome do parâmetro e o atributo “*index*” que define o seu endereço. O elemento “*menuType*” que é filho do “*menuValue*” contém os elementos que definem as várias propriedades do parâmetro como: fator multiplicativo; valor mínimo; valor máximo; unidade e

valor por defeito, ver Figura 3.7, este último elemento tem ainda um atributo "*base*" qual o formato do valor do parâmetro: "*Number*"; "*Time*"; "*Boolean*", entre outros.

- "*menus*" - Elemento que contém vários elementos filhos do tipo "*menu*", que possuem uma descrição única no seu atributo "*id*", estes elementos por sua vez contêm elementos filhos do tipo "*line*" que definem as várias linhas de um menu, uma por cada elemento "*line*". O parâmetro relativo a cada linha é identificado no elemento "*valueRef*" que deve ter um valor igual ao descrito no "*id*" do "*menuValue*" respetivo. Os elementos "*mask*" e "*maskval*" definem dependências de outros parâmetros, o elemento "*text*" define o texto a apresentar no menu e o elemento "*action*" define uma ação que deve ser iniciada na seleção da linha de menu. A estrutura pode ser observada na Figura 3.7.

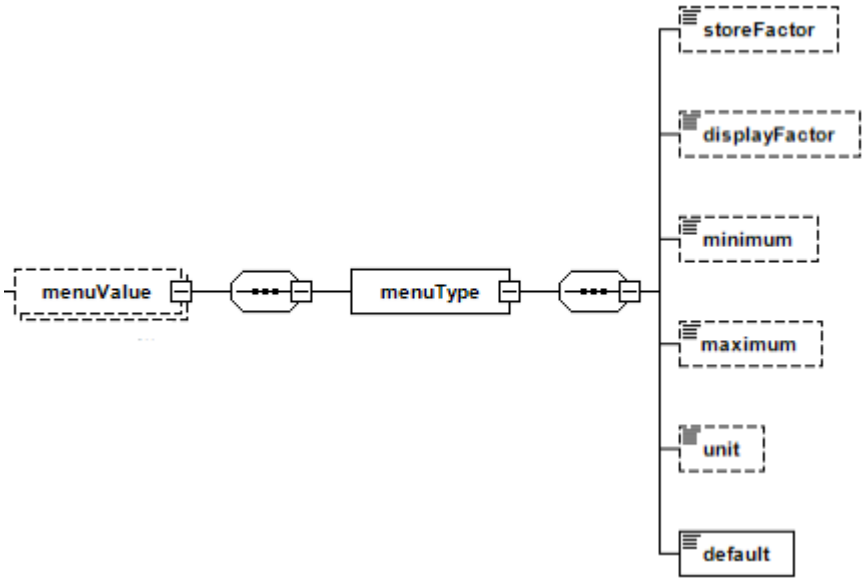

Figura 3.7 - Estrutura do elemento "*menuValue*"

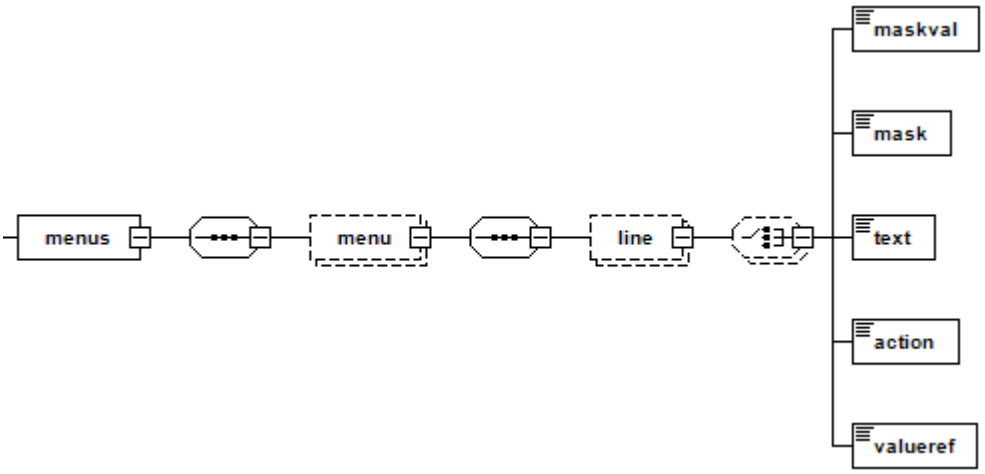

Figura 3.8 - Estrutura do elemento "*menu*"

4. Aplicação SCIS servidor

A aplicação SCIS servidor é o componente do sistema que comunica com o controlador DeltaSol M através do adaptador de barramento USB/VBus (ver seção 2.2), com o objetivo de transmitir os dados à aplicação SCIS cliente. Esta aplicação foi desenvolvida em linguagem C# usando o *framework* .NET 4.0 que disponibiliza um conjunto de bibliotecas específicas para o desenvolvimento de aplicações para o sistema operativo Windows. O ambiente de desenvolvimento (IDE) escolhido foi o *Visual Studio 2013* que tal como *framework* .NET é disponibilizado pela empresa Microsoft.

Neste capítulo será feita uma descrição das funcionalidades desta aplicação e dos pontos mais relevantes do seu desenvolvimento.

4.1. Funcionalidades da aplicação

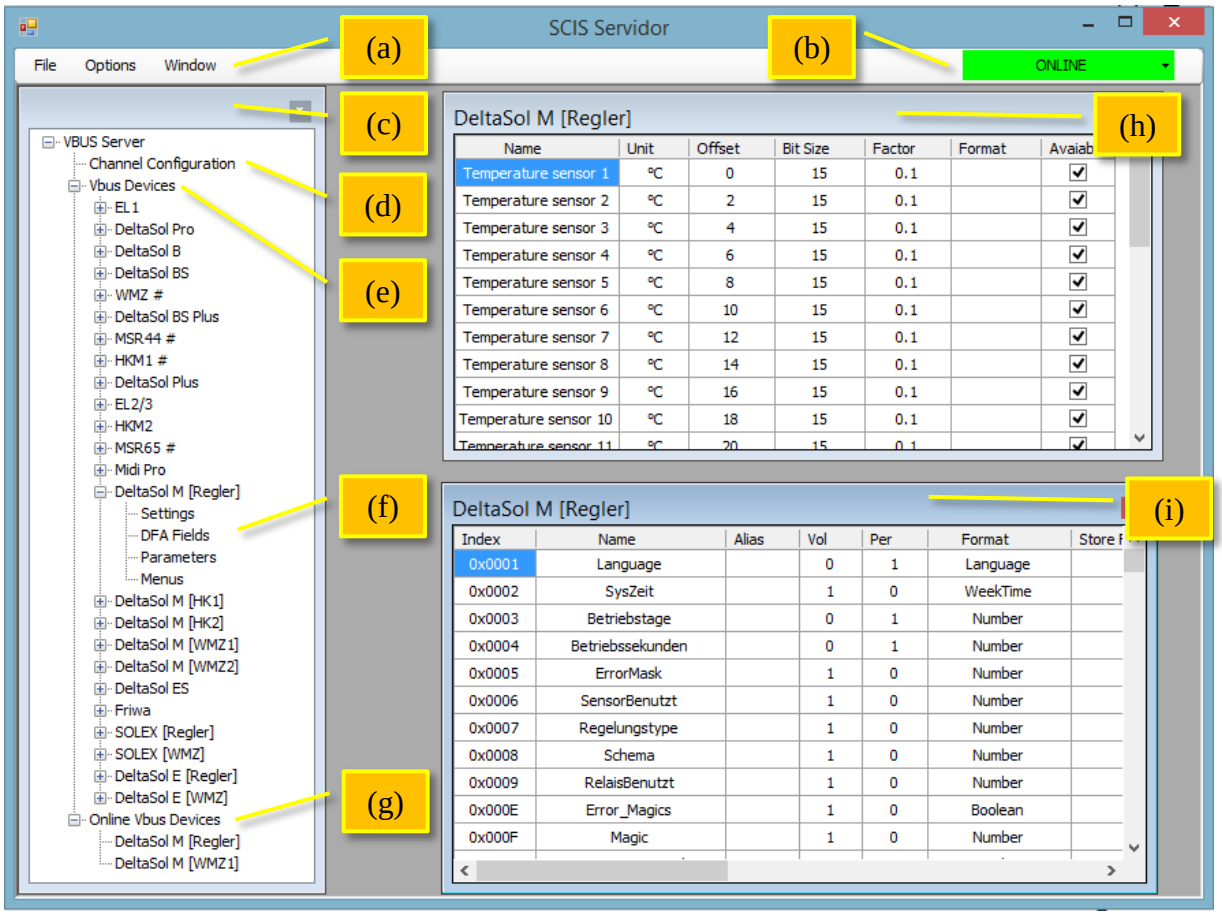

Figura 4.1 - Aplicação SCIS servidor

Como se pode observar na Figura 4.1, a aplicação SCIS servidor é composta por:

- Uma barra de ferramentas (a), com um botão "*File*" que permite gravar as configurações e sair da aplicação, um botão "*Options*" que permite importar as configurações do ficheiro XML com as definições dos dispositivos VBus e um botão "*Window*" que permite aceder às janelas de *debug* e de navegação (c).
- Um controlo (b) que permite visualizar e alterar o estado da conexão com o controlador DeltaSol M através da porta série, este controlo pode assumir os seguintes estados:
    - Cor cinzenta com o texto "*Disconnected*" - Aplicação sem ordem para conectar ao controlador;
    - Cor verde com o texto "*Connected*" - Aplicação conectada com sucesso ao controlador;
    - Cor vermelha com o texto "*PORT ERROR*" - Aplicação não consegue conectar ao controlador devido a um erro na porta série, que pode já estar a ser usada ou não existir.
- Uma janela de navegação (c) onde é possível aceder às configurações dos canais de comunicação (d) porta série e TCP/IP, e a uma árvore (e) com todos os dispositivos VBus importados do ficheiro XML referido na subseção 3.6.1. A expansão do nó de cada dispositivo (f) dá acesso às seguintes opções:
    - "*Settings*" - Dá acesso à janela com as propriedades do dispositivo VBus, ver Figura 4.2.
    - "*DFA Fields*" - Dá acesso à janela com as propriedades dos dados transmitidos deste dispositivo para o dispositivo de visualização remota RESOL DFA, ver Figura 4.1 (h).
    - "*Parameters*" - Dá acesso à janela com os valores e as propriedades dos parâmetros do controlador, ver Figura 4.1 (i). Estas propriedades são as importadas do ficheiro XML referido na subseção 3.6.2.
    - "*Menus*" - Dá acesso à janela de navegação no menu do controlador, ver Figura 4.3.

  Através da janela de navegação (c) é também possível visualizar o nó "*Online VBus Devices*" (g) que mostra todos os dispositivos a transmitir pacotes de dados por difusão no momento. Um duplo clique num destes dispositivos dá acesso à janela dos

dados para visualização remota que esse dispositivo está a transmitir no momento, ver Figura 4.4.

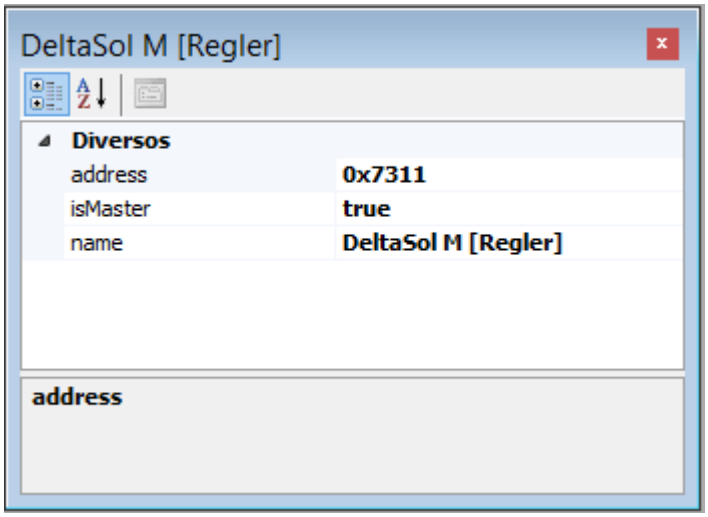

Figura 4.2 - Janela de propriedades do dispositivo VBus

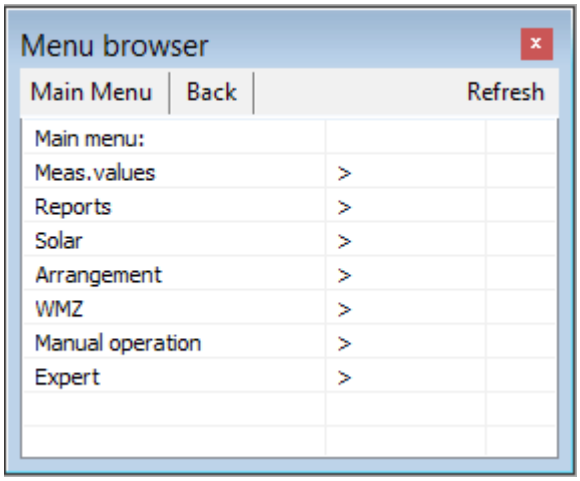

Figura 4.3 - Janela de navegação no menu

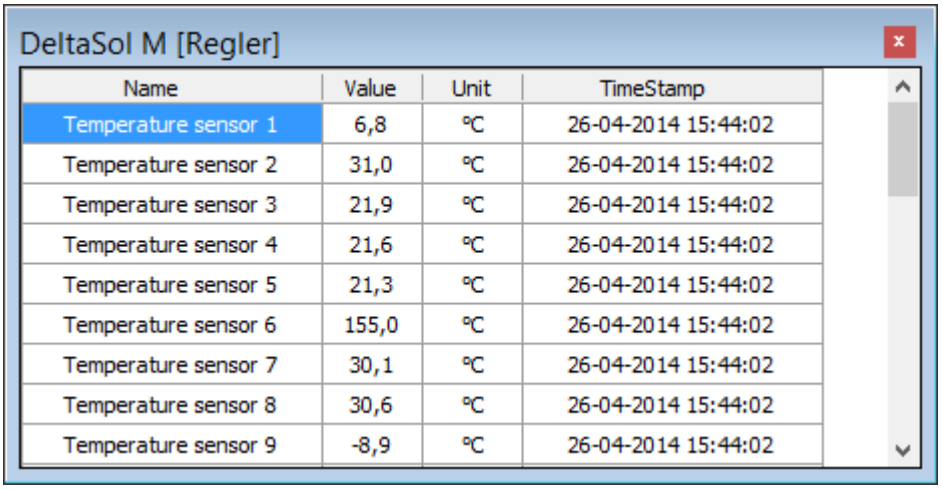

Figura 4.4 - Janela com os dados para visualização remota

4.2. Comunicação com o controlador solar

Na Figura 4.5 está representado o fluxograma das comunicações implementadas entre a aplicação SCIS e o controlador solar através do protocolo VBus, nas subseções seguintes são descritas as classes e métodos usados nesta comunicação.

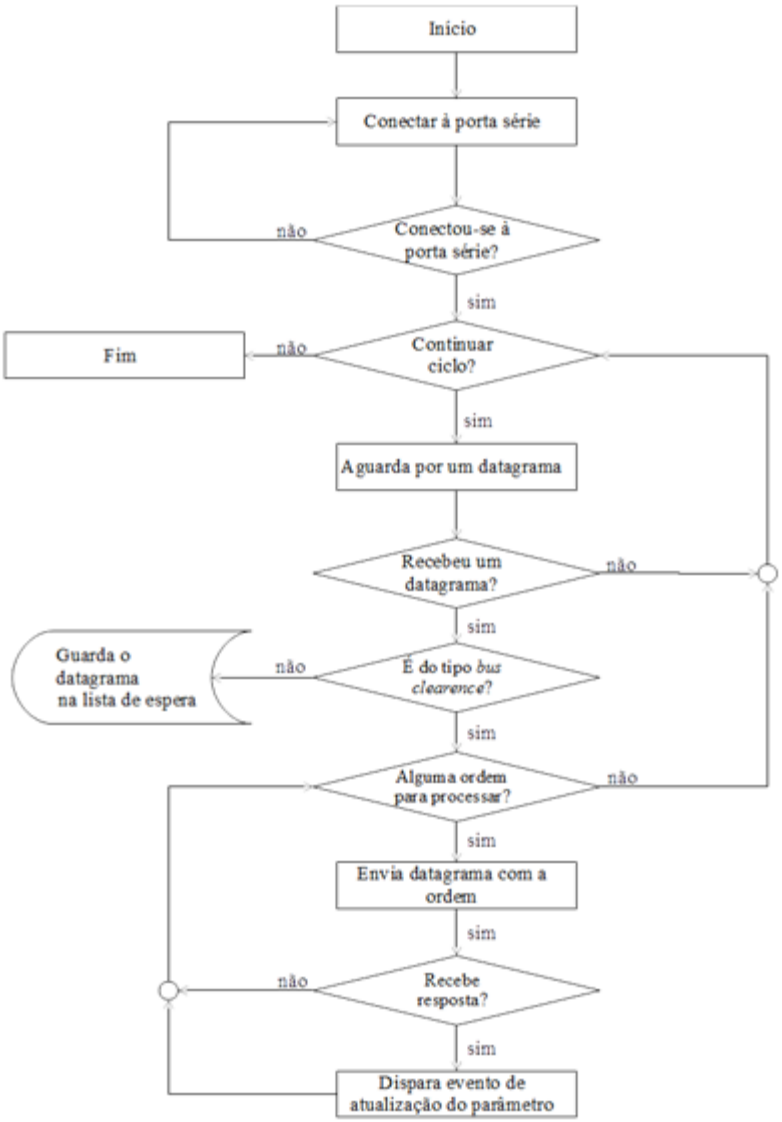

Figura 4.5 - Fluxograma das comunicações VBus

4.2.1. Classe "*VbusConnection*"

De forma a criar uma camada de abstração entre as comunicações de baixo nível VBus e os métodos de transmissão remota dos dados, foi criada a classe "*VbusConnection"* representada na Figura 4.6. Esta classe contém todos os métodos e definições necessárias à gestão das comunicações VBus através do adaptador de barramento VBus/USB. O *driver* USB deste

adaptador cria uma porta série virtual no sistema operativo Windows, através desta é feita a transmissão de dados entre a aplicação SCIS servidor e o controlador DeltaSol M. A ligação do servidor a esta porta série é feita através de uma instância da classe "*System.IO.Ports.SerialPort"* do *framework .NET,* esta classe implementa todos métodos necessários para a comunicação através deste canal de dados. A porta série a usar na conexão ao controlador é configurada na janela representada na Figura 4.7 disponível na aplicação SCIS servidor.

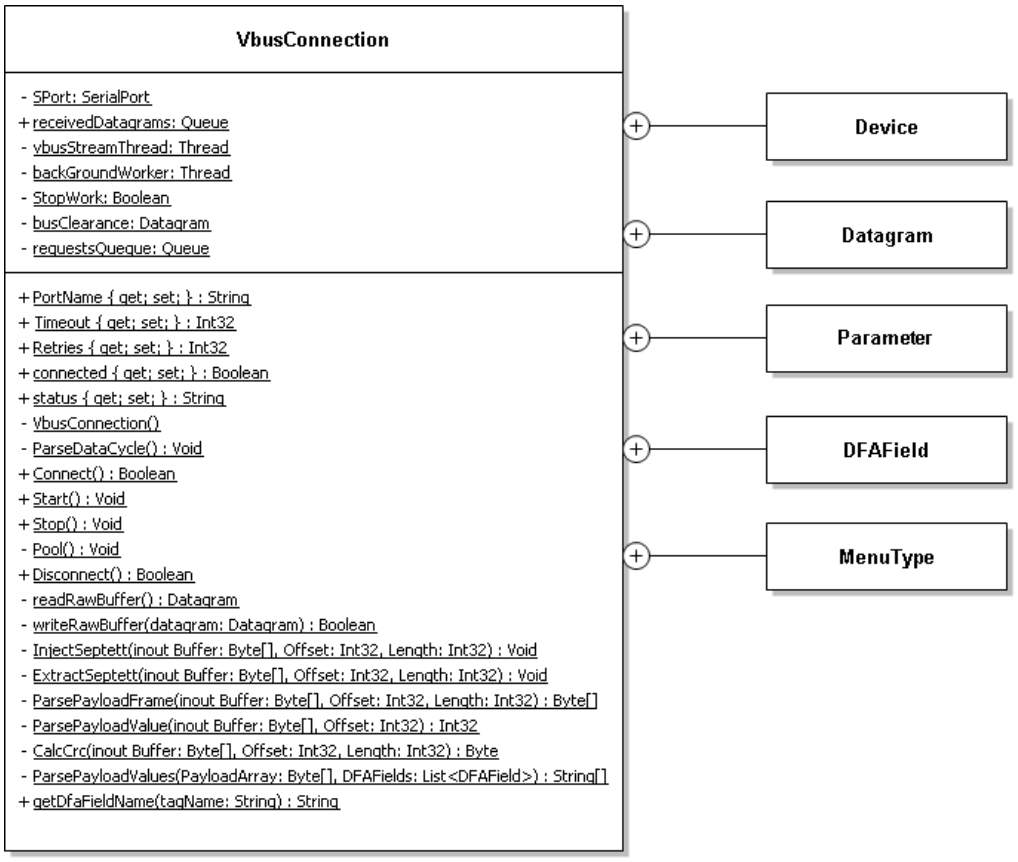

Figura 4.6 - Diagrama da classe *VbusConnection* e as suas classes auxiliares

Apesar da análise aprofundada do código fonte não fazer parte dos objetivos deste relatório, é importante referir os métodos principais da classe *VbusConnection*:

- *bool Connect()* - Inicia a comunicação com a porta série
- *void Pool()* - É o núcleo das comunicações, é chamado numa *thread* diferente da *thread* principal e no seu ciclo são feitos os pedidos aos dipositivos VBus.
- *void Start()* - Inicia o ciclo *Pool()*
- *void Stop()* - Termina o ciclo *Pool()*

- *Datagram readRawBuffer()* - Nível mais baixo das comunicações, este método lê os dados diretamente da porta série, processa-os segundo as especificações VBus referidas na seção 3.4 e devolve um objeto do tipo "*Datagram*".
- *bool writeRawBuffer(Datagram datagram)* - Semelhante ao anterior, trata os dados do objeto do tipo "*Datagram"* passado por argumento e envia-os pela porta série.
- *void InjectSeptett(ref byte[] Buffer, int Offset, int Length)* - Injecta o *Septett byte* conforme referido na seção 3.4.
- *void ExtractSeptett(ref byte[] Buffer, int Offset, int Length)* - Extrai o *Septett byte* conforme referido na seção 3.4.
- *byte CalcCrc(ref byte[] Buffer, int Offset, int Length)* - Cálculo do CRC de um *array* de *bytes* passado por referência.

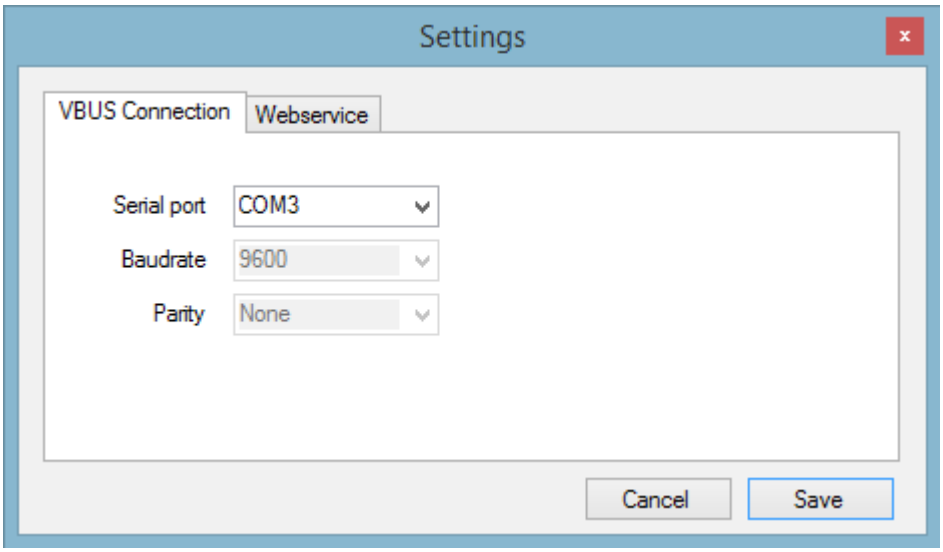

Figura 4.7 - Janela de configuração das comunicações VBus

4.2.2. Classe "*Device*"

A classe "*Device*", representada na Figura 4.8, contém todas as definições relativas a um dispositivo VBus, por exemplo, o seu nome e endereço. Nesta classe são também guardadas as listas de parâmetros, menus, e "DFA *Fields"* (conjunto de variáveis transmitidas para o dispositivo RESOL DFA na versão 1.0 do protocolo), também chamados "dados para visualização remota" ao longo deste relatório. Tal como referido na seção 3.6, todas as definições de parâmetros de um dispositivo VBus estão guardadas em ficheiros XML na aplicação "RESOL *service center*", no caso do controlador DeltaSol M usado neste projeto, existem mais de 400 parâmetros. A introdução manual de todas as propriedades destes parâmetros seria uma tarefa penosa para o utilizador, pelo que foram programados métodos

que interpretam os ficheiros XML referidos nas subseções 3.6.1 e 3.6.2 e que importam as definições diretamente para instância da classe "*Device"*, esses métodos são respetivamente: *"ImportDeviceFromXML(string filepath)* e *"ImportMenuFromXML(string filepath)*.

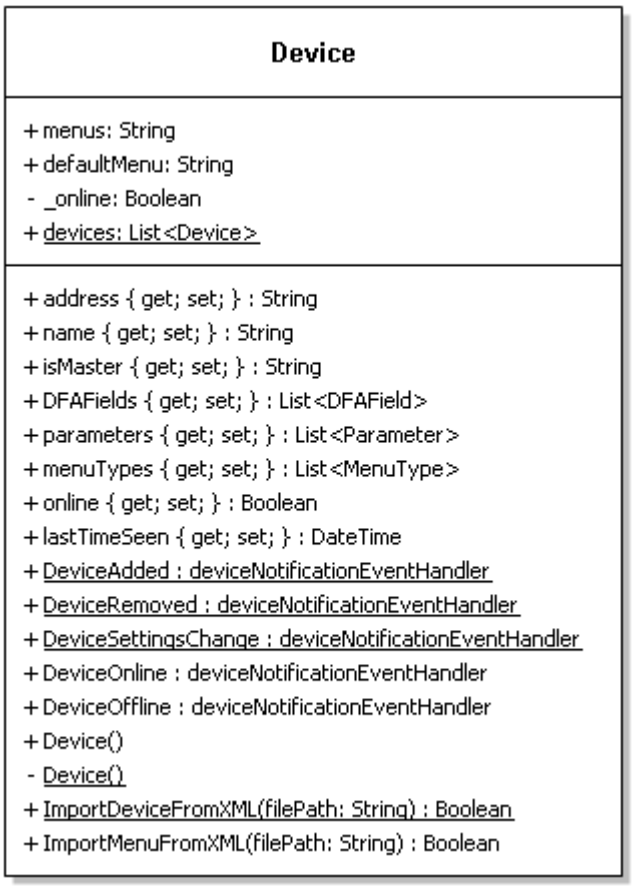

Figura 4.8 - Diagrama da classe "*Device*"

4.2.3. Classe "*Datagram*"

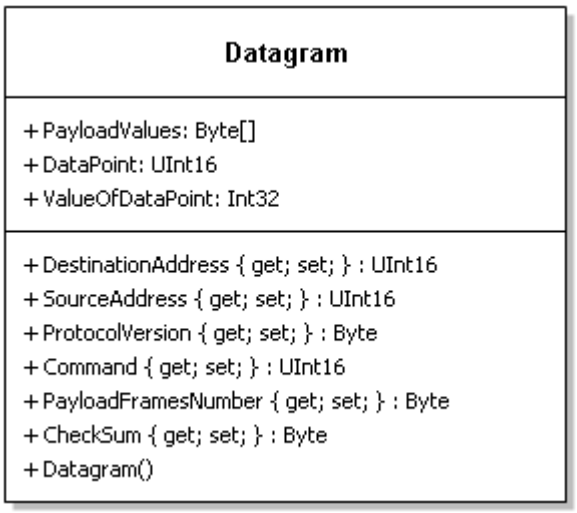

Figura 4.9 - Diagrama da classe "*Datagram*"

A classe "*Datagram*", representada na Figura 4.9, define o pacote VBus na sua forma mais elementar, contém as propriedades do "*header*" e as propriedades dos pacotes da versão 1.0 e 2.0, que apesar de não serem usadas em simultâneo, coexistem na classe. As instâncias desta classe são datagramas VBus, recebidos ou a transmitir.

4.2.4. Classe "*DFAField*"

A classe "*DFAField*", representada na Figura 4.10, define as propriedades das variáveis VBus transmitidas através da versão 1.0 do protocolo para o dispositivo de visualização remota de dados. As instâncias desta classe contêm variáveis cujas propriedades são importadas do ficheiro XML, referido na subseção 3.6.1, através do método *"ImportDeviceFromXML(string filepath)"* da classe "*Device*". Como pode ser observado na Figura 4.1 (h), através do interface gráfico da aplicação SCIS servidor é possível consultar e modificar as propriedades de cada parâmetro.

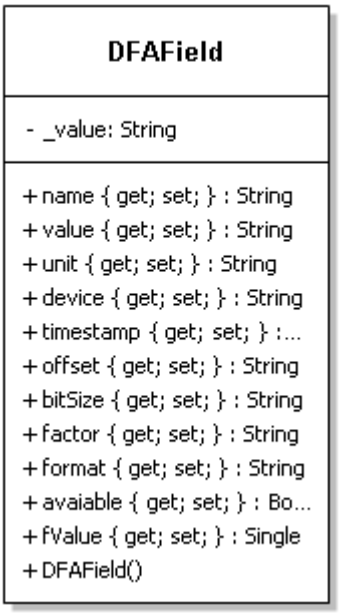

Figura 4.10 - Diagrama da classe "*DFAField*"

4.2.5. Classe "*Parameter*"

A classe "*Parameter*", representada na Figura 4.11, define as propriedades das variáveis VBus transmitidas através da versão 2.0 do protocolo. As instâncias desta classe contêm variáveis cujas propriedades são importadas do ficheiro XML, referido na subseção 3.6.2, através do método *"ImportMenuFromXML(string filepath)"* da classe "*Device*". Através do

interface gráfico da aplicação SCIS servidor é possível consultar e modificar as propriedades de cada parâmetro, ver Figura 4.1 (i).

Figura 4.11 - Diagrama da classe "*Parameter*"

4.3. Processamento dos menus

Uma das possibilidades do SCIS é disponibilizar na aplicação cliente o menu disponível através do *display* físico do controlador DeltaSol M. Este menu permitirá ao utilizador ter acesso a todas as funções documentadas no manual do controlador DeltaSol M [8] e possibilitará um controlo remoto equivalente ao controlo local, inclusive o controlo manual dos equipamentos ligados ao controlador DeltaSol M através do sub-menu "*Manual operation*".

Tendo em conta a grande quantidade de dados importados do ficheiro XML que contém as definições do menu (ver subseção 3.6.2), e o processamento necessário para a sua interpretação, optou-se por atribuir ao SCIS servidor todo o esforço computacional envolvido

nesta tarefa, e implementar neste todas as funções de processamento dos menus. A disponibilização do menu do controlador na aplicação servidor não tem muita utilidade, no entanto, durante o desenvolvimento surgiu a necessidade de testar as funções relativas ao seu processamento. Para esse efeito, foi criada a janela que recebeu o nome de "*Menu Browser"* representada na Figura 4.3, esta janela permitiu testar todo o processamento dos menus antes do desenvolvimento da aplicação SCIS cliente. O conteúdo dos menus apresentados nesta janela de navegação é semelhante ao apresentado ao utilizador da aplicação SCIS cliente.

Como analisado na subseção 3.6.2, os elementos do ficheiro XML definem cada linha do menu (ver Figura 3.8) e no seu elemento filho "*action*" está definida a ação a executar em caso de seleção dessa linha. Os tipos de ação referidos no ficheiro XML, foram identificados e programados, sendo descritos na Tabela 4.1.

Ação	Função
""	A não existência de ação define que um item é só de leitura
"*edit*"	Editar o valor do item selecionado
"*menuback*"	Voltar ao menu anterior
"*menujump*"	Ir para o menu selecionado

Tabela 4.1 - Tipos de ação do menu do controlador

menuType	**Valores**	**Formato para visualização**
Number	[-32768;32767]	[-32768; 32767] * Fator multiplicativo
Time	[0;1339]	[00:00; 23:59]
WeekTime	[0; 10079]	[00:00; 23:59] + [Mo; Tu; We; Th; Fr; Sa; Su]
Language	[0;4]	[Deutsch; English; Francais; Castellano]
ErrorSensorNr	[0;65536]	[0;65536]
SensorNr	[0;65536]	[0;65536]
Boolean	[0;1]	[No; Yes]
State	[0;1]	[Off; On]
Tristate_Relais	[0;2]	[Off; Auto; On]
SolarType	[0;9]	[A; B; C; D; F; G; H; I; J; K]
Regelungstyp	[0;2]	[None; Rise; PI-Reg.]
Bypass_Type	[0;1]	[Valve; Pump]
HkNachheizungType	[0;2]	[None; Term; Store]
OptionHkType	[0;2]	[No; HCM 1; HCM 2]
Modi_WSU	[0;2]	[Night/Day; Off/Day; Without]

Tabela 4.2 - Formatos de apresentação dos parâmetros do menu

O formato em que o valor dos parâmetros deve ser apresentado e introduzido é definido no atributo "*base*" de cada elemento *"menuType"* do ficheiro XML (ver Figura 3.6) e pode variar entre os apresentados na Tabela 4.2.

Sempre que o utilizador seleciona uma linha de menu em que o elemento "*action*" tem atributo "*menujump"*, é processado e apresentado um novo menu (identificado pelo valor desse elemento "*action*"), apenas neste instante são solicitados ao controlador DeltaSol M os valores dos parâmetros do sub-menu de destino, desta forma garante-se que o SCIS tem o mínimo de impacto no funcionamento normal do barramento VBus, pois como explicado na subseção 3.5.2 as solicitações ao controlador necessitam que seja transmitido um datagrama a requisitar o acesso exclusivo ao barramento, ficando os restantes dispositivos impedidos de comunicar até que as solicitações sejam concluídas. No entanto, sempre que um parâmetro é lido ou escrito, é gerado um evento na instância da classe "*Parameter*" relativa a esse parâmetro, este evento é subscrito pelos métodos de atualização do menu, desta forma evita-se a dessincronização entre o valor real e o apresentado no menu.

4.3.1. Classe VbusMenuBrowser

O processamento dos menus é feito pela classe "*VbusMenuBrowser"* representada na Figura 4.12, esta classe implementa o método "*String LoadMenu(String deviceAddress, String menuId, String lang, Boolean refreshValues)"*, que permite carregar qualquer menu de um dispositivo, recebe como argumentos: o endereço do dispositivo; a identificação do menu a carregar; a linguagem e uma *flag* que dá indicação para que os valores do menu sejam solicitados ao controlador. Após o processamento, este método devolve uma *string* com o menu a apresentar definido por uma coleção de instâncias da classe "*ParsedMenuLine"* serializada e formatada em JSON. Desta forma as funções de transmissão remota para a aplicação cliente apenas têm que executar este método quando um menu é solicitado, como todo o processamento é feito na própria classe, há uma camada de abstração entre estas duas funcionalidades.

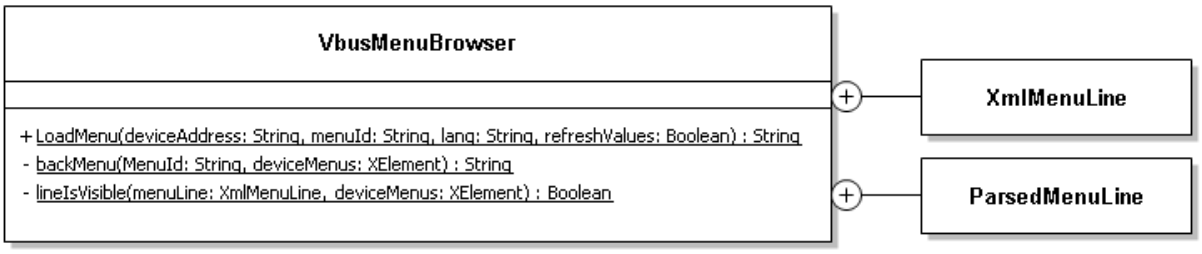

Figura 4.12 - Diagrama da classe "*VbusMenuBrowser*" e classes auxiliares

4.3.2. Classe ParsedMenuLine

Para a formatação das linhas do menu para a aplicação SCIS cliente, é usada a classe *ParsedMenuLine*, representada na Figura 4.13, esta classe define qual o texto a mostrar nas várias posições de uma linha de menu como representado na Figura 4.14, a propriedade "*action"* define qual a ação associada a cada linha, ao definir no servidor as ações de cada linha de menu, estas poderão ser alteração sem necessidade de modificar a aplicação cliente.

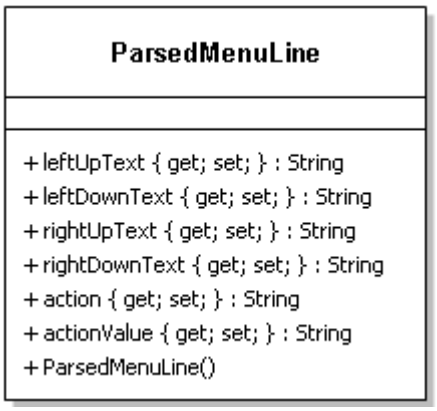

Figura 4.13 - Diagrama da classe "*ParsedMenuLine*"

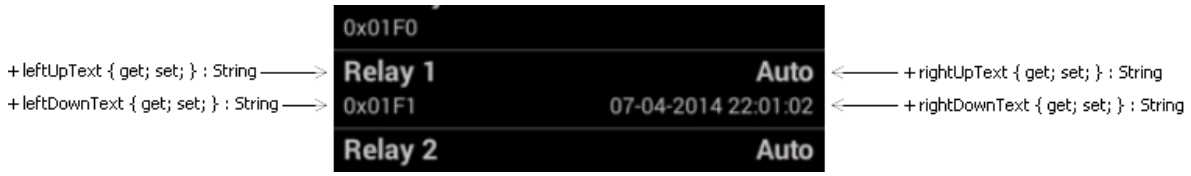

Figura 4.14 - Posições do texto numa linha de menu da aplicação cliente

4.3.3. Classe XmlMenuLine

A classe *XmlMenuLine* é uma classe auxiliar, uma coleção de instâncias desta classe contém as definições do menu importadas do ficheiro XML referido na subseção 3.6.2. Esta classe permite armazenar os dados no ficheiro de configurações da aplicação após a importação do mesmo. As propriedades da classe são as definidas no ficheiro XML para os elementos "*line*" representados na Figura 3.8.

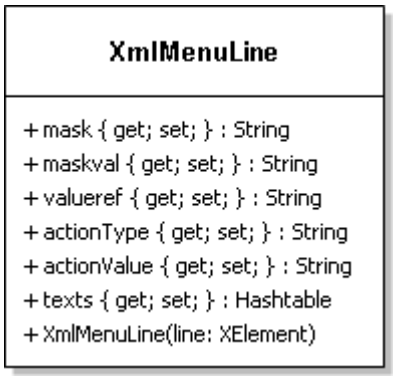

Figura 4.15 - Diagrama da classe "*XmlMenuLine*"

4.4. Comunicação com o cliente

Na aplicação SCIS servidor foi criada a classe "*WebService"* representada na Figura 4.16, nela foi implementada toda a gestão das solicitações HTTP [12] iniciadas pelos clientes (ver Figura 4.18), para a gestão das conexões usou-se a classe "*TcpListener"* do *framework* .NET. No método *Listener* é processado um ciclo que aguarda por conexões dos clientes, sempre que uma nova conexão é detetada é lançado o método *Respond* num novo *thread*, este método processa a resposta às solicitações em função dos parâmetros passados pelo cliente na solicitação HTTP GET.

Na aplicação SCIS servidor é possível selecionar a porta de escuta de solicitações, essa configuração pode ser feita na janela *Settings* no separador *Webservice*, ver Figura 4.17.

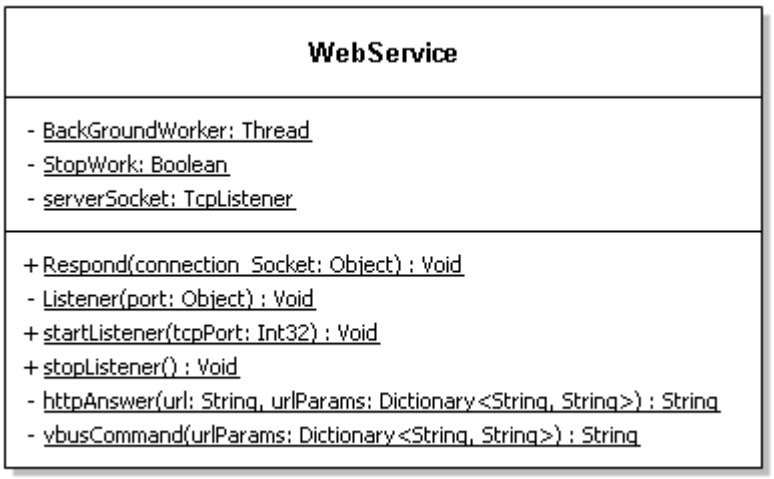

Figura 4.16 - Diagrama da classe "*WebService*"

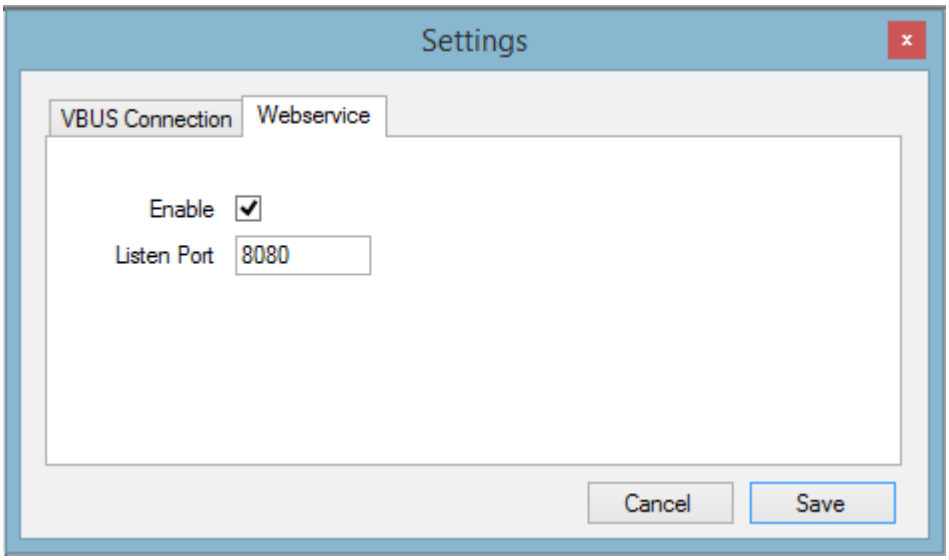

Figura 4.17 - Janela de configuração do *Webservice*

As solicitações são feitas à aplicação SCIS servidor através do comando HTTP GET que é um método de solicitação de dados através do protocolo HTTP. Uma solicitação efetuada através do método GET contém o endereço do servidor (URL), seguido de um conjunto de parâmetros organizados por pares nome/valor, na forma "nome=valor", separados pelo caracter "&".

Assim, todas as solicitações feitas pela aplicação SCIS cliente são feitas através de um endereço na forma:

"http://[Endereço do servidor]/vbus?command=[comando a executar]&[parâmetro adicional 1]=[valor]&[parâmetro adicional 2]=[valor]"

Em que o parâmetro "*command*" pode ter um dos seguintes valores:

- "***listdfafields***" - Solicita uma lista com todos os valores de visualização remota transmitidos pelo controlador, ver seção 3.5.1. Exemplo:
  "http://host/vbus?command=listdfafields"
- "***listmenu"*** - Acompanhado dos parâmetros "*device*", "*lang*" e "*menu*" solicita um menu de um dispositivo, definindo a linguagem. Exemplo:
  "http://host/vbus?command=listmenu&device=0x7311&menu=Solar_Experte"
- "***editparameter"*** - Acompanhado dos parâmetros "*device*" e "*parameter*" solicita o tipo e uma lista dos valores possíveis para um determinado parâmetro de um dispositivo. Exemplo:
  "http://host/vbus?command=editparameter&device=0x7311&parameter=0x0141"

  "***setparameter"*** - Acompanhado dos parâmetros "*device*", "*parameter*" e "*value*" dá ordem escrita do valor num parâmetro do dispositivo definido. Exemplo: "*http://host/vbus?command=setparameter&device=0x7311&parameter=0x0141&value=1"*

Como referido na subseção 3.5.1, o controlador DeltaSol M está continuamente a transmitir valores para o dispositivo de visualização remota RESOL DFA, a aplicação SCIS servidor mantém uma lista com dos últimos valores recebidos para consulta por parte dos clientes remotos. Esta lista referida no fluxograma da Figura 4.5 como a ação "Guarda o datagrama na lista de espera", é consultada sempre que são solicitados os valores de visualização remota ao servidor através do parâmetro HTTP "*command=listdfafields*".

Quando o utilizador solicita um menu através do parâmetro HTTP "*command=listmenu*", são transmitidas para a aplicação SCIS cliente as linhas de texto desse menu, e a aplicação mostra-as pela mesma ordem e formato. Da mesma forma, sempre que há um evento de navegação, por exemplo, escolher um novo menu, é enviado para o servidor esse evento na forma de solicitação HTTP, este será processado de acordo com o seu tipo e será enviada a resposta referente ao tipo de solicitação. Se o servidor receber um evento do tipo "saltar para um novo menu", então as linhas de texto transmitidas para o cliente serão modificadas para representarem o menu escolhido.

Este método tem a desvantagem do cliente precisar de fazer solicitações constantes ao servidor para apresentação dos dados, mas tem a vantagem de qualquer alteração na estrutura ou conteúdo do menu ser imediatamente refletida na aplicação SCIS cliente bem como uma menor necessidade de processamento por parte desta aplicação, o que é desejável em termos de autonomia do dispositivo móvel. O fluxograma das comunicações com o cliente móvel pode ser observado na Figura 4.18.

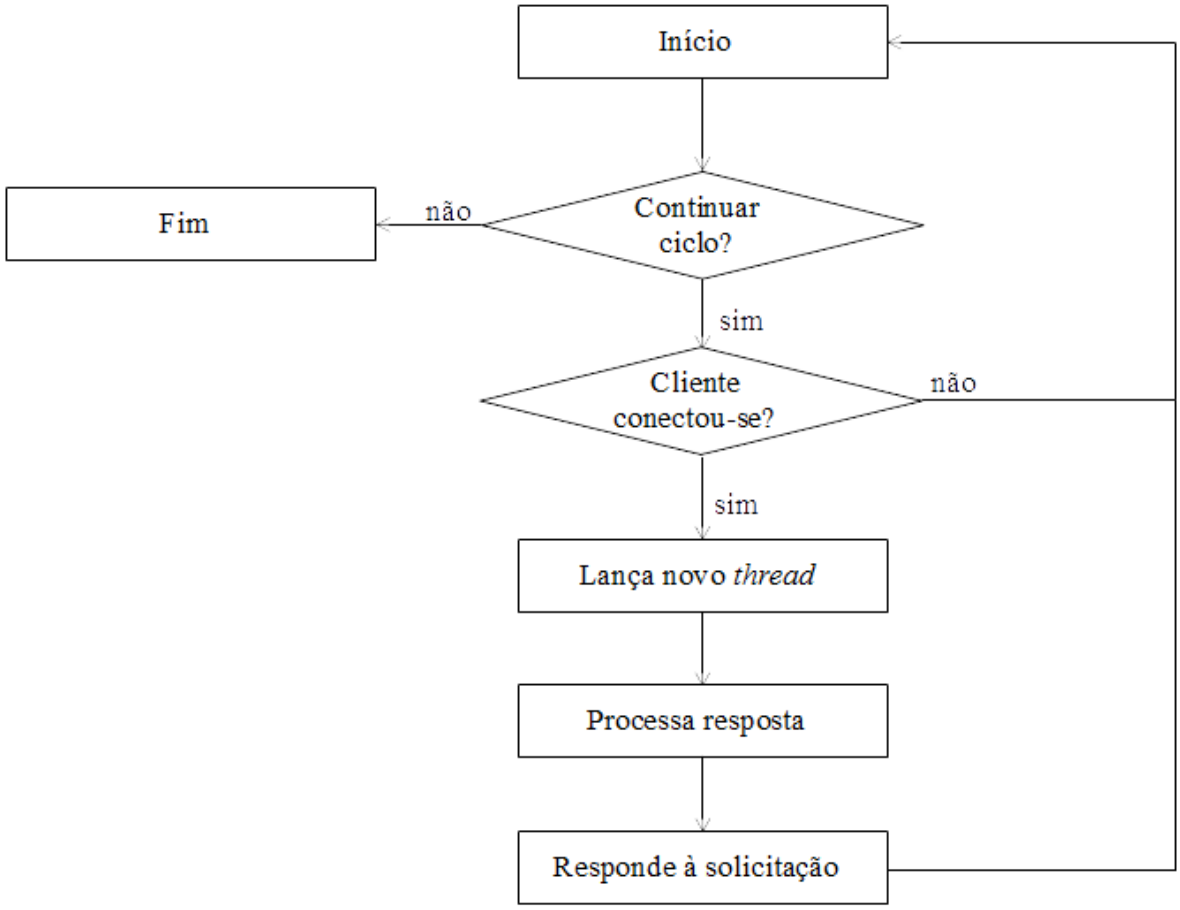

Figura 4.18 - Fluxograma das comunicações com o cliente Android

5. Aplicação SCIS cliente

A aplicação SCIS cliente é o componente do sistema que permite ao utilizador consultar os valores do controlador DeltaSol M através do ecrã do dispositivo móvel. Esta aplicação foi desenvolvida em linguagem Java usando o Android SDK disponibilizado pela Google, este SDK inclui o IDE Eclipse e um conjunto de bibliotecas específicas para o desenvolvimento de aplicações para o sistema operativo Android, neste capítulo será feita uma descrição do interface gráfico de cada *activity* (entenda-se por *activity* uma vista única da aplicação, ver seção 1.5.4) e serão abordados os pontos mais relevantes do desenvolvimento da aplicação.

5.1. Comunicação com o servidor

A aplicação SCIS cliente faz solicitações ao servidor através do protocolo HTTP, passando os parâmetros da solicitação através de um comando HTTP GET, este comando é processado pelo servidor que responde com os dados solicitados ao cliente no formato JSON como referido na seção 4.4.

A classe "*DefaultHttpClient*" incluída no pacote *"org.apache.http.impl.client"*, foi a escolhida para a solicitação de dados HTTP ao servidor através do protocolo HTTP, no entanto os métodos desta classe são síncronos, isto significa que a *activity* de onde é feita a solicitação fica bloqueada até receber uma resposta do servidor ou até se esgotar o tempo máximo para a operação *(Timeout).* Este comportamento de bloqueio causa uma má experiência na utilização da aplicação e por esse motivo as solicitações HTTP devem ser feitas de forma assíncrona.

Como referido na subseção 1.5.6, o Android SDK disponibiliza a classe *"AsyncTask"* que permite fazer qualquer operação de forma assíncrona, executando o método "*onPostExecute()"* quando a operação é concluída. Estendendo esta classe foi criada a classe "*AsyncHTTPGet*", representada na Figura 5.1, que usa uma instância de "*DefaultHttpClient*" para fazer solicitações de forma assíncrona ao servidor. Desta forma, as instâncias desta nova classe apenas têm que executar o método "*execute(httpaddress)"*, para receber a resposta no *callback* "*onPostExecute(result)"*. Em cada solicitação feita pelas instâncias desta classe é também atualizado o estado da conexão com o servidor, e conforme a solicitação tenha resposta ou não, a conexão é marcada como ativa ou inativa, simultaneamente é apresentada ao utilizador uma mensagem do tipo *Toast* (ver seção 1.5.7) a informar qualquer alteração ao estado da conexão.

Como as solicitações ao servidor são feitas em intervalos regulares e de forma transversal a todas as *activities* da aplicação SCIS cliente, foi programada a superclasse "*BaseActivity*" com a classe interna "*AsyncHTTPGet*" que serve de base a todas as *activities.* O diagrama destas classes está representado na Figura 5.1.

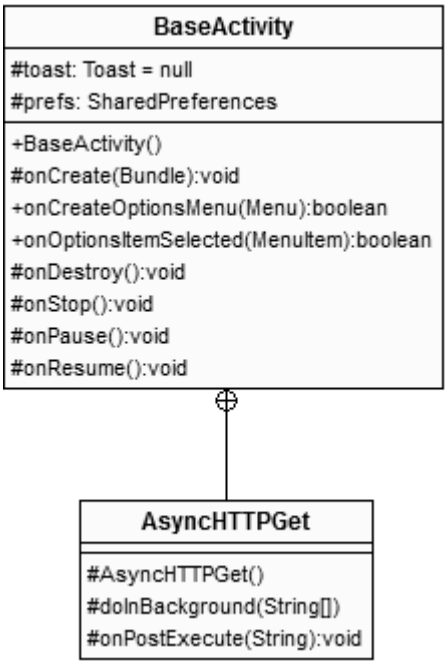

Figura 5.1 - Diagrama das classes "*BaseActivity*" e "*AsyncHTTPGet*"

5.2. Menu Principal

O menu principal da aplicação, Figura 5.2, é uma *activity* que permite ao utilizador gerir e conhecer o estado da conexão com o servidor e serve de ponto de partida para todas as outras *activities*.

O primeiro controlo a contar de cima na *activity* do menu principal representada na Figura 5.2, é uma caixa de texto (*TextView*) que indica o estado da conexão com o servidor, esta caixa pode assumir os seguintes textos:

- "***Offline***" - Aplicação sem conexão ao servidor e sem ordem do utilizador para conectar, Figura 5.2 estado A;
- "***Waiting***" - A aplicação recebeu ordem do utilizador para conectar ao servidor e aguarda que a conexão seja concluída, Figura 5.2 estado B;
- "***Connected***" - A aplicação está conectada ao servidor, Figura 5.2 estado C.

O segundo controlo, é um botão que permite ao utilizador iniciar e parar a conexão com o servidor, este botão pode assumir os seguintes estados:

- "***Connect***" - Neste estado um toque do utilizador inicia a conexão, Figura 5.2 estado A;
- "***Disconnect***" - Neste estado um toque do utilizador para a conexão, Figura 5.2 estado B.

Figura 5.2 - Vários estados do menu principal

Sempre que uma conexão está ativa ficam disponíveis os restantes 3 botões do menu, estes são ligações para as restantes *activities* da aplicação:

- "***List online data"*** - Abre a *activity* com a listagem dos dados de transmissão contínua;
- "***Device menu"*** - Abre a *activity* de acesso ao menu do controlador e onde é possível ver e alterar os seus parâmetros;
- "***Device Sinoptic"*** - Abre a *activity* com a representação gráfica do sistema, ou sinóptico.

Esta *activity* está associada à classe "*MainMenu*", representada no diagrama da Figura 5.3, e o seu *layout* está definido no ficheiro "*activity_main_menu.xml*". A classe "*MainMenu*" é uma subclasse de "*BaseActivity*" e a sua classe interna "*getDevicesFromServer*" é por sua vez uma subclasse de "*AsyncHTTPGet*". As solicitações ao servidor são feitas através de uma instância da classe "*getDevicesFromServer*" com o nome "*getDevices*", estas solicitações são feitas em intervalos regulares pelo método "*pullServer*". Sempre que é recebida uma resposta do servidor no método "*onPostExecute*" de "*getDevices*", os dados são processados pelo método "*ParseJsonToVBusDevices*" que desserializa a resposta numa coleção de objetos que representam dispositivos VBus usados para informar as outras *activities* dos dispositivos

conectados. A resposta a estas solicitações também permite conhecer o estado da conexão com o servidor para atualização dos controlos através do método "*syncButtons*".

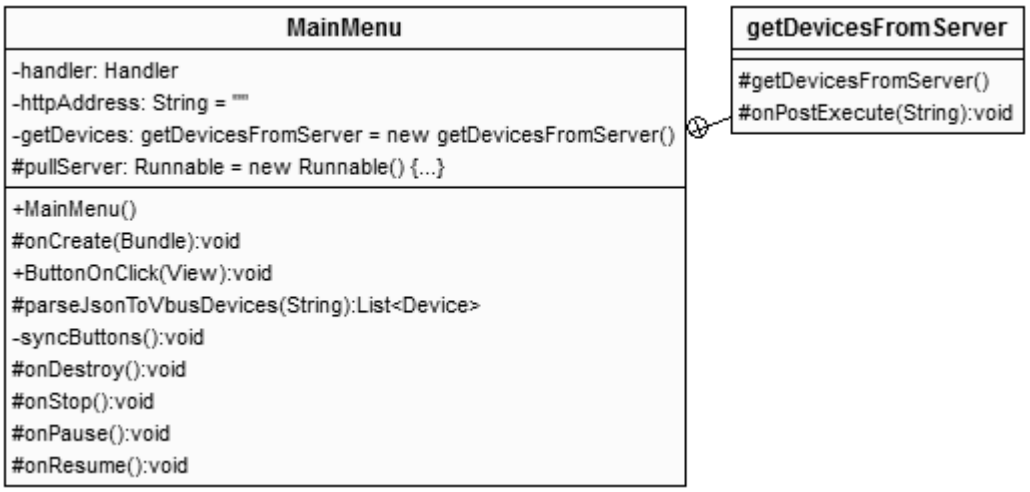

Figura 5.3 - Diagrama da classe "*MainMenu*"

5.3. Lista de dados de visualização remota

A *activity* da Figura 5.4 disponibiliza ao utilizador uma lista com dados transmitidos continuamente pelo controlador DeltaSol M para o dispositivo de visualização remota RESOL DFA (ver subseção 3.5.1), em baixo de cada valor existe a indicação da hora a que foi recebido, desta forma, na consulta da lista, têm-se uma noção imediata do estado e velocidade da conexão, pois estes valores são atualizados a cada resposta do servidor. O nome de cada variável é também transmitido pelo servidor, este por sua vez importou-os do ficheiro XML com as definições fornecidas pela RESOL (ver subseção 3.6.1), qualquer atualização ao nome, endereço ou unidade dos parâmetros no servidor será imediatamente refletida na lista da Figura 5.4. Também se observa que, imediatamente abaixo do nome de cada variável, está disponível um texto que a identifica de forma única, esta identificação é composta pelo endereço do dispositivo de origem em codificação hexadecimal, seguido do carácter "&", seguido do valor decimal do *offset* do parâmetro no pacote transmitido pelo controlador (ver subseção 3.5.1). Apesar de não ser o caso deste projeto, é possível que vários dispositivos possam transmitir em simultâneo este tipo de dados no barramento VBus, nestes casos, esta identificação permitirá ao utilizador distinguir o dispositivo de origem de cada variável.

Esta *activity* está associada à classe "*ListOnlineData*" e o seu *layout* está definido no ficheiro "*activity_menu.xml*". A classe "*ListOnlineData*", representada no diagrama da Figura 5.5, é

uma subclasse de "*BaseActivity*" e a sua classe interna "*getListFromServer*" é por sua vez uma subclasse de "*AsyncHTTPGet*".

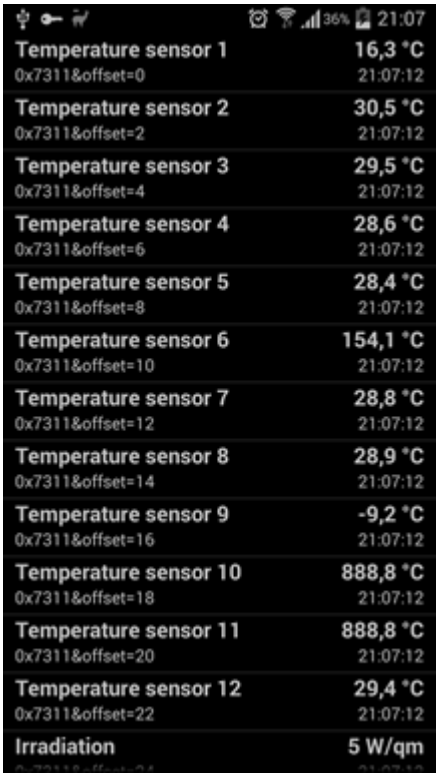

Figura 5.4 - Lista de dados para visualização remota

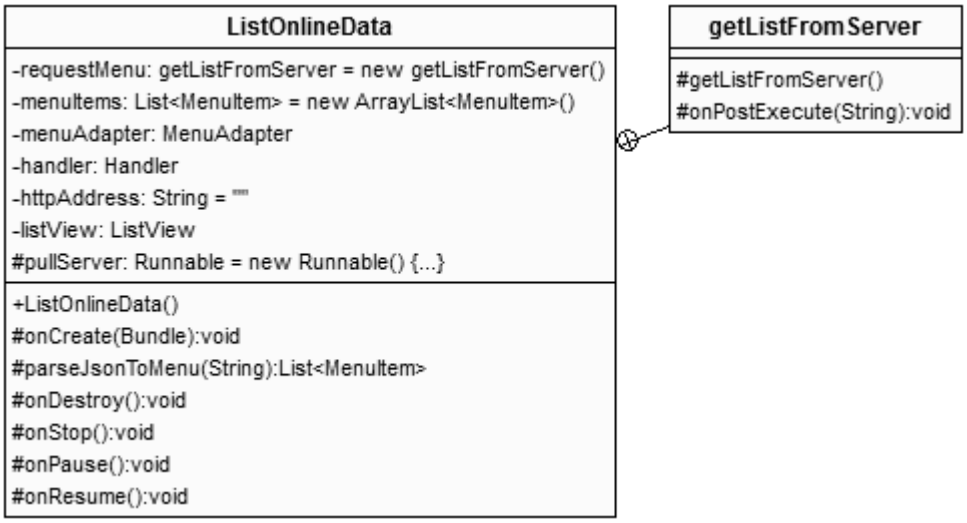

Figura 5.5 - Diagrama da classe "*ListOnlineData*" e respetivas classes internas

As solicitações ao servidor são feitas através de uma instância da classe "*getListFromServer*" com o nome "*requestMenu*", estas solicitações são feitas em intervalos regulares pelo método "*pullServer*". Sempre que é recebida uma resposta do servidor no método "*onPostExecute*" de

"*requestMenu*", os dados são processados pelo método "*ParseJsonToMenu*" que desserializa a resposta na coleção de objetos "*menuItems*", os objetos desta coleção são instâncias da classe "*menuItem*", representada na Figura 5.6. Cada instância da classe "*menuItem*" representa uma linha da lista e contém as informações que servirão para o "*menuAdapter*" a povoar, sendo o "*menuAdapter*" o "*Custom adapter*" da lista definido na classe "*MenuAdapter*" (ver subseção 1.5.5).

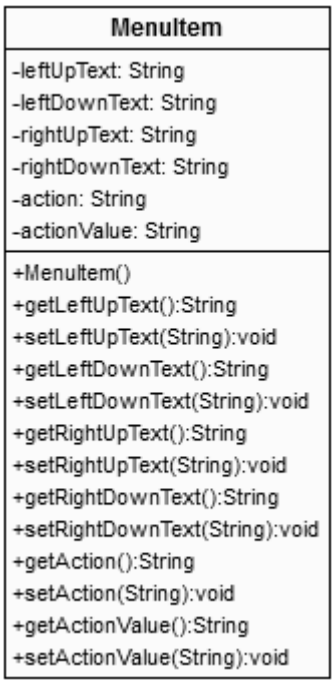

Figura 5.6 - Diagrama da classe "*MenuItem*"

5.4. Menu de parâmetros do controlador

A *activity* da Figura 5.7 replica o menu disponível no *display* físico do controlador e desta forma disponibiliza remotamente todas as funções e parametrizações acessíveis localmente através deste. Na Figura 5.8 está representado o sub-menu "*Solar adjustment values"*, cada linha de um sub-menu contém as seguintes informações, por parâmetro:

- **Canto superior esquerdo** - Nome do parâmetro
- **Canto superior direito** - Valor e unidade
- **Canto inferior esquerdo** - Endereço do parâmetro em codificação hexadecimal
- **Canto inferior direito** - Data e hora da última leitura

Esta *activity* está associada à classe "*ListMenu*", representada na Figura 5.10 - Diagrama da classe "*ListMenu"* e respetivas classes internas, e o seu *layout* está definido no ficheiro

"*activity_menu.xml*". O método de apresentação dos dados nas linhas do menu é semelhante ao usado na *activity* da subseção 5.3.

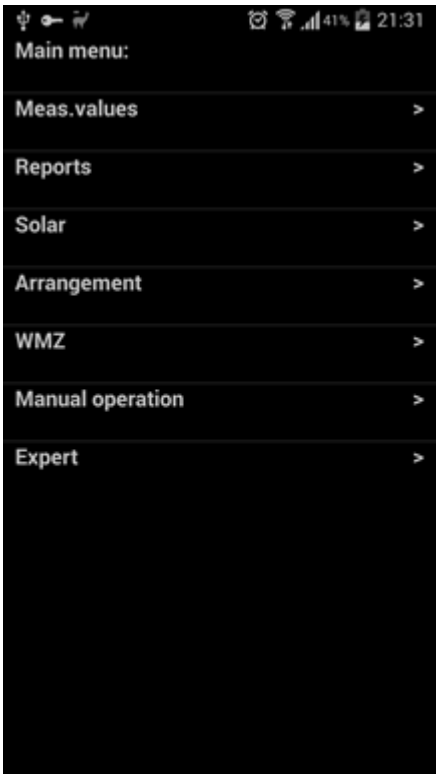

Figura 5.7 - *Activity* com o menu do controlador

Figura 5.8 - Sub-menu "*Solar adjustment values*"

No entanto, na classe "*ListMenu*", a classe interna responsável pela solicitação dos dados que povoam o menu chama-se "*getMenuFromServer*" e na subseção 5.3 a classe com funções equivalentes chama-se "*getListFromServer*". No diagrama da classe "*MenuItem*", representada na Figura 5.6, podemos observar que cada item, ou linha, do menu tem uma propriedade "*action*" que define o tipo de ação associado à seleção do item, estas ações são transmitidas pelo servidor de acordo com o explicado na seção 4.3. No método "*onCreate*" que é executado na criação da *activity*, é associado um "*onClickListener*" a cada item do menu, este "*Event Listener*" (ver subseção 1.5.4) executa o método "*MenuAction*" que recebe como argumento o objeto "*menuItem*" associado à linha selecionada e processa a sua ação, que pode ser de três tipos:

a) **"*menujump*"** - Solicita um novo submenu ao servidor, através da alteração do endereço "*httpAddress*" que é usado para as solicitações feitas em intervalos regulares pelo método "*pullServer*"

b) **"*request*"** - Solicita ao servidor os dados necessários para apresentação dos diálogos da Figura 5.9. A solicitação é feita numa instância da classe "*ParameterDataHttpRequest*" e o endereço desta solicitação é previamente definido no servidor e transmitido na propriedade "*actionValue*" da instância "*menuItem*" associada à linha do menu.

c) **"*readonly*"** - Sempre que a ação associada à linha é a informação "*readonly*" o método "*MenuAction*" não faz qualquer solicitação e apresenta uma mensagem do tipo *"Toast"* com o texto "*read only"* para informação do utilizador.

Como explicado em b), após a seleção de um parâmetro com acesso de escrita, são recebidas as definições que permitem apresentar um dos diálogos de introdução de dados representados na Figura 5.9, estes permitem visualizar e introduzir o valor do parâmetro devidamente formado de acordo com o tipo (ver Tabela 4.2) de *"menuType"* definido para o parâmetro (no ficheiro XML referido na subseção 3.6.2). Os diálogos de introdução de dados são criados numa instância da classe "*InputDialog*" e podem ser de quatro tipos:

a) **Diálogo para entrada de dia da semana e hora** - Representado na imagem central da Figura 5.9, é usado para o tipo "*WeekTime*". O *layout* deste diálogo está definido no ficheiro "*input_time_dialog.xml*"

b) **Diálogo para entrada de hora** - Semelhante ao anterior mas na criação deste diálogo a caixa de seleção do dia da semana é escondida, é usado para o tipo "*Time*".

c) **Diálogo para entrada de valor decimal** - Representado na imagem mais à direita da Figura 5.9, usado para o tipo "*Number*". O *layout* deste diálogo está definido no ficheiro "*input_dialog.xml*"

d) **Diálogo com várias opções para seleção única** - Representado na imagem mais à esquerda na Figura 5.9, usado para todos os tipos não referidos nas alíneas anteriores, usa um *layout* pré-definido para itens de opção única.

Sempre que o utilizador introduz um novo valor para o parâmetro usando um dos diálogos de introdução de dados, uma instância da classe "*DialogHttpRequest*" pede ao servidor essa alteração usando a solicitação HTTP GET com o parâmetro "*setparameter"* (ver seção 4.4). Esse valor é posteriormente enviado pelo servidor para

o controlador DeltaSol M, após a escrita efetiva o submenu será atualizado com o novo valor para o parâmetro.

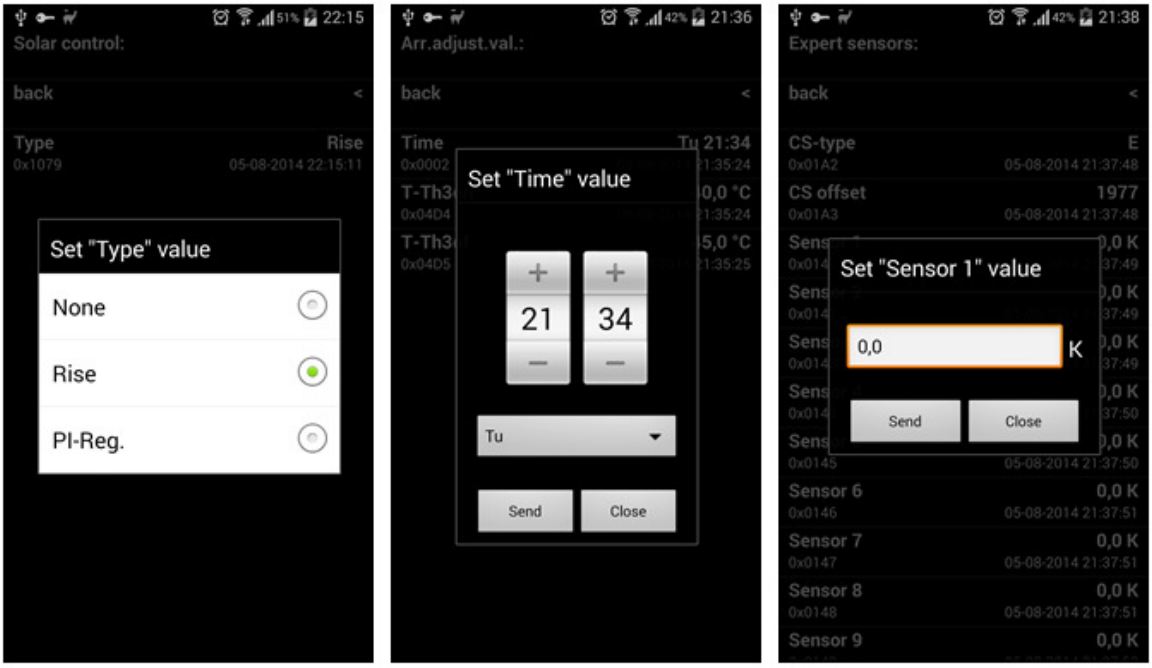

Figura 5.9 - Exemplos de diálogos de entrada de dados

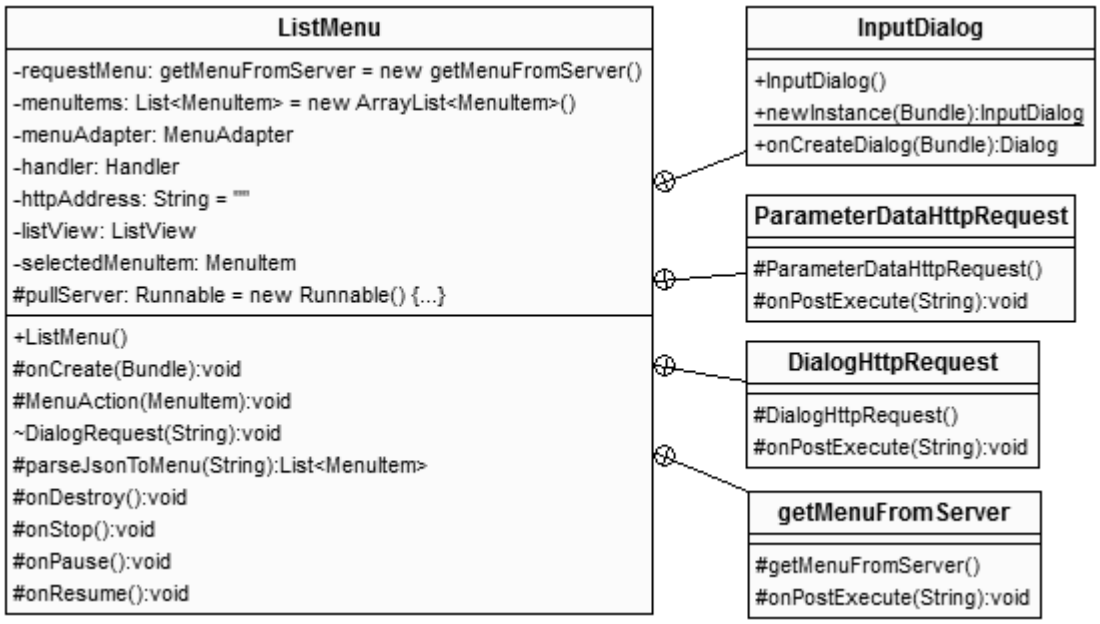

Figura 5.10 - Diagrama da classe "*ListMenu*" e respetivas classes internas

5.5. Sinóptico do sistema

Os valores representados na *activity* do sinóptico do sistema são obtidos da mesma forma que os valores de visualização remota disponíveis na respetiva lista (ver subseção 5.3), no entanto, disponibilizados de forma gráfica. O controlador DeltaSol M suporta sistemas com 17 configurações diferentes [8]. Para efeitos de implementação e tendo em conta o âmbito genérico deste projeto optou-se por representar na aplicação o sistema mais simples, a Figura 5.11 mostra a *activity* com o sinóptico do sistema em dois estados diferentes, de forma a ser percetível ao leitor a atualização dos valores nos objetos e a alteração da cor de uma eletrobomba de recirculação em função do seu estado.

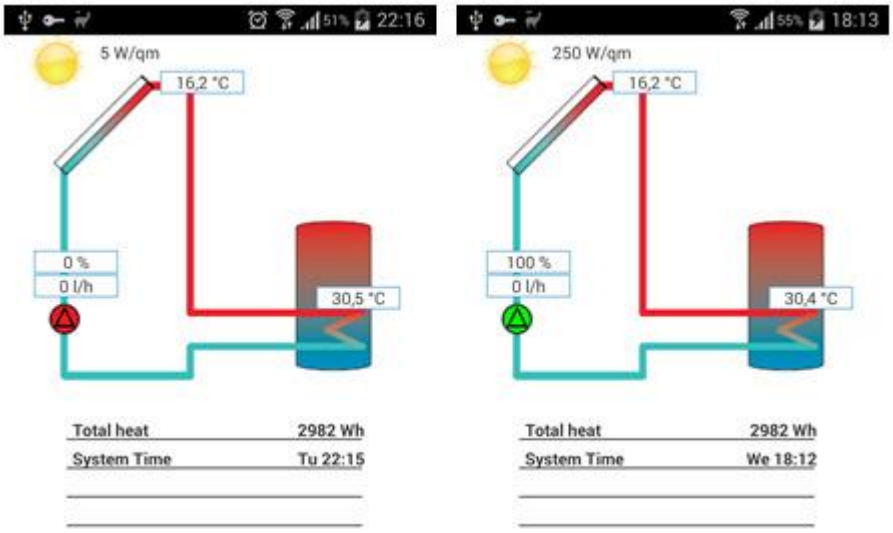

Figura 5.11 - *Activity* com o sinóptico do sistema

Esta *activity* está associada à classe "*Sinop*" representada no diagrama da Figura 5.12, e o seu *layout* está definido no ficheiro "*sinop.xml*". Na criação deste *layout* foi usada a propriedade "*Tag*" de cada objeto gráfico para definir qual o parâmetro associado ao mesmo. Essa associação foi feita usando um texto com o seguinte formato:

"device=**[endereço do dispositivo VBus]**&offset=**[Offset da variável VBus]**"

O exemplo seguinte associa a um objeto gráfico a variável com o valor da radiação solar do dispositivo DeltaSol M:

"device=0x7311&offset=24"

Á semelhança das outras *activities* a solicitação de dados ao servidor é feita em intervalos regulares pelo método "*pullServer*" de uma instância da classe "*getDataFromServer*", sempre que os dados solicitados são recebidos é executado o método "*onPostExecute*" desta classe. Nela é efetuado todo o processamento que permite atualizar os objetos gráficos, em função dos dados recebidos do servidor.

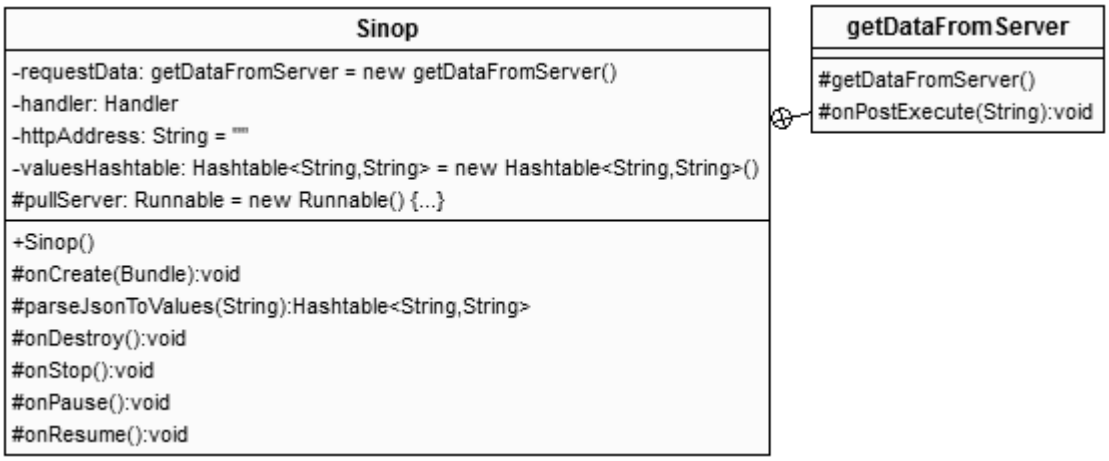

Figura 5.12 - Diagrama da classe "Sinop"

5.6. Menu de opções

Em todas as *activities* da aplicação SCIS cliente está disponível o menu representado na parte inferior da Figura 5.13. Este menu pode ser acedido a partir do botão "Menu" do dispositivo móvel e tem dois botões com as seguintes funções:

- **Botão "*Menu*"** - Permite voltar ao menu principal da aplicação
- **Botão "*Settings*"** - Permite aceder à lista de configurações da aplicação

As funções relativas à apresentação deste menu estão definidas na classe "*BaseActivity*" representada no diagrama da Figura 5.1, como todas as *activities* da aplicação estendem esta classe e herdam o seu menu. As funções usadas para a criação deste menu, permitiram defini-lo completamente através do ficheiro "/menu/main.xml". As ações a executar após a seleção de um dos seus itens são processadas no método "*onOptionsItemSelected*" da classe "*BaseActivity*".

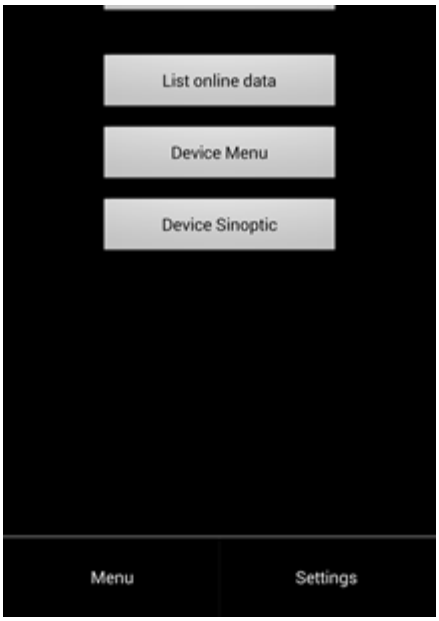

Figura 5.13 - Menu de opção da aplicação SCIS cliente

5.7. Lista de configurações da aplicação

A lista de configurações da aplicação permite guardar valores definidos pelo utilizador de forma permanente. Estas configurações são: o endereço do servidor e o tempo entre as solicitações, como representado na lista, à esquerda da Figura 5.14, após a escolha da opção a configurar é apresentada uma caixa de diálogo onde o utilizador pode introduzir um novo valor para a configuração escolhida, ver Figura 5.14 à direita.

Para o armazenamento das configurações da aplicação foi usada a classe "*SharedPreferences*" disponibilizada pelo Android SDK. Esta classe permite armazenar e solicitar pares "chave-valor" de tipos de variáveis primitivos como: *boleans*, *floats*, *ints*, *longs* e *strings*. Os valores são persistentes mesmo após o encerramento da aplicação [15]. A *activity* que permite a consulta e edição desta lista está associada à classe "*Settings*", que estende a classe "*PreferenceActivity"* disponibilizada pelo Android SDK, e que já implementa todas as funcionalidades de apresentação dos dados na lista.

As propriedades dos parâmetros de configuração estão definidas no ficheiro "/xml/settings.xml". Este ficheiro é usado como argumento do método "*addPreferencesFromResource(R.xml.settings)"* que povoa a lista da Figura 5.14 (imagem da esquerda) com os dados de configuração da aplicação.

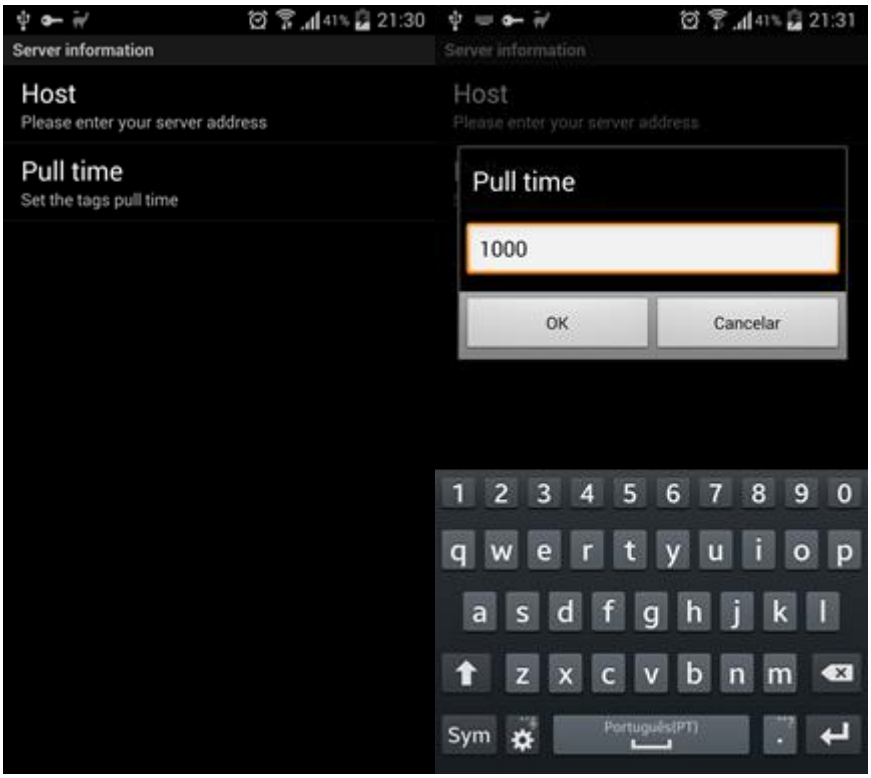

Figura 5.14 - Configurações da aplicação

6. Conclusão

Como referido na subseção 3.3.3, os ensaios ao sistema realizaram-se no 3º e no 6º e último estágio do seu desenvolvimento, este capítulo relata os resultados desses ensaios, as conclusões que deles resultaram e também as várias funcionalidades que seriam úteis ao sistema, mas que não foram implementadas por estarem fora do âmbito deste projeto.

6.1. Ensaio da aplicação SCIS servidor

No 3º estágio do desenvolvimento do sistema foi feito um ensaio com o objetivo de verificar o correto funcionamento da comunicação entre o controlador DeltaSol M e a aplicação SCIS servidor através do protocolo VBus, este ensaio realizou-se no laboratório de Engenharia Mecânica, local onde está instalado o controlador (ver Figura 2.1).
Este ensaio consistiu em vários testes de comparação entre os valores disponíveis no *display* do controlador e os valores disponíveis na aplicação SCIS servidor, para essa comparação foram utilizadas as janelas da aplicação criadas especificamente para esse propósito.
Os testes à implementação da versão 1.0 do protocolo VBus, foram feitos através da janela de consulta dos dados para visualização remota, representada na Figura 4.4, nela foi possível verificar que:

- ✓ Os valores apresentados na aplicação refletem os valores das leituras instantâneas disponíveis no *display* do controlador.
- ✓ Quando se alteram as temperaturas lidas pelo controlador através do simulador de sondas (ver Figura 6.1), os valores apresentados pela aplicação refletiram as alterações num tempo máximo de 2 segundos.

O teste à implementação da versão 2.0 do protocolo VBus foi feito através da janela de navegação do menu, representada na Figura 4.3, nela foi possível verificar que:

- ✓ A estrutura do menu da aplicação, e os valores dos parâmetros a ele associados, refletem os disponíveis no menu do controlador.
- ✓ Quando se modificam valores de parâmetros através do menu do controlador, os valores do menu da aplicação refletem essas modificações.
- ✓ Quando se modificam valores de parâmetros através do menu da aplicação, os valores do menu do controlador refletem essas modificações.

Tendo em conta que os resultados do ensaio foram os esperados, foi possível concluir que a leitura e escrita de valores no controlador através do protocolo VBus funciona corretamente.

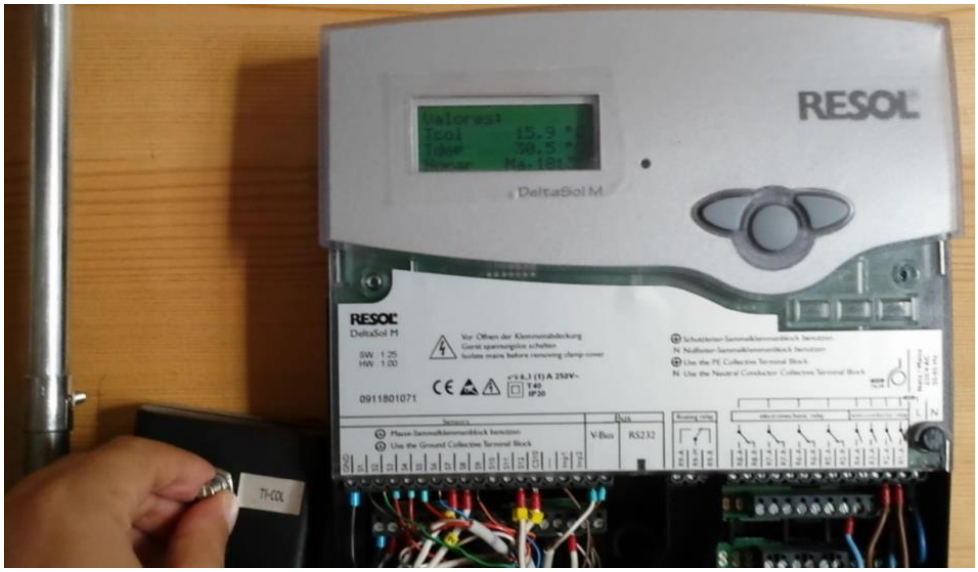

Figura 6.1 - Fotografia tirada durante o ensaio do 3º estágio

6.2. Ensaio da aplicação SCIS cliente

No 6º estágio do desenvolvimento do sistema foi feito um ensaio com o objetivo de verificar o correto funcionamento da comunicação entre a aplicação SCIS cliente e o controlador DeltaSol M através da aplicação SCIS servidor, este ensaio realizou-se em dois cenários distintos:

a) No laboratório de Engenharia Mecânica, local onde está instalado o controlador (ver Figura 2.1). A conexão entre o SCIS cliente e o SCIS servidor foi feita através da rede *Wi-Fi* "*eduram*", disponível no *campus* da Penha.
   Neste cenário, o ensaio consistiu na comparação entre os valores disponíveis no *display* do controlador e os valores disponíveis nas *activities* da aplicação SCIS cliente.

b) Fora do *campus* da Penha. A conexão entre o SCIS cliente e o SCIS servidor foi feita através da internet, utilizando a ligação VPN da UALG, devidamente configurada no dispositivo móvel.
   Neste cenário, o *display* do controlador não está acessível para a comparação dos valores, mas, através do "ambiente de trabalho remoto" disponibilizado pelo sistema

operativo Windows, foi possível aceder remotamente ao computador e fazer a comparação entre os valores disponíveis nas aplicações SCIS servidor e SCIS cliente.

Assim, relativamente às *activities* "Lista de dados para visualização remota" e "Sinóptico do sistema" (referidas nas seções 5.3 e 5.4 respetivamente), foi possível verificar que:

- ✓ Os valores apresentados nas *activities* refletem os valores das leituras instantâneas disponíveis no *display* do controlador (cenário a)), e na aplicação SCIS servidor (cenário b)).
- ✓ Quando se alteram as temperaturas lidas pelo controlador através do simulador de sondas (ver Figura 6.1), os valores disponíveis nas *activities* refletem os valores apresentados no *display* do controlador (cenário a)), e na aplicação SCIS servidor (cenário b)).

No caso da *activity* "Menu de parâmetros do controlador" (referida na seção 5.4) foi possível verificar que:

- ✓ A estrutura do menu da *activity*, e os valores dos parâmetros a ele associados, refletem os disponíveis nos menus do controlador (cenário a)), e da aplicação SCIS servidor (cenário b)).
- ✓ Quando se modificam valores de parâmetros através do menu do controlador (cenário a)) ou através do menu da aplicação SCIS servidor (cenário b)), os valores disponíveis no menu da *activity* refletem essas modificações.
- ✓ Quando se modificam valores de parâmetros através do menu da *activity*, essas modificações são refletidas no menu do controlador (cenário a)) e no menu da aplicação SCIS servidor (cenário b)).

6.3. Conclusões

Uma das funcionalidades mais relevantes da aplicação desenvolvida, é o acesso ao menu de parâmetros do controlador (referido na seção 5.4), que possibilita uma supervisão completa do mesmo, inclusive a consulta de condições de alarme no sistema solar térmico e no próprio controlador, função essencial numa aplicação de supervisão. A disponibilização deste menu deu à aplicação um potencial não previsto inicialmente, e que superou as espectativas do autor do projeto. Curiosamente, esta disponibilização só foi possível devido a uma lacuna identificada nas especificações do protocolo VBus [9], estas ao remeterem para o *software*

"RESOL *Service Center*" para a consulta dos endereços dos vários parâmetros de configuração do controlador, não identificam algumas das suas propriedades mais importantes, por exemplo, o seu fator multiplicativo. Este imprevisto implicou uma análise de engenharia inversa aos ficheiros XML referidos nas subseções 3.6.1 e 3.6.2, que consumiu uma grande parte considerável do tempo de desenvolvimento devido à falta de documentação relativamente ao significado dos seus elementos XML. O tempo despendido foi útil, pois para além da disponibilização do menu na aplicação cliente, a interpretação destes ficheiros também possibilitou a importação automática das definições de todos os controladores neles definidos, sem necessidade de qualquer introdução manual de dados na aplicação SCIS servidor por parte do utilizador.

Apesar do objetivo deste projeto focar a supervisão do controlador DeltaSol M, todas as funções das aplicações SCIS servidor e cliente, foram criadas de forma a serem compatíveis com outros controladores que suportem o protocolo VBus, no entanto, por estar fora do âmbito deste projeto e por falta de acesso a outros equipamentos, este cenário não foi testado.

Os ensaios referidos nas seções 6.1 e 6.2, permitem concluir que foram alcançados os objetivos propostos no início do projeto. A aplicação de supervisão desenvolvida para dispositivo móvel Android permite visualizar todos os valores do controlador DeltaSol M em tempo real, alterar os seus parâmetros de configuração e visualizar o sistema solar térmico na forma gráfica através de um sinóptico.

6.4. Trabalho futuro

Durante o desenvolvimento, foram identificadas várias funcionalidades úteis à aplicação cliente, mas que não foram implementadas por estarem fora do âmbito deste projeto. Estas funcionalidades são:

- ✓ Notificar o utilizador de qualquer condição de alarme no controlador, para isso devem ser solicitadas regularmente ao controlador as variáveis que contém as informações relativas às suas anomalias.
- ✓ Permitir que o utilizador escolha quais as anomalias que vão gerar notificações.
- ✓ Permitir a visualização de gráficos com a evolução das principais grandezas ao longo do tempo, estes dados ficariam armazenados numa base de dados a desenvolver na aplicação servidor, e que seria consultada pela aplicação cliente.
- ✓ Criar sinópticos que representem vários sistemas, e permitir que o utilizador escolha qual deve ser usado.

Bibliografia

[1] Canalys, *Press release 2014/011*, Canalys, 2014.

[2] Android Open Source Project, "Licenses," [Online]. Available: https://source.android.com/source/licenses.html. [Acedido em 03 2014].

[3] Android developers, "Keeping Your App Responsive," [Online]. Available: http://developer.android.com/reference/android/os/AsyncTask.html. [Acedido em 05 2014].

[4] Android developers, "AsyncTask," Android developers, [Online]. Available: http://developer.android.com/reference/android/os/AsyncTask.html. [Acedido em 05 2014].

[5] National Instruments, "Smartphones e Tablets para Medição e Controle," 12 2013. [Online]. Available: http://www.ni.com/newsletter/51387/pt/. [Acedido em 05 2014].

[6] Schneider Electric, "Vijeo design'Air brochure," 2013. [Online]. Available: http://download.schneider-electric.com/files?L=pt&p=&p_docId=&p_docId=&p_Reference=DIA5ED1130402EN_998-1193082_GMA-GB&p_EnDocType=Brochure&p_File_Id=61800675&p_File_Name=DIA5ED1130402EN_998-1193082_GMA-GB.pdf [Acedido em 05 2014].

[7] M. Nicolae, "Embedding Android devices in automation systems," em *International Symposium for Design and Technology in Electronic Packaging*, Galaţi, 2013.

[8] RESOL - Elektronische Regelungen GmbH, RESOL DeltaSol® M, Hattingen: RESOL - Elektronische Regelungen GmbH, 2007.

[9] RESOL - Elektronische Regelungen GmbH, VBus Protocol Specification, RESOL - Elektronische Regelungen GmbH, 2011.

[10] RESOL - Elektronische Regelungen GmbH, "RESOL Software," [Online]. Available: http://www.resol.de/index/software/sprache/en. [Acedido em 04 2014].

[11] RESOL - Elektronische Regelungen GmbH, RESOL VBUS Protokollspezifikation, Hattingen: RESOL - Elektronische Regelungen GmbH, 2003.

[12] World Wide Web Consortium (W3C), “Hypertext Transfer Protocol -- HTTP/1.1,” 06 1999. [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616.html. [Acedido em 08 2014].

[13] http://json.org, “Introdução ao JSON,” 2014. [Online]. Available: http://json.org/json-pt.html. [Acedido em 04 2014].

[14] World Wide Web Consortium (W3C), “Extensible Markup Language (XML) 1.0 (Fifth Edition),” 26 11 2008. [Online]. Available: http://www.w3.org/TR/2008/REC-xml-20081126/. [Acedido em 08 2014].

[15] Android Developers, “Storage Options,” [Online]. Available: http://developer.android.com/guide/topics/data/data-storage.html.[Acedido em 05 2014].