Faculdade de Engenharia da Universidade do Porto

SCADA em Android

Rafael Pisco

V1

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

Orientador: Mário Jorge Rodrigues de Sousa (Professor)

29-06-2015

U.PORTO
FEUP

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES | **2014/2015**

A Dissertação intitulada

"SCADA em Androide"

foi aprovada em provas realizadas em 14-07-2015

o júri

Presidente Professor Doutor Hélio Mendes de Sousa Mendonça
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores
da Faculdade de Engenharia da Universidade do Porto

Professor Doutor Paulo Bacelar Reis Pedreiras
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática
da Universidade de Aveiro

Professor Doutor Mário Jorge Rodrigues de Sousa
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores
da Faculdade de Engenharia da Universidade do Porto

O autor declara que o presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, paragrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

Rafael Gonçalves Pisco

Autor - Rafael Gonçalves Paralta da Silva Pisco

Faculdade de Engenharia da Universidade do Porto

Resumo

Atualmente com a grande expansão do mercado dos *smartphones*/*tab*lets quase todas as pessoas possuem um. Isto permite que os *smartphones*/*tablets* sejam usados para realizar muitas tarefas que antigamente eram apenas realizadas por computadores ou dispositivos de porte médio/grande.

Os sistemas SCADA, que permitem a monitorização e o controlo dos processos de um sistema, têm vindo a ser cada vez mais usados e melhorados. Aliado a este aumento do uso dos sistemas SCADA, têm vindo a ser desenvolvidos cada vez mais equipamentos para todas as componentes destes sistemas.

Além da evolução do mercado dos *smartphones*/*tablets*, também as comunicações usando o protocolo 802.11 têm evoluído.

Estas evoluções permitiram o uso dos *smartphones*/*tablets* para controlar equipamentos, por exemplo através do protocolo *Modbus TCP*/*IP*. Isto abriu a possibilidade de usar os dispositivos *mobile* como *Master Terminal Units* de um sistema SCADA com interfaces Homem/Máquina incorporadas.

Substituindo a compra das MTUs e das interfaces Homem/Máquina por tablets/smartphones iria baixar substancialmente o custo de aquisição, bem como o custo de manutenção.

Iria também permitir ao cliente ter uma maior escolha na hora de efetuar a compra, pois existe uma quantidade enorme de dispositivos *Android* com características diferentes. Isto iria permitir ao cliente escolher um dispositivo completamente adaptado às suas necessidades, em vez de ter de escolher equipamentos de uma lista pré-definida.

No entanto, para isto tudo ser possível, é necessário adaptar as MTUs e interfaces, dos sistemas SCADA existentes para os dispositivos *Android*.

Esta dissertação assenta então sobre esta necessidade. A dissertação visa a criação de uma aplicação, que funcione como as MTUs e interfaces dos sistemas SCADA existentes. Esta aplicação deve ter uma performance e escalabilidade semelhantes às encontradas nos sistemas SCADA atuais.

Abstract

Nowadays with the big expansion of the smartphones'/tablets' market almost every person was one. This allows that smartphones/tablets to be used to accomplish many tasks that were formerly accomplished by computers or medium/large size devices.

The SCADA systems, that allow the supervision and the control of the system's processes, has been more used and improved. Allied with the increase of the use of the SCADA systems, more equipment had been developed for every component of the SCADA systems.

Beside the big expansion of the smartphones'/tablets' market, communications using the protocol 802.11 had evolved too.

These evolutions allowed these devices to control equipment using for example the Modbus TCP/IP protocol. This had opened the possibility to use the mobile devices as Master Terminal Units, present in SCADA system, with an incorporated Man/Machine interface.

Replacing the bought of the MTUs and the interfaces with smartphones/tablets would lower significantly the cost of acquisition and the cost of maintenance.

This would give a bigger pool of choices to the client because there is a larger quantity of Android devices with different characteristics. This large choice would allow the client to choose a device fully adapted to is necessities instead of choosing from a predefined list of devices.

Nonetheless for this to be possible it's necessary to adapt the MTUs and the interfaces of the existent SCADA systems to the Android devices.

This dissertation is based on that necessity. This dissertation is aimed to create an application that work like the MTUs and interfaces in the existent SCADA systems. This application should have the performance and scalability similar to that of the actual SCADA systems.

Agradecimentos

Gostava de agradecer a toda a minha família e à minha namorada pela paciência que tiveram comigo nos momentos de maior *stress*, pela ajuda que me prestaram e pela palavras de força que foram dizendo.

Quero também agradecer ao meu orientador pelo tempo que disponibilizou para discutir comigo o trabalho e pelas ideias que foi dando.

Índice

Lista de figuras

Lista de tabelas

Abreviaturas e Símbolos

Lista de abreviaturas (ordenadas por ordem alfabética)

ADT	*Android Development Tools*
AOSP	*Android Open Source Project*
API	*Application Programming Interface*
ASCII	*American Standard Code for Information Interchange*
CPU	*Central Processing Unit*
DDE	*Dynamic Data Exchange*
DLL	*Dynamic-link library*
GNU	*Gnu is Not Unix*
IDE	*Integrated Development Environment*
IP	*Internet Protocol*
LAN	*Local Area Network*
LTS	*Long-Term Support*
ODBC	*Open Data Base Connectivity*
OLE	*Object Linking and Embedding*
OOM	*Out-Of-Memory*
PLC	*Programmable Logic Controllers*
QEMU	*Quick Emulator*
RTDB	*Real-Time DataBase*
RTU	*Remote Terminal Unit*
SCADA	*Supervisory Control and Data Acquisition*
SDK	Software Development Kit
WLAN	Wireless Local Area Network

Capítulo 1

1 Introdução

Neste capítulo foi feita uma pequena introdução ao tema e explicado o porquê de ser um bom tema. São apresentados também neste capítulo os objetivos da dissertação bem com a metodologia usada para os atingir. Por último, é apresentada a estrutura da dissertação.

1.1 Enquadramento

O uso dos *smartphones*/*tablets* pelas pessoas no seu dia-a-dia está a sofrer um elevado crescimento, levando a que a maioria possua pelo menos um. Este enorme crescimento levou ao aumento do número de dispositivos diferentes no mercado.

Os sistemas SCADA, que permitem a monitorização e o controlo dos processos de um sistema, têm vindo a ser cada vez mais usados e melhorados. Aliado a este aumento do uso dos sistemas SCADA, têm vindo a ser desenvolvidos cada vez mais equipamentos para todas as componentes destes sistemas.

Por outro lado a evolução tecnológica levou também a uma grande evolução nas comunicações *Modbus TCP*/*IP* via protocolo 802.11. Esta evolução tecnológica permitiu que aparelhos equipados com protocolo 802.11 pudessem controlar equipamentos que comunicassem através do protocolo *Modbus*.

Este aumento do mercado de *smartphones* e do avanço nas comunicações *Modbus,* podem ser aproveitados em conjunto, possibilitando a substituição dos *Master Terminal Units* e das interfaces Homem/Máquina dos sistemas SCADA por *smartphones*/*tablets*. Esta substituição tem a enorme vantagem de os *smatphones*/*tablets* terem um custo de aquisição muito menor que os equipamentos normalmente usados. Por outro lado, os *smartphones*/*tablets* tem também um menor custo de manutenção e eliminam a restrição da manutenção apenas pelos fornecedores dos equipamentos. Os *smartphones*/*tablets* apresentam ainda uma enorme variedade de dispositivos, com características distintas, quando comparado à quantidade dos equipamentos usados nos sistemas SCADA atuais. Esta enorme variedade permite ao utilizador comprar um dispositivo, que melhor se adapte as suas necessidades e não um equipamento predefinido que não cumpre ou ultrapassa as suas necessidades.

Apesar de ser uma ótima ideia substituir as *Master Terminal Units* e as interfaces por dispositivos *mobile*, é necessário primeiro criar uma aplicação que realize o trabalho das

MTUs e das interfaces existentes nos sistemas SCADA atuais. Esta aplicação tem de ter também uma performance e escalabilidade igual ou melhor ao dos sistemas SCADA existentes.

1.2 Objetivos

Os objetivos para esta dissertação são a criação de uma *Master Terminal Unit* em conjunto com uma interface Homem/Máquina e a simulação de uma *Remote Terminal Unit* para verificação do sistema.

A *Master Terminal Unit* deve ser capaz de comunicar com a *Remote Terminal Unit*, através do protocolo *Modbus TCP/IP*. A *Master Terminal Unit* deve ainda ser capaz de atualizar a interface consoante as informações recebidas pela *Remote Terminal Unit*.

A interface Homem/Máquina deve incluir alguns dos objetos existentes nas interfaces industriais usadas atualmente, como por exemplo botões e luzes. Deve ainda ser possível configurar os objetos presentes na interface através de um ficheiro de configuração.

1.3 Metodologia

Como a realização da dissertação tem limites de tempo rígidos, foi necessário usar uma metodologia que permitisse realizar o trabalho de maneira estruturada e concisa. Por este motivo, a metodologia adotada foi a seguinte:

- Aprender a programar em ambiente *Java* e *Android*;
- Interpretar o problema e pensar em soluções possíveis;
- Estruturar uma arquitetura do sistema usando a melhor solução possível para o problema;
- Programar o sistema tendo por base a arquitetura estruturada anteriormente, mantendo uma capacidade critica e resolutiva na resolução de problemas;
- Verificar o sistema em conjunto com a simulação de uma *Remote Terminal Unit*;
- Realizar testes de funcionalidade, de performance e de escalabilidade;

1.4 Estrutura da tese

Esta dissertação foi estruturada em cinco capítulos.

No primeiro capítulo é dada uma pequena introdução ao tema e são explicados os objetivos e as metodologias.

No segundo capítulo são apresentados os conceitos necessários para entender-se o tema. Este capítulo divide-se nos conceitos de *Android*, de Sistemas SCADA e de *Modbus*.

No terceiro capítulo é descrito o problema e os seus requisitos. É também estruturada uma solução e uma planificação temporal.

No quarto capítulo é mostrado como foi implementada a solução, bem como os *softwares* e *hardwares* usados. É também mostrado os testes de funcionalidade, de performance e de escalabilidade realizados.

Por fim, no último capítulo, são apresentadas as conclusões retiradas do projeto bem como o trabalho que pode ser realizado no futuro, para melhorar e completar o projeto.

Capítulo 2

2 Tecnologias e conceitos associados

Neste capítulo foi feita uma introdução ao tema, apresentando os conceitos mais importantes das tecnologias usadas.

2.1 *Android*

2.1.1 Introdução

A plataforma *Android* é uma plataforma *open source,* desenvolvida com principal foco nos dispositivos *mobile*.

Esta plataforma foi comprada pela *Google* e mais tarde desenvolvida pela *Open Handset Alliance* (conjunto de algumas das maiores empresas no setor dos dispositivos *mobile*). A criação desta aliança possibilitou o avanço das *open standarts* no mercado dos dispositivos *mobile* [1].

A plataforma *Android* é *open source* desde as suas camadas mais baixas (ao nível do *kernel*) até as suas camadas mais altas (ao nível da *Application Framework*). O uso de licenças *open source* permite que a plataforma seja alterada a qualquer nível, consoante as necessidades do programador.

Por ser uma plataforma compreensiva, a plataforma *Android* permite uma utilização fácil por todo o tipo de utilizadores. [2]

Ao utilizador regular, a plataforma permite um uso intuitivo e uma quantidade considerável de configurações.

Ao programador de aplicações, a plataforma fornece todas as *frameworks* e ferramentas necessárias para uma criação rápida e fácil de aplicações.

Finalmente, aos fabricantes de dispositivos *mobile,* a plataforma é facilmente aplicada aos dispositivos desses fabricantes e permite uma configuração completa. A plataforma está pronta a ser usada nos dispositivos criados pelos fabricantes sendo apenas necessário a criação de algumas *drivers* para o *hardware* específico do fabricante.

2.1.2 História

A plataforma *Android* veio sofrendo grandes alterações com o passar dos anos.

Em 2005 a empresa *Android, Inc* foi comprada pela *Google*, mas apenas em 2007 foi apresentada a plataforma *Android* ao público, como uma plataforma *open source* para dispositivos *mobile*. Esta apresentação da plataforma deu-se aquando da apresentação da *Open Handset Alliance*, uma associação sem fins lucrativos de alguns dos maiores produtores e distribuidores de *software* e *hardware* para dispositivos *mobile*. Esta associação foi criada com o intuito de desenvolver a plataforma *Android* e permitir o avanço das *open standarts* para o mercado dos dispositivos *mobile*.

Em 2008 foi então apresentado o primeiro *Android Software Development Kit (SDK)*. Este SDK foi apresentado ao público antes de ter sido lançado qualquer dispositivo *mobile* que usa-se a plataforma *Android*. Isto pode acontecer pois o SDK permite a programação de aplicações sem o recurso a um dispositivo físico. O primeiro dispositivo a usar a plataforma *Android* foi lançado apenas próximo do fim do ano (Outubro) e era designado por *T-Mobile G1*.

Desde então até a 2014, o número de dispositivos que integram a plataforma *Android* têm vindo a aumentar, tendo atingido o seu pico máximo em 2014 (81.2% do mercado dos dispositivos *mobile*). No primeiro quarto do ano de 2015 a sua percentagem do mercado manteve-se, mostrando que a plataforma *Android* continua a integrar mais dispositivos que todos os seus competidores juntos [3].

2.1.3 Software Stack

A plataforma *Android* é constituída por várias camadas de *software*. Estas camadas têm cada uma a sua função e caraterísticas. Apesar de cada camada ter a sua função, estas camadas interligam-se umas com as outras para as realizarem.

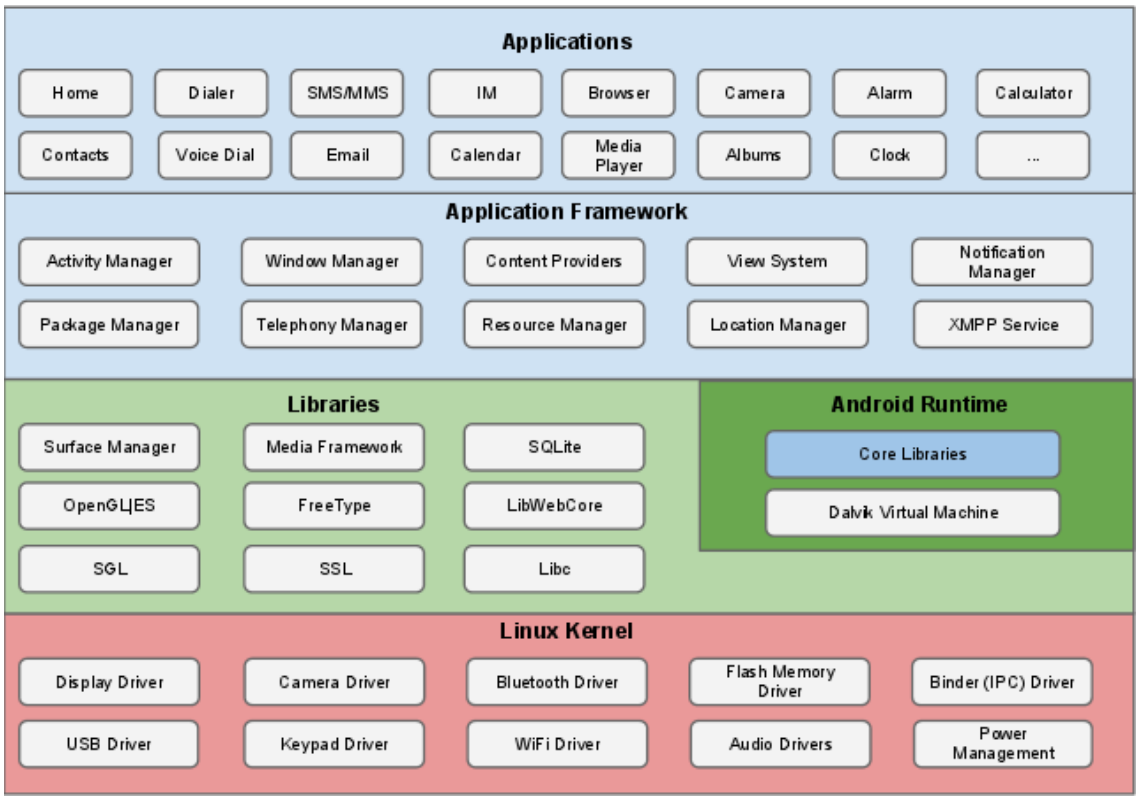

Figura 1 - *Android's Software Stack* [4]

2.1.3.1 *Linux Kernel*

Esta camada é a base do da plataforma *Android*, pois é sobre esta camada que foi construída toda a plataforma. O *kernel* do *Linux* foi usado por ser *open source*, algo necessário para poder ser a base de uma plataforma totalmente *open source*. Este *kernel* permite à plataforma uma abstração ao nível do *hardware*, maior segurança e mais funcionalidades.

Permite uma abstração ao nível do *hardware* pois o sistema *Linux*, no qual a plataforma é baseada, é fácil de compilar em várias arquiteturas distintas de *hardware*.

O sistema *Linux* é um sistema muito seguro que passou por muitos testes difíceis durante toda a sua existência. As aplicações da plataforma *Android* são executadas em processos distintos do *Linux*, cujas permissões são determinadas pelo sistema *Linux*. Por esta razão os processos da plataforma *Android* dependem principalmente da segurança do sistema *Linux*.

Por fim o sistema *Linux* possui algumas funcionalidades úteis que a plataforma *Android* usa, como por a gestão de memória e de potência.

2.1.3.2 *Libraries*

Esta camada contém as bibliotecas padrão usadas pela plataforma *Android* para permitir o uso de certos serviços. Estas bibliotecas incluem por exemplo a biblioteca *OpenGL* usada para os gráficos 3D e a biblioteca *SQLite* usada para a criação de uma base de dados completamente funcional.

2.1.3.3 *Android Runtime*

Esta camada é onde se encontra a máquina virtual *Dalvik* e as bibliotecas essenciais para o seu funcionamento. Esta máquina virtual foi desenvolvida pela *Google* unicamente para ser usada pela plataforma *Android*. Ao contrário das máquinas virtuais de *Java* (JVM), que foram criadas com o propósito de poderem ser usadas em qualquer dispositivo, a máquina virtual *Dalvik* foi criada para ser usada apenas para dispositivos *mobile*. Ao ser criada apenas com os dispositivos *mobile* em vista, foi possível melhorar funcionalidades das JVM, de maneira a serem mais otimizadas para dispositivos *mobile*.

As máquinas virtuais *Java* compilam o seu *source code* em *byte code* e executam-no. Tal como as máquinas virtuais *Java*, as máquinas virtuais *Dalvik* compilam também o *source code* em *byte code*. No entanto as máquinas virtuais *Dalvik* pegam no *byte code* e compilam-no em *Dalvik byte code*. Este código binário resultante comporta-se como um executável só que é ainda mais otimizado para ser executado em dispositivos *mobile* [5]. A comparação da transformação do *source code* por estas duas máquinas virtuais está demonstrada na seguinte figura.

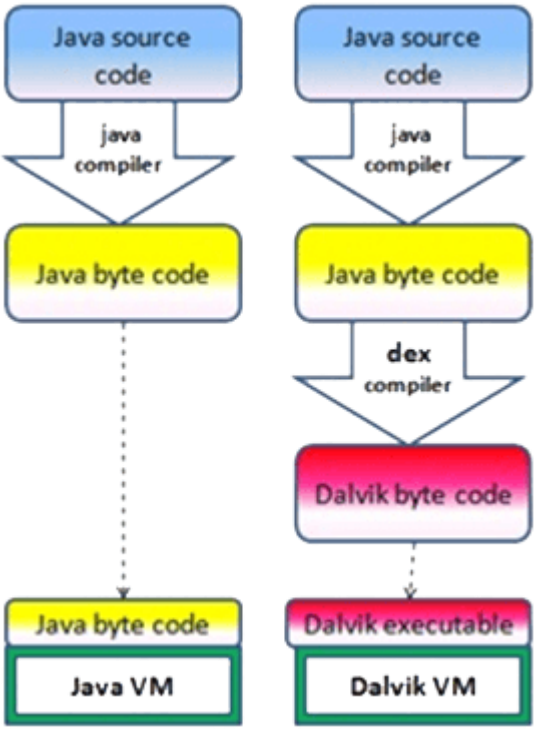

Figura 2 - Comparação entre *Java Virtual Machine* e *Dalvik Virtual Machine* [6]

2.1.3.4 *Application Framework*

Esta camada é a que possibilita a criação de aplicações. Nesta camada podem ser encontradas várias bibliotecas criadas especificamente para a plataforma *Android*. Podem também ser encontrados vários serviços que permitem às aplicações aceder a sistemas como por exemplo sensores, localização, *wifi*, etc.

2.1.3.5 *Applications*

Esta é a camada onde se encontram as aplicações criadas pelos utilizadores. Estas aplicações podem já se encontrar nos dispositivos ou podem ser descarregadas e instaladas.

As aplicações são instaladas através do uso de *Android Application Packages* (.apk). Estes *packages* são constituídos por:

- *Dalvik executable* - executável criada através do *Java source code*;
- *Resources* – recursos que não sejam *Java source code,* como por exemplo os vários ficheiros de configuração XML e as imagens usadas pela aplicação;
- *Native libraries* – bibliotecas escritas em código nativo *C/C++*, que podem ou não fazer parte da aplicação;

2.1.4 *Main Blocks*

As aplicações da plataforma *Android* são constituídas por blocos. Cada bloco tem a sua utilidade, podendo estar ou não presente consoante a aplicação.

2.1.4.1 *Activities*

Uma atividade é uma coleção dos elementos de *design* que, ao trabalharem juntos, compõem cada interface da aplicação [5]. Ou seja a atividade é tudo o que o utilizador vê no ecrã, como por exemplo os elementos gráficos, o texto, o fundo, os vídeos, etc.

As atividades são compostas por um *Layout Container* que contém todas as *widgets* da aplicação. Estas *widgets* contém todos os elementos gráficos presentes no ecrã e podem ou não conter imagens. Estas imagens na plataforma *Android* são designadas *drawables*.

As atividades podem também criar ou receber eventos. Estes eventos permitem à aplicação alterar os seus elementos gráficos enquanto a aplicação está a ser executada.

2.1.4.2 *Services*

Enquanto as atividades são o *front-end* da aplicação, os serviços são o *back-end*. Estes serviços correm normalmente em paralelo com as atividades e são invisíveis ao utilizador. Estes serviços podem ser por exemplo músicas que correm em plano de fundo ou operações matemáticas necessárias para a atividade.

2.1.4.3 *Broadcast Receivers*

Os *broadcasts* são mensagens que são criadas em resposta a certos eventos, como por exemplo uma chamada telefónica ou bateria fraca. As aplicações podem ou não conter *broadcast receivers,* que servem para receber os *broadcasts* A receção destes *broadcasts* permite à aplicação realizar certas ações em resposta a esses *broadcasts*, como por exemplo emitir um som quando a bateria estiver fraca.

2.1.4.4 *Intents*

Os *intents* são pequenas mensagens transmitidas entre aplicações diferentes. Estas mensagens permitem a uma aplicação iniciar atividades, iniciar ou parar serviços ou simplesmente enviar um *broadcast*. Um exemplo do uso de *intents* pode ser visto na figura 3.

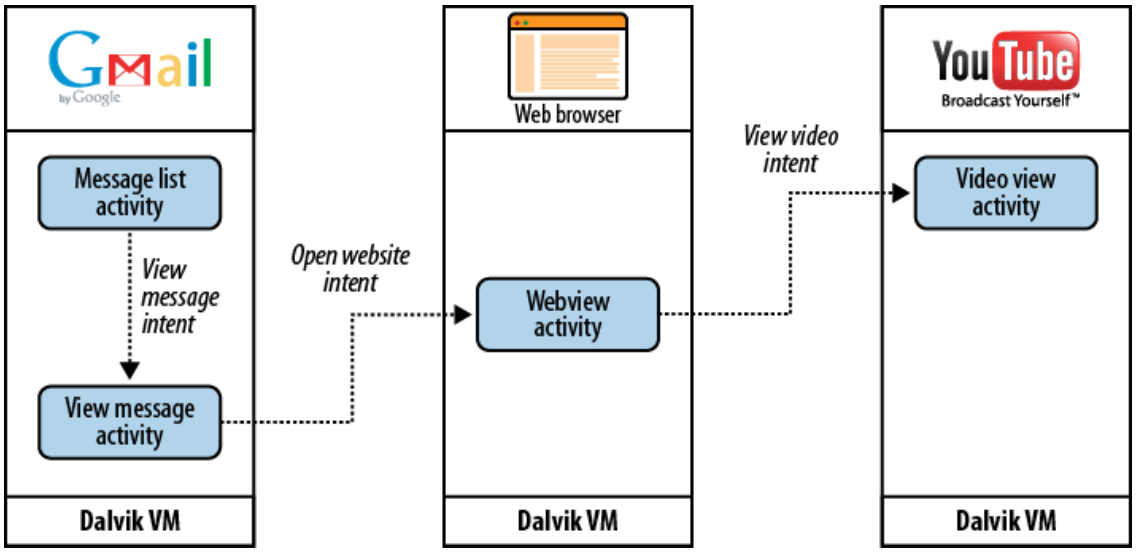

Figura 3 - Exemplo do uso de *intents* [2]

2.1.4.5 *Content Provider*

Os *Content Providers* são uma interface entre as bases de dados de cada aplicação. Esta interface permite realizar operações de inserção, atualização, remoção e interrogação às bases de dados das várias aplicações. Esta interface permite então a troca indireta de uma grande quantidade de dados entre aplicações, como por exemplo a troca dos dados de contatos entre duas aplicações diferentes, do tipo lista de contatos.

2.2 Sistema SCADA

2.2.1 Introdução

SCADA é um acrónimo para *Supervisory Control and Data Acquisition*. Estes sistemas são usados para monitorizar e controlar a planta ou equipamentos nas indústrias como as telecomunicações, água e controlo de desperdícios, energia, refinarias de óleo e gás e transporte [6]. Estes tipos de sistemas envolvem a transferência de dados entre *Master Terminal Units* e *Remote Terminal Units*. Envolvem também a transferência de dados entre a *Master Terminal Units* e as interfaces Homem/Máquina.

Estes sistemas podem ser relativamente simples (sistemas para monitorizar as condições ambientais de um pequeno escritório) ou muito complexos (sistemas para monitorizar todas as atividades numa central nuclear ou as atividades do sistema de águas municipal).

Estes sistemas são constituídos por:

- Interfaces Homem/Máquina – Interface que recebe os dados da *Master Terminal Unit* e apresenta-os ao utilizador;
- *Master Terminal Unit* – Computador central que recolhe todos os dados presentes nas *Remote Terminal Units* e envia comandos para as mesmas;
- *Remote Terminal Unit* – Equipamento que recolhe todos os dados dos sensores e os transforma em dados digitais, de maneira a serem enviados para as *Master Terminal Units*;
- Infraestrutura de comunicação – Infraestrutura usada para realizar as comunicações entre as *Remote Terminal Units* e as *Master Terminal Units*;

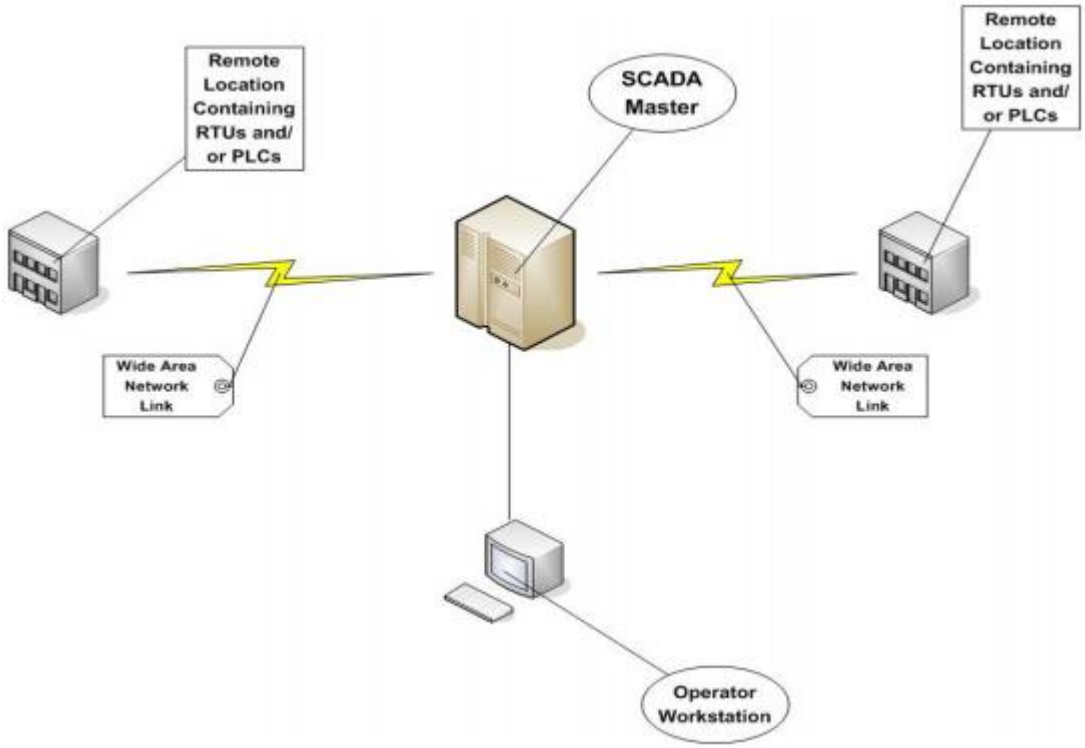

Figura 4- Sistema SCADA básico [7]

2.2.2 Componentes do sistema SCADA

2.2.2.1 *Remote Terminal Unit*s (RTUs)

As *Remote Terminal Units* são os "olhos e ouvidos" de um sistema SCADA [8]. Estas RTUs são os "olhos e ouvidos" do sistema, pois recebem informação sobre o estado deste através dos sensores. Esta informação é transformada em dados digitais através de conversores analógico-digitais. Estes dados, por sua vez, são transmitidos para as *Master Terminal Units* que os interpreta.

Além de servirem para informar o estado do sistema às *Master Terminal Units*, as RTUs servem também para atuar sobre o sistema. Esta atuação é feita sobre os atuadores

presentes no campo, mediante a receção de comandos de controlo, provenientes das *Master Terminal Units*.

No entanto, atualmente, já é possível realizar operações nas RTUs. Estas operações permitem às *Remote Terminal Units* atuar automaticamente em certos parâmetros dos sistemas.

2.2.2.2 Redes de comunicação

As redes de comunicação são usadas para transmitir os dados entre as *Master Terminal Units* e as *Remote Terminal Units*. São usadas também para transmitir os dados entre as *Master Terminal Units* e as interfaces Homem/Máquina.

As redes de comunicação podem ser estabelecidas usando cabo terrestres, cabos telefónicos ou rádio. Os cabos terrestres têm a vantagem de permitirem trocas rápidas de dados, mas têm a desvantagem de terem um elevado custo, quando usados sobre grandes distâncias. Os cabos telefónicos têm a vantagem de poderem ser arrendados e terem uma grande abrangência geométrica. No entanto, certos locais geométricos não permitem o uso de cabos telefónicos nem terrestres, o que leva à utilização das comunicações via rádio.

Além dos vários tipos de redes de comunicação, existem também um grande número de protocolos que podem ser usados nas comunicações entre as *Master Terminal Units* e as *Remote Terminal Units*.

2.2.2.3 *Master Terminal Units* (MTUs)

As *Master Terminal Units* são um computador ou um conjunto de computadores, que recebem e processam os dados dos processos, provenientes das *Remote Terminal Units*. Estas MTUs comunicam com as *Remote Terminal Units* usando um protocolo previamente estabelecido.

As MTUs enviam também mensagens para as RTUs de maneira a controlarem os processos. Este controlo dos processos passa pela modificação do estado dos vários atuadores do sistema.

Estas MTUs servem também para enviar os dados dos processos para as interfaces Homem/Máquina, de maneira a que o estado do sistema possa ser mostrado ao utilizador.

Além disto as MTUs podem guardar os dados do processo em *logs,* de maneira a permitir ao utilizador observar os gráficos das várias variáveis pelo tempo.

2.2.2.4 Interfaces Homem/Máquina

As interfaces Homem/Máquina mostram ao utilizador o estado do sistema. O estado do sistema advém dos dados dos processos, que as *Master Terminal Units* enviam para a interface.

De maneira a mostrar o estado do processo de uma maneira mais percetível, são usados objetos que facilitam a compreensão dos dados. Estes objetos podem ser quaisquer tipos de objetos que forem convenientes, como por exemplo luzes, imagens, gráficos, etc.

2.2.3 Evolução das arquiteturas do sistema SCADA

Os sistemas SCADA têm vindo a evoluir em paralelo com a evolução da tecnologia. Durante a sua evolução apareceram três grandes gerações de arquiteturas *SCADA*:

- Primeira Geração – Monolítica;
- Segunda Geração – Distribuída;
- Terceira Geração – *Networked*;

2.2.3.1 Sistemas SCADA monolíticos

Quando os sistemas SCADA foram desenvolvidos o conceito de computação estava geralmente centrado em sistemas "*mainframe*" e as redes geralmente não existiam, o que levava a que cada sistema centralizado fosse autónomo sem quase nenhuma ligação a outros sistemas.

As redes WAN que eram implementadas para comunicar com os *RTUs* serviam apenas para comunicar com os RTUs no campo e nada mais, isto pois os protocolos *WAN* usados hoje em dia eram desconhecidos na altura.

Os protocolos de comunicação destes sistemas eram desenvolvidos pelos vendedores dos *RTUs* e eram maioritariamente proprietários. Essas características, levavam a que esses protocolos fossem muito rígidos não suportando qualquer funcionalidade para além de pesquisar e controlar os dispositivos remotos e impossibilitando outros tipos de tráfego de dados com comunicação *RTU* nessa rede.

A conectividade ao computador central era também muito limitada pelos vendedores, isto pois geralmente essa ligação só podia ser efetuada ligando um adaptador próprio ao *CPU*.

A redundância nesta geração era feita usando dois sistemas mainframe, o principal e o de *backup*. A função do sistema de backup era apenas a de monitorizar o sistema primário e detetar um evento de falha, o que significava que quase nenhum processamento era realizado neste sistema de backup [7][9].

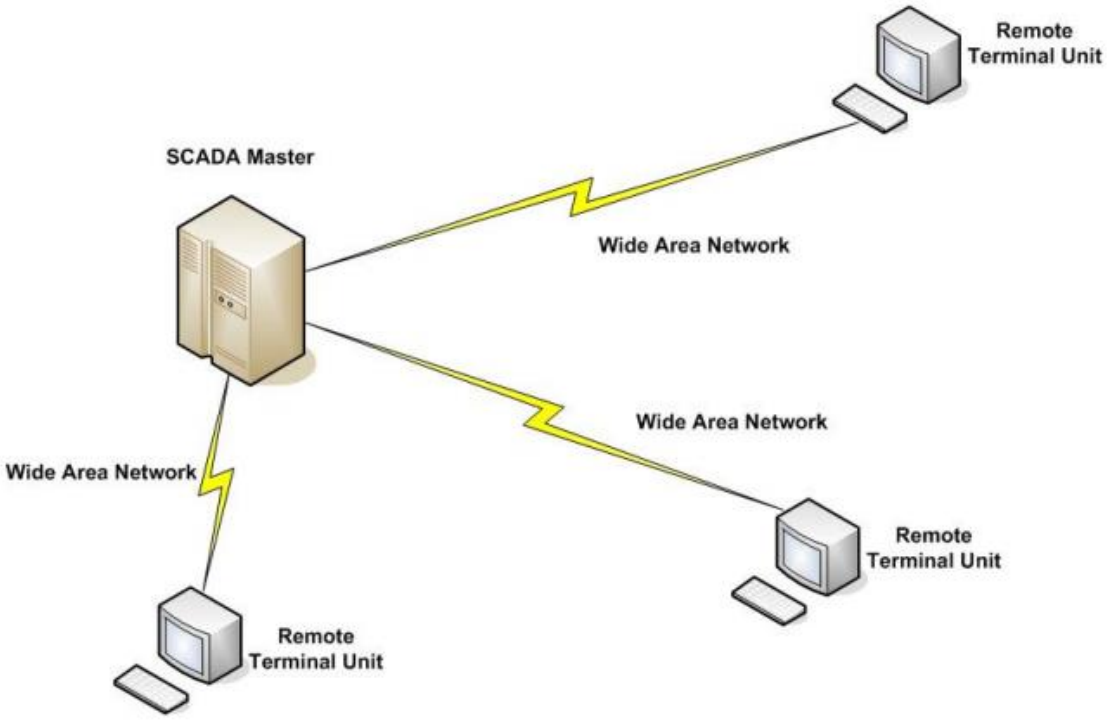

Figura 5 - Arquitetura SCADA de primeira geração [7]

2.2.3.2 Sistemas SCADA distribuídos

Esta geração tirou partido dos avanços tecnológicos ao nível da miniaturização dos sistema e ao nível da tecnologia das redes *LAN* para distribuir o processamento por vários sistemas. Várias estações, cada uma com a sua função, eram ligadas a uma *LAN* e partilhavam informação umas com as outras em tempo real. Como cada uma tinha a sua função as estações passaram a ser tipicamente mini computadores, muito mais baratos e pequenos que os da primeira geração.

Esta distribuição das funcionalidades do sistema por sistema ligados a rede *LAN* serviu não apenas para aumentar a capacidade de processamento (uso de vários processadores com simples funcionalidades contra o uso de apenas um processador), mas também para aumentar a redundância e a confiança no sistema como um todo. Isto permite a que ao contrário do sistema primário/*backup* usado nos sistemas de primeira geração, se tivesse uma arquitetura distribuída que mantinha todas as estações num estado online, levando a substituição de uma estação por outra em caso de falha, sem necessidade de esperar da transição do sistema primário para o de backup.

No entanto a comunicação entre estações tinha de ser realizada localmente na rede *LAN* e a rede *WLAN* era apenas usada unicamente para comunicação entre os *RTUs* e um servidor de comunicações na rede *LAN*, não havendo qualquer estação remota.

Outro problema era que os protocolos usados na rede *LAN* eram normalmente proprietárias, por um lado permitindo otimizar o tráfego em tempo real da rede, mas por outro lado limitando ou até mesmo eliminando a possibilidade de conexão com redes *LAN* de outros vendedores. Logo a segunda geração de sistemas SCADA continuava a ser limitada ao nível do *hardware*, software e dispositivos periféricos pelos vendedores [7][9].

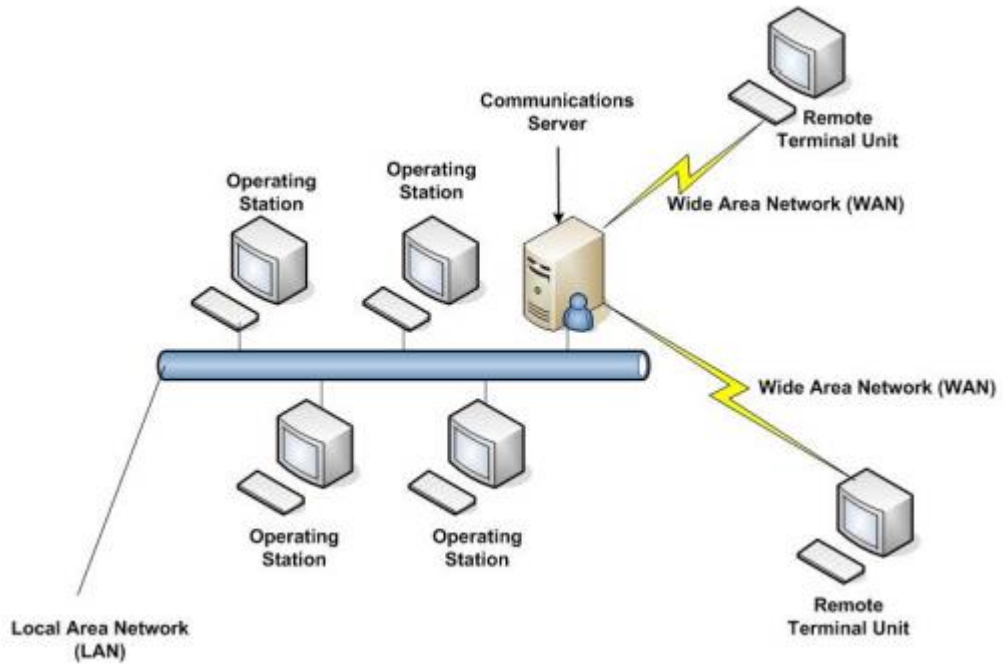

Figura 6 - Arquitetura SCADA de segunda geração [7]

2.2.3.3 Sistemas SCADA Networked

A atual geração dos sistemas SCADA apresenta a maior diferença de usar protocolos abertos e não proprietários. Ainda existem múltiplos sistemas partilhando as funcionalidades do computador central, e RTUs que usam protocolos proprietários. O maior avanço desta geração foi tornar a arquitetura do sistema aberta, utilizando *standards* e protocolos abertos o que possibilitou distribuir as funcionalidades através da rede *WLAN* e não apenas da rede *LAN*. Tornar o sistema aberto facilitou também a conexão de dispositivos periféricos ao sistema e/ou a rede.

As maiores vantagens destas geração advém da possibilidade do uso de protocolos *WLAN* como o *Internet Protocol* (IP) para comunicar entre estação principal e os equipamentos de comunicação. Isto permitiu a que uma parte da estação principal responsável pela comunicação com os dispositivos de campo pudesse estar dispersa por toda a rede *WLAN* [7][9].

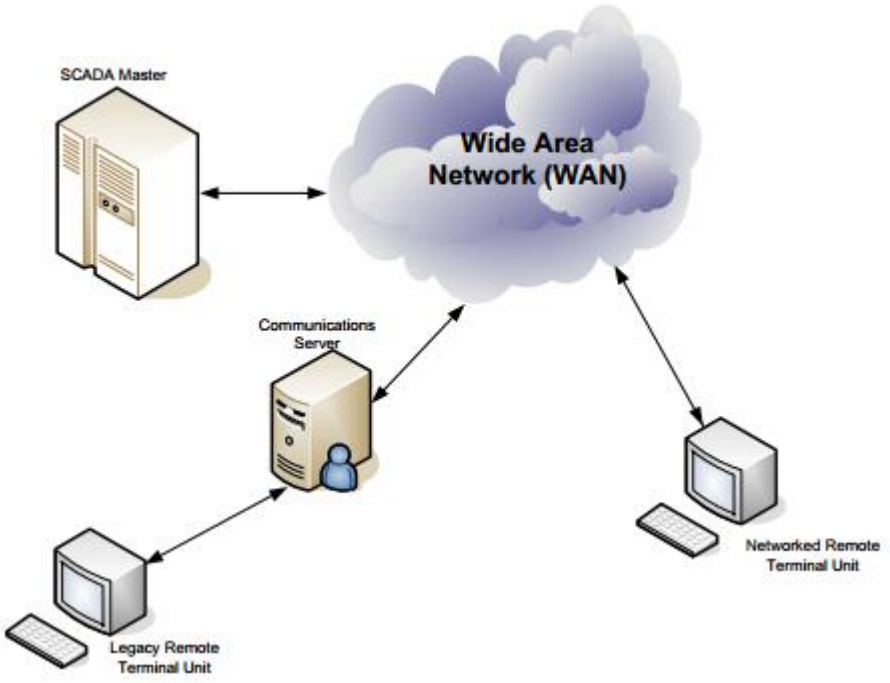

Figura 7 - Arquitetura SCADA de terceira geração [7]

2.3 Modbus

2.3.1 Introdução

O protocolo *Modbus* é um protocolo que assenta na arquitetura mestre/escravo. Neste tipo de arquitetura apenas o mestre pode enviar pedidos. Estes pedidos podem ser enviados de um mestre para outro mestre ou de um mestre para um escravo.

As mensagens enviadas usando este protocolo seguem uma *Application Data Unit* bem definida, podendo existir apenas pequenas alterações. Esta *Application Data Unit* pode ser observada na figura 8.

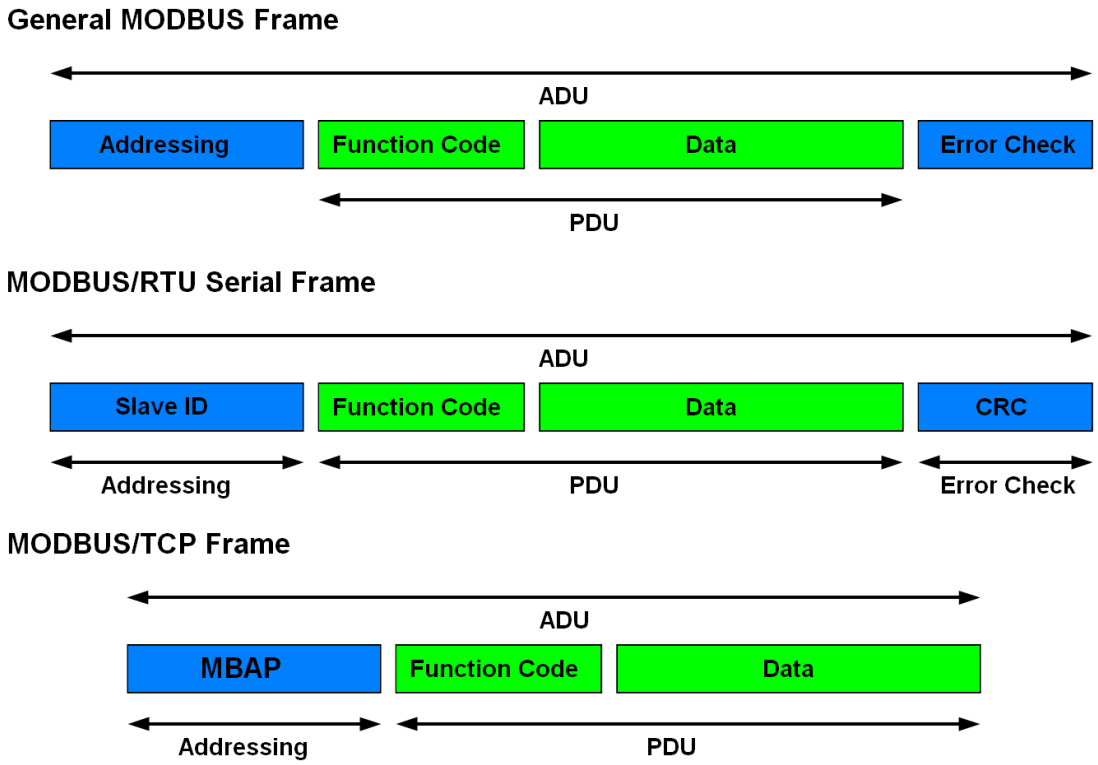

Figura 8 - *Application Data Unit* do protocolo *Modbus* [10]

Pode-se observar também pela figura 8 que o tipo das ADU é identificado por um *function code*. Estes *function code* identificam o tipo de mensagem que esta a ser enviada e é definido por um id único, como se pode ver pela figura 9.

				code	Sub code	(hex)	Section
Data Access	Bit access	Physical Discrete Inputs	Read Discrete Inputs	02		02	6.2
		Internal Bits Or Physical coils	Read Coils	01		01	6.1
			Write Single Coil	05		05	6.5
			Write Multiple Coils	15		0F	6.11
	16 bits access	Physical Input Registers	Read Input Register	04		04	6.4
		Internal Registers Or Physical Output Registers	Read Holding Registers	03		03	6.3
			Write Single Register	06		06	6.6
			Write Multiple Registers	16		10	6.12
			Read/Write Multiple Registers	23		17	6.17
			Mask Write Register	22		16	6.16
			Read FIFO queue	24		18	6.18
	File record access		Read File record	20		14	6.14
			Write File record	21		15	6.15
	Diagnostics		Read Exception status	07		07	6.7
			Diagnostic	08	00-18,20	08	6.8
			Get Com event counter	11		OB	6.9
			Get Com Event Log	12		0C	6.10
			Report Slave ID	17		11	6.13
			Read device Identification	43	14	2B	6.21
Other			Encapsulated Interface Transport	43	13,14	2B	6.19

Figura 9 - *Function Codes* possíveis no protocolo *Modbus* [10]

Neste tipo de protocolo o mestre envia um pedido para o escravo. Quando recebe o pedido, o escravo decifra a mensagem, através do *function code*, e executa as ações necessárias.

Quando terminar de realizar as ações necessárias, o escravo envia uma resposta ao pedido do mestre ou uma mensagem de erro, como se pode observar pelas figuras 10 e 11 respetivamente.

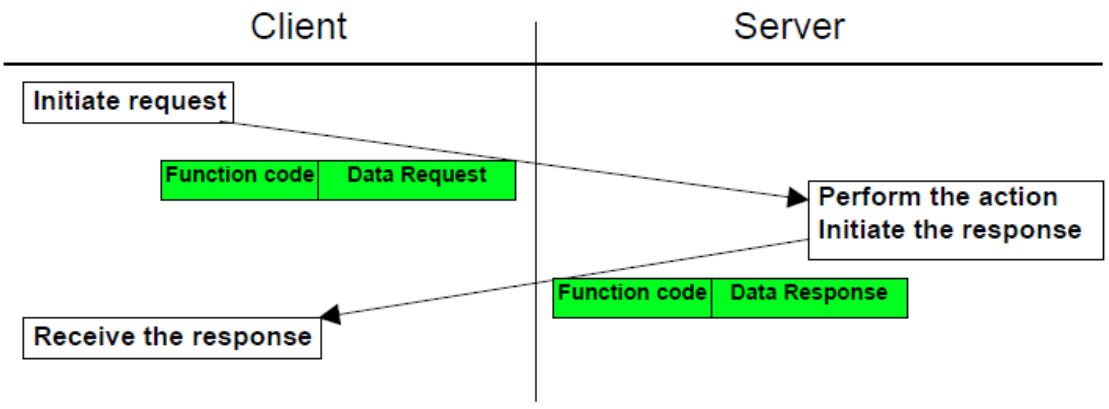

Figura 10 - Pedido e resposta sem erro [10]

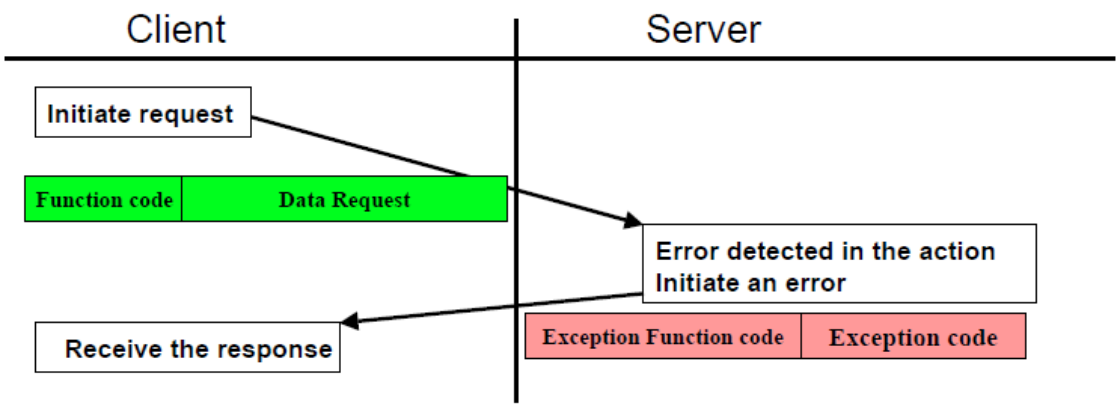

Figura 11 - Erro ao executar a ação [10]

Esta mensagem de erro é facilmente identificada pela presença do valor 0x80 no sétimo *byte* da trama. Esta mensagem de erro permite ainda saber qual foi o erro que ocorreu, através do uso de *error codes*. O id dos *error codes* e o seu significado estão presentes na tabela seguinte.

Tabela 1 – *Error codes* possíveis no protocolo *Modbus*

Exception Code	Nome
0x01	Illegal Function
0x02	Illegal Data Address
0x03	Illegal Data Value
0x04	Slave Device Failure
0x05	Acknowledge
0x06	Slave Device Busy
0x07	Negative Acknowledge
0x08	Memory Parity Error
0x0A	Gateway Path Unavailable
0x0B	Gateway Target Device Failed to Respond

Capítulo 3

3 Estrutura do sistema

Neste capítulo são descritos o sistema e os seus requisitos. É também apresentada neste capítulo uma possível solução para o sistema, usando um diagrama de classes *UML* e um diagrama de atividades *UML*.

Estes diagramas mostram a estrutura da solução encontrada, bem como as suas ações no tempo.

Por fim, é ainda apresentado um diagrama de *Gantt* com uma possível planificação temporal da implementação da solução.

3.1 Descrição do sistema

O sistema deve ser constituído por um *Master Terminal Unit* e uma interface Homem/Máquina.

O *Master Terminal Unit* deve ser capaz de enviar pedidos e receber respostas das *Remote Terminal Units* usando o protocolo *Modbus TCP/IP*. Deve também conseguir enviar e receber pedidos de outras *Master Terminal Units*. Além de receber as respostas, a *Master Terminal Unit* deve ser capaz de interpretar as respostas e guardá-las nas variáveis especificas para isso. A *Master Terminal Unit* deve ainda usar os valores guardados nas variáveis para efetuar alterações na interface Homem/Máquina.

A interface Homem/Máquina deve mostrar todos os objetos especificados num ficheiro de configuração. Esta interface deve ainda alterar os estados dos seus objetos consoante as ordens enviadas pela *Master Terminal Unit*.

3.2 Requisitos do sistema

Os requisitos do sistema foram identificados e divididos entre requisitos da *Master Terminal Unit* e requisitos da interface Homem/Máquina.

3.2.1 Requisitos da *Master Terminal Unit*

Tabela 2 - Requisitos da *Master Terminal Unit*

ID	Definição
MTU0	A *Master Terminal Unit* deve conseguir enviar pedidos e receber respostas de leitura de *coils*, através do protocolo *Modbus TCP/IP*
MTU1	A *Master Terminal Unit* deve conseguir enviar pedidos e receber respostas de leitura de entradas discretas, através do protocolo *Modbus TCP/IP*
MTU2	A *Master Terminal Unit* deve conseguir enviar pedidos e receber respostas de leitura de registos de entradas discretas, através do protocolo *Modbus TCP/IP*
MTU3	A *Master Terminal Unit* deve conseguir enviar pedidos e receber respostas de escrita de *coils*, através do protocolo *Modbus TCP/IP*
MTU4	A *Master Terminal Unit* deve conseguir interpretar as respostas recebidas, através do protocolo *Modbus TCP/IP*, e guardá-las nas respetivas variáveis
MTU5	A *Master Terminal Unit* deve conseguir ler um ficheiro de configuração e inicializar as comunicações *Modbus TCP/IP*
MTU6	A *Master Terminal Unit* deve conseguir ler um ficheiro de configuração e inicializar os objetos da interface Homem/Máquina
MTU7	A *Master Terminal Unit* deve conseguir alterar os estados dos objetos da interface consoante o valor das variáveis

3.2.2 Requisitos da Interface Homem/Máquina

Tabela 3 - Requisitos da Interface Homem/Máquina

ID	Definição
HMI0	A interface Homem/Máquina deve incluir objetos do tipo botão
HMI1	A interface Homem/Máquina deve incluir objetos do tipo luz
HMI2	A interface Homem/Máquina deve incluir objetos do tipo imagem
HMI3	A interface Homem/Máquina deve incluir objetos do tipo texto
HMI4	A interface Homem/Máquina deve incluir objetos do tipo forma
HMI5	A interface Homem/Máquina deve incluir objetos do tipo imagem
HMI6	A interface Homem/Máquina deve conseguir alterar os estados dos objetos, consoante as ordens recebidas pela MTU
HMI7	A interface Homem/Máquina deve conseguir alterar a visibilidade dos objetos, consoante as ordens recebidas pela MTU
HMI8	A interface Homem/Máquina deve conseguir alterar a posição dos objetos nos eixos x e y, consoante as ordens recebidas pela MTU
HMI9	A interface Homem/Máquina deve conseguir dispor os objetos ao longo do eixo dos z (alterar o *depth*)

3.3 Diagrama de classes UML

De maneira a desenvolver uma solução de maneira estruturada que cumprisse todos os requisitos, foi criado um diagrama de classes.

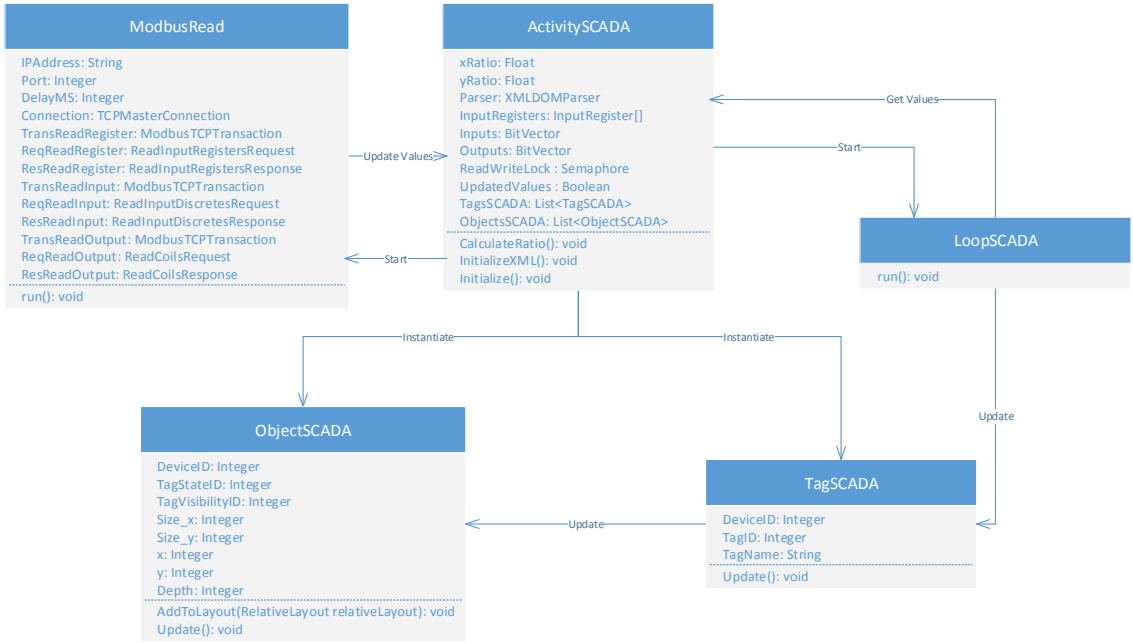

Figura 12 - Diagrama de classes do sistema

Como se pode observar pela Figura 15 existe uma classe que trata da parte das comunicações (*ModbusRead*) e uma classe que trata da parte de enviar ordens para a interface Homem/Máquina (*LoopSCADA*). Estas duas classes são *threads* que correm em paralelo a interface, de maneira a manter a *thread* da interface Homem/Máquina o menos carregada possível. Isto é necessário para que as alterações à interface aconteçam sem nenhum atraso percetível ao utilizador.

Como se pode observar também pela figura existe uma classe principal (*ActivitySCADA*) que inicializa as outras duas classes referidas anteriormente. Esta classe inicializa também a interface e todos os objetos nela presente.

Existe também uma classe designada *TagSCADA* que guarda todos os objetos que dependam de uma determinada variável, bem como o valor anterior dessa variável.

Tanto a classe *ObjectSCADA* como a classe *TagSCADA* têm classes que dependem delas como se pode observar pelas figuras 16 e 17, respetivamente.

As classes que dependem da classe *ObjectSCADA* representam todos os tipos de objetos da interface Homem/Máquina, bem como todas as características únicas de cada objeto.

As classes que dependem da classe *TagSCADA* representam todos os tipos de *tags* que podem existir, bem como o tipo de variável da qual dependem.

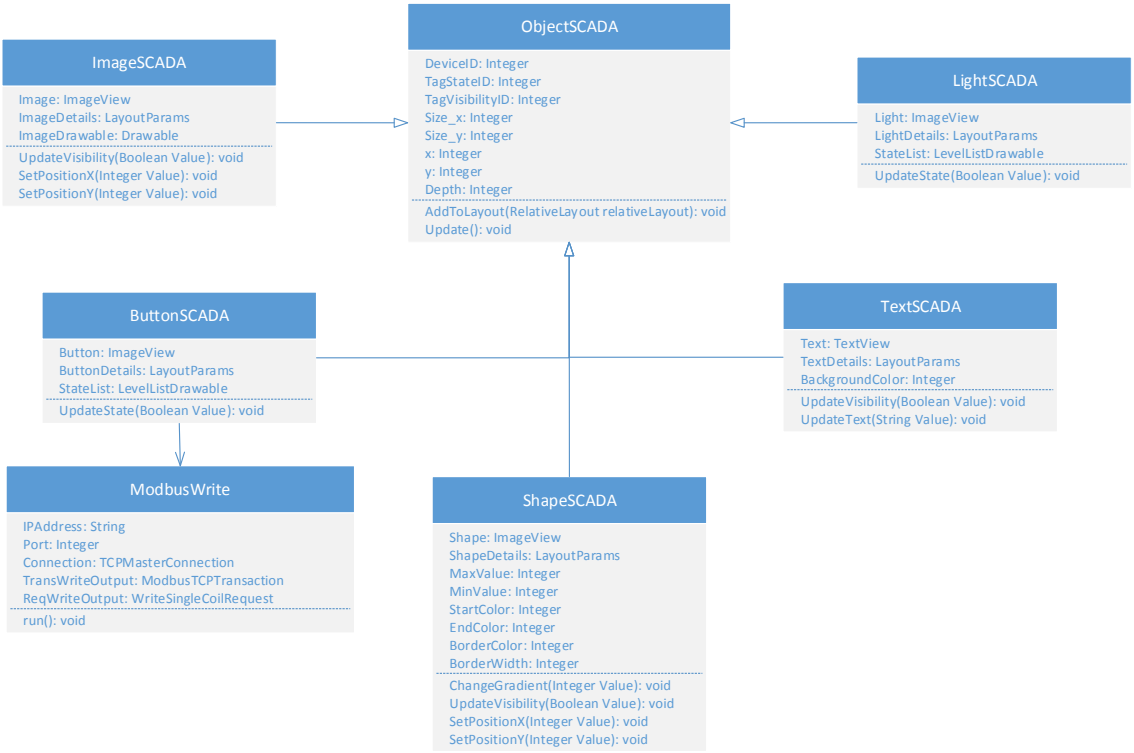

Figura 13 - Diagrama de classes dos objetos

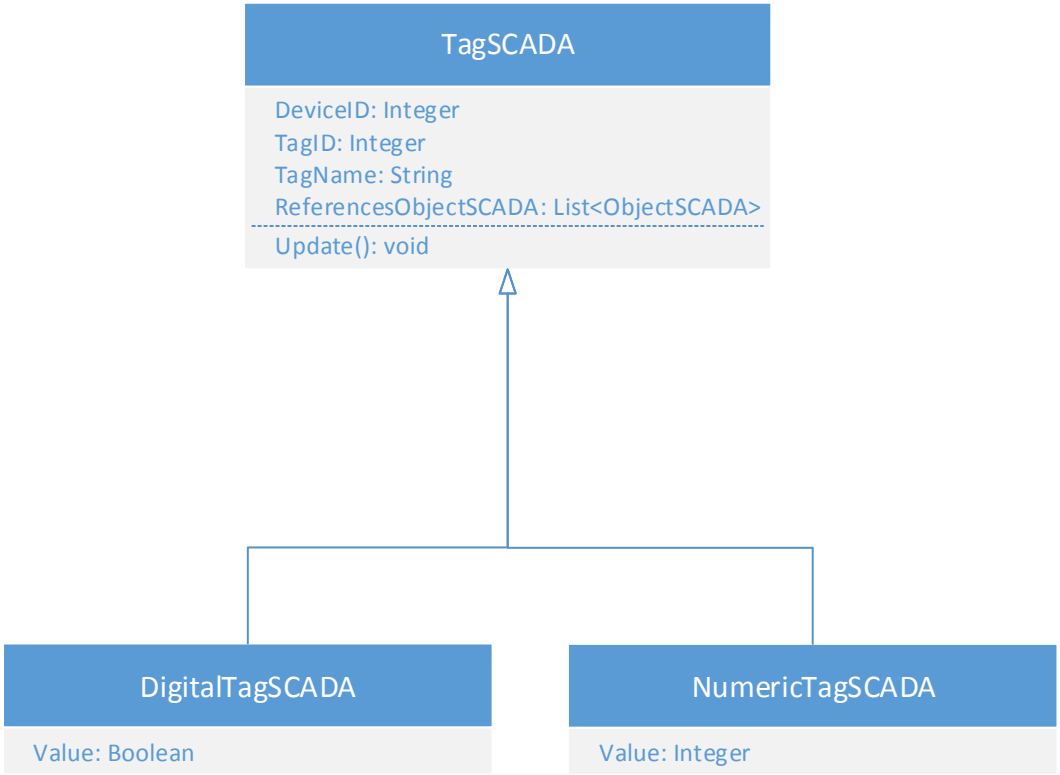

Figura 14 - Diagrama de classes das *tags*

3.4 Diagrama de atividades UML

De maneira a estruturar como a solução descrita no ponto anterior deveria agir temporalmente, foram criados diagramas de atividade UML como se pode observar pelas seguintes figuras.

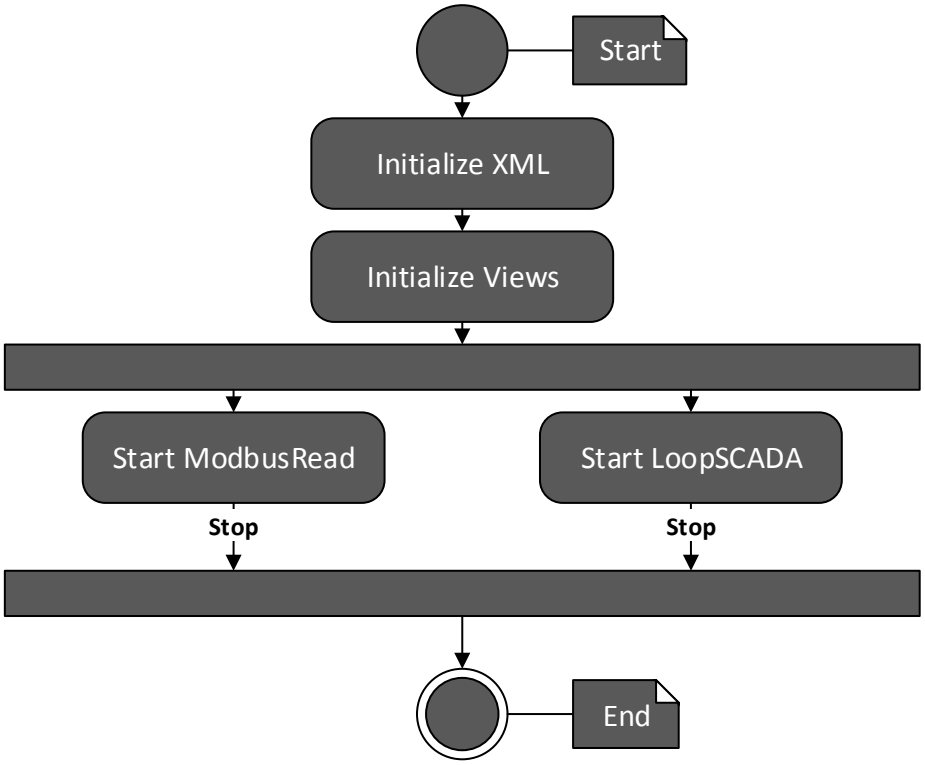

Figura 15 – Diagrama de atividades (Inicialização)

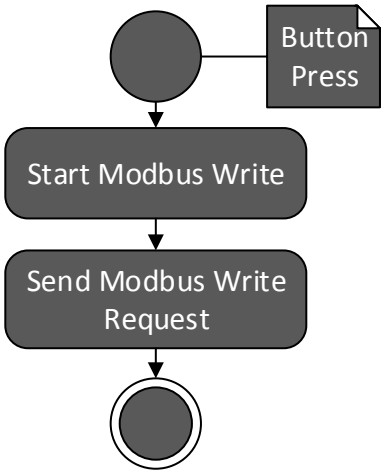

Figura 16 – Diagrama de atividades (pedido de escrita de *coils*)

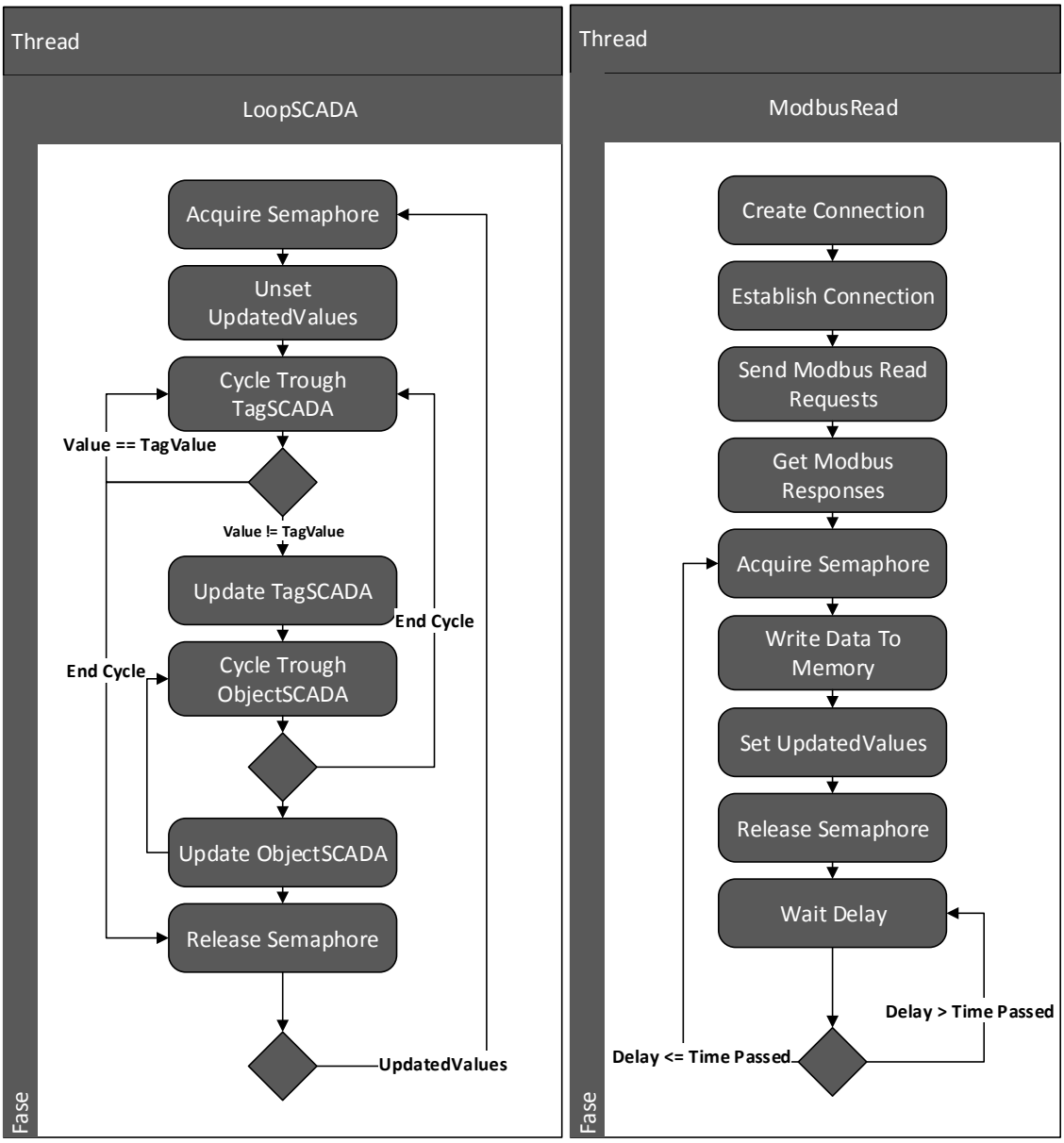

Figura 17 - Diagrama de atividades (*threads* paralelas)

Como se pode observar pelo diagrama da Figura 18, quando o sistema é iniciado é lido o ficheiro de configuração e iniciada a interface Homem/Máquina com os objetos presentes no ficheiro de configuração. Podemos observar também que são iniciadas as duas *threads* responsáveis pelas comunicações e pela atualização da interface.

Ao observar a figura 19, vemos que é realizado um pedido de escrita de *coils* quando um objeto do tipo botão é pressionado. Este pedido de escrita é executado por uma *thread* paralela denominada *ModbusWrite*.

Ao observar a figura 20, vemos que a *thread ModbusRead* cria uma conexão ao servidor, ao qual vai realizar os pedidos de leitura usando o protocolo *Modbus TCP/IP*. Para criar esta conexão a *thread* usa os valores de IP e porta presentes no ficheiro de configuração. Esta conexão é então usada para enviar pedidos (leitura de *coils*, leitura de entradas discretas e leitura de registos de entradas discretas) e receber as respostas, através do protocolo *Modbus*

TCP/IP. Após serem recebidas todas as respostas, a *thread* espera até o semáforo de leitura e escrita estar livre e de seguida bloqueia-o. Depois de estar bloqueado o semáforo, a *thread* escreve o valor das respostas nas variáveis respetivas e avisa a *thread LoopSCADA* que os valores foram atualizados. De seguida a *thread liberta o semáforo*. Este processo é então repetido em ciclo, usando um valor de atraso estipulado no ficheiro de configuração.

Ao observar a figura 20, vemos também que a *thread LoopSCADA* espera até o semáforo de leitura e escrita estar livre e bloqueia-o. De seguida, a *thread* percorre a lista de todos os objetos do tipo *TagSCADA* e verifica se o valor antigo guardado nelas é igual ao valor atual, lido pela *thread ModbusRead*. No caso de serem diferentes a *thread LoopSCADA* chama a função *Update* dessa *TagSCADA*. Esta função *Update* altera o valor guardado pela *TagSCADA* para o seu valor atual e percorre a lista de todos os *ObjectSCADA* que dependem dessa *tag*. Por cada *ObjectSCADA* da lista é chamada a sua função *Update*. Esta função *Update* recebe da *TagSCADA* apenas o valor da *tag*, o tipo de *tag* e o id correspondente. Esta função *Update* utiliza os valores passados pela *TagSCADA* para perceber qual as características do objeto é que devem ser alteradas. Depois de percorrer todos os objetos *TagSCADA* da lista, a *thread* avisa a *thread ModbusRead* que os valores já não são atuais e liberta o semáforo. Este ciclo é então repetido sempre que novos valores forem lidos pela *thread ModbusRead*.

3.5 Planificação

De maneira a distribuir corretamente a quantidade de trabalho, pelo tempo disponível, foi usado o diagrama de *Gantt* da figura 21. Este diagrama permitiu uma melhor gestão do tempo, possibilitando assim, manter uma carga de trabalho idêntica durante todas as semanas.

Figura 18 - Diagrama de *Gantt*

Como se pode observar pelo diagrama, o plano temporal do trabalho realizado coincidiu mais ou menos com o plano temporal previsto. No entanto, foram necessárias algumas alterações e a adição de uma nova tarefa de maneira a responder a dificuldades que foram surgindo.

Capítulo 4

4 Implementação da solução

Neste capítulo podem ser observados os procedimentos usados na implementação e verificação da solução, previamente estruturada.

4.1 *Software* usado

4.1.1 Android Studio

Integrated Development Environment (IDE) usado para desenvolver o código e compilar a aplicação. Este IDE foi escolhido por ser um dos IDEs mais completos para desenvolver aplicações para a plataforma *Android* e por ser o IDE recomendado pela *Google*.

O *Android Studio* tem as vantagens de permitir *debugging* em tempo real quando conectado por um cabo *usb* ao dispositivo, de compilar diretamente o código em *Android Package* (*.apk*) e de ter embutido um sistema de criação e emulação de dispositivos *Android* [11].

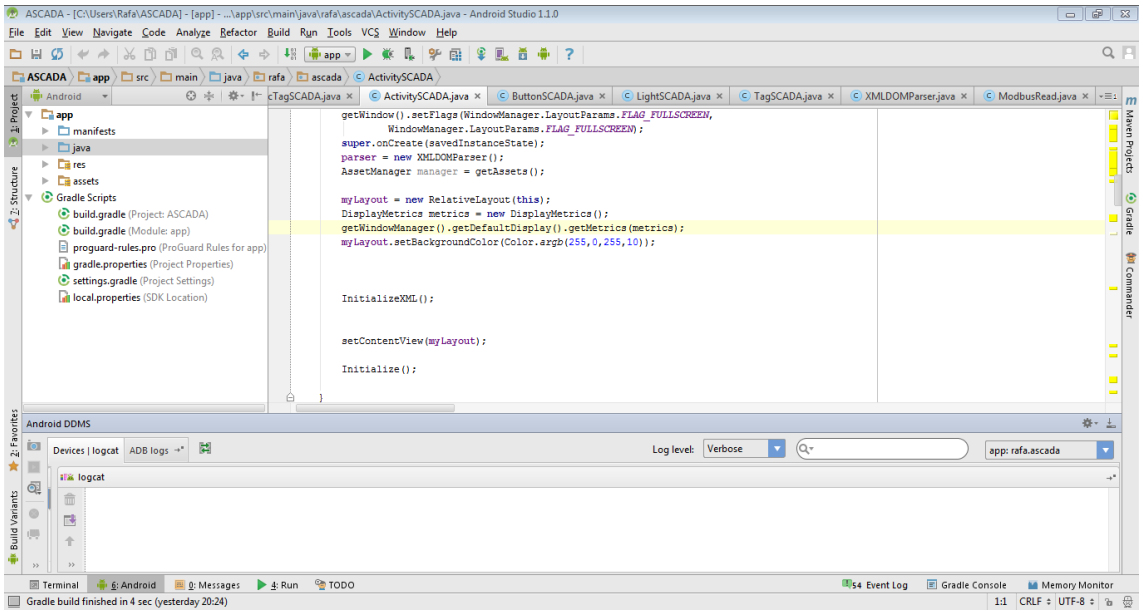

Figura 19 - *Android Studio*

4.1.2 Linha de Produção Flexível

A linha de produção flexível é uma linha de produção de peças que existe no laboratório de automação da FEUP. Esta linha é composta por cinco módulos como se pode ver pela figura 20.

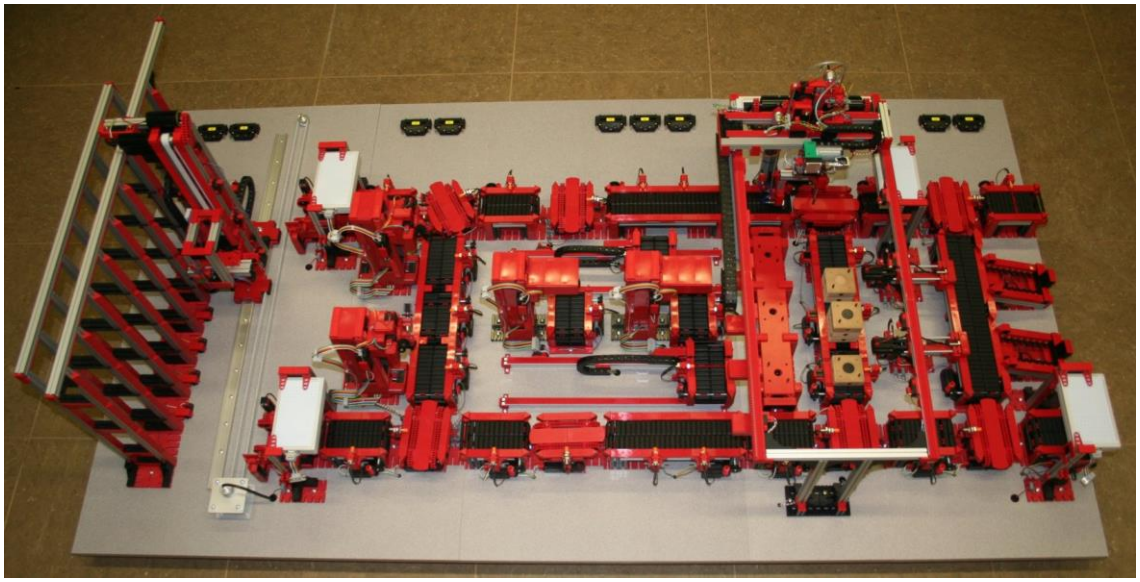

Figura 20 - Linha de produção flexível [12]

Estes módulos são, da esquerda para a direita, o armazém, a célula de maquinação em série, a célula de maquinação em paralelo, a célula de montagem e a célula de cargas e descargas.

4.1.2.1 Armazém

O armazém é o local onde são armazenadas as peças da linha de produção flexível. É possível inserir e retirar peças do armazém sejam elas simples ou compostas.

O armazém é constituído por dois tapetes lineares (um para a inserção de peças no armazém e outro para a remoção de peças do armazém), uma *stacker* ('torre' vertical) e dezoito posições para armazenamento de peças (distribuídas em colunas e linhas), como se pode observar pela figura 21.

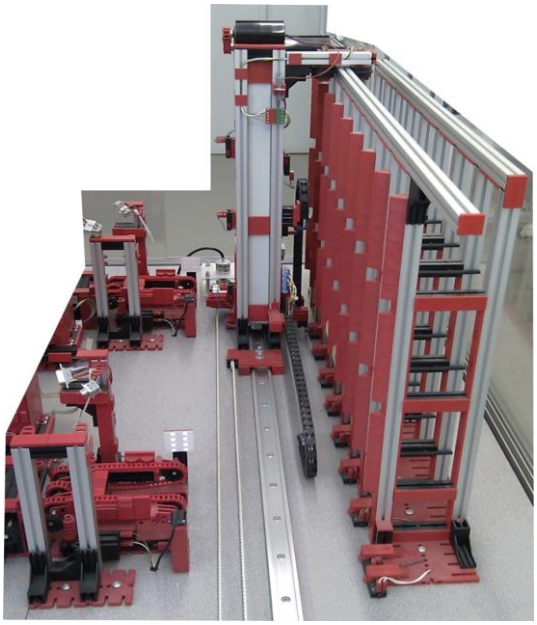

Figura 21 - Armazém da linha de produção flexível [12]

4.1.2.2 Célula de maquinação em série

A célula de maquinação em série é uma das células onde são maquinadas as peças simples. Estas peças simples são maquinadas nas duas máquinas presentes na célula. O tipo de peças simples resultante da maquinação depende do tempo de maquinação e da ferramenta que a máquina esteja a usar.

Como se pode observar pela figura 22 a célula de maquinação em série é constituída por cinco tapetes lineares e dois tapetes rotativos todos equipados com um sensor de presença. Estes tapetes permitem a movimentação das peças pela célula bem como saber a posição das peças na célula. É também constituído por duas máquinas, cada uma com três opções de ferramentas de maquinação diferentes. Estas máquinas movem-se pelo eixo dos x e dos z de maneira a se posicionarem corretamente sobre as peças a maquinar.

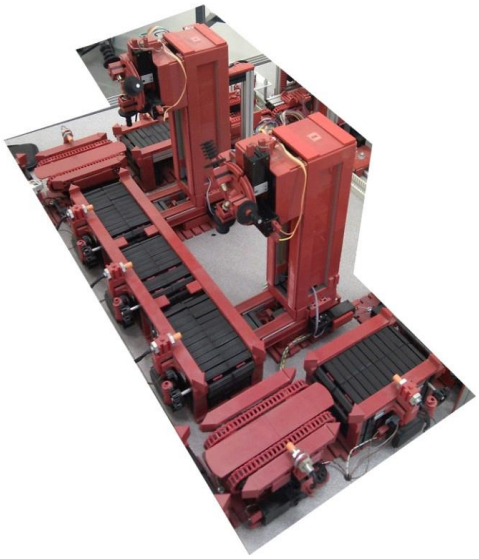

Figura 22 - Célula de maquinação em série da linha de produção flexível [12]

4.1.2.3 Célula de maquinação em paralelo

A célula de maquinação em paralelo é a outra célula onde são maquinadas as peças simples. Estas peças simples também são maquinadas nas duas máquinas presentes na célula. Tal como acontece na célula de maquinação em série, o tipo de peças simples resultante da maquinação também depende do tempo de maquinação e da ferramenta que a máquina esteja a usar.

Como se pode observar pela figura 23, esta célula é constituída por cinco tapetes lineares, dois tapetes rotativos e dois tapetes deslizantes que possibilitam a movimentação das peças por toda a célula. O uso destes tapetes deslizantes permite a maquinação simultânea e paralela de duas peças diferentes. É também constituído por duas máquinas iguais às máquinas presentes na célula de maquinação em série.

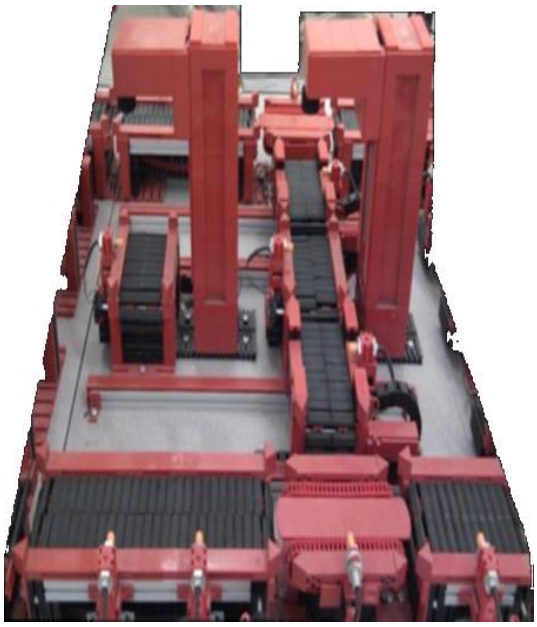

Figura 23 - Célula de maquinação em paralelo da linha de produção flexível [12]

4.1.2.4 Célula de montagem

A célula de montagem é a célula onde as peças simples são transformadas em peças compostas. Estas peças compostas são um conjunto de peças simples empilhadas, umas em cima das outras (máximo três peças simples).

Como se pode observar pela figura 24, esta célula é constituída por quatro tapetes lineares, dois tapetes rotativos e três mesas de trabalho. Estas mesas de trabalho são onde as peças são empilhadas. A célula é também constituída por um robô 3D (movimenta-se nos três eixos), que pega nas peças simples e as empilha em cima das peças que tiverem na mesa de trabalho criando peças compostas. Este robô também pega nas peças compostas, que se encontram nas mesas de trabalho, e transporta-as para os tapetes rotativos para estas poderem movimentar-se para a célula de descarga ou para o armazém.

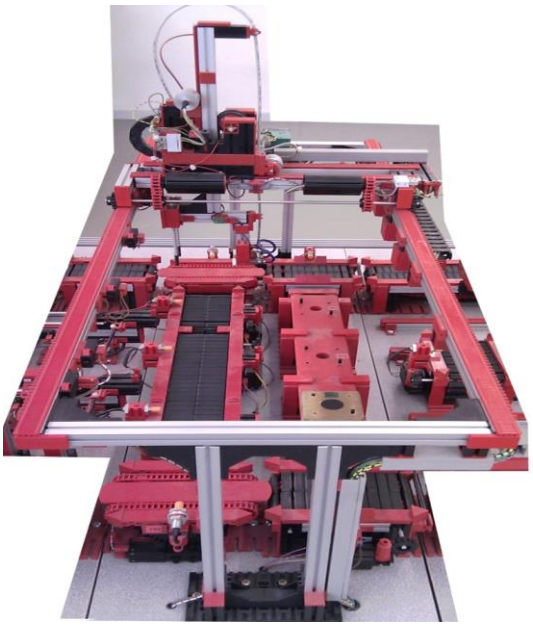

Figura 24 - Célula de montagem da linha de produção flexível [12]

4.1.2.5 Célula de cargas e descargas

A célula de cargas e descargas é a célula onde são as peças carregadas ou descarregadas. A operação de carga é realizada manualmente através da colocação de peças nos tapetes lineares, no extremo da célula. Por outro lado a operação de descarga é realizada também manualmente através da remoção das peças que se encontrem nos tapetes de descarga.

Como se pode observar pela figura 25, a célula é constituída por seis tapetes lineares, dois tapetes rotativos e dois tapetes de descarga. Estes tapetes de descarga são tapetes sem motor, que servem apenas para armazenar as peças até estas serem removidas manualmente. A célula é também constituída por dois *pushers*, que servem para empurrar as peças nos tapetes lineares para os tapetes de descarga.

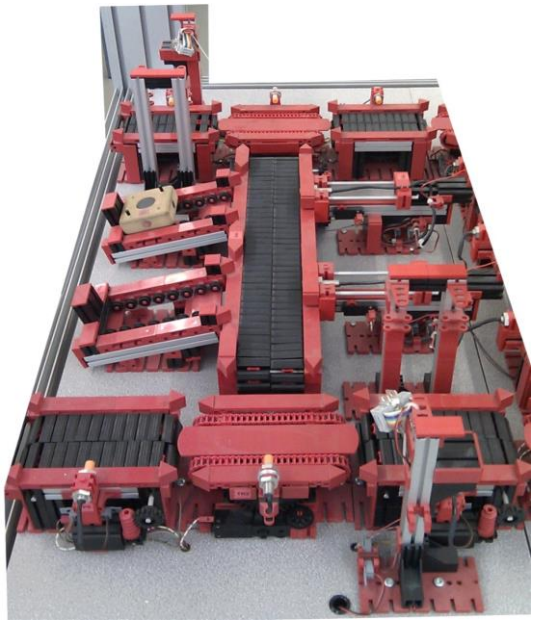

Figura 25 - Célula de cargas e descargas da linha de produção flexível [12]

4.1.2.6 Simulador

No entanto como nem sempre é possível aceder à linha de produção física, um simulador da fábrica pode ser usado. Este simulador simula toda a linha de produção bem como as comunicações *Modbus* que o sistema físico utiliza.

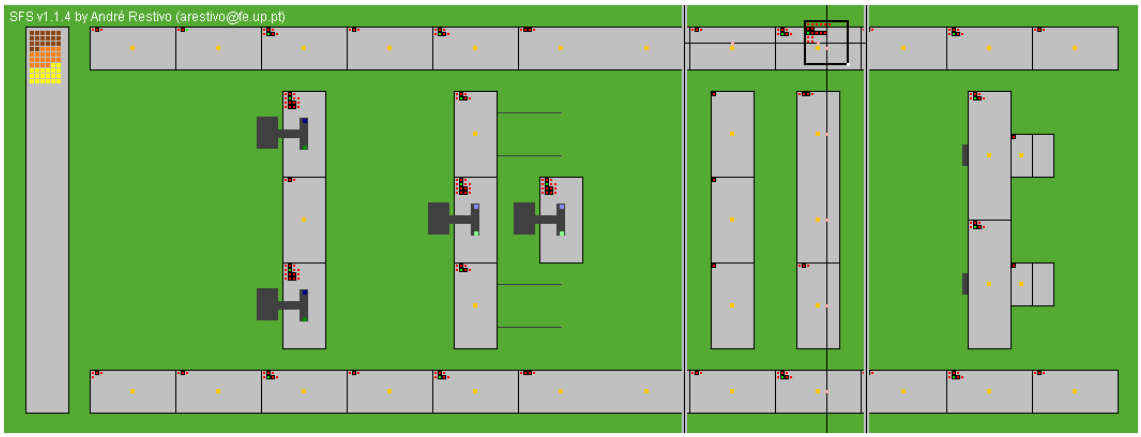

Figura 26 - Simulador da linha de produção flexível

4.1.3 Unity Pro XL

Software que permite programar usando linguagem *IEC61131-3*, bem como simular um *PLC* no computador. Este *software* foi usado em conjunto com o simulador, de maneira a automatizar a linha de produção apresentada no ponto anterior. A vantagem de automatizar a linha de produção é poder apresentar no ecrã do dispositivo *Android*, simulações de movimentos das peças e não apenas dos sensores da fábrica.

O *Unity Pro XL* foi usado por ser um *software* de fácil compreensão e bastante completo, tendo em conta as necessidades do nosso sistema.

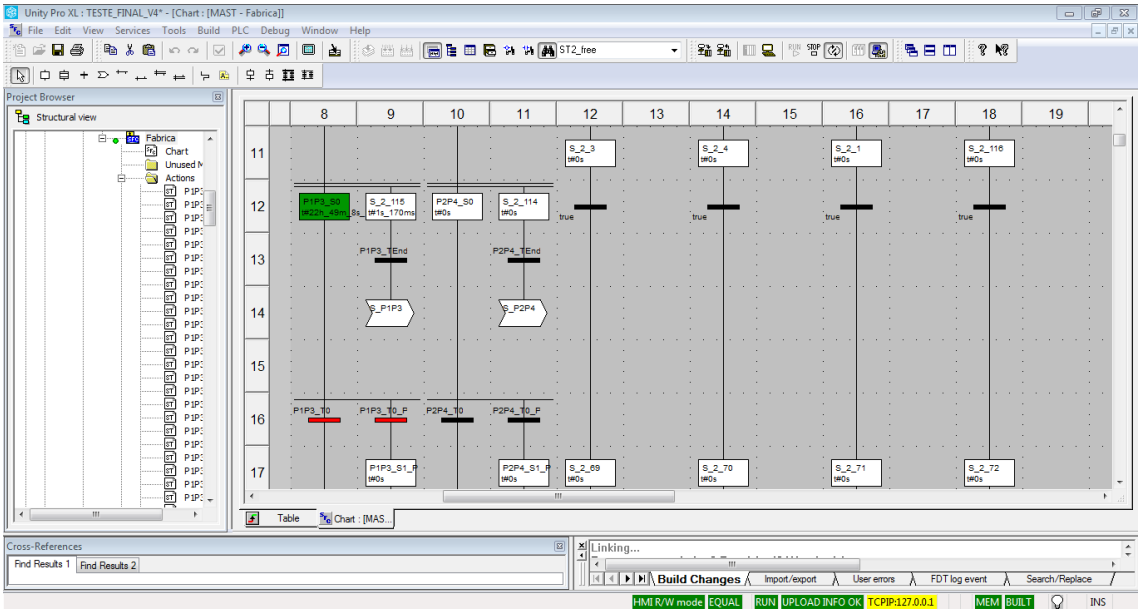

Figura 27 - *Unity Pro XL* em funcionamento

4.2 *Hardware* usado

4.2.1 Computador portátil

Computador usado para desenvolver o código do projeto. Foi usado também para correr o programa do *Unity Pro XL* e o simulador da linha de produção.

Além da utilização referida anteriormente o computador foi usado também para correr o emulador de dispositivos *Android*. A utilização do emulador permitiu testar o sistema numa grande gama de dispositivos. Esta gama de dispositivos de teste variou desde os de gama baixa até aos de gama alta e desde os dispositivos com ecrãs mais reduzidos até aos dispositivos com ecrãs maiores.

Por último, o computador foi usado para compilar e transferir a aplicação para os dispositivos físicos e para realizar operações de *debbuging* via cabo *usb*.

4.2.2 Smartphone E-Star x35

Dispositivo físico usado para testar o sistema. Este dispositivo foi escolhido por ser um dispositivo de gama baixa e com um tamanho de ecrã reduzido [13]. Este dispositivo permite testar como o sistema se comporta quando usado num dispositivo físico de gama baixa. Permite também observar como se comporta a função de manter a proporcionalidade em todos os tamanhos de ecrã.

4.2.3 Tablet E-Star Beauty Dual-Core

Dispositivo físico também usado para testar o sistema. Este dispositivo também foi escolhido por ser de baixa gama [14]. No entanto este dispositivo, ao contrário do *smartphone,* foi escolhido por ter um tamanho de ecrã médio. Este dispositivo permite então testar o sistema noutro dispositivo de gama baixa, mas permite também testar como se comporta a função de manter a proporcionalidade, num ecrã de tamanho médio.

4.3 Trabalho desenvolvido

4.3.1 *Master Terminal Unit*

A *Master Terminal Unit* foi programada de maneira a ser orientada a objetos, tornando-a modular. Esta programação, orientada a objetos, permite que as *tags* atualizem os objetos que dependem delas, sem ter qualquer conhecimento dos métodos usados para isso. Permite também que a adição de novas classes passe apenas por criar a classe em si e alterar a função que lê o ficheiro de configuração.

Ao nível da *thread* que realiza as comunicações, usando o protocolo *Modbus TCP/IP,* foi integrada uma biblioteca de comunicações *Modbus* chamada *jamod* [15]. Esta biblioteca permite realizar os pedidos necessários através do protocolo *Modbus TCP/IP*. A biblioteca apresentou bons resultados e por ser *open source* permite a alteração de código, quando necessário.

Ao nível da *thread* que realiza a atualização da interface, foram criadas classes para cada tipo de objetos. Estas classes são referenciadas pela *tag* respetiva, podendo assim as *tags* enviar pedidos de atualização para os objetos que dependem delas. Estas *tags* podem ser de dois tipos diferentes:

- *DigitalTagSCADA* (implementação da *tag* digital dos sistemas SCADA)
  - Dois estados possíveis (*True*/*False*);
  - Nome da *tag* – *string*;
  - ID da *coil* – *integer*;
- *NumericTagSCADA* (implementação da *tag* digital dos sistemas SCADA)
  - Duzentos e cinquenta e seis estados possíveis (0…255);
  - Nome da *tag* – *string*;
  - ID do registo – *integer*;

4.3.2 Interface Homem/Máquina

A *interface Homem/Máquina foi* programada de maneira a manter a proporcionalidade dos objetos em diferentes tamanhos de ecrã, como se pode ver pelas figuras 28 e 29. Esta proporcionalidade permite ao utilizador ter uma experiência similar em qualquer dispositivo, aquela do dispositivo para onde foi desenhada inicialmente a interface.

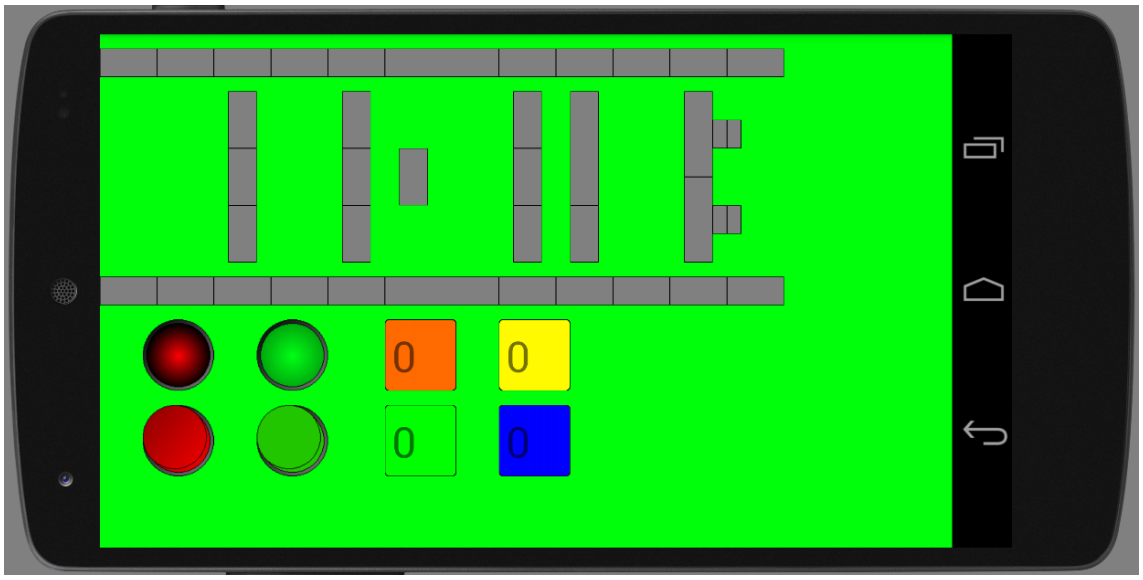

Figura 28 - Sistema SCADA num ecrã de 4,95 polegadas

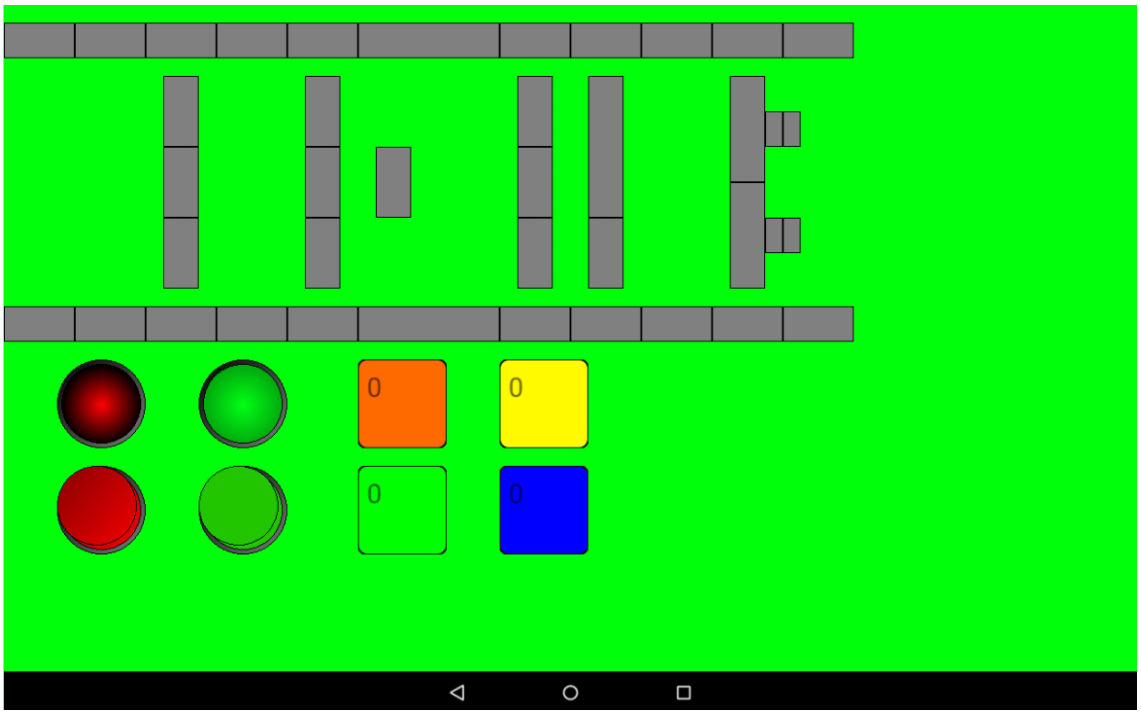

Figura 29 - Sistema SCADA num ecrã de 10,1 polegadas

Este interface permite também o uso de cinco tipos de objetos diferentes que podem ter o seu estado alterado, por pedidos realizados pela MTU. Estes cinco objetos são:

- *ButtonSCADA* (implementação do objeto botão dos sistemas SCADA)
  - Dois estados possíveis (Ligado/Desligado) – *tag* digital;
  - Dois tipos possíveis (*Switch*/Pressão) - *byte*;
  - Escrita de *coil* – *tag* digital;
  - Posição no eixo dos Z – *integer*;

- *LightSCADA* (implementação do objeto led dos sistemas SCADA)
  - Dois estados possíveis (Ligado/Desligado) – *tag* digital;
  - Posição no eixo dos Z – *integer*;
- *ImageSCADA* (implementação do objeto imagem dos sistemas SCADA)
  - Um estado possível;
  - Dois estados de visibilidade (Visível/Invisível) – *tag* digital;
  - Movimento pelo eixo dos x – *tag* numérica;
  - Movimento pelo eixo dos y – *tag* numérica;
  - Posição no eixo dos Z – *integer*;
- *ShapeSCADA* (implementação do objeto forma dos sistemas SCADA)
  - Vários estados possíveis (gradiente de duas cores) – *tag* numérica;
  - Três tipos possíveis (Retângulo/Retângulo redondo/Oval) - *byte*;
  - Dois estados de visibilidade (Visível/Invisível) – *tag* digital;
  - Movimento pelo eixo dos x – *tag* numérica;
  - Movimento pelo eixo dos y – *tag* numérica;
  - Posição no eixo dos Z – *integer;*
- *TextSCADA* (implementação do objeto texto dos sistemas SCADA)
  - Um ou vários estados possíveis (estático/variável) – *tag* numérica;
  - Dois estados de visibilidade (Visível/Invisível) – *tag* digital;
  - Movimento pelo eixo dos x – *tag* numérica;
  - Movimento pelo eixo dos y – *tag* numérica;
  - Posição no eixo dos Z – *integer*;

4.3.3 Configuração

Ao nível da configuração, foi criado um ficheiro XML que a *Master Terminal Unit* consegue interpretar. A *Master Terminal Unit* usa então este ficheiro para inicializar os objetos da interface, bem como todos os seus parâmetros (estados, imagens, texto, cores de fundo, cores do texto, tipos de forma, posição, *depth*, etc). Usa também o ficheiro para inicializar as *tags* digitais e numéricas, das quais os objetos da interface dependem.

Além disto o ficheiro permite ainda configurar o IP, a porta e o atraso do ciclo da *thread ModbusRead*. Permite também configurar o endereço inicial e a quantidade de entradas, saídas e registos a ler pela *Master Terminal Unit*.

4.3.4 Primeira utilização do sistema

Quando o sistema é utilizado pela primeira vez, este cria um diretório de pastas público, como se pode ver pela figura 30. Este diretório é composto por duas pastas chamadas *Config* e *Drawables*, onde serão colocados, respetivamente, o ficheiro de configuração inicial e as imagens.

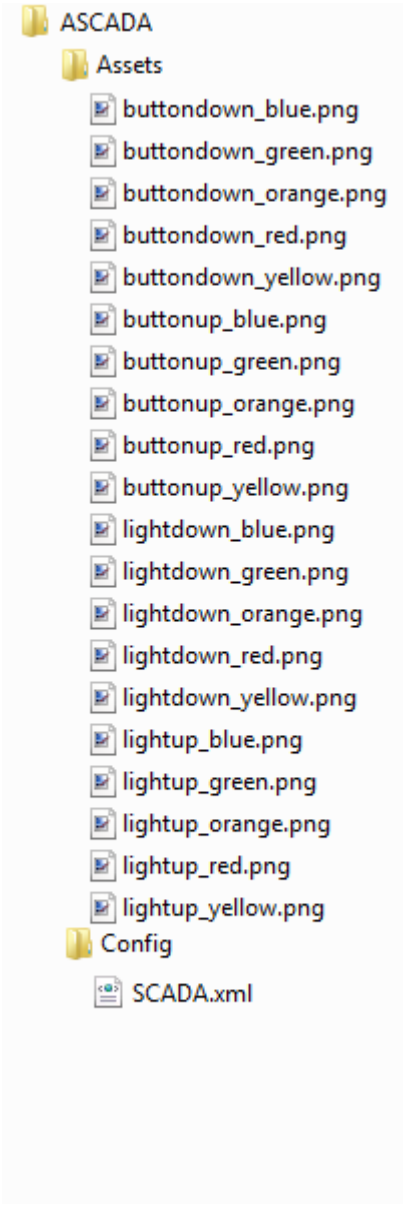

Figura 30 – Diretório de pastas

O ficheiro de configuração criada na primeira utilização contém então exemplos de como definir cada objeto, bem como uma explicação para cada um dos exemplos. Este ficheiro pode ser consultado nos anexos da dissertação.

As imagens criadas na primeira utilização são o conjunto de botões e luzes que se encontram na Figura 31. Estas luzes e botões permitem ao utilizador criar aplicações de teste básicas.

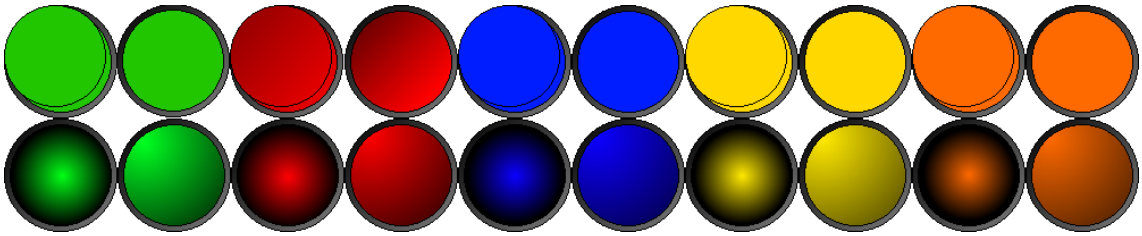

Figura 31 - Botões e luzes criados na primeira utilização do sistema

4.3.5 Verificação da *Master Terminal Unit* em conjunto com a Interface Homem/Máquina

De maneira a verificar a *Master Terminal Unit* em conjunto com a interface foi criado um programa do *Unity Pro XL* em conjunto com o simulador da linha de produção. O programa do *Unity* em conjunto com o simulador da linha de produção simula uma *Remote Terminal Unit*. Esta *Remote Terminal Unit* comunica com a *Master Terminal Unit* usando o protocolo *Modbus TCP/IP*. Por sua vez, a *Master Terminal Unit* comunica à interface alterações que sejam necessárias realizar.

O programa do *Unity* usa *Sequencial Function Charts* (SFC) e *Structured Text* (ST) para controlar a produção de peças na célula de maquinação de série. Este programa controla também a movimentação das peças desde a célula de maquinação em série até à célula de cargas e descargas.

Para criar este programa de teste foi necessário usar mais de quatrocentas *tags* digitais e oito registos. A *Master Terminal Unit* usa então os valores destas *tags* para alterar o estado dos objetos da interface, como se pode observar pelas Figuras 32 e 33.

A interface Homem/Máquina foi criada de maneira a ser semelhante ao simulador da linha de produção. Esta interface foi então constituída por:

- Formas retangulares para todos os tapetes;
- Imagens para cada estado possível dos tapetes (Frente/Trás/Roda para a direita/Roda para a esquerda);
- Imagens para cada estado possível das máquinas (Trabalhar/Desligada/Mover para cima/Mover para baixo/Mover para a direita/Mover para a esquerda);
- Formas para cada posição possível das peças;
- Botões para paragem de emergência e início do sistema;
- Luzes para indicar o estado do sistema (A funcionar/Parado);
- Formas e texto para cada uma das quatro peças que podem ser produzidas, para indicar os números de peças produzidas;

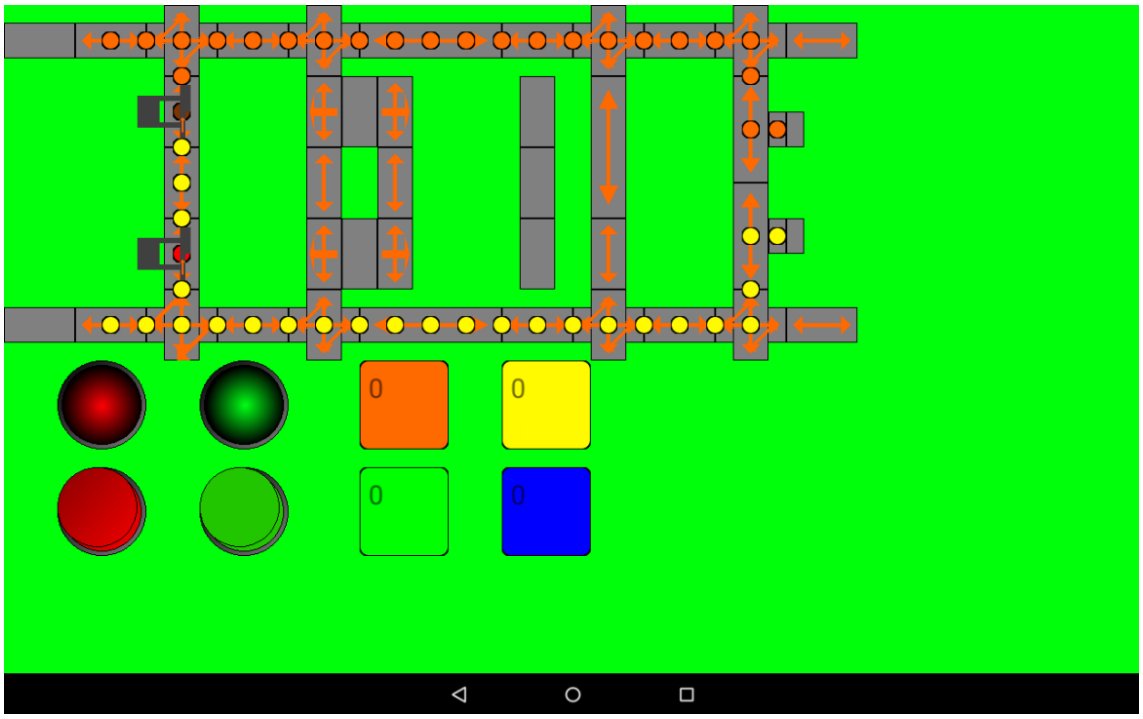

Figura 32 - Sistema SCADA completo sem as *tags* de visibilidade

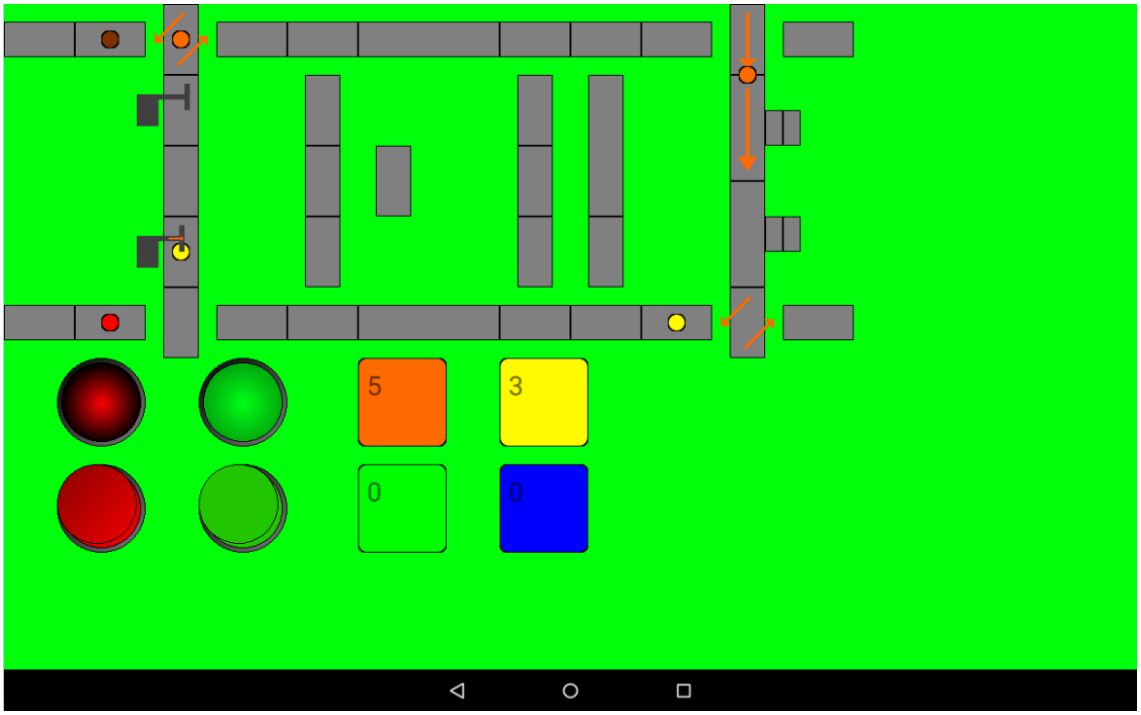

Figura 33 - Sistema SCADA completo com as *tags* de visibilidade e a funcionar

4.4 Testes

4.4.1 Funcionalidade

Para verificar a ocorrência de erros foi criado um objeto luz especial, com dois estados possíveis (luz verde/luz vermelha). Este objeto mostra uma luz verde quando não há nenhuma ocorrência de erros e mostra uma luz vermelha quando existem erros.

Foi então executado o programa de seis em seis horas, num total vinte e quatro horas, e verificado o estado da luz. Foi também executado o programa de seis em seis horas, num total vinte e quatro horas, mas desta vez com pedidos constantes de escrita de *coils* e verificado o estado da luz.

No caso da luz se encontrar vermelha foram observados os *logs* do *Android Studio*, para se perceber qual tinha sido o erro. Os resultados obtidos foram os seguintes:

Tabela 4 - Teste de funcionalidade sem pedidos de escrita de *coils*

Horas	Estado da Luz	Erro
0 - 6 horas	Verde	Nenhum
6 - 12 horas	Vermelho	Erro a conectar-se ao escravo *Modbus*
12 - 18 horas	Verde	Nenhum
18 - 24 horas	Verde	Nenhum

Tabela 5 - Teste de funcionalidade com pedidos constantes de escrita de *coils*

Horas	Estado da Luz	Erro
0 - 6 horas	Verde	Nenhum
6 - 12 horas	Verde	Nenhum
12 - 18 horas	Verde	Nenhum
18 - 24 horas	Verde	Nenhum

4.4.2 Performance

Para testar a performance do sistema foi usado o dispositivo físico *E-star X35* e a aplicação "Cool Tool - system stats", para registar o valor de processador e RAM livre com e sem o uso do sistema SCADA, e obtive os seguintes resultados:

Tabela 6 - Performance do dispositivo com e sem o sistema SCADA

	S/ SCADA			C/ SCADA		
	Min	Média	Max	Min	Média	Max
CPU (%)	0	2	6	0	4	8
RAM Livre (MB)	106	109	117	81	82	85

Foi medido também o tempo que o programa demora a executar, com e sem o tempo da comunicação *Modbus* (120 entradas/320 saídas/4 registos), em milissegundos e obtive os seguintes resultados:

Tabela 7 - Tempo de execução do sistema com e sem *Modbus*

	S/ *Modbus*			C/ *Modbus*		
	Min	Média	Max	Min	Média	Max
Tempo de execução (ms)	0	0	1	63	70	103

4.4.3 Escalabilidade

Para testar a robustez do sistema foi usado o mesmo dispositivo e a mesma aplicação dos testes anteriores, mas desta vez foi usado um ficheiro de configuração com três mil objetos *LightSCADA*. Todos estes objetos dependem da mesma *tag*. Depois de iniciado o sistema, foi medida a performance com o uso do sistema SCADA. Foi medida também a performance aquando da alteração do valor lógico da *tag* de qual dependem os três mil objetos e obtive os seguintes resultados:

Tabela 8 - Performance do dispositivo com sistema SCADA e com ou sem alteração do valor da *tag*

	C/ SCADA e S/*tag*			C/ SCADA e C/*tag*		
	Min	Média	Max	Min	Média	Max
CPU (%)	0	3	8	26	35	40
RAM Livre (MB)	77	78	80	51	53	60

Foi medido também o tempo que demora o programa a executar, com e sem o tempo da comunicação *Modbus* (1 entrada), aquando da alteração do valor da *tag* e obtive os seguintes resultados:

Tabela 9 - Tempo de execução do sistema com e sem *Modbus* aquando da alteração do valor da *tag*

	S/ *Modbus*			C/ *Modbus*		
	Min	Média	Max	Min	Média	Max
Tempo de execução (ms)	0	0	2	12	16	23

Capítulo 5

5 Conclusão e trabalho futuro

Neste capítulo são apresentadas as conclusões retiradas deste projeto e o trabalho que pode ser realizado futuramente para melhorar e completar o sistema.

5.1 Conclusão

Depois de terminado o sistema foram retiradas algumas conclusões do mesmo.

A primeira conclusão retirada foi que o sistema cumpria todos os requisitos estipulados pelo cliente. Apenas foi possível atingir este resultado porque o sistema tinha sido bem pensado e estruturado.

Outra conclusão retirada foi que o sistema se mantem funcional mesmo quando são executados pedidos constantes de escrita de *coils*. O sistema falhou apenas uma vez, no entanto esta falha deveu-se a uma falha do escravo e não da *Master Terminal Unit*.

Observando a Tabela 6, outra conclusão que foi retirada, foi que o sistema se comporta como esperado e apresenta uma boa performance quando comparado ao telemóvel sem nenhuma aplicação aberta, isto usando um dispositivo de gama baixa.

Observando a Tabela 8, a conclusão retirada foi que mesmo sobrecarregando o sistema com uma enorme quantidade de objetos ele mantém uma boa performance, diminuindo apenas quando os três mil objetos são atualizados ao mesmo tempo.

No entanto, observando a Tabela 9 na coluna sem as comunicações *Modbus*, verificamos que apesar do aumento do uso da CPU aquando da alteração do valor da *tag* o tempo de execução manteve-se idêntico. Observando agora as tabelas 7 e 9 em conjunto e comparando os valores com e sem o uso do *Modbus* verificamos que o *bottleneck* do sistema é as comunicações *Modbus*. Para verificar isto foi usado o *Wireshark*, que mostrou que o tempo de execução com o uso das comunicações *Modbus* era semelhante ao tempo de resposta do escravo. A conclusão retirada disto foi que o tempo de execução elevado se devia a processamento no computador e não ao tempo de transmissão do pedido ou de receção da resposta.

A última conclusão retirada foi que apesar da programação em *Android* ter algumas funcionalidades únicas e não ter qualquer experiência passada a programar em *Java* ou em

Android, consegui resolver os problemas à medida que foram aparecendo e concluir o programa. Isto deveu-se apenas ao meu gosto por programar, a uma capacidade critica e alguma ajuda do meu orientador.

5.2 Trabalho futuro

Apesar do *Master Terminal Unit* e da interface encontrarem-se funcionais e com ótima performance é sempre possível melhorá-los.

Ao nível da interface seria vantajoso adicionar mais objetos extra. Esta adição proporciona ao utilizador um maior número de opções para apresentar e tratar os dados do processo.

Ainda ao nível da interface seria também vantajoso criar uma aplicação que possibilitasse a configuração da interface recorrendo a métodos gráficos.

Ao nível do *Master Terminal Unit* seria uma mais valia incluir novos protocolos de comunicação. Estes novos protocolos permitiriam o uso do *Master Terminal Unit* para utilizadores que não possuem comunicações que suportem o protocolo *Modbus TCP/IP*.

Seria também vantajoso adicionar ao *Master Terminal Unit* o suporte para alarmes e a criação de *logs* e *backups*.

Referências

[1] “Android (operating system)”. Disponível em https://en.wikipedia.org/wiki/Android_(operating_system). Acesso em 16/Fevereiro/2015.

[2] “Learning Android”, Marko Gargenta, Março de 2011.

[3] “Smartphone OS Market Share, Q1 2015”. Disponível em http://www.idc.com/prodserv/smartphone-os-market-share.jsp. Acesso em 23/Julho/2015

[4] “CS 355 - Systems Programming: Android + Sphero”. Disponível em http://www.cs.ccsu.edu/~stan/classes/CS355/notes/02-AndroidAppArchitecture.html. Acesso em 23/Julho/2015

[5] “Learn Android App Development”, Wallace Jackson, Maio de 2013.

[6] “What is Android”. Disponível em http://www.edureka.co/blog/what-is-android/. Acesso em 23/Julho/2015.

[7] “The Benefits of Networked SCADA Systems Utilizing IP Enabled Networks”, McClanahan, Maio de 2002

[8] “SCADA: Supervisory Control And Data Acquisition”, Stuart A. Boyer, 2004

[9] “TECHNICAL INFORMATION BULLETIN 04-1 - Supervisory Control and Data Acquisition (SCADA) Systems”, Office of the Manager National Communications System, Outubro de 2004

[10] “MODBUS in a Nutshell”. Disponível em http://gridconnect.com/blog/tag/modbus-tutorial/. Acesso em 23/Julho/2015

[11]“Android Developer Tools”, Donn Felker, Agosto de 2013.

[12]“Linha de Produção Flexível”, Faculdade de Engenharia da Universidade do Porto, 2012.

[13]“eSTAR X35 3.5″ Specifications”. Disponível em http://www.estar.eu/en/estar-x35-3-5-specifications/. Acesso em 16/Fevereiro/2015

[14]“eSTAR BEAUTY Dual Core 7.0″ Specifications”. Disponível em http://www.estar.eu/en/estar-beauty-dual-core-7-0-specifications/. Acesso em 16/Fevereiro/2015

[15]“Java Modbus Library (jamod)”. Disponível em http://jamod.sourceforge.net/. Acesso em 16/Fevereiro/2015

Anexos

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SCADA>
    <!-- Help -->
    <!-- tags' addresses can be discrete inputs, output coils and discrete input registers -->
    <!-- tags' addresses of the type discrete input should be equal to its addresses in modbus. example: TagID = 20 | Modbus address = 20 -->
    <!-- tags' addresses of the type output coils should be equal to its addresses in modbus plus 10000. example: TagID = 10050 | Modbus address = 50 -->
    <!-- tags' addresses of the type discrete input registers should be equal to its addresses in modbus plus 20000. example: TagID = 20030 | Modbus address
    = 30 -->
    <!-- color's value should be in hex values and must follow the patern AARRGGBB. example: semi transparent red | 80FF0000 -->
    <!-- AA can go from 00 to FF and is the value of the alpha (00 -> trasparent, FF -> opaque) -->
    <!-- RR can go from 00 to FF and is the value of the red present in the color -->
    <!-- GG can go from 00 to FF and is the value of the green present in the color -->
    <!-- BB can go from 00 to FF and is the value of the blue present in the color -->
    <!-- image name must include the type extension. example: buttonupred.png -->
    <!-- all images must be inside Drawables folder -->

    <!-- configurations of the modbus' communications -->
    <SCADAModbus>
        <ip></ip> <!-- ip for modbus' communications -->
        <port></port> <!-- port for modbus' communications -->
        <delay></delay> <!-- delay for modbus' communications -->
        <inputStartAddress></inputStartAddress> <!-- start address of discrete inputs to read -->
        <inputQuantity></inputQuantity> <!-- quantity of discrete inputs to read -->
        <outputStartAddress></outputStartAddress> <!-- start address of output coils to read -->
        <outputQuantity></outputQuantity> <!-- quantity of output coils to read -->
        <inputRegisterStartAddress></inputRegisterStartAddress> <!-- start address of discrete input registers to read -->
        <inputRegisterQuantity></inputRegisterQuantity> <!-- quantity of discrete input registers to read -->
    </SCADAModbus>

    <!-- configurations of the layout -->
    <SCADALayout>
        <BackgroundColor></BackgroundColor> <!-- background color for the application -->
    </SCADALayout>

  <!-- configurations of the digital tags -->
  <SCADADigitalTag>
        <TagID></TagID> <!-- address of the tag -->
        <TagName></TagName> <!-- name of the tag -->
        <TagDescription></TagDescription> <!-- description of the tag -->
  </SCADADigitalTag>

  <!-- configurations of the numeric tags -->
  <SCADANumericTag>
        <TagID></TagID> <!-- address of the tag -->
        <TagName></TagName> <!-- name of the tag -->
        <TagDescription></TagDescription> <!-- description of the tag -->
  </SCADANumericTag>

  <!-- configurations of the light objects  -->
  <SCADALight>
        <TagID></TagID> <!-- address of the tag used to update light's state -->
        <imageon></imageon> <!-- image used when light is on -->
        <imageoff></imageoff> <!-- image used when light is off -->
        <x></x> <!-- position of the light in the x's axe -->
        <y></y> <!-- position of the light in the y's axe -->
        <size_x></size_x> <!-- size of the light in the x's axe -->
        <size_y></size_y> <!-- size of the light in the y's axe -->
        <depth></depth> <!-- depth of the light in the z's axe -->
  </SCADALight>
```

```
    <!-- configurations of the button objects  -->
    <SCADAButton>
        <TagID></TagID> <!-- address of the tag used to update button's state and to write single coil -->
        <imageon></imageon> <!-- image used when button is on -->
        <imageoff></imageoff> <!-- image used when button is off -->
        <type></type> <!-- type of button (0 -> switch, 1 -> pressure) -->
        <x></x> <!-- position of the button in the x's axe -->
        <y></y> <!-- position of the button in the y's axe -->
        <size_x></size_x> <!-- size of the button in the x's axe -->
        <size_y></size_y> <!-- size of the button in the y's axe -->
        <depth></depth> <!-- depth of the button in the z's axe -->
    </SCADAButton>

    <!-- configurations of the image objects  -->
    <SCADAImage>
        <TagVisibilityID></TagVisibilityID> <!-- address of the tag used to update image's visibility -->
        <TagXID></TagXID> <!-- address of the tag used to update image's position in the x's axe -->
        <TagYID></TagYID> <!-- address of the tag used to update image's position in the y's axe -->
        <Image></Image> <!-- image to display -->
        <x></x> <!-- position of the image in the x's axe -->
        <y></y> <!-- position of the image in the y's axe -->
        <size_x></size_x> <!-- size of the image in the x's axe -->
        <size_y></size_y> <!-- size of the image in the y's axe -->
        <depth></depth> <!-- depth of the image in the z's axe -->
    </SCADAImage>


    <!-- configurations of the text objects  -->
    <SCADAText>
        <TagID></TagID> <!-- address of the tag used to update text's value -->
        <TagVisibilityID></TagVisibilityID> <!-- address of the tag used to update text's visibility -->
        <TagXID></TagXID> <!-- address of the tag used to update text's position in the x's axe -->
        <TagYID></TagYID> <!-- address of the tag used to update text's position in the y's axe -->
        <TextDefault></TextDefault> <!-- default text to show -->
        <TextColor></TextColor> <!-- color of the text -->
        <TextSize></TextSize> <!-- size of the text -->
        <x></x> <!-- position of the text in the x's axe -->
        <y></y> <!-- position of the text in the y's axe -->
        <size_x></size_x> <!-- size of the text in the x's axe -->
        <size_y></size_y> <!-- size of the text in the y's axe -->
        <depth></depth> <!-- depth of the text in the z's axe -->
    </SCADAText>

    <!-- configurations of the shape objects  -->
    <SCADAShape>
        <TagID></TagID> <!-- address of the tag used to update shape's gradient -->
        <TagVisibilityID></TagVisibilityID> <!-- address of the tag used to update shape's visibility -->
        <TagXID></TagXID> <!-- address of the tag used to update shape's position in the x's axe -->
        <TagYID></TagYID> <!-- address of the tag used to update shape's position in the y's axe -->
        <StartColor></StartColor> <!-- start color of the shape's gradient -->
        <EndColor></EndColor> <!-- end color of the shape's gradient -->
        <StrokeColor></StrokeColor> <!-- color of the shape's border -->
        <StrokeWidth></StrokeWidth> <!-- width of the shape's border -->
        <x></x> <!-- position of the shape in the x's axe -->
        <y></y> <!-- position of the shape in the y's axe -->
        <size_x></size_x> <!-- size of the shape in the x's axe -->
        <size_y></size_y> <!-- size of the shape in the y's axe -->
        <depth></depth> <!-- depth of the shape in the z's axe -->
        <MinValue></MinValue> <!-- minimun value of the shape's gradient -->
        <MaxValue></MaxValue> <!-- maximun value of the shape's gradient -->
        <Type></Type> <!-- type of shape's form (0 -> rectangle, 1 -> rounded rectangle, 2 -> oval) -->
    </SCADAShape>
</SCADA>
```