Departamento de Engenharia Electrotécnica e de Computadores

Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Licenciatura em Tecnologias de Informação Visual

2007/2008

Interface de Simulação e Visualização da Instrumentação de Aeronaves não Tripuladas

Orientador: Doutor Paulo Coimbra

Cláudio Roberto Pereira Macedo Campos Fernandes, aluno nº 501032904

José Alexandre Vaz de Paiva Nunes, aluno nº 501032907

Tiago José Miranda dos Santos, aluno nº 501032935

Resumo

Neste relatório descreve-se o processo de desenvolvimento de um sistema informático que proporciona uma interface de simulação e visualização da instrumentação de aeronaves não tripuladas. O resultado do trabalho desenvolvido insere-se no âmbito do projecto DIVA (1). A plataforma informática desenvolvida compreende dois programas distintos:

- **Simulator**: Programa que possibilita a leitura e comunicação de dados, de uma aeronave, previamente guardados. A sua função principal é representar para a interface gráfica um dirigível virtual, simulando missões já realizadas.
- **DIVAgui:** Interface gráfica que permite a monitorização dos instrumentos de uma aeronave não tripulada. Na instrumentação virtual são apresentados velocímetros, um altímetro, um indicador de direcção e um horizonte artificial. O estado da comunicação da aeronave e dos seus sensores é indicado e a sua localização GPS é representada num mapa. A informação é actualizada e interpretada nas várias componentes da interface gráfica para que seja facilmente compreendida por parte do utilizador.

O sistema foi desenvolvido em C++, utilizando a *framework* Qt. Dada a complexidade e heterogeneidade dos sistemas envolvidas no processo, utilizou-se uma arquitectura padrão para a comunicação entre objectos: o protocolo CORBA.

A aplicação implementada foi testada com êxito em monitorização real do DIVA, produzindo bons resultados.

Agradecimentos

O relatório de projecto vem concluir um ciclo importante de formação, resultado de muito trabalho e sacrifício. Por isso gostaríamos de agradecer aos nossos pais pelo apoio e carinho, aos colegas e amigos pela alegria, amizade e bons momentos. À Carla e à Joana um agradecimento especial pela paciência e amor. Agradecemos também ao nosso orientador Dr. Paulo Coimbra e ao investigador Luiz Mirisola pela paciência e acompanhamento ao longo do projecto.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Diagramas

Lista de Abreviações

ACE – *Adaptive Communication Environment*

CDR – *Common Data Representation*

CORBA – *Common Object Request-Broken Architecture*

CVS – *Concurrent Version System*

DCE - *Distributed Computing Environment*

DCOM – *Distributed Communication Object Model*

DEEC – Departamento de Engenharia Electrotécnica e de Computadores

DIVA – Dirigível Instrumentado para Vigilância Aérea

DIVA GS – DIVA *Ground Station*

DSI - *Dynamic Skeleton Interface*

FCTUC – Faculdade de Ciências e Tecnologia da Universidade de Coimbra

gcc - GNU *Compiler Collection*

GIOP - *General Inter-ORB Protocol*

GNU - GNU's *Not Unix*

GUI – *Graphical User Interface*

ICE - *Information and Content Exchange*

IDL – *Interface Description Language*

IIOP - *Internet Inter-Orb Protocol*

IOR - *Interoperable Object Reference*

ISR – Instituto de Sistemas e Robótica

JRMP - *Java Remote Method Protocol*

MS – Microsoft

MSDN – Microsoft *Developer Network*

OOP – *Object Oriented Programming*

ORB – *Object Request Broker*

OSM – *Open Street Maps*

QWT – *Qt Widget for Technical Applications*

RMI – *Remote method invocation*

RPC - *Remote procedure call*

SSI - *Static Skeleton Interface*

SVN – Subversion

TAO – *The ACE Orb*

TCP/IP - *Transmission Control Protocol over Internet Protocol*

UC – Universidade de Coimbra

UML – *Unified Modeling Language*

WMS – *Web Map Service*

xml - *Extensible Markup Language*

XP – *Extreme Programming*

Capítulo 1. Introdução

Este relatório constitui a documentação principal de todo o processo de desenvolvimento de uma plataforma informática para a simulação, comunicação e visualização da instrumentação de uma aeronave não tripulada, no âmbito do projecto DIVA (1). Nesta plataforma constam as aplicações: Simulator e DIVAgui. A primeira consiste numa aplicação que faz a leitura de ficheiros de missão previamente criados e comunica estes dados ao servidor que os faz chegar à interface. A interface principal DIVAgui apresenta um conjunto de mecanismos de visualização e controlo importantes na óptica do utilizador.

1.1. Descrição Geral

A concepção de sistemas informáticos de comunicação e visualização para dispositivos aeronáuticos está sujeita a um grande número de factores críticos relacionados com a segurança e fiabilidade. É fundamental que a informação seja apresentada ao utilizador de forma válida e em tempo útil.

Nesta perspectiva, foi desenvolvido um *software* que permite a monitorização de fenómenos físicos e do comportamento de uma aeronave instrumentada. A plataforma consiste numa interface gráfica principal que apresenta um vasto conjunto de instrumentos virtuais e outros elementos de interacção/visualização. Foi contemplada também a possibilidade de simular missões anteriores, cuja funcionalidade é assegurada pela aplicação Simulator. O *software* de simulação está ligado ao desenvolvimento da interface gráfica principal e foi também construído no âmbito deste projecto. A comunicação entre o servidor e a aplicação é assegurada pela utilização de uma *framework* baseada na arquitectura CORBA.

Inicialmente o sistema foi preparado para o aparelho aéreo desenvolvido pelo projecto DIVA. Contudo, a aplicação não depende inteiramente do DIVA, pelo que a

comunicação e representação da informação é suficientemente versátil para a sua utilização noutras aeronaves.

1.2. Motivação

Actualmente o auxílio robótico tem vindo a desempenhar um papel mais relevante e abrangente, permitindo ao homem abstrair-se do esforço físico e de riscos inerentes a algumas actividades. A robótica aérea é uma área onde esse risco é mais evidente. Assim torna-se importante a utilização de aeronaves não tripuladas auxiliadas por instrumentação virtual. O seu domínio de aplicabilidade passa pela monitorização de estradas, detecção de incêndios ou mesmo a inspecção de cabos aéreos de distribuição de energia eléctrica.

Este projecto serve de apoio às aplicações da aeronave não tripulada, monitorizando e representando os seus comportamentos.

1.3. Objectivos

O principal objectivo do projecto é criar uma aplicação informática estável e segura que permitia a monitorização remota de uma aeronave. Deste modo, os nossos esforços foram orientados no sentido de cumprir os seguintes objectivos:

- Criação de uma interface gráfica que receba e apresente os dados num formato facilmente interpretado pelo utilizador;
- Criação de uma plataforma de simulação de missões que possa ser utilizada para rever ou analisar em pormenor dados específicos do voo;
- Desenvolvimento suficientemente versátil para a inclusão de funcionalidades cuja necessidade possa surgir no futuro;
- Desenvolvimento de um sistema de comunicação eficiente e fiável;
- Desenvolver a aplicação de forma a assegurar a sua portabilidade nos vários sistemas existentes.

1.4. Estrutura e organização do relatório

Este relatório está dividido em cinco capítulos, proporcionando uma descrição organizada do trabalho desenvolvido.

No capítulo primeiro é feita uma introdução ao projecto, através da descrição sucinta de todo o trabalho desenvolvido, bem como a apresentação dos nossos objectivos e motivações que estiveram na sua origem. Posteriormente, o capítulo segundo, é introduzido com o processo de desenvolvimento do *software*, que vai compreender toda a parte de engenharia de requisitos, design conceptual e as metodologias para as fases de implementação, testes e manutenção. Nesta fase são descritas as práticas de engenharia de *software* utilizadas no nosso projecto.

Depois de apresentado o tema, os objectivos e o processo de desenvolvimento da aplicação, é abordado no capítulo terceiro a escolha das tecnologias baseada na análise de requisitos e das tecnologias disponíveis. Feitas as escolhas, a arquitectura do sistema é descrita em diferentes abordagens.

No capítulo quarto são apresentadas as técnicas que foram utilizadas na implementação da aplicação. Finalmente, no último capítulo são descritos, de uma forma sumária, os objectivos atingidos, as competências adquiridas e conclusões a que chegámos com o desenvolvimento do trabalho.

Capítulo 2. Processo de desenvolvimento de *software*

O processo de desenvolvimento de *software* segue procedimentos e metodologias definidas com intenção de promover um ciclo contínuo de trabalho e pesquisa. Deste modo é possível a criação de sistemas informáticos capazes de responder às necessidades do cliente de uma forma simples e efectiva. Tal como qualquer processo de engenharia, a produção de *software* exige sempre um estudo cuidado de todas as necessidades, objectivos e restrições.

Com isto, ao longo da nossa actividade processual procurámos ter uma interacção constante com futuros utilizadores e membros de outras equipas de desenvolvimento envolvidas, com o objectivo de apreender os requisitos mais importantes que devem estar assegurados na aplicação final.

Este capítulo vai apresentar o resultado da planificação conceptual e técnica do nosso *software*. Vão ser analisados os requisitos principais da aplicação, a sua especificação técnica e serão definidas as metodologias para implementação, testes e técnicas de manutenção.

2.1. Análise de requisitos

No processo de engenharia de *software* identifica-se um conjunto de funcionalidades, qualidades e restrições que devem ser consideradas no desenvolvimento da aplicação. Um requisito simboliza um comportamento desejável durante a execução de um *software*.

A identificação de requisitos é essencial no planeamento e satisfação de um projecto de *software*.

2.1.1. Identificação de requisitos

Alguns requisitos foram determinados já na fase de implementação.

Segundo a taxonomia de ESW os requisitos podem ser divididos em quatro tipos principais:

- **Funcionais**: Descrevem o comportamento do sistema e a funcionalidades que este deve ter;
- **Não funcionais**: Descrevem qualidades ou características que o *software* deve apresentar;
- **Design**: Decisão ao nível da concepção, um requisito de componentes específicos de interface;
- **Processo**: Descrevem uma restrição sobre tecnologias ou processos no desenvolvimento do *software*.

De seguida apresenta-se uma tabela que identifica os requisitos principais da aplicação, o tipo e um índice de prioridade:

Nº	Descrição de requisito	Prioridade
1	Recepção de dados em tempo real da aeronave	* * * * *
2	Detecção de erros e de sensores presentes na aeronave	* * * * *
3	Detecção de dados inválidos	* * * * *
5	Configuração de nova missão	* * * *
6	Possibilidade de gravação de configuração da missão	* * * *
7	Configuração de mapas	* * * *
8	Configuração de referências para tanques e baterias	* * * * *
9	Configuração de comunicação	* *
10	Possibilidade de carregamento e configuração de mapas através de imagem	* * * *
11	Possibilidade de carregamento e configuração de mapas através de internet	* * *
12	Simulação de missões previamente guardadas	* * * *
13	Configuração de missões previamente guardadas	* *

Tabela 1 - Listagem de requisitos funcionais e sua importância

Nº	Descrição de requisito	Prioridade
1	Interface em Língua Inglesa	* * * *
2	Interacção principal com o uso do rato	* * * * *
3	Existência de pequenos atalhos de teclado	* *
4	Existência de mecanismos de feedback para eventuais avisos	* * *
5	Inclusão de Manual de Utilizador em língua Inglesa, sucinto e explicativo	* * * *
6	Inclusão de Manual de Referência	* * *
7	As unidades podem ser alteradas para outros formatos internacionais	* *
8	Portabilidade da aplicação para os sistemas operativos mais comuns	* * * *

Tabela 2 - Listagem de requisitos não funcionais e sua importância

Nº	Descrição de requisito	Prioridade
1	Apresentação de trajectória em mapa	* * *
2	Apresentação de instrumentação analógica (semelhante a cockpit real)	* * * * *
3	Apresentação de instrumentação digital	* * * * *
4	Apresentação da informação de forma identificada e seccionada	* * * *
5	Apresentação de estado de baterias e tanques	* * * *
6	Sistema adequado a multi-plataforma	* * * *

Tabela 3 - Listagem de requisitos de Design e sua importância

Nº	Descrição de requisito	Prioridade
1	Aplicação desenvolvida em *opensource*	* * * *
2	Compatibilidade de comunicação com o sistema DIVAGS	* * * * *

Tabela 4 - Listagem de requisitos de processo e sua importância

2.1.2. Especificação de requisitos

A especificação de requisitos pretende interpretar os requisitos identificados, bem como prever dificuldades que possam aparecer durante a implementação do projecto.

Os requisitos de processo exigem que a aplicação seja multi-plataforma e restringem-nos à utilização de ferramentas *opensource*.

De modo a cumprir os requisitos funcionais identificados, especifica-se que deverão ser implementados mecanismos que impeçam que o utilizador cometa erros de utilização. A utilização de mensagens de confirmação é um desses mecanismos. Esse tipo de confirmações deverá prevenir o fim abrupto da monitorização de um voo ou da aplicação. O processo de configuração deverá guiar o utilizador de modo a especificar correctamente todos os elementos necessários à monitorização do voo. Esse processo deve permitir carregar configurações anteriores, configurar mapas (locais ou internet), definir capacidade de combustível, autonomia energética e especificar a origem dos dados. O processo de configuração é comum para simulações e voos em tempo real. Quando finalizado e assim que os dados da aeronave são recebidos, começam imediatamente a ser mostrados.

Relativamente aos funcionais é de referir também que a recepção dos dados, a monitorização dos sensores, a detecção de erros, a configuração de mapas, definir, salvar e carregar definições de missões são requisitos que directamente afectam e determinam o desenrolar da aplicação.

A comunicação com o utilizador deverá ser simples e intuitiva, usando metáforas que lhe permitam a memorização do modo como se atingem as diferentes funcionalidades. O uso de instrumentos aeronáuticos irá recriar o ambiente dentro do cockpit de uma aeronave. A inclusão de atalhos de teclado permite a utilizadores avançados melhorar a velocidade de utilização.

O desempenho da aplicação deverá ser optimizado na gestão da memória e processamento, pois a aplicação está constantemente a receber dados vindos do servidor. Para tal é necessário usar algoritmos simples e uma gestão eficaz de memória.

Uma interacção simples, com metáforas adequadas e um design atractivo, aliado a baixo custo processual são mais-valias para o projecto, fazendo parte dos requisitos não funcionais especificados.

Entre os requisitos de Design especificou-se que a interface do utilizador deverá primar por apresentar toda a informação em inglês, sendo desejável uma interacção maioritariamente feita através do rato. Deverão ser apresentadas metáforas de interacção de modo a envolver o utilizador na aplicação, como se estivesse dentro do cockpit de uma aeronave. Além de manter as principais funcionalidades da aplicação sempre disponíveis, é também importante que a aplicação informe o utilizador sobre toda a sua actividade. A interface deverá ainda ter meios de prevenção para evitar más configurações.

As limitações de design prendem-se essencialmente com a capacidade gráfica das máquinas onde a aplicação possa ser executada. Apesar de hoje em dia a maioria dos computadores apresentar uma resolução igual ou superior a 1024*768 pixéis, o tamanho, por defeito, da janela da aplicação é de 800*600.

Dado que o âmbito desta aplicação está inserido num projecto multinacional, a especificação de requisitos deve incluir a utilização de metáforas e elementos que sejam socialmente aceites pelo público-alvo.

2.1.3. Design Conceptual

O design conceptual é a abordagem do plano de desenvolvimento da aplicação dirigida ao requerente do projecto.

A aplicação principal é uma interface que recebe dados enviados por duas entidades exteriores: aeronave ou simulador. Os dados têm a sua origem consoante o modo de operação pretendido. Os dados provêm da aeronave, através de uma ligação de rede local *wireless* quando se pretende monitorizar o seu voo ou verificar o funcionamento correcto de todos os seus sensores. Se pretendermos rever um voo anterior os dados provêem do simulador, sendo este uma aplicação externa à interface, que envia os dados para a aplicação. O protocolo de comunicação entre a interface gráfica e estas duas entidades é o CORBA.

Os dados, quando recebidos dentro da aplicação, são tratados e representados na interface através dos mostradores correspondentes. A informação de voo é representada por instrumentos aeronáuticos como velocímetros, horizonte artificial, altímetro e indicador de direcção, bem como através de mostradores digitais. As informações

relativas aos sensores e autonomia da aeronave são representados por via de LEDS (verde para funcional, vermelho para não funcional). As barras verticais são utilizadas para representar o combustível e carga restante das baterias.

O utilizador terá de definir vários parâmetros antes de iniciar uma missão. A escolha do tipo de mapas é muito importante para a visualização correcta da posição da aeronave na interface. Um mapa local necessita de ter as coordenadas de imagem relacionadas com as coordenadas GPS. Um mapa de internet necessita de um ponto de partida para ser enquadrado até se começar a receber os primeiros dados relativos à posição do dirigível. É também necessário indicar as capacidades de combustível e bateria máximas para fazer uma estimativa correcta da restante autonomia.

Nas figuras seguintes podemos observamos a evolução das maquetas durante o processo de design conceptual. As maquetas procuraram responder afirmativamente a todos os requisitos identificados. Nos anexos está incluída a documentação relativa à primeira fase de planeamento e identificação de requisitos, sendo visível o planeamento do funcionamento da interface.

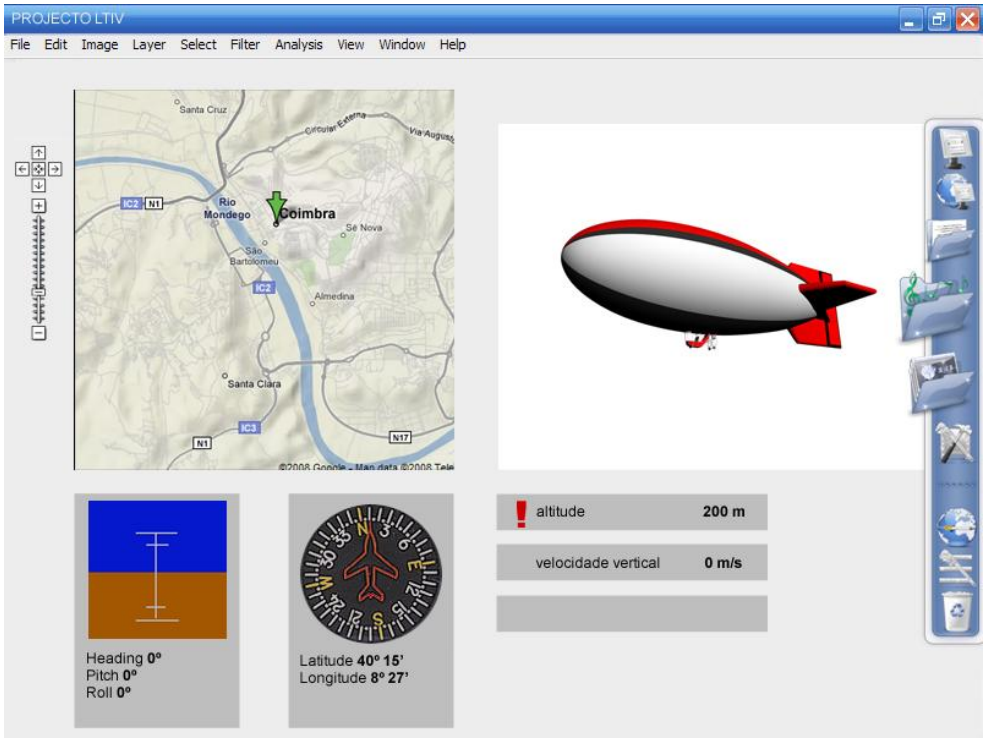

Figura 1 - Primeira concepção do interface gráfico da aplicação

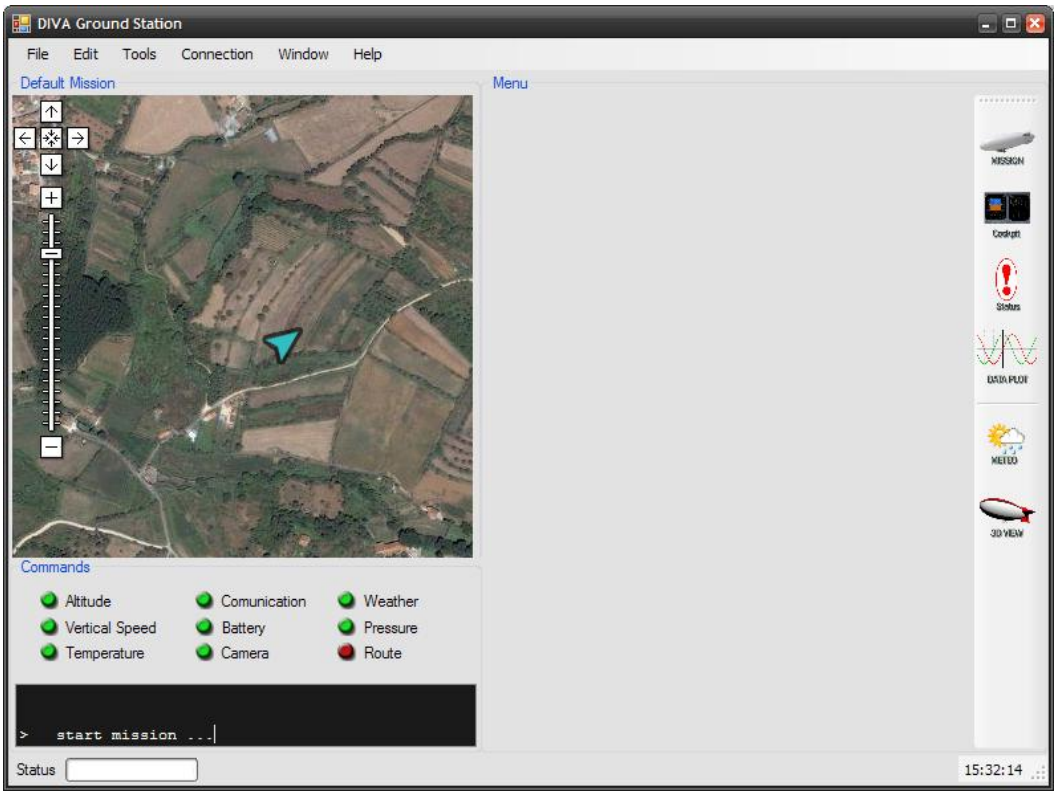

Figura 2 - Segunda Maqueta da aplicação

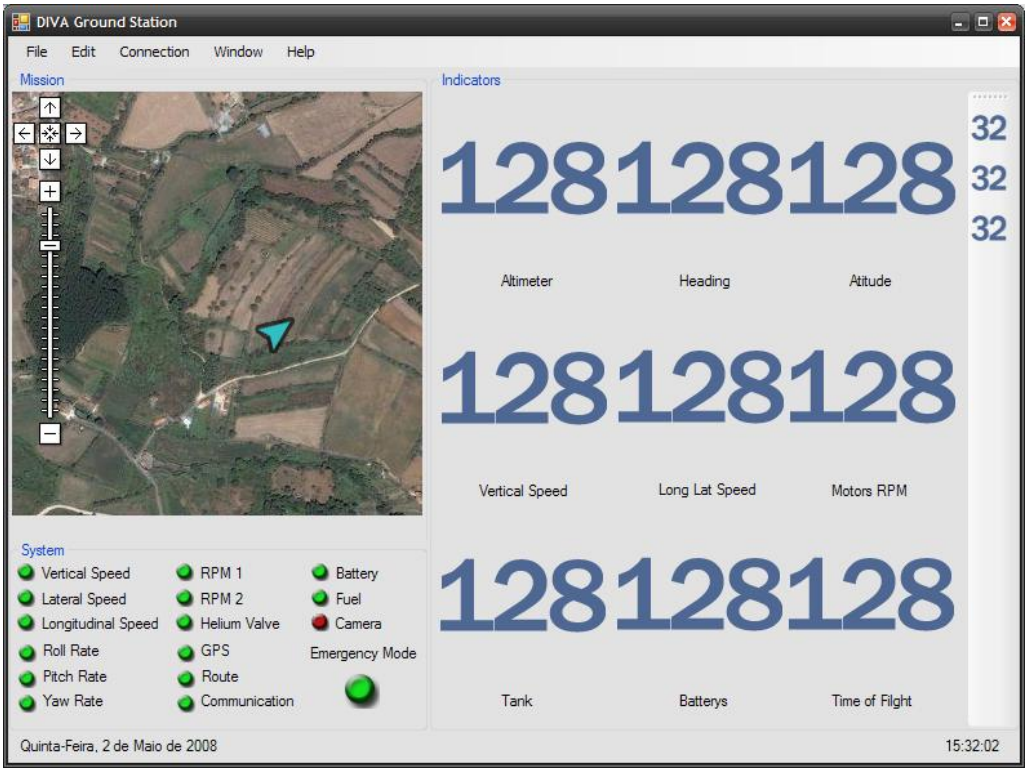

Figura 3 - Terceira maqueta do interface gráfico entregue na fase de planeamento e identificação de requisitos.

2.2. Processo de Implementação

De seguida vamos abordar as metodologias que escolhemos para o processo de implementação, validação e manutenção da aplicação. O processo de desenvolvimento compreende, sobretudo, a adopção de práticas comuns aos domínios da engenharia de *software* e gestão de projectos.

2.2.1. Metodologias para Implementação

No processo de implementação de *software* é essencial adoptar boas práticas de estruturação, documentação e produção de código. Este procedimento permite maior clareza na escrita e leitura do código, maior coerência com o design idealizado e maior facilidade na implementação e modificação de funcionalidades. Nestas práticas de implementação destacam-se:

- Aplicação de normas de programação;
- Aplicação de regras sobre as estruturas de dados, estruturas de controlo e algoritmos;
- Correspondência directa do código com o *design* proposto;
- Produção de documentação interna;
- Utilização de pseudo-código para sistematizar os algoritmos mais importantes.

A ideia principal na estruturação do código é Dividir e Conquistar ou *Divide and Conquer* – qualquer tarefa é atomizada e pode ser descrita.

Numa abordagem mais prática é muito importante que, ao longo de todo o código produzido, se mantenha coerência na nomenclatura e definição de objectos, de modo a facilitar a leitura e compreensão do código. Conscientes da importância deste tipo de implementação procurámos adoptar estas medidas no código produzido.

A estruturação e documentação do código adequadas contribuíram positivamente para o processo de desenvolvimento técnico da aplicação. A aplicação desta metodologia trouxe grandes vantagens ao nível da organização de ideias, localização de falhas, criação de mecanismos para eventual reestruturação e auxílio na produção de documentação interna e externa.

As fases anteriores de análise de requisitos, design conceptual e modelação técnica contribuíram para uma especificação do código mais eficaz. A própria modelação considerada para o sistema permitiu-nos melhor interoperabilidade na nossa equipa, pelo que cada elemento do grupo pode trabalhar em áreas distintas da fase de código.

Dada a natureza e âmbito do projecto, foi necessário utilizar um método de Engenharia de *Software*, chamada *Extremme Programming (XP)*. Este método consiste numa abordagem de desenvolvimento que visa a construção de código à medida que se contacta o cliente e se compreendem os requisitos e necessidades. A sua utilização promove a comunicação, a interoperabilidade entre membros internos e externos à equipa e a simplicidade na resolução de problemas. O ciclo da implementação da aplicação torna-se suficientemente versátil à alteração de requisitos. (2)

2.2.2. Metodologias para teste da aplicação

O processo de validação e teste de unidades de *software* é uma prática comum que tem como objectivo detectar e corrigir erros e eventuais falhas. Estes defeitos surgem muitas vezes de uma análise ineficiente dos requisitos ou de falhas inerentes ao design e código concebidos. Entre os métodos mais importantes de teste de *software* existem:

Designação	Unidades testadas	Vantagens	Desvantagens
***Closed Box* ou *Black Box* (2)**	Entradas e saídas sob o ponto de vista de interface, comportamento e funcionalidades	Eficientes em grandes sistemas, detecção de falhas em requisitos funcionais, teste do ponto de vista de utilização	Dificuldade em conceber testes para todas as entradas possíveis na interface
***Clear Box* ou *White Box* (3)**	Código produzido, algoritmos, caminhos, lógica interna, entradas e saídas de funções	Optimização de código, controlo sobre entrada e saída, desempenho	Dificuldade em analisar toda a estrutura interna e lógica do software

Tabela 5 – Comparação de métodos para teste de aplicações

No nosso *software* foram aplicados os dois tipos de testes enunciados. A sua utilização permitiu a aferição de resultados que se reflectiram no refinamento necessário às fases de análise de requisitos e design. Pela natureza e condições em que o projecto pôde ser testado na sua plena funcionalidade, a existência de um simulador de missões foi fundamental para preparar a aplicação para a monitorização real.

2.2.3. Metodologias para manutenção de *software*

"Em Engenharia de *Software*, a manutenção é a modificação de um produto de *software* depois da entrega ao cliente, com vista a corrigir falhas, melhorar o desempenho ou adaptar o produto a ambiente com características diferentes." (2)

Consideramos que a possibilidade de um *software* se manter ou adaptar facilmente a diferentes condições é de extrema importância para qualquer produto deste tipo. No projecto DIVA, o aparelho aéreo é susceptível a pequenas alterações na instrumentação e sensores. É importante que a tarefa de manutenção do *software* seja relativamente facilitada.

Neste sentido, procurámos facilitar o processo de aprendizagem para futuros utilizadores e desenvolvedores, criando documentação adequada para esse propósito. (3)

Foi criado um manual de utilizador que procura descrever todas as funcionalidades do sistema na óptica do utilizador. Este documento introduz o *software*, com uma descrição geral, apresentando um glossário com a definição dos termos técnicos utilizados. Depois são indicadas as funções do *software* e o modo como o utilizador pode aceder a estas. O manual inclui suportes visuais que auxiliam o utilizador no processo de aprendizagem.

O manual de referência foi criado com o propósito de facilitar a compreensão da estrutura interna da aplicação. É um documento orientado para futuros desenvolvedores que pretendam adaptar, corrigir ou modificar funcionalidades do sistema.

Capítulo 3. Sistema desenvolvido

Neste capítulo será exposto o resultado funcional do trabalho desenvolvido no âmbito deste projecto. A apresentação será iniciada por uma descrição qualitativa das tecnologias utilizadas, indicando as principais motivações que nos levaram a adoptar as tecnologias referidas. As razões podem ser influenciadas por requisitos específicos impostos inicialmente, por vantagens ou desvantagens de aplicação em determinadas situações. Outro factor importante foi a nossa motivação em trabalhar com ferramentas que nos proporcionassem novos conhecimentos que ainda não havíamos adquirido.

Na segunda fase deste capítulo, o sistema é apresentado sob o ponto de vista prático e funcional. Serão descritas as principais características práticas e funcionais do nosso *software* e considerados os mecanismos de interacção e de comunicação presentes.

3.1. Introdução às tecnologias utilizadas

Com base nos requisitos encontrados, estabelecemos uma base tecnológica para cumprir com os objectos deste projecto. A opção baseou-se em tecnologias que garantissem a portabilidade da aplicação, assim como, a facilidade na interoperabilidade entre objectos num sistema distribuído. Foi utilizado um sistema de controlo de versões de modo a proporcionar um método de programação mais versátil e seguro.

3.1.1. Ambiente de desenvolvimento

Atendendo ao conjunto de funcionalidades e características de qualidade que o *software* deve possuir, a escolha do ambiente de desenvolvimento é fundamental, pelo que esta pode exercer grande influência nas características do sistema final e limitar as tecnologias disponíveis.

Actualmente existem muitas plataformas adequadas ao desenvolvimento de *software*. Contudo, considerámos para o projecto apenas os sistemas Microsoft e GNU/Linux.

Característica	WINDOWS	GNU/LINUX
Estabilidade	As versões baseadas no *kernel* NT são mais estáveis que as versões anteriores. O uso de drivers não certificados pode causar instabilidade. É necessário reiniciar o sistema sempre que se efectua um *update*. Se ocorrer uma falha critica todo o sistema pára.	O *kernel* herda a estabilidade do UNIX devido à sua arquitectura modular. Varia consoante o gestor de janelas e o emulador de terminal que se usa. Existem métodos para terminar aplicações instáveis. Base operativa em modo de texto mais fiável
Performance	Existe decréscimo de performance devido à fragmentação de disco.	É usada uma partição unicamente dedicada à paginação da memória, evitando a fragmentação de disco.
Segurança	Existem inúmeras vulnerabilidades para este sistema.	Existe algum software malicioso para Linux propagado via internet.
Desenvolvimento *Cross-Platform*	Maiorias das aplicações são implementadas em tecnologias específicas para Windows.	Compatibilidade entre sistemas tipo UNIX. É possível emular programas Windows.
IDE e Compiladores	Existe uma oferta diversificada de compiladores e IDE comerciais e grátis.	Existem alguns IDE comerciais, mas os mais conhecidos são grátis, sendo a referência o *gcc*
Proprietário Vs *Open-Source*	Afirma ser mais seguro devido à sua implementação usando *Security Development Lifecycle*, mas apenas os programadores da Microsoft podem corrigir *bugs*, devido a ser código proprietário.	Por ser implementado em código aberto, é mais rápido obter *patches* que assegurem a segurança
Documentação	A documentação existente destinada ao desenvolvimento em plataformas Windows está disponível no MSDN	Leque muito alargado, desde de desenvolvimento a manuais de utilizador. Acessível via shell de sistema.

Tabela 6 - Comparação entre Windows e GNU/Linux contemplando aspectos como a segurança, estabilidade , ferramentas disponíveis e documentação existente. (4)

A escolha do sistema operativo para desenvolvimento deve respeitar as condições impostas pela fase de análise de requisitos. Neste sentido, alguns requisitos apresentados nas Tabela 1,Tabela 2,Tabela 3 Tabela 4 foram reconsiderados na análise do ambiente de desenvolvimento.

A escolha recaiu no ambiente de desenvolvimento GNU/Linux, justificada em grande parte pelo nosso interesse em dominar os sistemas baseados nesta arquitectura. Outros factores importantes que motivaram a escolha de ambiente estão directamente ligados a requisitos específicos do *software* e à boa adaptação destes sistemas a linguagens *open-source* portáveis e a IDEs eficientes.

A utilização deste ambiente desenvolvimento permitiu-nos compreender melhor, na prática, alguns conceitos que foram abordados nas áreas curriculares da nossa licenciatura. Esta aprendizagem e compreensão foram possíveis através da transparência ao nível de estrutura e de funcionalidade deste tipo de sistemas.

Apesar da existência de variados sistemas que apresentam também boas condições para a prática de programação e desenvolvimento, a utilização do GNU/Linux revelou-se uma boa escolha.

3.1.2. Definição de Linguagem e Plataforma

A definição da linguagem de programação base para o desenvolvimento do *software* é um passo importante que deve ser considerado nas vantagens e desvantagens que pode trazer. A escolha deve reflectir-se sobre os requisitos que foram apreendidos nas fases anteriores de análise e especificação.

As linguagens de programação distinguem-se pelo conceito que aplicam no desenvolvimento, na forma com são aplicadas e estruturadas e na funcionalidade que pretendem atingir.

Na tabela seguinte destacamos os principais paradigmas de programação considerados:

Paradigma	Exemplos de Linguagens	Vantagens	Desvantagens
Procedimental	C, Pascal, Fortran	Modularidade, flexibilidade	Dificuldade de compreensão e gestão de grandes quantidades de informação
Orientado a Objectos (7)	C++, Java, C#, PHP	Reutilização e Herança de objectos, Abstracção, Polimorfismo	Pode ser computacionalmente mais exigente

Tabela 7 - Paradigmas de programação. (5)

A nossa escolha acabou por incidir no C++, justificada em parte pela nossa experiência académica com esta linguagem e pela vontade de aprofundar os nossos conhecimentos de programação. Além disso, o C++ apresenta muitas características que são favoráveis ao desenvolvimento do nosso projecto. O facto de ser uma programação orientada a objectos (OOP) facilita o processo de concepção, permitindo melhor coerência com o design proposto e fomentando a abstracção.

Dentro das opções disponíveis e atendendo aos factores dos requisitos, ambiente e plataforma, o C++ revelou-se uma boa escolha. A experiência que já possuíamos com esta linguagem orientou-nos nas fases primárias de desenvolvimento.

3.1.3. Tecnologias de suporte para visualização

As interfaces gráficas ou *Graphical User Interfaces* (GUIs) surgiram no sentido de facilitar a interacção do homem com dispositivos electrónicos (6). Uma GUI apresenta as funcionalidades do sistema em forma de ícones, gráficos e instrumentos virtuais, habitualmente representados por metáforas e que podem ser acedidos por manipulação directa. A criação de GUIs é uma fase importante na implementação de um *software* interactivo. Consequentemente, as tecnologias que vão suportar a implementação desta fase devem ser alvo de forte consideração.

Com a crescente popularidade das interfaces gráficas nos sistemas de operação e nas respectivas aplicações, existem muitas *frameworks* disponíveis que auxiliam o desenvolvimento de GUIs. As *frameworks* gráficas dependem normalmente da linguagem base utilizada. Na tabela seguinte são apresentadas algumas *toolkits* gráficas e algumas das suas características A tabela seguinte apresenta as *toolkits* consideradas:

Toolkit	Linguagem	Ferramentas	Licença	Vantagens	Desvantagens
Tk	Tcl	ASED, Tiny, Tloona	Open source	Portabilidade	Limitado em widgets
Qt	C++	QT Designer, Qdevelop, Kdevelop, QAssistant	GPL 2/3 (open source), Q Public License	Portabilidade, grande variedade de widgets, estável, documentação	Licença comercial não é grátis
GTK+	C	Glade, Geany,...	LGPL	Portabilidade, licença grátis	Portabilidade, Fraca documentação.
MFC / WinAPI	.NET	VisualStudio	Proprietário	Familiaridade	Não portabilidade
Windows Forms	.NET	VisualStudio	Proprietário	Familiaridade	Não portabilidade
Mono	.NET	Mono	GPLv2	Portabilidade	Integração com outras tecnologias
wxWidgets	C++	VisualWx, PythonCard	Baseado na L-GPL	Portabilidade, cross-compiling, emulação widgets nativos	Pouca documentação

Tabela 8 - Toolkits Gráficas Consideradas

Procurámos fazer uma análise cuidada acerca das ferramentas de desenvolvimento de interfaces gráficas. Chegámos à conclusão que o Qt é a plataforma mais indicada para implementar as funcionalidades requeridas e que melhor se adapta às tecnologias já adoptadas anteriormente (C++ e GNU/Linux).

Funcionamento básico do Qt

O Qt é uma *framework* multi-plataforma para desenvolvimento de interfaces humanas com suporte para as linguagens Java e C++. Possui um vasto conjunto de ferramentas próprias para o *design* de Interfaces gráficas e uma biblioteca de classes em C++ (7). Na figura seguinte apresentamos um esquema que mostra o funcionamento inter-modular do Qt nas diferentes plataformas:

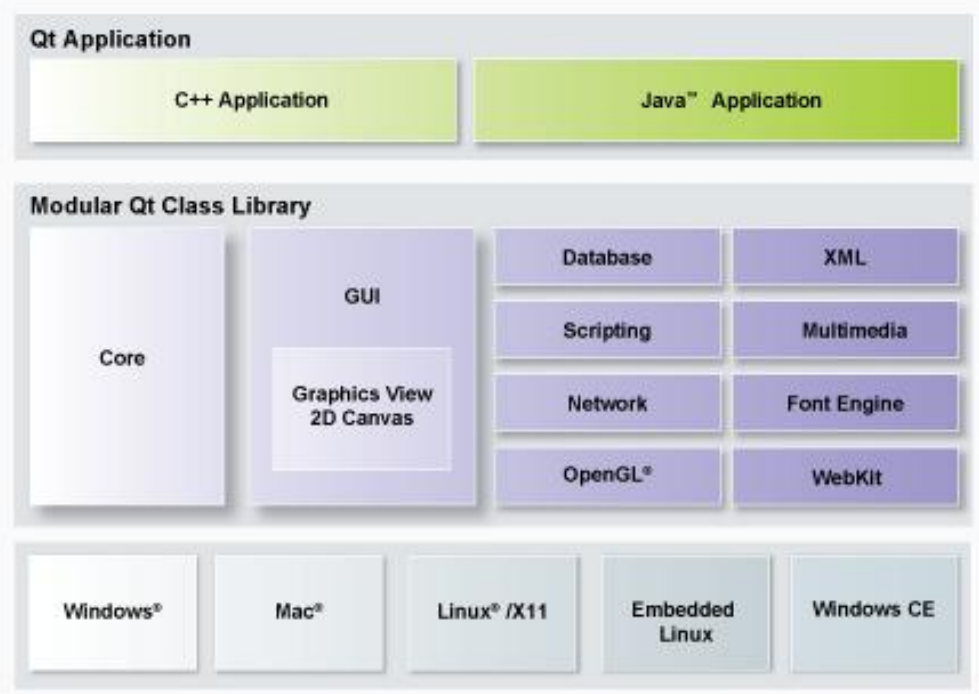

Diagrama 1 - Arquitectura Modular do Qt

As bibliotecas presentes no Qt não se limitam a acrescentar funcionalidades para o desenvolvimento de interfaces, elas permitem por exemplo, a integração de Redes e comunicação, OpenGL, XML e Bases de Dados SQL.

Tipicamente, o Qt compreende as seguintes bibliotecas:

- QtCore – classes básicas (suporte para declaração de Listas ligadas, Arrays,...);
- QtGui – classes para o desenvolvimento de GUI;
- QtNetwork – classes para programação de rede;
- QtOpenGL – classes para integração de rotinas OpenGL;
- QtXML – classes para integração de XML;
- QtSQL – classes para integração de bases de dados relacionais SQL.

Qt Designer

Esta aplicação faz parte do Qt e consiste, fundamentalmente, num ambiente de desenvolvimento que permite a criação de janelas e ambientes gráficos. Através desta ferramenta, o desenvolvedor tem a possibilidade de construir janelas e outros *widgets*, utilizando uma abordagem mais visual. Sendo uma aplicação multi-plataforma, possui mecanismos que se adaptam e permitem prever os objectos criados no sistema operativo utilizado. O Qt Designer possibilita, em parte, a criação de *widgets* personalizados a partir de *widgets* já existentes do Qt.

QMake

O suporte multi-plataforma do Qt é simplificado pela aplicação QMake. Este programa automatiza a geração do *makefile* necessário à compilação nas diferentes plataformas (7). A geração depende do ficheiro de projecto. É gerado também um conjunto de regras para a compilação de ficheiros de interface (.ui) .

O uso do Qmake não está limitado a código QT podendo ser também aplicado a qualquer tipo de projecto ou linguagem.

Bibliotecas e *Widgets* Externos

Para além das funcionalidades de QT descritas anteriormente, esta *framework* permite também a integração de *widgets* e a inclusão de bibliotecas externas.

Para melhor satisfazer os requisitos dos utilizadores, sentimos a necessidade de recorrer a *widgets* e funções externas. Na parte que se segue abordaremos sucintamente as tecnologias adicionais que utilizámos e a forma como estas contribuíram para o desenvolvimento da aplicação.

QWT

Esta biblioteca vem acrescentar ao Qt uma colecção bastante vasta de *widgets*. A aplicação destes *widgets* em *software* técnico acarreta grandes vantagens, pois estes consistem, principalmente, em mostradores analógicos, termómetros e bússolas. Com isto, a implementação gráfica deste tipo de elementos virtuais torna-se bastante simplificada. (11)

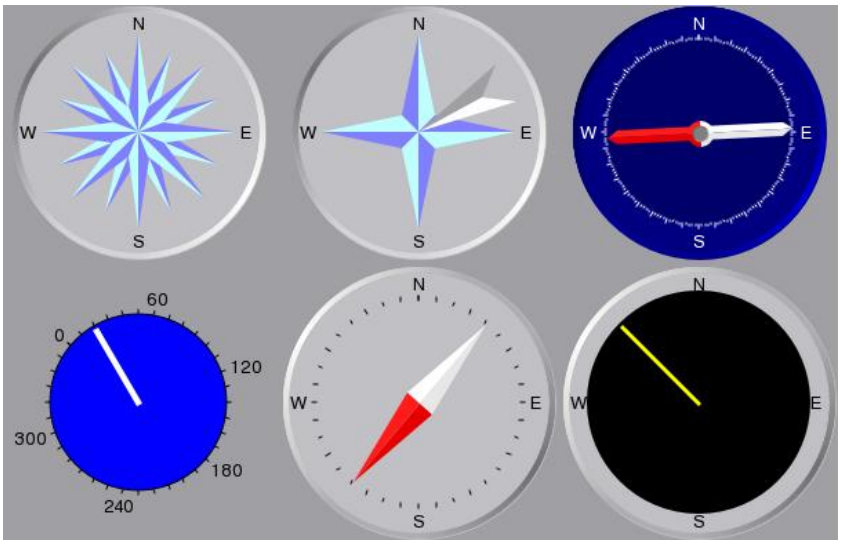

Figura 4 - Classes de instrumentação presentes no QWT

QMapcontrol

O QMapcontrol é um *widget* para o Qt, que torna possível a utilização de mapas e informações geoespaciais nas aplicações desenvolvidas. As imagens cartográficas são obtidas através de um serviço online de mapas (OpenStreetMap, WMS-Server, ...) e podem ser conjugadas com outros elementos gráficos simples (linhas, pontos, imagens ou *widgets*). A interacção com o mapa pode ser implementada de forma simples, de modo a facilitar o processo de navegação.

QLed

Este *widget* constitui a representação virtual de uma pequena luz de estado.

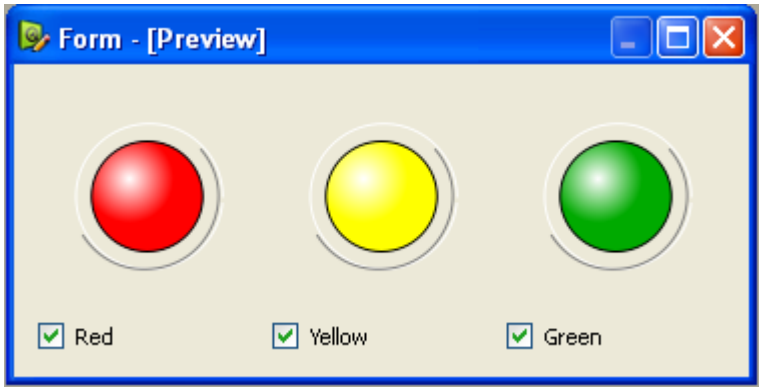

Figura 5 – Aspecto gráfico do QLed

A escolha do Qt como tecnologia principal de suporte à visualização esteve, em parte, ligada ao nosso interesse em explorar e aprender esta plataforma. A sua utilização acabou por se revelar muito vantajosa para o nosso trabalho. A documentação completa, a facilidade de integração de novas funcionalidades, a existência de ferramentas adequadas ao desenvolvimento e a rápida adaptação a sistemas heterogéneos foram factores chave que influenciaram positivamente o processo de desenvolvimento do nosso projecto. Embora não tenhamos utilizado todas as funcionalidades das ferramentas aqui apresentadas, estas foram essenciais à implementação. O aproveitamento das aplicações auxiliares para desenvolvimento Qt, como o Qt Designer, foi bastante útil para manter a coerência do design conceptual com a implementação técnica dos requisitos impostos inicialmente.

3.1.4. Tecnologias de suporte para comunicação

A comunicação com a aeronave é um aspecto essencial para a nossa aplicação devido à necessidade da monitorização em tempo real. Sem a comunicação com o sistema da aeronave, seria impossível a comunicação entre aplicações (DIVAgui e DIVAGS). Para que tal seja possível é necessário utilizar uma tecnologia que funcione num sistema distribuído e que permita a troca de objectos comuns entre as aplicações.

Toolkit	Protocolo	Linguagens Suportadas	Plataformas execução	Vantagens	Desvantagens
TAO	CORBA	C, C++, Java, Python, …	Windows, GNU/Linux, Solaris, vxWorks	Multi-plataforma, melhorias no serviço de eventos do CORBA	Suspensão de execução enquanto se delegam tarefas
DCOM (10)	RPC/DCE	C, C++, java, Python, …	Windows	controlo de memória	Não portável
TwRICE, RICE	ICE (14)	XML	Portável	Pensado para uso na internet, Elevada performance	Informação tem de ser transaccionada via ficheiros XML
Java RMI	JRMP	Java	Windows, GNU/Linux, Solaris, MacOS, …	Portabilidade, controlo de memória	Não é totalmente compatível com CORBA

Tabela 9 – Pequeno comparativo entre vários Toolkit de desenvolvimento para sistemas distribuídos. (9)

A decisão do *toolkit* a usar recaiu sobre a que melhor se adequava aos requisitos iniciais do projecto. Optou-se por usar o *toolkit* TAO que usa o protocolo CORBA pois a estação de trabalho DIVAGS comunica por este protocolo.

O protocolo CORBA é uma especificação da OMG que padroniza as comunicações de objectos em sistemas distribuídos, permitindo a comunicação de aplicações feitas em linguagens e plataformas diferentes. Para possibilitar as trocas de objectos é necessário especificar os objectos a transaccionar. A linguagem de programação IDL é puramente declarativa, estando ausentes algoritmos nos seus scripts.

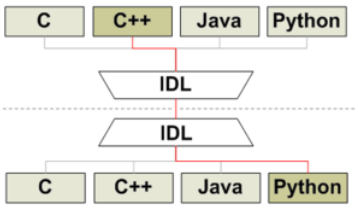

Figura 6 – Exemplo do funcionamento do IDL.

O IDL permite a interoperabilidade entre sistemas heterogéneos, definindo a separação entre interfaces de cada sistema. A cada objecto corresponde uma interface que permite efectuar pedidos a um outro objecto, sem que se conheça a sua especificação. O compilador IDL gera dois componentes: o STUB do lado do cliente e o SKELETON do lado do servidor. Durante a sua criação, cada um dos componentes é adaptado à linguagem de implementação de cada um dos elementos do sistema.

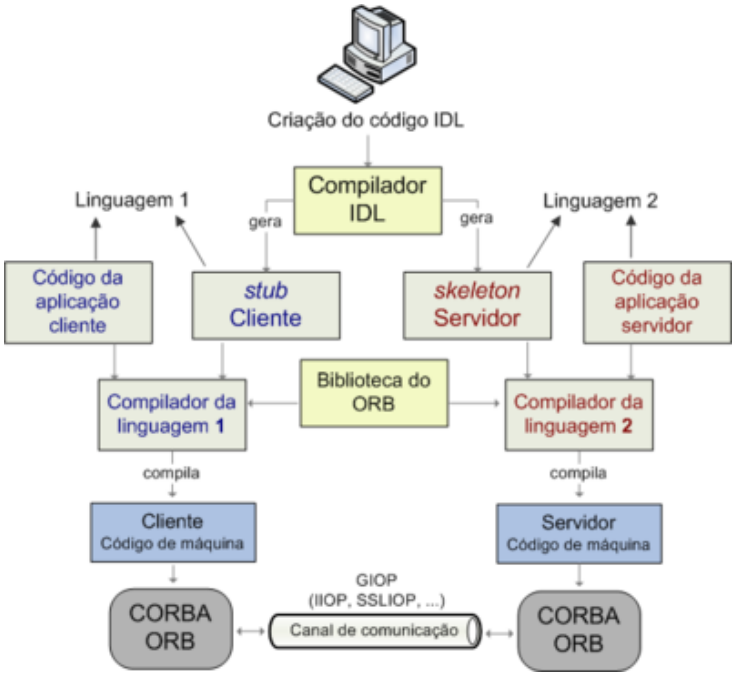

Diagrama 2 - Relação entre os vários componentes de um sistema CORBA. (6)

Os componentes do sistema comunicam-se através do ORB. O ORB é responsável por responder aos pedidos de objectos vindos do cliente e servidor. Os ORB têm ainda capacidade de se comunicar com outros ORBs através do protocolo GIOP. (13) Existe a implementação do GIOP sobre o protocolo TCP/IP denominado IIOP.

O protocolo GIOP é definido em três partes:

- CDR (*common* data representation) - representa os dados ou objectos transaccionados entre cliente e servidor;
- IOR - Indica a localização dos objectos comuns a servidor e cliente (referência) de modo a serem efectuadas transacções entre cliente e servidor (8);
- Formato das mensagens – Os formatos das mensagens transaccionadas controlam o fluxo de dados (*request*, *cancel*, …).

O CORBA pode ser usado dentro de qualquer rede de dados, sendo o seu protocolo de transmissão de dados definido pelo seu utilizador, libertando-o de toda a implementação de baixo nível do sistema distribuído.

3.1.5. Sistema de controlo de versões

É fundamental que qualquer equipa de desenvolvimento de *software* possua ferramentas que permitam assegurar o controlo de versões produzidas. Quando adequadas e bem aplicadas, estas ferramentas fomentam o trabalho entre todos os elementos da equipa e possibilitam um registo seguro com todas as versões e alterações a que o projecto foi sujeito.

Com o reconhecimento da importância destes sistemas para projectos de *software*, existem, actualmente, muitas opções *opensource* disponíveis: CVS, Subversion, openCVS, entre outras.

Para o desenvolvimento do nosso projecto, foram consideradas as ferramentas CVS e Subversion. Através de alguma pesquisa e experimentação, concluímos que o sistema SVN era o mais adequado para o nosso projecto. Além da rapidez e fiabilidade na execução das tarefas normais de revisão, o SVN opera sobre redes, permitindo uma utilização remota e por parte de todos os elementos da equipa (12). A integração simples do cliente KSvn para ambiente LINUX foi também um dos factores chave na nossa decisão. Com as dificuldades em encontrar um servidor seguro para o armazenamento de informação, decidimos utilizar a plataforma Googlecode.

O Googlecode é um sítio desenvolvido pelo Google que permite o alojamento e consulta de projectos sob licença *Opensource*. É uma plataforma bastante organizada e intuitiva na utilização. Apresenta informações relevantes (descrição geral, objectivos, linguagem) que podem ser introduzidas pelos desenvolvedores.

3.2. Arquitectura do Sistema

De modo a compreender o funcionamento interno da aplicação devemos entender a sua implementação. É neste contexto que introduzimos este capítulo, descrevendo toda a linguagem técnica necessária para entender a sua implementação. O manual do utilizador (7) deve ser consultado para uma abordagem mais funcional e descritiva da aplicação. Este documento é completamente orientado para o utilizador e pretende auxiliá-lo nas actividades do programa.

3.2.1. Descrição Geral do sistema

Concluída a fase de análise de requisitos, concepção de arquitectura e escolha de tecnologias vamos agora apresentar genericamente o *software* desenvolvido. A arquitectura interna do sistema que possibilitou a implementação das funcionalidades e qualidades pode ser descrita através dos diagramas seguintes:

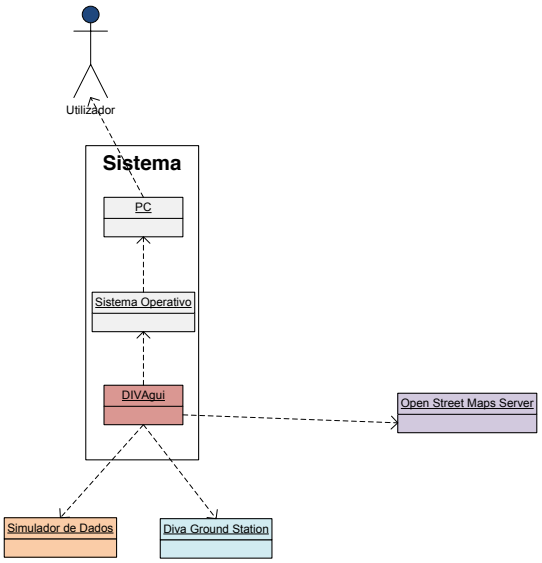

Diagrama 3 - Decomposição modular do sistema DIVAgui

O diagrama modular apresenta o sistema e as suas dependências externas. A aplicação DIVAgui está dependente das fontes de dados (DIVA GS ou Simulator). Os vários módulos estão identificados por cores de modo a facilitar a sua identificação nos próximos diagramas UML.

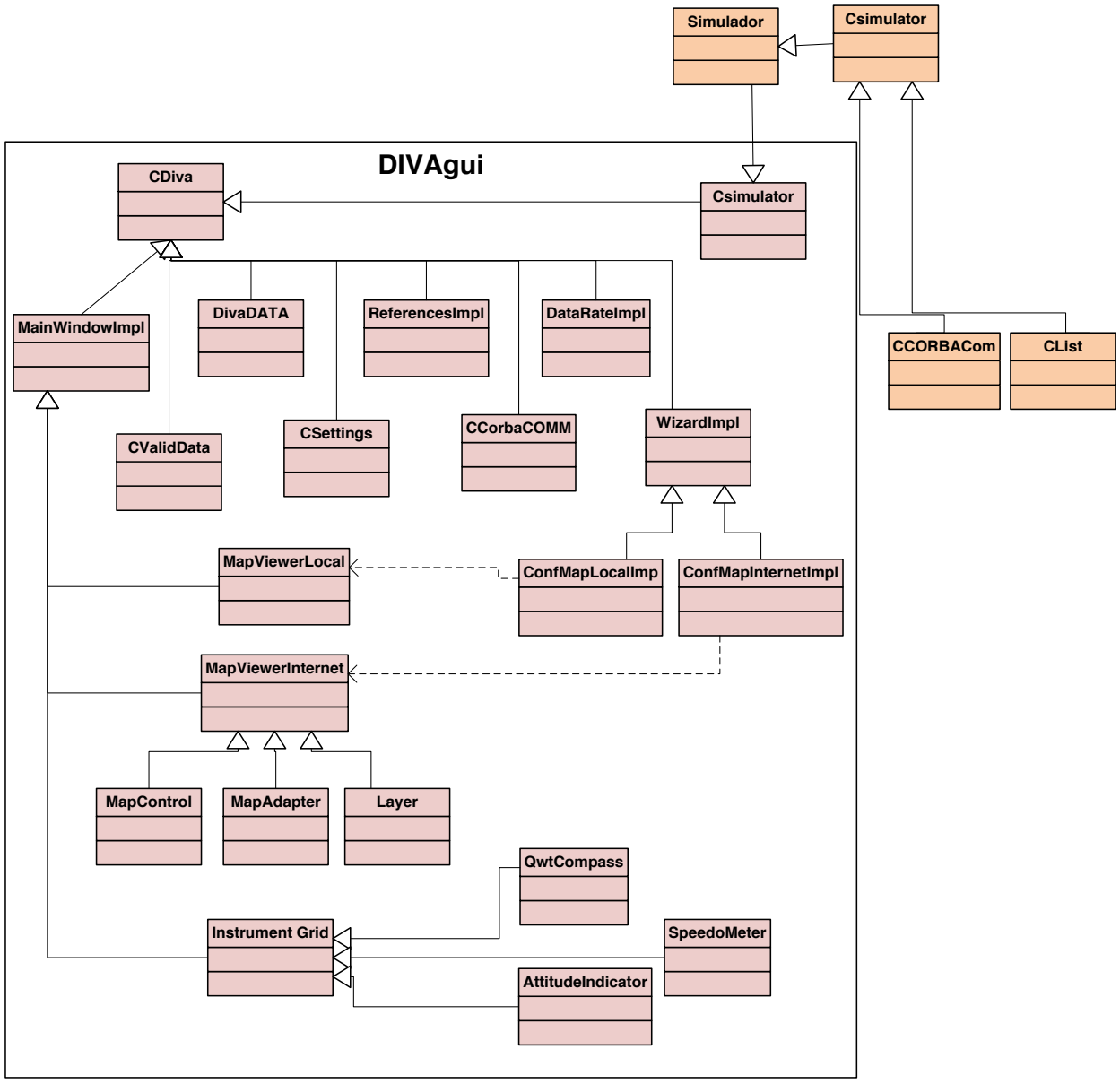

Diagrama 4 - Arquitectura interna do sistema DIVAgui

Como vemos pelo **Error! Reference source not found.** a classe CDiva é instanciada na rotina principal. Este objecto é responsável pela criação das restantes classes que desempenham métodos específicos.

As classes podem dividir-se em três níveis:

- **Comunicação:** CCorbaComm e Stock_consumer;
- **Interface**: MainWindowImpl, *Wizard*Impl, MapViewer_local, MapViewer_net, datarateimpl, referencesimpl, MapControl, MapAdapter, Layer, Instruments Grid, QwtCompass, AttitudeIndicator e SpeedoMeter;
- **Processamento de dados:** DivaData, CValidData, CSettings, ConfMapLocalImpl, ConfMapInternetImpl.

Numa abordagem mais prática vamos descrever, de seguida, as aplicações DIVAgui e Simulator.

3.2.1.1. DIVAgui

A interface principal do sistema apresenta grande parte das qualidades e funcionalidades impostas pelos requisitos enunciados nas Tabela 1,Tabela 2,Tabela 3 Tabela 4.

A nível da actividade da aplicação, o utilizador pode configurar uma missão real ou simulada, alterar a taxa de aceitação de blocos de dados, reconfigurar as referências para indicadores específicos e aceder a menus de ajuda e "About".

Independentemente do tipo de missão, a sua configuração é realizada através de um conjunto de passos que auxiliam o utilizador no preenchimento da informação necessária. Os passos compreendem: a configuração do nome da missão, a escolha do mapa, a definição de referências para determinados sensores e a configuração da comunicação. Estes dados podem ser guardados ou carregados a partir do disco.

A apresentação dos dados recebidos – principal função do sistema – é garantida pelos elementos visuais presentes no lado direito da janela. O utilizador escolhe a forma de visualização de dados, podendo optar entre a apresentação analógica ou digital. A instrumentação virtual só começa a mostrar informação a partir do momento em que é configurada uma missão.

Na parte esquerda é exibido um mapa configurável e interactivo que mostra a posição da aeronave segundo a sua posição geoespacial. O mapa escolhido pode ser

obtido a partir dos servidores OSM (internet) ou a partir de um ficheiro de imagem local. Para este último caso, o utilizador terá que calibrar a imagem, definindo as coordenadas GPS do canto superior esquerdo e inferior direito. As funções de ligar/desligar o desenho da rota e centrar o *viewport* na aeronave estão contempladas neste elemento. O estado da comunicação da aplicação cliente com o servidor é mostrado com a designação do sensor (XSENS, GPS, WIND, entre outros). Se o *software* for utilizado para simulação, o controlo sobre a missão é garantido pelos botões presentes.

3.2.1.2. Simulador

O programa que permite a simulação de missões é chamado pela aplicação DIVAgui quando é accionada essa opção. Como tal, o Simulator não possui interface gráfica. A sua implementação contempla a possibilidade de parar, acelerar ou moderar a velocidade da simulação. Para tal, a interface principal envia comandos específicos para a aplicação externa de simulação

O Simulator faz a leitura de um ficheiro de missão devidamente formatado e envia os dados para a interface. Na óptica da interface, o Simulator representa um dirigível virtual.

3.2.2. Interacção com o utilizador

De acordo com a nossa experiência e formação académica nas áreas de interacção com o utilizador, a aplicação desenvolvida procura tirar partido de mecanismos de usabilidade adequados.

A interface DIVAgui apresenta regras de interacção e mecanismos semelhantes aos que encontramos nos sistemas operativos comuns. Deste modo, o processo de aprendizagem na utilização do *software* é mais facilitado.

A utilização de ícones simples e facilmente identificáveis permite uma melhor usabilidade e controlo sobre a aplicação e visualização. Na janela principal, os ícones presentes nos botões principais referem-se ao controlo da simulação. O seu desenho deve ser suficientemente explicativo e universal em relação à função que desempenham.

Figura 7 - *Screenshots* à iconografia presente na interface

Com vista a facilitar a identificação da função década botão são utilizados mecanismos de feedback – *tooltips*, mensagens, entre outros.

Os mecanismos de informação (*feedback*) que o utilizador dispõe, nomeadamente ao nível do estado dos sistemas aeronáuticos, são constituídos por *widgets* que representam um LED virtual. Através da cor do LED, o utilizador tem a percepção do estado do sistema. As cores utilizadas foram o vermelho (estado crítico/desligado) e o verde (estado normal/ligado), que embora não completamente universais no seu significado, abrangem grande parte dos utilizadores.

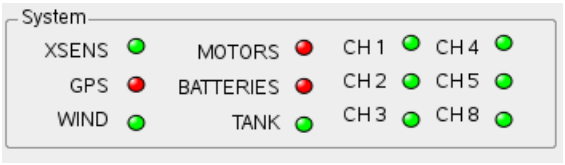

Figura 8 - *Screenshot* aos leds de diagnostico da comunicação

Em relação à configuração de missões, facilitámos o processo com um sistema baseado em *wizards*. A informação é preenchida pelo utilizador, caso haja algum dado incorrecto, não é possível avançar.

De uma forma geral, os mecanismos de interacção presentes procuram abstrair o utilizador da parte técnica da aplicação, apresentado metáforas e ambiente simples.

3.2.3. Comunicação do sistema

Como pudemos verificar no Diagrama 3 o sistema está dependente da comunicação entre módulos internos e externos.

O módulo principal – DIVAgui – comunica com o utilizador devolvendo o resultado das suas interacções com o programa. Como resultado dos pedidos vindos do utilizador, a aplicação irá processar esses pedidos interna e externamente, produzindo o comportamento desejado. Neste sub-capitulo vamos descrever tecnicamente a comunicação inter-modular.

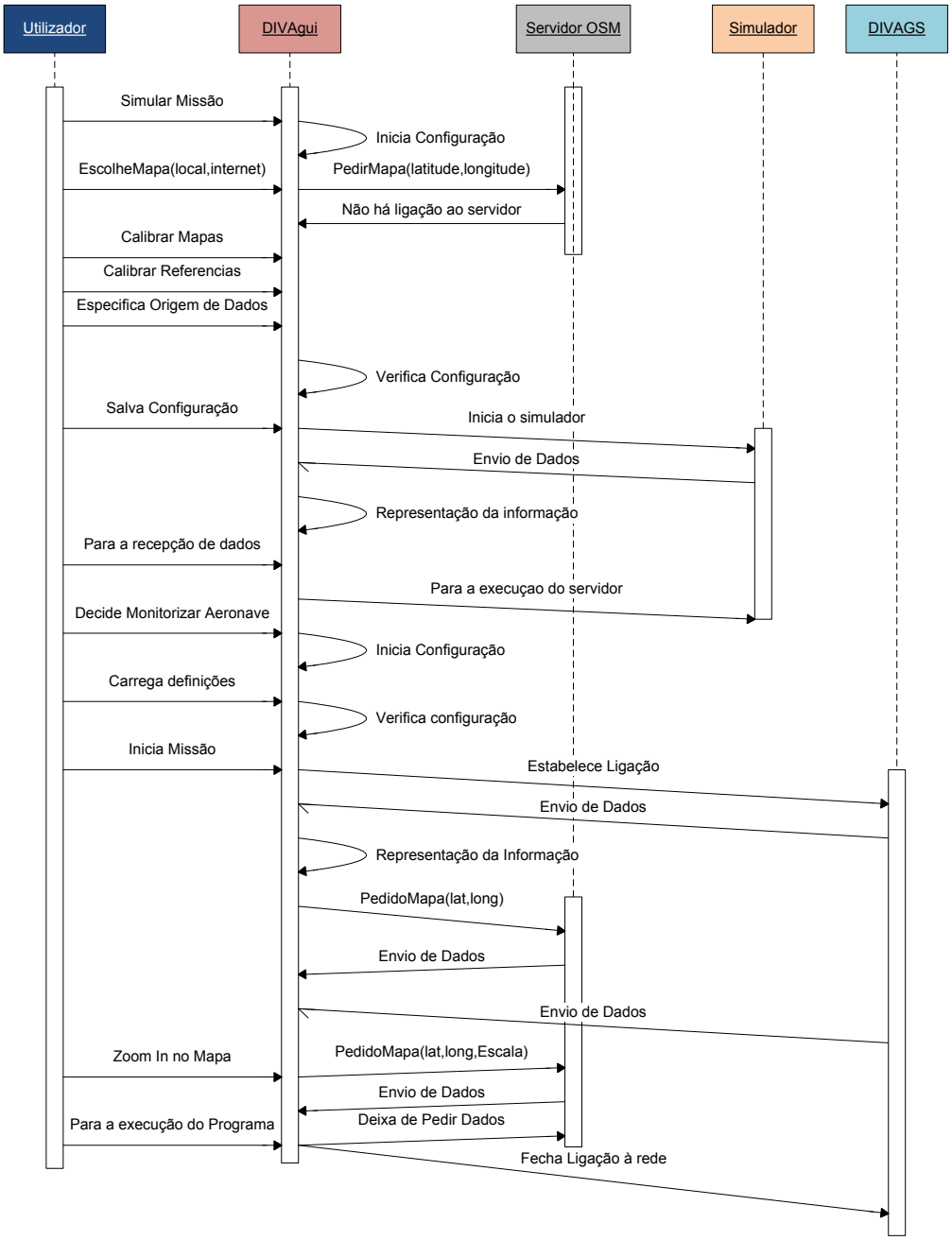

Diagrama 5 - Diagrama de sequência de eventos ocorridos no uso do DIVAgui

O Diagrama 5 representa uma sequência de eventos provocados pelo utilizador durante a execução da aplicação. A acção pretendida pelo utilizador é satisfeita através da implementação dos acontecimentos necessários até à sua conclusão.

A comunicação interna do sistema aquando de uma sequência de eventos pode ser descrita através dos seguintes diagramas de actividade:

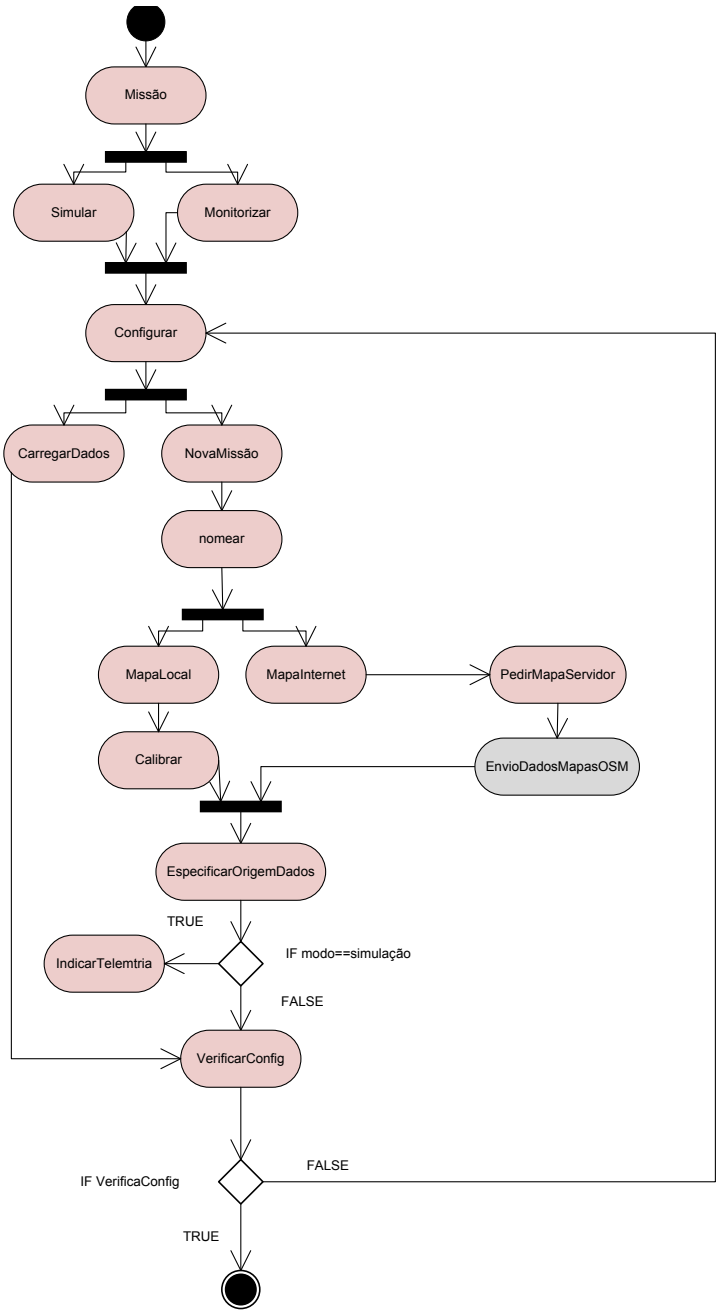

Diagrama 6 - Diagrama de Actividade durante a configuração de missão

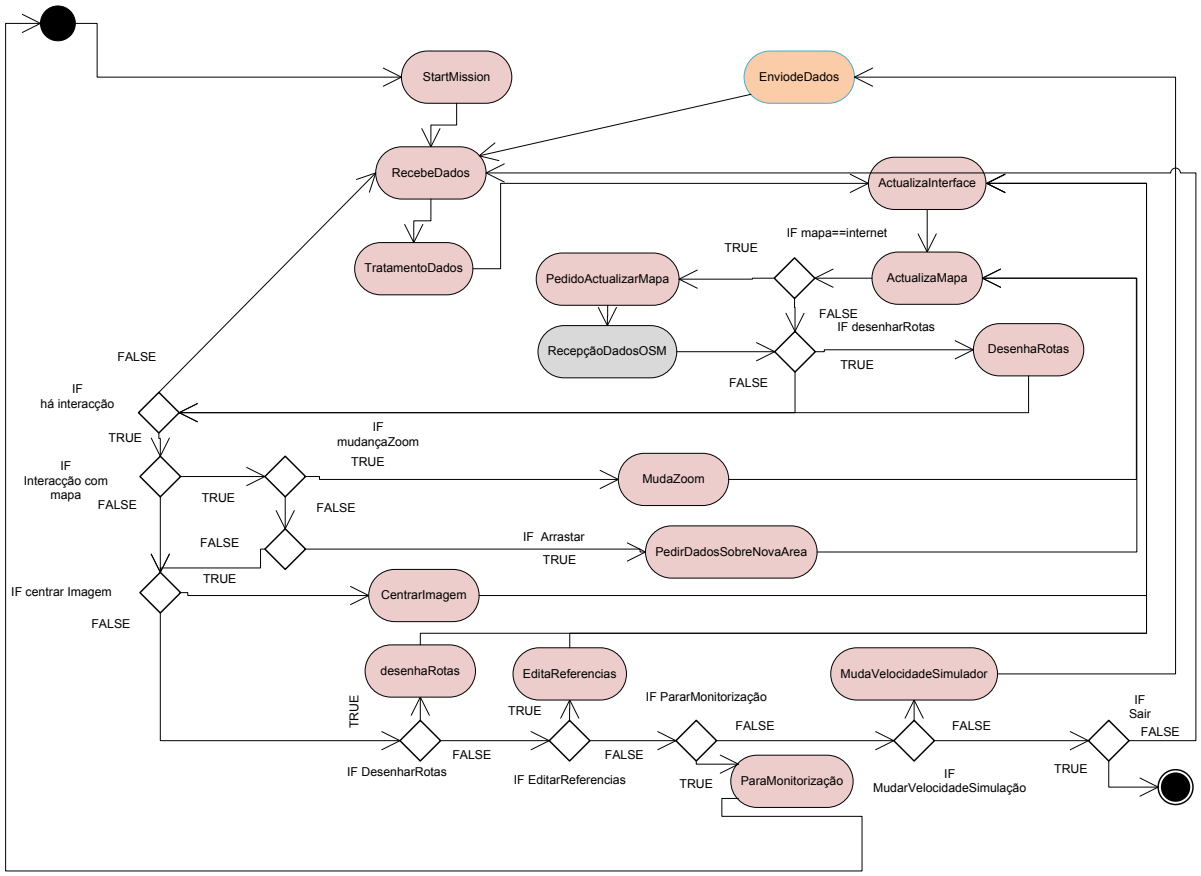

Diagrama 7 - Actividade do sistema DIVAgui durante a sua execução.

Capítulo 4. Desenvolvimento do sistema

Neste capítulo vamos apresentar as técnicas que foram utilizadas para a implementação da aplicação. Os módulos de interface e simulação serão apresentados em abordagens diferentes.

Interface

4.1.1. Qt

O Qt foi utilizado para a implementação das componentes de processamento de informação e de interface gráfica. Ao contrário de outras *framework*s, o Qt utiliza um método de comunicação baseada em sinais e *slots* (12). A emissão de um sinal é realizada com a ocorrência de um evento específico. O destinatário é chamado de *slot*. Este método é seguro e garante maior encapsulamento da informação.

4.1.2. Mostradores QWT

O conceito destes *widgets*, que estão disponíveis na biblioteca QWT, está em tudo relacionado com os *widgets* nativos do Qt. A comunicação entre os objectos QWT é também assegurada pelo método de sinais e *slots*.

4.1.3. Mapas e calibração

A construção do mapa de internet está implementada na classe MapViewer_net que instancia o *widget* QMapControl. Deste modo, todas as funcionalidades que permitem a configuração e interacção são responsabilidade da implementação deste *widget*. Este mapa possui duas camadas: a primeira contém o mapa e a segunda a rota.

O funcionamento básico deste *widget* é descrito no diagrama seguinte:

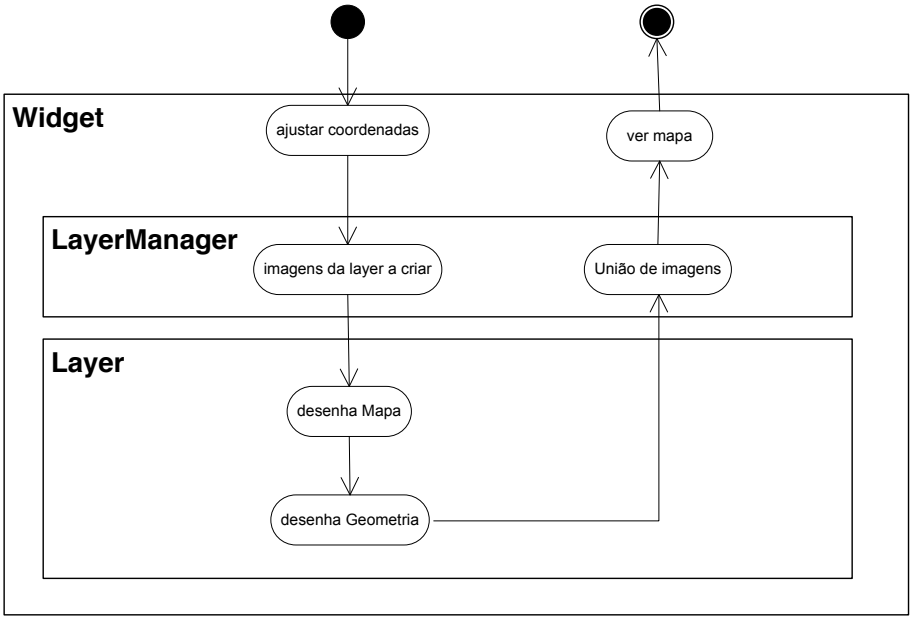

Diagrama 8 - Funcionamento básico do QMapControl

Por sua vez, a construção do mapa local foi implementada a partir da classe MapViewer_local que utiliza ficheiros de imagem, carregados para uma instância da classe QPixMap. A calibração de mapas é feita com os dados relativos às coordenadas GPS do canto superior esquerdo e do canto inferior direito do mapa, inseridos pelo utilizador. O mapeamento das coordenadas GPS para o sistema de coordenadas do mapa é feito do seguinte modo:

$$x = \frac{(GPSlongitude - LSClongitude) \times comprimento\_mapa}{RIClongitude - LSClongitude} \quad (1)$$

$$y = \frac{(GPSlatitude - LSClatitude) \times altura\_mapa}{LSClatitude - RIClatitude} \quad (2)$$

Legenda:

LSC – Canto superior esquerdo do mapa

RIC – Canto inferior direito do mapa

4.1.4. *Wizards*

O sistema de configuração de missões baseado em *wizards* foi implementado a partir da classe QDialog do Qt. De forma a validar os dados introduzidos, foram utilizadas instâncias da classe QValidator e algumas subclasses já implementadas.

4.1.5. CORBA

A classe CCorbaComm é responsável pela implementação do código que permite a comunicação com o servidor CORBA. Para isso foi utilizada a *framework* TAO. A classe Stock_Consumer, instanciada na classe CCorbaComm estabelece a ligação com a ORB, aguarda a recepção de dados e posteriormente envia-os para a classe CDiva.

4.1.6. Validação de dados

A validação dos dados recebidos está assegurada pela classe CValidData. Esta classe é instanciada na classe CDiva. Cada vez que um bloco de dados é recebido, a informação é validada neste objecto. Cada sensor é avaliado de acordo com as suas características e valores padrão. Alguns valores serão adaptados para serem mostrados na interface principal e outros são rejeitados pelo facto de não serem válidos.

Simulador

4.1.7. Princípios de desenvolvimento

A função principal do simulador recebe os dados de inicialização da aplicação. Estes parâmetros são: o caminho para o ficheiro com os dados de missão e o caminho IOR do servidor. Já durante o ciclo de execução do programa, esta função recebe comandos que controlam o simulador. A classe principal instancia a classe CDiva e esta é responsável pela direcção e velocidade da leitura de dados.

4.1.8. CORBA

A comunicação de dados do simulador para o programa de interface é também especificada pelo protocolo CORBA. O ORB é inicializado, o servidor é localizado pelo *Naming Service* e os dados são enviados a partir da execução de uma função do IDL.

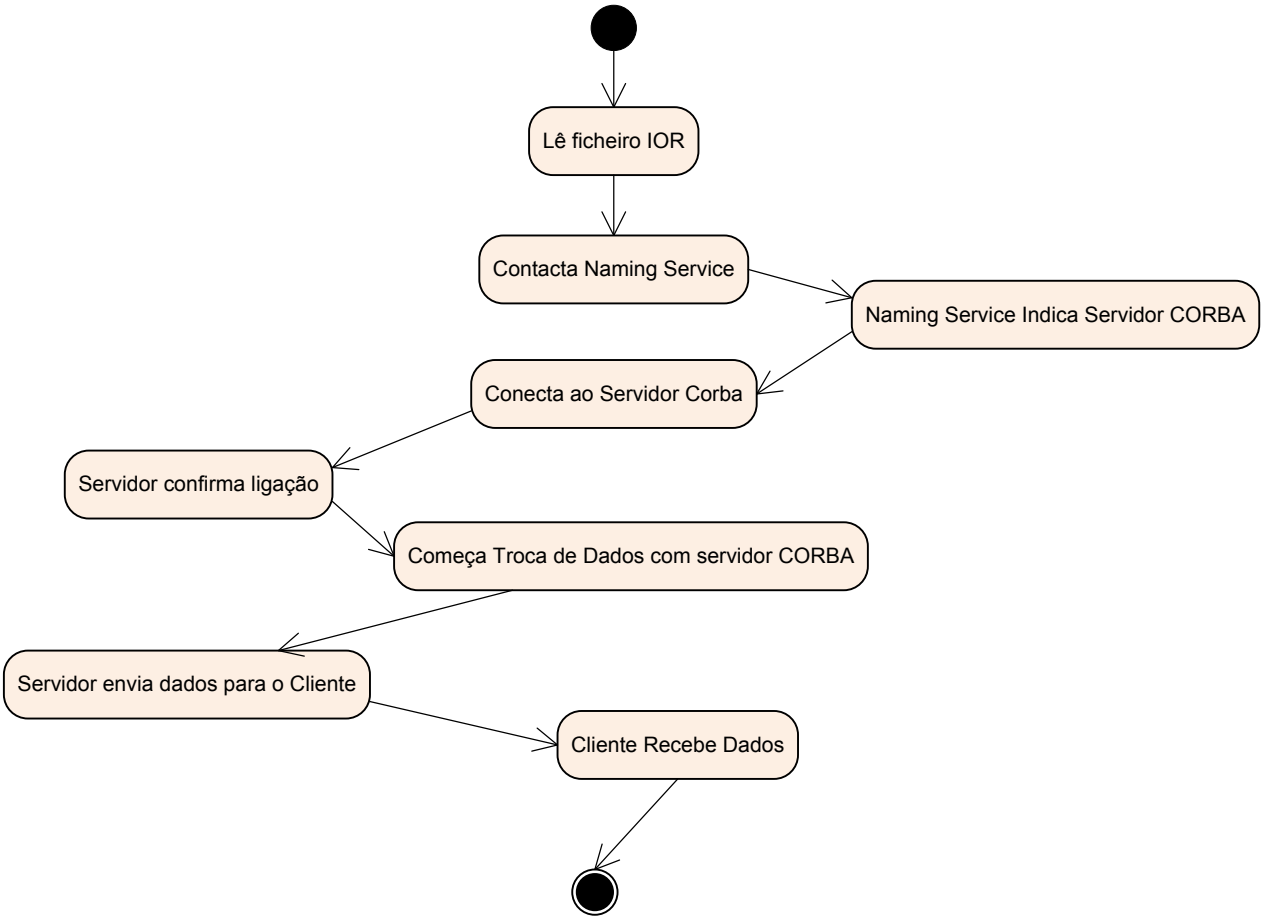

Diagrama 9 – Diagrama de actividades da ligação do simulador ao servidor até receber o primeiro bloco de dados.

Capítulo 5. Resultados e Conclusões

Este capítulo tem como objectivo descrever os efeitos da aplicação de determinados métodos no processo de desenvolvimento do projecto, apresentar os resultados da aplicação do *software* no contexto solicitado e expôr os conhecimentos que consolidámos com a sua realização.

A abordagem deste trabalho como um projecto de engenharia de *software* trouxe grandes vantagens, tanto na forma metódica e estruturada como organizámos os nossos planos como na forma consciente como preparámos os passos posteriores. A interacção com os elementos da equipa do DIVA foi fulcral para uma boa análise de requisitos, o que deu origem a um *design* de sistema adequado às necessidades. O estudo das opções a nível tecnológico foi também uma fase importante que acabou por conduzir todo o processo de implementação.

Em relação às competências e conhecimentos que adquirimos ao longo deste projecto, destacamos a aprendizagem das *frameworks* QT e TAO. Foi também muito importante a experiência que tivemos com o Linux, com o C++ e com as ferramentas *opensource* no geral. A formação que tivemos ao longo do curso, nomeadamente nas unidades curriculares de Interacção Homem-Máquina, Algoritmos e Estruturas de Dados, Sistemas Operativos e Engenharia de *Software*, foi imprescindível na realização deste projecto.

As principais dificuldades encontradas no desenvolvimento do projecto foram a familiarização com o funcionamento interno do ambiente de desenvolvimento e a instalação da *framework* TAO. A implementação dos clientes CORBA, apesar de ter sido uma das principais dificuldades, foi recompensada pelo contacto obtido com este protocolo de sistemas distribuídos.

A gestão da informação e do conhecimento foram práticas constantes ao longo do projecto devido a toda a documentação e planeamento desenvolvido na fase preliminar do projecto. O contacto permanente entre os membros do grupo através de ferramentas de difusão e recolha de conhecimento foi outro factor crítico no desenrolar do projecto, permitindo distribuição de tarefas.

No final deste projecto tivemos a oportunidade de testar o *software* em situação real, no âmbito das missões realizadas pelo DIVA. O resultado desses testes foram bastante satisfatórios, pelo que a aplicação teve um bom desempenho e permitiu o uso de grande parte das suas funcionalidades.

Bibliografia

1. **ISR-DEEC-FCTUC.** DIVA Homepage. *DIVA Website*. [Online] [Citação: 06 de Fevereiro de 2008.] http://paloma.isr.uc.pt/diva/.

2. **ISO Copyright Office .** *Software Engineering — Software Life Cycle Processes — Maintenance* . Software & Systems Engineering Standards Committee of the IEEE Computer Society , ISO. Geneve : www.iso.org, 2006. p. 13, International Standart. ISO/IEC 14764:2006(E).

3. **Fernandes, Cláudio, Nunes, José e Santos, Tiago.** DIVA GS - Meta 1 - Introdução ao Projecto. *Análise de Requisitos do Projecto DIVA GS* . [PDF]. Coimbra, Portugal : s.n., 20 de 04 de 2008. Identificação de Requisitos do Projecto;Design Conceptual;Planeamento de Projecto, p. 17.

4. **Wikipedia.** Comparison of Windows and Linux. *Wikipedia*. [Online] Comparison of Windows and Linux. [Citação: 8 de Junho de 2008.] http://en.wikipedia.org/wiki/Comparison_of_Windows_and_Linux.

5. —. Comparison of programming languages. *Wikipedia*. [Online] Wikipedia. [Citação: 03 de Setembro de 2008.] http://en.wikipedia.org/wiki/Comparison_of_programming_languages.

6. **Wikibooks.** Sistemas de Informação Distribuídos/Interoperação/Common Object Request Broker Architecture (CORBA). *Wikibooks, collection of open-content textbooks*. [Online] [Citação: 12 de Agosto de 2008.] http://pt.wikibooks.org/wiki/Sistemas_de_Informa%C3%A7%C3%A3o_Distribu%C3%ADdos/Interopera%C3%A7%C3%A3o/Common_Object_Request_Broker_Architecture_(CORBA).

7. **Fernandes, Claúdio, Nunes, José e Santos, Tiago.** DIVAgui - User's Manual. *User's Manual*. Coimbra : s.n., 2008.

8. **Pyarali, Irfan e Schmidt, Douglas C.** *An Overview of the CORBA Portable Object Adapter*. Department of Computer Science, Washington University. St. Louis : Washington University, 1998. p. 13, Paper.

9. **Wikipedia.** Information and Content Exchange. *Wikipedia*. [Online] Wikipedia. [Citação: 13 de Julho de 2008.] http://en.wikipedia.org/wiki/Information_and_Content_Exchange.

10. **OMG.** ORB Basics. *Object Management Group*. [Online] Object Management Group. [Citação: 12 de Agosto de 2008.] http://www.omg.org/gettingstarted/orb_basics.htm.

11. **Wikipedia.** CORBA. *Wikipedia*. [Online] Wikipedia. [Citação: 10 de Agosto de 2008.] http://pt.wikipedia.org/wiki/CORBA.

12. **Schmit, Douglas.** The ADAPTIVE Communication Environment (ACE). *The ADAPTIVE Communication Environment (ACE)*. [Online] [Citação: 15 de Junho de 2008.] http://www.cs.wustl.edu/~schmidt/ACE.html.

13. —. Real-time CORBA with TAO (The ACE ORB). [Online] [Citação: 15 de Junho de 2008.] http://www.cse.wustl.edu/~schmidt/TAO.html.

14. **OMG.** *General Inter-ORB Protocol*. [ed.] Object Management Group. EUA, Julho de 2002. Communication Protocol.

15. **Parekh, Nilesh.** Software Testing - White Box Testing Strategy. *Buzzle.com - Intelligent Life on the Web*. [Online] Byzzle, 04 de 11 de 2005. [Citação: 02 de 09 de 2008.] http://www.buzzle.com/editorials/4-10-2005-68350.asp.

16. **OMG.** CORBA FAQ. *The Object Management Group (OMG)*. [Online] [Citação: 12 de Agosto de 2008.] http://www.omg.org/gettingstarted/corbafaq.htm#HowWork.

17. **IP Devel.** Development Methodology. *IP Devel - Embedded Software Outsourcing*. [Online] IP Devel. [Citação: 03 de Setembro de 2008.] http://www.ipdevel.net/index.php?mid=53&pid=53.

18. **Wikipedia.** Graphical User Interface. *Wikipedia*. [Online] [Citação: 05 de Setembro de 2008.] http://en.wikipedia.org/wiki/Graphical_user_interface.

19. Interoperable Object Reference (IOR). *CORBA EXPLAINED SIMPLY*. [Online] ciaranmchale.com. [Citação: 2008 de Agosto de 12.] http://www.ciaranmchale.com/corba-explained-simply/interoperable-object-reference.html.

20. **Wikipedia.** Qmake. *Wikipedia*. [Online] [Citação: 06 de Setembro de 2008.] http://en.wikipedia.org/wiki/Qmake.

21. **Trolltech.** qmake Manual. *Qt Reference Documentation*. [Online] Trolltech, 2008. [Citação: 05 de Setembro de 2008.] http://doc.trolltech.com/4.4/qmake-manual.html.

22. **Wikipedia.** Qt (toolkit). *Wikipedia*. [Online] [Citação: 02 de Junho de 2008.] http://en.wikipedia.org/wiki/Qt_(toolkit).

23. **Trolltech.** Qt Cross-Platform Application Framework. *Trolltech - Code Less, Create More. Deploy Everywhere*. [Online] Trolltech, 2008. [Citação: 08 de Junho de 2008.] http://trolltech.com/products/qt.

24. —. Signals and Slots. *Qt Reference Documentation*. [Online] Trolltech, 2008. [Citação: 04 de Setembro de 2008.] http://doc.trolltech.com/4.2/signalsandslots.html.

25. **Wikipedia.** TAO (software). *Wikipedia*. [Online] Wikipedia. [Citação: 14 de Julho de 2008.] http://en.wikipedia.org/wiki/TAO_(software).

26. **Helmut, Herold.** Qt - Portable GUI Programming under Linux/Unix/Windows. *Linux Knowledge Portal*. [Online] 09 de Junho de 2008. [Citação: 28 de Setembro de 2008.] http://www.linux-knowledge-portal.org/en/content.php?&content/programming/qt.html.

27. **Koundinya.** Black Box Testing, Its advantages and disadvantages. *The Code Project*. [Online] The Code Project, 03 de Dezembro de 2003. [Citação: 02 de Setembro de 2008.] http://www.codeproject.com/KB/architecture/Black_Box_Testing.aspx.

28. **Microsoft.** *Distributed Component Object Model (DCOM) Remote Protocol Specification*. EUA, 14 de Março de 2007.

29. **Rathmann, Uwe e Wilgen, Josef.** Qwt - Qt Widgets for Technical Applications. *Qwt - Qt Widgets for Technical Applications*. [Online] sourceforge, 24 de Maio de 2008. [Citação: 30 de Maio de 2008.] http://qwt.sourceforge.net/.

30. **Trzewik, Artur, et al.** Advantages and Disadvantages of OOP. *Tcler's Wiki*. [Online] 24 de Janeiro de 2005. [Citação: 04 de Setembro de 2008.] http://wiki.tcl.tk/13398.

31. **Webber, Neil, O'Connell, Conleth e Hunt, Bruce.** The Information and Content Exchange (ICE) Protocol. *World Wide Web Consortium*. [Online] W3C, 26 de Outubro de 1998. [Citação: 2008 de Julho de 2008.] http://www.w3.org/TR/NOTE-ice.

32. **Wells, Don.** What is Extreme Programming? *Extreme Programming: A gentle introduction*. [Online] 1999. [Citação: 04 de Setembro de 2008.] http://www.extremeprogramming.org/what.html.

33. **Rodriges, Pimenta, Pereira, Pedro e Sousa, Manuela.** *Programação em C++ : Conceitos Básicos e Algoritmos*. 5ª Edição. Lisboa : FCA, 1998. ISBN: 972-722-038-X.

34. **Rodrigues, Pimenta, Pereira, Pedro e Sousa, Manuela.** *Programação em C++: Algoritmos e Estruturas de Dados*. 2ª Edição. Lisboa : FCA, 2000. p. 687. ISBN: 972-722-199-8.

35. **Molkentin, Daniel.** *The Book of Qt 4: The Art of Building Qt Applications*. s.l. : No Starch Press, 2006. p. 442. ISBN 978-3-937514-12-3.

36. **Blanchette, Jasmin e Summerfield, Mark.** *C++ GUI Programming with Qt 4*. s.l. : Prentice Hal, 2006. p. 560. ISBN: 978-0-13-187249-3.

37. **Hamilton, Marc e Kern, Harris.** *Version Control with Subversion : For Subversion 1.1*. s.l. : Prentice Hall PTR, 1999. p. 357. ISBN: 9780130812469.

38. **Dalheimer, Matthias Kalle.** *Programação em Qt*. [ed.] Paulo Marques. [trad.] Alessandra Duarte. Rio de Janeiro : Editora Ciência Moderna, 1999. p. 335. ISBN: 85-7393-053-5.

Anexos

De forma a complementar este relatório, como anexos, incluímos dois documentos: o manual de referência e o manual do utilizador.

- **Manual de Referência:** Apresenta a estrutura interna da aplicação. É um documento orientado para futuros desenvolvedores que pretendam adaptar, corrigir ou modificar funcionalidades do sistema.
- **Manual do Utilizador** (7)**:** Apresenta uma abordagem mais funcional e descritiva da aplicação. Este documento é completamente orientado para o utilizador e pretende auxiliá-lo nas actividades do programa.