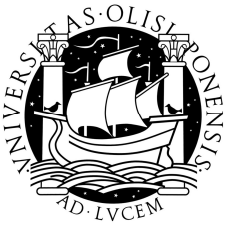
UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática




# FLIGHT DYNAMICS FACILITY – EXTERNAL INTERFACES

projecto realizado na

Critical Software S.A.

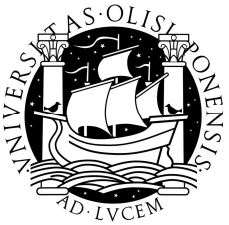
por

João Manuel Ribeiro Gonçalves

Mestrado em Engenharia Informática

2007

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática




# FLIGHT DYNAMICS FACILITY – EXTERNAL INTERFACES

projecto realizado na

Critical Software S.A.

por

João Manuel Ribeiro Gonçalves

Projecto orientado pelo Prof. Dr. Vasco Vasconcelos
e co-orientado pelo Eng. Luís Teixeira

Mestrado em Engenharia Informática

2007

DEPARTAMENTO DE INFORMÁTICA

Faculdade de Ciências – Universidade de Lisboa

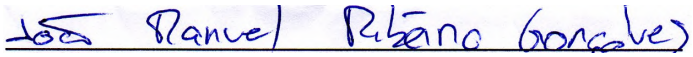
Bloco C6 – Piso 3 – Campo Grande, 1749-016 Lisboa

Tel. & Fax: 351.217500084

## Declaração

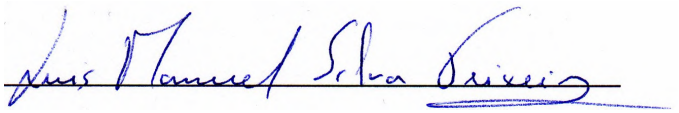
João Manuel Ribeiro Gonçalves, aluno n.º 30337 da Faculdade de Ciências da Universidade de Lisboa, declara ceder os seus direitos de cópia sobre o seu Relatório de Projecto em Engenharia Informática, intitulado "Flight Dynamics Facility – External Interfaces", realizado no ano lectivo de 2006/2007 à Faculdade de Ciências da Universidade de Lisboa para o efeito de arquivo e consulta nas suas bibliotecas e publicação do mesmo em formato electrónico na Internet.

FCUL, 05 de Junho de 2007



Luís Teixeira, supervisor do projecto de João Manuel Ribeiro Gonçalves, aluno da Faculdade de Ciências da Universidade de Lisboa, declara concordar com a divulgação do Relatório do Projecto em Engenharia Informática, intitulado "Flight Dynamics Facility – External Interfaces".

Critical Software S.A., 05 de Junho de 2007



# Agradecimentos

Ao amigo e gestor do projecto FDF, Luís Teixeira, por ter tido a paciência necessária para lidar com uma pessoa sem experiência profissional e por ter partilhado o extenso conhecimento, tornando este estágio numa experiência pedagógica única.

Ao amigo e colega Nuno Esculcas, que trabalhando no mesmo projecto contribuiu com todo o seu conhecimento de modo a que pudesse fazer frente a todas as adversidades enfrentadas ao longo do estágio.

Ao Professor Doutor Vasco Vasconcelos, orientador do projecto por parte da faculdade, pelo rigor imposto ao longo do estágio e pela compreensão demonstrada, nunca perdendo a noção que o estágio estava a ser realizado numa empresa e que por isso existem pormenores que nem sempre podem ser planeados.

Aos colegas da Critical Software que desde o início me acolheram como sendo um deles, disponibilizando-se a ajudar. A Critical Software é o caso em que as pessoas pela sua mentalidade e qualidade de trabalho fazem da empresa o que ela é: uma empresa de sucesso.

Aos meus pais, família e a todos amigos. Aos meus pais e irmão que desde o início mostraram orgulho no que faço e por me forçaram a acreditar nas minhas qualidades, levando-me a ultrapassar todos os obstáculos e levando-me a crer que com trabalho tudo é possível. Aos amigos Bruno Duarte e João Silva, colegas de projectos na faculdade, que me apoiaram em alturas difíceis, contribuindo cada um com o seu conhecimento nas mais diversas alturas.

Em especial à minha namorada Cátia, com quem tive o privilégio de partilhar o local de trabalho, pela sua dedicação e compreensão, pela força dada em todos os momentos e por me fazer acreditar.

# Resumo

O sistema Galileo consiste numa constelação de 30 satélites que disponibilizará diversos serviços, entre os quais, serviços de posicionamento e navegação alternativos aos actualmente disponibilizados pelo GPS.

Neste projecto será concebido o componente External Interfaces que é parte integrante do elemento Flight Dynamics Facility (FDF), responsável pelo cálculo das órbitas e comportamentos dos satélites que fazem parte da constelação GALILEO.

O componente External Interfaces é responsável por disponibilizar serviços de comunicação ao FDF, oferecendo a este a possibilidade de comunicar com elementos internos e externos ao Ground Control Segment (GCS) cuja função é tratar todas as operações relativas à manutenção e gestão dos satélites. Tipicamente o FDF irá receber dados e executar um conjunto de operações sobre esses mesmos dados, sendo que os resultados serão depois direccionados para os elementos internos ou externos através do componente External Interfaces.

Deste modo o projecto pretende consolidar os requisitos de software do componente External Interfaces, efectuando de seguida todas as alterações decorrentes dessa consolidação no desenho do componente, ou seja, consolidar a arquitectura e desenho detalhado. Foi ainda necessário elaborar um conjunto de documentos com os planos de testes (aceitação/integração e unitários) garantindo assim que seria possível durante e após a fase de implementação, também incluída no projecto, verificar e validar a mesma. Devido ao facto do componente desenvolvido incluir uma GUI, desenvolvida no decorrer do estágio, foi necessária a elaboração do manual de utilização da mesma.

PALAVRAS-CHAVE: External Interfaces, Galileo, Engenharia de Software, FDF, C++, CORBA.

# Abstract

The Galileo system, which consists of 30 satellites orbiting the earth, is expected to pinpoint a geographical position to within a single meter. Because the service's availability will be guaranteed in almost any circumstance, the system will be ideal for applications in which precision and reliability are critical, such as air traffic management (among many possible examples). This system pretends to be an alternative to the GPS.

This internship is about the development of the External Interfaces (EXIF) component, which is an integrant part of the Flight Dynamics Facility (FDF) element. This element is responsible for orbit determination and manoeuvre planning of the satellites that compose the Galileo constellation.

The component External Interfaces is responsible for providing communication services to the FDF, giving the FDF the possibility to establish interfaces between internal and external Ground Control Segment (GCS) elements. The GCS will handle spacecraft housekeeping and constellation maintenance. Typically the FDF will receive data, execute a set of operations on that data and store the produced results. Those results shall then be sent to external elements through the External Interfaces component.

To develop the External Interfaces all component's system and software requirements were consolidated, performing all the necessary changes in the External Interfaces Arquitecture and Detailed Design. In order to perform validation and verification of the developed code it shall also be necessary to produce the Acceptance, Integration and Unit Test plans. Since the component has a GUI, it's mandatory to produce the correspondent software user manual.

KEYWORDS: External Interfaces, Galileo, Software Engineering, FDF, C++, CORBA.

# Índice

# Índice de Figuras

# Índice de Tabelas

# Capítulo 1

## Introdução

Existem actualmente dois sistemas de posicionamento por satélite: o GPS e o GLONASS. Ambos são financiados e controlados por autoridades militares (dos Estados Unidos e da Rússia, respectivamente). A sua continuidade e fiabilidade dos dados que transmitem dependem, por conseguinte, das referidas autoridades que podem, por exemplo, interromper ou deteriorar o sinal a qualquer momento. Por esta e outras razões, a União Europeia decidiu construir o seu próprio sistema sob a designação de Galileo.

O projecto Galileo baseia-se numa constelação de trinta satélites, especificamente concebidos para uma utilização civil, e em estações terrestres que permitem fornecer informações relativas ao posicionamento de utentes de inúmeros sectores, como por exemplo, transportes (localização de veículos, busca de itinerário, controlo da velocidade, sistemas de orientação, etc.), serviços sociais (auxílio aos deficientes ou aos idosos), justiça e serviços aduaneiros (localização de suspeitos, controlos nas fronteiras), obras públicas (sistemas de informação geográfica) e assistência a pessoas em perigo ou actividades de lazer (orientação no mar e na montanha, etc.).

O programa Galileo será gerido e controlado por civis e oferece uma garantia de qualidade e continuidade essencial a várias aplicações sensíveis. A sua complementaridade com os actuais sistemas vai aumentar a fiabilidade e disponibilidade de serviços de navegação por satélite em todo o globo.

O projecto discutido neste documento refere-se a um componente, *External Interfaces,* parte integrante do elemento *Flight Dynamics Facility*, responsável pelo cálculo das órbitas e comportamento dos satélites que fazem parte da constelação Galileo. Este componente ficará responsável por fornecer os meios necessários de modo a que outros elementos do Galileo consigam estabelecer interface com o elemento *Flight Dynamics Facility*.

## 1.1. Âmbito

Este estágio foi realizado no âmbito da disciplina de Projecto de Engenharia Informática do Mestrado em Engenharia Informática, integrando-se no projecto Galileo da Agência Espacial Europeia que, como já foi referido, pretende dotar a União Europeia de um sistema de posicionamento por satélite totalmente independente dos sistemas dos Estados Unidos e da Rússia.

## 1.2. Acrónimos

Esta secção pretende fornecer uma lista dos acrónimos usados ao longo deste documento. A Tabela 1 contém uma descrição desses acrónimos. Este acrónimos podem ser encontrados em [1].

| Acrónimos | Descrição |
|---|---|
| CORBA | Common Object Request Broker Architecture |
| CMCF | Central Monitoring and Control Facility |
| CSW | Critical Software, S.A. |
| ESCC | External Satellite Control Centre |
| EXIF | External Interfaces |
| FDF | Flight Dynamics Facility |

| | |
|---|---|
| FTP | File Transfer Protocol |
| GCS | Ground Control Segment |
| GFTS | Generic File Transfer System |
| GLONASS | Global Navigation Satellite System |
| GMS | Ground Mission Segment |
| GPS | Global Positioning System |
| GSS | Galileo Sensor Stations |
| GSWS | GALILEO SW Standards |
| HTTPS | Secure Hypertext Transfer Protocol |
| SCCF | Spacecraft & Constellation Control Facility |
| SCPF | Spacecraft Constellation Planning Facility |
| SNMP | Simple Network Management Protocol |
| M&C | Monitoring & Control |
| OPF | Operations Preparation Facility |
| ORB | Object Request Broker |
| TBD | To Be Defined |
| TBC | To Be Confirmed |
| TCP-IP | Transmission Control Protocol – Internet Protocol |
| TT&C | Tracking Telemetry and Command |
| TTCF | Tracking Telemetry and Command Facility |
| XML | Extensible Mark-up Language |

*Tabela 1: Lista de acrónimos*

## 1.3. Objectivos

O objectivo do estágio realizado consistiu em desenvolver uma aplicação que funcione como uma *gateway* de comunicação entre vários elementos que constituem o sistema Galileo (para obter informação mais detalhada sobre o contexto do EXIF ver Capítulo 2). Essa aplicação, denominada *External Interfaces (EXIF)* deverá interagir com outros elementos de forma a receber *inputs* necessários ao seu funcionamento e a fornecer os resultados necessários ao funcionamento de outros elementos ([2] e [3]).

Neste contexto o componente *EXIF* deverá conter uma base de dados relacional onde toda a informação relevante será guardada, tendo ainda de implementar um conjunto de funcionalidades de modo a poder interagir com os componentes internos e externos ao sistema e dos quais depende.

## 1.4. Contexto Empresarial

A Critical Software fornece soluções, serviços e tecnologias para os sistemas de informação críticos das empresas, respondendo às necessidades de clientes de diversos mercados, designadamente, telecomunicações, sector público, indústria, sector aeroespacial e defesa. A empresa foi fundada em 1998 e actualmente tem escritórios em Coimbra, Porto e Lisboa, Portugal, no Reino Unido e em San Jose, Califórnia, EUA. O sucesso da Critical Software reside na utilização de níveis elevados de qualidade e inovação tecnológica como agentes na introdução de vantagens competitivas nos sistemas de informação e no negócio dos clientes e parceiros.

A inovação das soluções desenvolvidas está patente no facto de a empresa comercializar actualmente no mercado internacional produtos inovadores em áreas como a Confiabilidade e a Computação de Alto Desempenho.

A empresa opera ainda com base num sistema de qualidade certificado que segue a norma ISO 9001:2000 segundo o referencial TickIT (*British Standard Institute*), sendo actualmente a única empresa certificada na Península Ibérica.

## 1.5. Organização do Documento

A ordem pela qual este documento está organizado diz respeito à ordem das tarefas desempenhadas na Critical Software S.A. no decorrer do estágio. Esta secção apresenta a organização deste documento fornecendo uma breve descrição sobre o conteúdo de cada um dos capítulos que o compõem:

- O Capítulo 1 contém uma breve introdução sobre o trabalho efectuado no estágio, o seu âmbito, os objectivos e o contexto empresarial onde o estágio foi realizado.
- O Capítulo 2 (Contexto do Projecto – O Sistema Galileo) pretende oferecer um enquadramento do projecto no sistema Galileo, descrevendo a sua utilidade e todos os seus componentes relevantes, indicando onde será inserido o trabalho desenvolvido.
- O Capítulo 3 (Análise) refere-se ao trabalho de análise efectuado não só como forma de conhecer o projecto mas também como forma de consolidar o trabalho já realizado antes do início do estágio.
- O Capítulo 4 (Desenho) contém a interpretação em termos de desenho do componente do trabalho desenvolvido no capítulo anterior. O trabalho que se descreve neste capítulo é referente a toda uma fase de consolidação, quer da arquitectura quer do desenho detalhado do componente *EXIF.*
- O Capítulo 5 (Verificação & Validação) é relativo à elaboração dos vários documentos de testes necessários para validar e verificar todo o trabalho desenvolvido.
- O Capítulo 6 (GUI e Manual do Utilizador) contém uma breve descrição da GUI desenvolvida para o *EXIF* e o respectivo manual para a utilização da mesma.
- O Capítulo 7 (Implementação) é como o título indica um resumo do trabalho realizado na fase de implementação, enunciando as tecnologias usadas e algumas métricas relativas a esta fase.
- O Capítulo 8 pretende não só fazer um balanço final do estágio mas apresentar também o que poderá ser feito no componente *EXIF* em termos de trabalho futuro.

# Capítulo 2

## Contexto do Projecto – O Sistema Galileo

O sistema Galileo vai incorporar componentes globais, regionais e locais ([4]).

O componente global é o núcleo do sistema, compreendendo os satélites, *Space Segment*, e o *Ground Segment*.

O componente regional do Galileo irá conter um conjunto de *External Region Integrity Systems (ERIS),* ou seja, sistemas que serão implementados e operados por organizações, países ou conjuntos de países exteriores à Europa. Estes sistemas serão usados de modo a garantir a integridade de serviços independentes do sistema Galileo, de modo a, por exemplo, satisfazer restrições legais no que diz respeito a garantias do sistema.

Os componentes locais a serem criados terão por objectivo aumentar o desempenho do Galileo localmente. Este aumento de desempenho reflecte-se, por exemplo, na entrega de sinal onde os sinais provenientes dos satélites não conseguem chegar.

Visto que o trabalho deste estágio foi realizado no contexto do componente global do sistema Galileo, apenas será fornecida uma breve descrição deste componente e dos seus constituintes.

## 2.1. Serviços Oferecidos

Graças à compatibilidade e à interoperabilidade dos dois sistemas, Galileo e GPS, os utilizadores de todo o mundo terão facilmente acesso aos sinais emitidos pelos satélites de navegação e beneficiarão de um desempenho de muito maior qualidade. O Galileo oferecerá, além disso, uma precisão superior à do GPS.

Os serviços oferecidos pelo Galileo beneficiarão de uma garantia de continuidade que poderá mesmo ser objecto de contrato: uma inovação extremamente importante quando estão em causa vidas humanas, como é o caso no controlo do tráfego aéreo ou ferroviário.

O Galileo assegurará a independência estratégica da Europa e permitirá às empresas europeias participar num sector industrial em pleno desenvolvimento, cujo mercado anual poderá atingir mais de 200 000 milhões de euros em 2020, com 3 mil milhões de receptores em serviço.

O Galileo simboliza ainda o advento de uma revolução tecnológica superior à iniciada pelos telemóveis, ao permitir o desenvolvimento de uma nova geração de serviços universais de valor acrescentado em todas as áreas do nosso quotidiano, dos transportes às telecomunicações, da agricultura à pesca ou ao ensino.

Este sistema será colocado à disposição da comunidade aeronáutica para atender todas as suas exigências em matéria de navegação, incluindo a assistência à aterragem de precisão. Será também de grande utilidade para todos os utilizadores de sistemas de comunicações móveis, marítimas e terrestres, sistemas de navegação, posicionamento, salvamento, etc. ([4] e [5]).

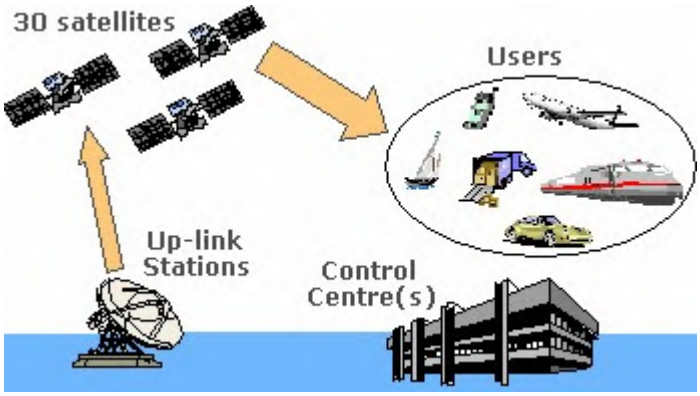



*Figura 1: Funcionamento geral do Galileo*

## 2.2. Componente Global

O componente global do Galileo encontra-se dividido em dois segmentos: o *Space Segment* e o *Ground Segment*.

O *Space Segment* fornecerá a constelação dos satélites Galileo, sendo que cada um transmitirá sinais de navegação juntamente com os sinais de dados de navegação. Os dados de navegação vão conter os dados de correcção essenciais não só para a navegação mas também os sinais da integridade que fornecem um aumento de serviço *space-based*.

O *Ground Segment* do Galileo complementa o *Space Segment*, compreendendo um par de centros de controlo e uma rede global de estações de transmissão e recepção de dados.

### 2.2.1 Ground Segment

O núcleo do Galileo *Ground Segment* será composto por dois centros de controlo. Cada um destes centros de controlo vai gerir funções de "controlo", suportadas por um *Ground Control Segment (GCS)* dedicado e funções de "missão", suportadas por um *Ground Mission Segment (GMS)* dedicado ([2], [3] e [5]).

#### 2.2.1.1 Ground Control Segment

O GCS tratará de todas as operações relativas à manutenção e gestão dos satélites. Esta propriedade é garantida através de uma combinação de telemetria enviada de modo intermitente e através do contacto telecomandado a satélites individuais, permitindo monitorizar e controlar o satélite e oferecendo ao GCS uma síntese do estado de toda a constelação de satélites.

Para além da monitorização e controlo de satélites, o GCS é também responsável pelo planeamento das operações a serem desempenhadas pelos satélites, incluindo um número de funções de suporte que vão permitir que estas sejam executadas de um modo correcto e seguro.

O GCS terá à sua disposição uma rede global de cinco estações *TTC* para comunicar com cada satélite usando um esquema regular de contactos planeados, expedições de teste a longo prazo e contactos de contingência.

A interacção do GCS com os restantes elementos encontra-se descrita na Figura 2:

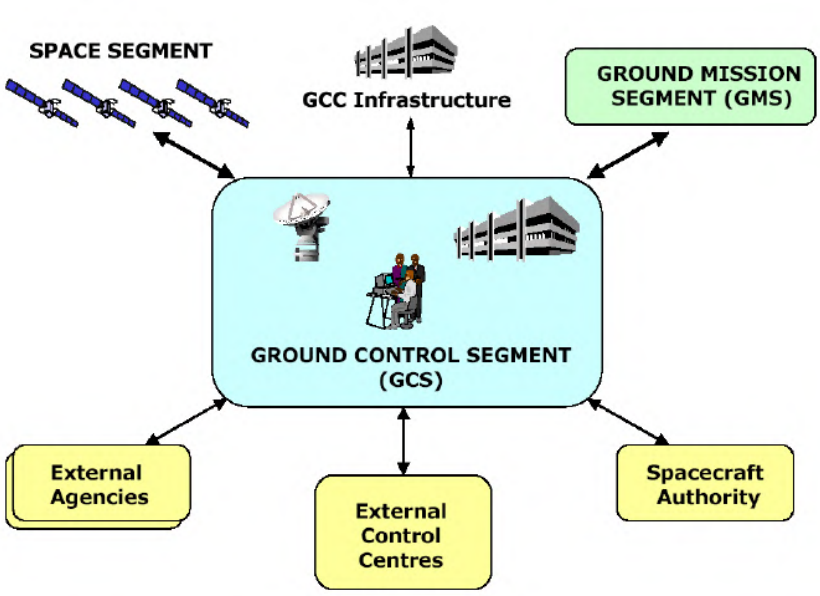



*Figura 2: Ground Control Segment – Visão Global*

Os sistemas externos ao GCS são:

- *Space Segment* – O GCS gere o *Space Segment* durante todas as fases da Constelação;
- *Ground Mission Segment* (GMS) – O GMS fornece dados das missões para serem enviados para o GCS;
- *Ground Control Centre (GCC) Infrastructure* – Fornece serviços básicos essenciais para o correcto funcionamento do GCS;
- *External Agencies, External Control Centres, Spacecraft Authority* - Fornecem serviços ao GCS, mas não estão em contacto permanente com este.

### *Flight Dynamics Facility*

O *Flight Dynamics Facility,* responsável pelo cálculo das órbitas e comportamentos dos satélites que fazem parte da constelação Galileo, encontra-se inserido no GCS e faz interface não só com elementos externos ao GCS mas também com elementos internos. Esta funcionalidade poderá ser observada na Figura 3, figura esta que apresenta de um modo mais detalhado a constituição do GCS podendo assim observar-se os elementos com os quais o FDF/EXIF terá de interagir ([2], [3] e [5]):

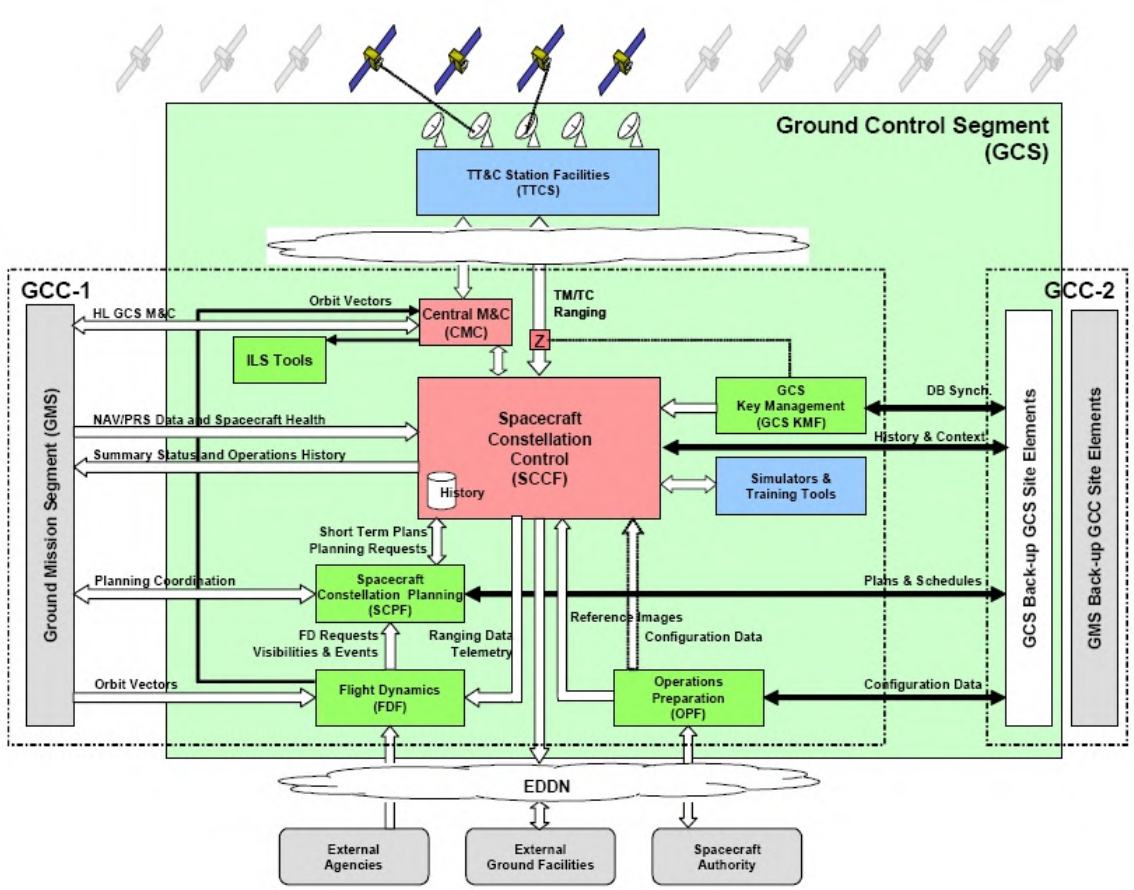



*Figura 3: Ground Control Segment – Visão detalhada*

Os elementos observáveis na figura são:

- *Spacecraft & Constellation Control Facility* (SCCF) – Efectua operações de monitorização e controlo dos satélites.
- *Satellite Constellation Planning Facility* (SCPF) – Este elemento calendariza o contacto regular com os satélites (1 contacto por órbita) de modo a serem efectuadas operações de rotina.
- *Flight Dynamics Facility* (FDF) – O FDF suporta o cálculo de órbitas, planeamento de manobras e monitorização da comportamento individual dos satélites e da própria constelação. É também capaz de determinar órbitas.
- *Operations Preparation Facility* (OPF) – O OPF é responsável pela preparação e configuração de controlo de todos os procedimentos operacionais, incluindo aqueles que virão a ser realizados de um modo automático.
- *Key Management Facility* (GCS KMF) – Este elemento está directamente relacionado com os aspectos de segurança das missões.
- Central Monitoring and Control Facility (CMCF) – O CMCF suporta a monitorização e controlo de todos as estações terrestres do GCS.
- *Simulator, Training and Integrated Logistic Support Tools*.

O *EXIF*, interno ao FDF, é um elemento crítico para o desempenho geral do sistema. É responsável pela comunicação entre todas as GMS, GCS, elementos externos e elementos internos

ao FDF. Recebe informação proveniente de vários elementos externos e internos ao FDF, e reenvia essa informação para componentes internos/externos.

A Figura 4 ilustra os tipos de dados com os quais o FDF terá de lidar e, consequentemente, aqueles que o *EXIF* terá de transmitir ([2]):

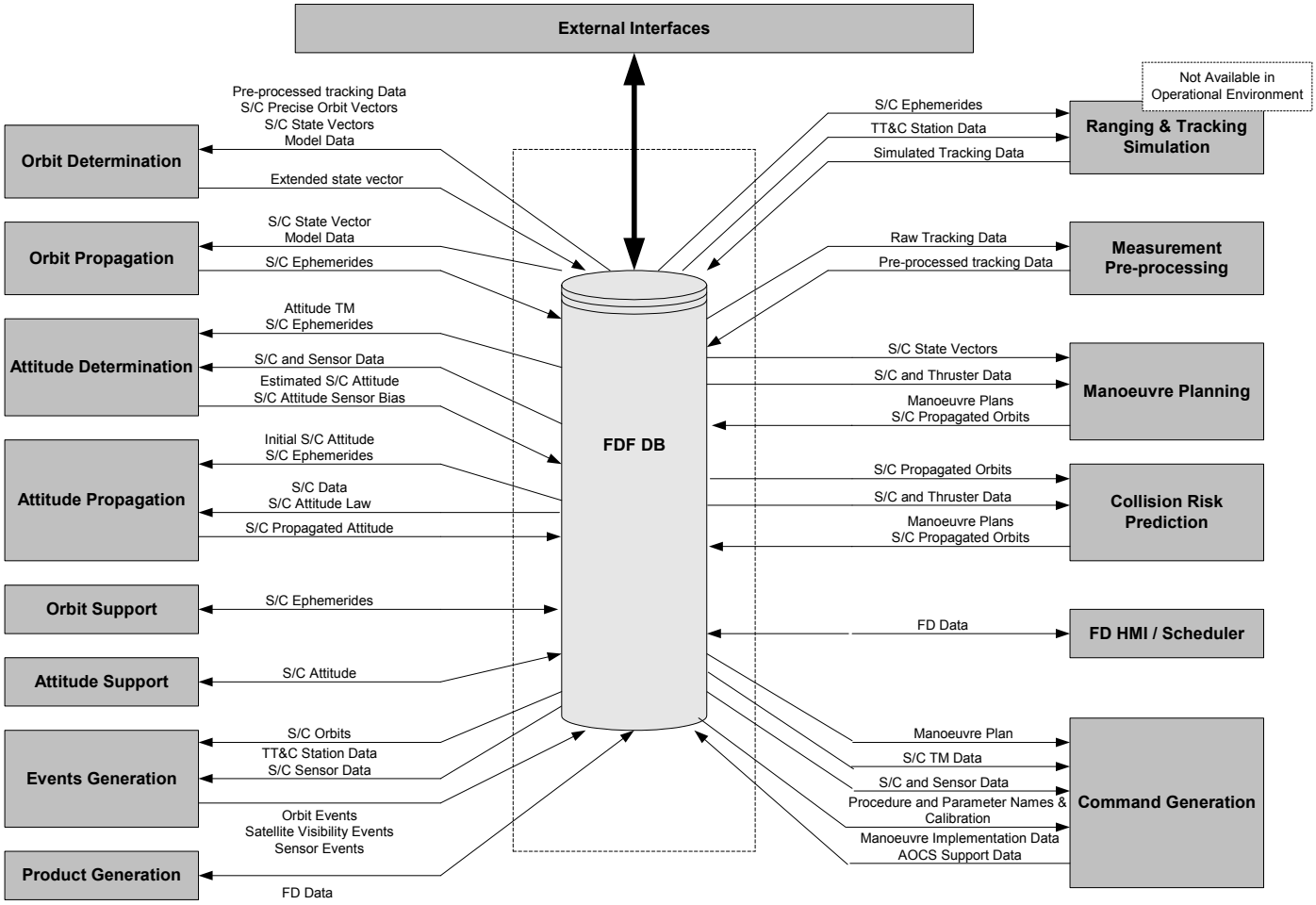



*Figura 4: Flight Dynamics Facility*

O facto de todas as interfaces necessárias para comunicar com os componentes externos se encontrarem reunidas num só componente, o *EXIF,* vai fazer com que o mesmo tenha uma arquitectura modular, extensível e reutilizável, onde detalhes como pormenores da interface, implementação de protocolos, portos e outros pormenores de localização são implementados e conhecidos. Assim a adição de novos elementos externos é conseguida de um modo mais fácil e intuitivo, minimizando o impacto nos componentes internos.

#### 2.2.1.2 Ground Mission Segment

O *Ground Mission Segment (GMS)* utilizará uma rede global de trinta *Galileo Sensor Stations (GSS)* de modo a monitorizar os sinais de navegação provenientes de todos os satélites. Esta monitorização será efectuada através de comunicações pormenorizadas que são trocadas usando satélites comerciais e ligações por cabo onde cada ligação será duplicada para se obter redundância.

O GMS vai usar a rede de GSS com dois propósitos independentes:

- Através da função de *Orbitography Determination and Time Synchronisation* (OD&TS). Esta função vai fornecer um processamento, a cada dez minutos, de todas as observações de todos os satélites, calculando as órbitas precisas e o *clock offset* de cada satélite, válidas por várias horas. Será feito o *upload* desta informação para os respectivos satélites a cada 100 minutos.
- O segundo uso da rede GSS é de *Integrity Processing Function* (IPF). O propósito do IPF é o de fornecer observações instantâneas de todos os GSS de cada satélite de modo a garantir a integridade dos sinais. O resultado desta computação, para toda a constelação, será enviado para os satélites seleccionados e transmitida de modo a que qualquer utilizador receba sempre, no mínimo duas *Integrity Messages*.

### 2.2.2 Space Segment

O *Space Segment* do Galileo será constituído por uma constelação de 30 satélites. O desenho desta constelação tem por objectivo fornecer os serviços mencionados em 2.1, pretendendo ser robusta o bastante para conseguir manter o seu normal funcionamento mesmo quando um satélite falha.

Todos os satélites serão colocados a uma altitude de 23,616 km, o que quer dizer que cada satélite irá demorar 15 horas e 4 minutos para completar uma órbita. Esta órbita será rastreada a cada 10 dias, ou seja, a cada 10 dias será efectuado um *groundtrack*. O *groundtrack* não é mais que o "desenho" de uma linha imaginária que liga o centro da terra a um satélite, linha essa que será criada com base no período orbital, na altitude e na inclinação de um satélite ([4]).

## 2.3. Sumário

Este capítulo serviu de introdução ao sistema Galileo, focando-se essencialmente nos aspectos relevantes para o projecto desenvolvido. Deste modo todo o sistema Galileo foi superficialmente descrito, entrando com maior detalhe no segmento onde o *External Interfaces* está integrado, no *Ground Segment*.

Deste capítulo retira-se que o Galileo é composto por componentes globais, regionais e locais. O componente global divide-se no *Space Segment* e no *Ground Segment.* Resumidamente o *Space Segment* é composto pelos satélites e o *Ground Segment* por dois centros de controlo, um que gere funções de controlo e outro funções directamente relacionadas com as missões, *Ground Control Segment* e *Ground Mission Segment*, respectivamente.

O trabalho desenvolvido será inserido no GCS e diz respeito a um elemento de um dos elementos que constituem o GCS, o *Flight Dynamics Facility*. O FDF, responsável pelo cálculo das órbitas e comportamentos dos satélites que fazem parte da constelação Galileo irá usar o componente a desenvolver, o *External Interfaces,* para estabelecer interfaces com vários elementos que compõem o sistema Galileo.

# Capítulo 3

# Análise

Nesta primeira fase de trabalho o objectivo primário foi o de conhecer o projecto, obtendo assim um conhecimento mais profundo sobre o contexto do mesmo.

A fase de análise de um projecto é de extrema importância uma vez que é nesta fase que se define todo o trabalho a ser realizado no futuro. É nesta fase que se define, por exemplo, as bibliotecas *third-parties* a usar e o porquê das mesmas terem sido seleccionadas. É também nesta fase que se começa a ter uma primeira noção da arquitectura, tendo por base a análise feita aos requisitos de sistema da aplicação a desenvolver.

Neste caso as actividades de análise consistiram na elaboração do documento *Software Procurement Justification* (3.1) e à consolidação e análise dos requisitos de software (3.2).

É comum que o cliente não saiba o que realmente deseja, que existam problemas na comunicação e ainda que exista uma mudança constante de requisitos. Esta fase é então crucial para o normal desenvolvimento de todo o projecto.

## 3.1. Software Procurement Justification

### 3.1.1 Abordagem

Em projectos de larga magnitude a necessidade de justificar a utilização/compra de software é de extrema importância. É necessário garantir que as ferramentas escolhidas servem o propósito pretendido, estando as mesmas dentro do orçamento do projecto a desenvolver.

Esta justificação deve ser efectuada por alguém que se encontre familiarizado com os requisitos do projecto, tendo assim em sua posse toda a informação sobre onde é necessário investir para a correcta implementação do projecto.

O documento em questão deverá conter:

- Uma descrição de que o novo software é necessário por oferecer compatibilidade com software já em uso, e/ou;
- Uma descrição das características do software em estudo e uma explicação detalhada que explique o porquê deste software ser essencial/único para a correcta implementação de certos requisitos, e/ou;
- Um explicação sobre como é que as propriedades/funções do software são necessárias para implementação do projecto, e/ou;
- Um conjunto de justificações que explicite o porquê de determinado software que não contenha certas características não possa ser usado;
- Informação baseada em factos e não em opiniões. Só desta forma se garante que o software utilizado é de facto o mais indicado para determinada função.

Sabendo isto, o documento realizado forneceu uma descrição das ferramentas que serão usadas no *EXIF* oferecendo ainda explicações detalhadas das razões que levam a que as mesmas sejam necessárias para o desenvolvimento do componente.

### 3.1.2 Desenvolvimento

Depois de explicada na secção 3.1.1 a abordagem que deveria ser seguida para a criação deste documento o mesmo foi então elaborado fornecendo uma introdução às tecnologias de desenvolvimento a ser utilizadas, sendo depois cada uma delas sujeita a uma análise que permitiu perceber o porquê da sua escolha.

A Tabela 2 oferece um breve resumo do documento elaborado.

| Software a Utilizar | Categoria | Justificação |
|---|---|---|
| Linux | Software Operacional | Sistema operativo do FDF e como tal o EXIF terá de ser integrado neste sistema operativo. |
| SNMPv3 | Software Operacional | Esta solução foi escolhida porque apresenta uma série de avanços relativamente a outras bibliotecas, nomeadamente a SNMPv2. A biblioteca SNMPv3 oferece além das funcionalidades presentes na v2 suporte para autenticação e funcionalidades de segurança. |
| XML Xerces C++ Library | Software Operacional | Xerces-C++ é uma biblioteca que oferece a uma aplicação a habilidade de ler e escrever dados XML de um modo simples e eficaz. Esta biblioteca é fiel à recomendação XML 1.0 e a diversos standards que têm de ser cumpridos no projecto FDF ([6] e [7]). |
| MySQL | Software Operacional | A base de dados MySQL tornou-se mundialmente famosa por ser *open source*, por apresentar um desempenho rápido e consistente, uma alta fiabilidade e por ser de fácil utilização/integração.<br>O MySQL corre também em mais de 20 plataformas oferecendo ao FDF a flexibilidade desejada.<br>Sendo o MySQL uma *Standard Query Language (SQL) open source* que segue os ANSI-Standards é então a ferramenta indicada para aplicações de qualquer magnitude. |
| GNU C/C++ Compiler | Ferramenta de Desenvolvimento e compilador | O GCC oferece muitos níveis de detecção de erros no código, produz informação para *debugging* e oferece vários níveis de optimização ao código.<br>Além das funcionalidades mencionadas o GNU C/C++ poderá ser usado também para fazer testes de cobertura do código, para garantir que não há 'lixo' no mesmo. Outra das grandes vantagens é o facto desta ferramenta ser software livre.<br>Por todas estas razões o GNU C/C++ é o indicado para o projecto FDF. |
| CppUnit | Framework utilizada para a validação do *EXIF* | Esta *framework* é usada para executar testes unitários em ambiente C++ e pelo facto destes mesmos testes terem de ser feitos automaticamente, fornecendo no final da execução um relatório sobre o sucesso ou insucesso dos mesmos.<br>CppUnit é um membro da família "XUnit". Este nome é dado a um grupo de *frameworks* de testes que é bem visto por engenheiros de software. Isto deve-se ao factor de oferecer um modo fácil de definir os testes e de se poder escolher executar todos os testes ou só um pequeno grupo de testes.<br>Esta framework oferece assim todas as funcionalidades necessárias para a validação deste projecto, evitando assim o recurso a ferramentas de teste caras e com funcionalidades que não acrescentam nada de novo aos testes unitários. |
| ActiveTcl | Software Operacional | O ActiveTcl foi escolhido por ter uma qualidade elevada, por ser completo e por oferecer uma fácil integração entre linguagens |

| | | |
|---|---|---|
| | | *cross-platform.* |

*Tabela 2: Software Utilizado no EXIF*

Este documento tal como os restantes foi sujeito a revisões constantes como descrito em [8], [9] e [10].

## 3.2. Consolidação de Requisitos de Software

### 3.2.1 Abordagem

O projecto Flight Dynamics Facility – External Interfaces teve, ainda antes da minha inclusão no projecto, um desenvolvimento bastante atribulado. Devido ao facto dos requisitos de sistema mapeados ao *EXIF* sofrerem diversas alterações, este trabalho seguiu um processo bastante iterativo. Sendo a definição dos requisitos de software de extrema importância para o que viria a ser desenvolvido, era necessário ter uma base sólida para desse modo desenvolver também um produto sólido.

É arriscado desenvolver um produto onde não se sabe, ao pormenor, a função do mesmo e o que este necessita suportar para o seu correcto funcionamento. No caso do Flight Dynamics Facility – External Interfaces o problema maior foi definir aquilo que teria de ser implementado uma vez que os requisitos que especificavam o comportamento do *EXIF* sofriam mudanças constantes. Ao longo de todo o estágio acontecimentos deste género foram uma constante. Desse modo o trabalho realizado nesta área é, como já mencionado, um trabalho iterativo.

### 3.2.2 Desenvolvimento

Antes de definir a arquitectura e o desenho detalhado da aplicação foi necessário perceber quais as funcionalidades a implementar. Para isso um conjunto de documentos que continham a definição das interfaces com as quais o *EXIF* teria de interagir e os requisitos que o mesmo teria de cumprir foram analisados ao pormenor.

Numa primeira fase foi feito um levantamento de todos os requisitos de sistema que eram aplicáveis ao elemento FDF. Posteriormente foi efectuada uma triagem (*tailoring)* de modo a definir o conjunto desses requisitos que seriam também aplicáveis ao *EXIF.*

Depois de definidos os requisitos de sistema que realmente interessavam procedeu-se então à tradução dos mesmos para requisitos de software. O resultado foi então um conjunto de requisitos que permitem não só perceber a funcionalidade do componente como perceber as alterações que os documentos de arquitectura e desenho detalhado já existentes teriam de sofrer. Na Tabela 3 é apresentado um exemplo de como a "tradução" de requisitos de sistema para requisitos de software funciona. Neste exemplo dois requisitos de sistema dão origem a três requisitos de software.

| Requisitos de Sistema (Exemplo) | Requisitos de Software (Exemplo) |
|---|---|
| **Requisito de sistema 1:** Todos os elementos que trocarem ficheiros devem usar o protocolo FTP.<br>**Requisito de sistema 2:** Todos os elementos devem conseguir executar outras operações enquanto recebem ou enviam ficheiros por FTP. | **Requisito de software 1:** O componente X deverá suportar o protocolo FTP para a troca de ficheiros, não ficando o seu normal funcionamento afectado pelas trocas ocorridas.<br>**Requisito de software 2:** O componente X deverá ser *multithread* de modo a suportar concorrência.<br>**Requisito de software 3:** O componente X necessita dos seguintes parâmetros de configuração relativos ao protocolo FTP:<br>- Endereço do servidor FTP:<br>- Porto do servidor FTP;<br>- Nome do utilizador para aceder ao servidor FTP;<br>- Palavra-chave do utilizador para aceder ao servidor FTP;<br>- Directório onde guardar os ficheiros recebidos por FTP. |

*Tabela 3: Exemplo do processo de "tradução" de requisitos*

Os requisitos definidos contemplam o comportamento que o componente deve ter, qual a performance que o mesma deverá atingir, o *input* que irá receber e qual o *output* esperado, quais os protocolos necessários para uma correcta interacção com todos os elementos, etc. Os requisitos de software foram definidos com 4 objectivos ([11]):

1. Fornecer ao cliente *feedback*, provando assim que a aplicação irá funcionar de acordo com as necessidades estabelecidas por eles;
2. Decompor o problema a resolver em várias partes, consolidando assim as ideias e ajudando a organizar, de início, os componentes a desenvolver;
3. Servir de *input* para a especificação da arquitectura e desenho detalhado da aplicação. O documento de requisitos pode ser visto como o documento "mestre" para todos os outros que se irão realizar. Desse modo é necessário que os requisitos definidos já contenham o detalhe necessário de modo a que as decisões de desenho sejam mais fáceis de idealizar.
4. Servir como documento de validação. É necessário que, no decorrer da validação, seja possível estabelecer uma relação entre as funcionalidades implementadas e os requisitos de software que levou a que essas funcionalidades existissem.

O resultado de todo este processo de engenharia levou a que fossem criados um conjunto de requisitos de software que cobrissem todos as funcionalidades e propriedades pretendidas pelo cliente. Na criação destes requisitos tentou-se que os mesmos fossem ([12]):

- Correctos – É necessário que os requisitos de software criados sejam direccionados ao problema a resolver;
- Não ambíguos – Para evitar futuros mal-entendidos é essencial que os mesmos requisitos de software não dêem azo a interpretações diferentes;
- Completos – Os requisitos de software devem cobrir todos os aspectos do problema. Muitas vezes um só requisito de sistema pode dar origem a vários requisitos de software;
- Consistentes – É importante que não existam requisitos contraditórios, ou que possam ser interpretados de maneiras contraditórias. Para o desenvolvimento da aplicação todos os requisitos são tidos em conta por isso é importante que acha consistência entre todos eles;

- Verificáveis – Em situações normais é desejável que os requisitos de software tenham esta propriedade mas no projecto em questão é essencial. Devido ao contexto do projecto é de elevada importância que cada requisito seja verificável, sendo que neste caso, esta propriedade será garantida através da execução de testes de sistema/aceitação;
- Estruturados – Os requisitos devem ser bem estruturados, evitando que os mesmos sejam compostos por vários parágrafos. Deste modo, em caso de modificações futuras, estas ficarão facilitadas devido ao cuidado de não se ter enumerado no mesmo requisito várias propriedades/funcionalidades;
- Mapeáveis – Tal como a propriedade enunciada no ponto 5, esta é de extrema importância. É necessário "provar" o porquê de se implementar qualquer uma das funcionalidades da aplicação. Isso é feito oferecendo uma matriz que estabelece uma relação entre os requisitos de software e os requisitos de sistema que lhes deram origem;

Tal como o documento *Software Justification Procurement* também o documento de requisitos foi sujeito a revisões constantes de modo a verificar se o mesmo cumpria com o definido em [8], [9] e [10].

## 3.3. Sumário

A fase de análise, descrita neste capítulo, é essencial para uma correcta implementação da aplicação, garantindo que ela cumpre os objectivos esperados. Como mencionado ao longo deste capítulo, todo o trabalho relacionado com esta fase permite definir logo à partida quer a fase de desenho, quer a fase de implementação, descritas nos capítulos seguintes.

Este capítulo aborda em pormenor dois documentos elaborados no início do estágio que contribuem para uma melhor percepção da aplicação a desenvolver.

O primeiro documento, o *Software Justification Procurement*, abordou as tecnologias a usar no desenvolvimento da aplicação, tecnologias essas que vão desde o compilador e linguagem de programação a usar, até às bibliotecas *third-parties* que serão usadas para implementar funcionalidades específicas.

O segundo documento, referente aos requisitos de software afectos ao componente *External Interfaces*, é encarado como um documento mestre em todo o processo de desenvolvimento do *External Interfaces.* Este documento especifica, já com algum detalhe, pormenores referentes ao desempenho, desenho, tecnologias, protocolos, standards, etc., que o componente a desenvolver terá de cumprir.

# Capítulo 4

# Desenho

Depois de feita a análise é necessário interpretar os resultados dessa mesma fase, ou seja, os requisitos de software, para desse modo começar a especificar em termos de desenho o componente *EXIF*.

Com os requisitos de software já definidos e sabendo de antemão as características que o software a desenvolver necessitava ter, foi possível definir/especificar a aplicação em si.

Esta fase teve o seu início com a consolidação da arquitectura. Na fase de análise novos requisitos de software foram adicionados, resultado da mudança dos requisitos de sistemas, sendo então necessário colocar a arquitectura e o desenho detalhado do software a desenvolver de acordo com as novas especificações. Para isto foi necessário recorrer aos conhecimentos de UML aprendidos na Faculdade de modo a que o resultado desta fase fosse de fácil entendimento, oferecendo todo o detalhe necessário para as fases descritas no Capítulo 5 e Capítulo 7. Toda a parte de desenho foi desenvolvida recorrendo à ferramenta *Enterprise Architect* ([13]).

## 4.1. Consolidação da Arquitectura

### 4.1.1 Abordagem

Ao ser inserido num projecto em curso é normal haver documentação que já se encontra finalizada ou perto de ser finalizada. O objectivo nesta fase foi assim consolidar a arquitectura, definir interfaces em falta, etc. A instabilidade deste projecto, já referida na secção 3.2.1 levou também a que o processo de consolidação fosse um processo iterativo. Assim o trabalho realizado é uma junção de um processo de consolidação e definição da arquitectura.

### 4.1.2 Desenvolvimento

Depois de definidos os requisitos de software é necessário mapear os mesmos à arquitectura, mais concretamente aos componentes que vão compor a mesma. Desse modo é preciso garantir que a arquitectura contempla todos os componentes necessários para satisfazer os requisitos definidos e é preciso especificar que componente implementa determinado requisito(s).

A arquitectura do *EXIF* tem por base uma arquitectura cliente/servidor. É estabelecida uma relação entre processos que estão ou podem estar a correr em diferentes máquinas. O *EXIF* pode ser visto como sendo cliente e servidor, dependo dos elementos com os quais está a interagir. Existem casos em que é responsável pelo envio de fluxos de dados (cliente) e outros em que espera que um cliente o contacte (servidor).

A Figura 5 descreve a arquitectura definida para o *EXIF*:

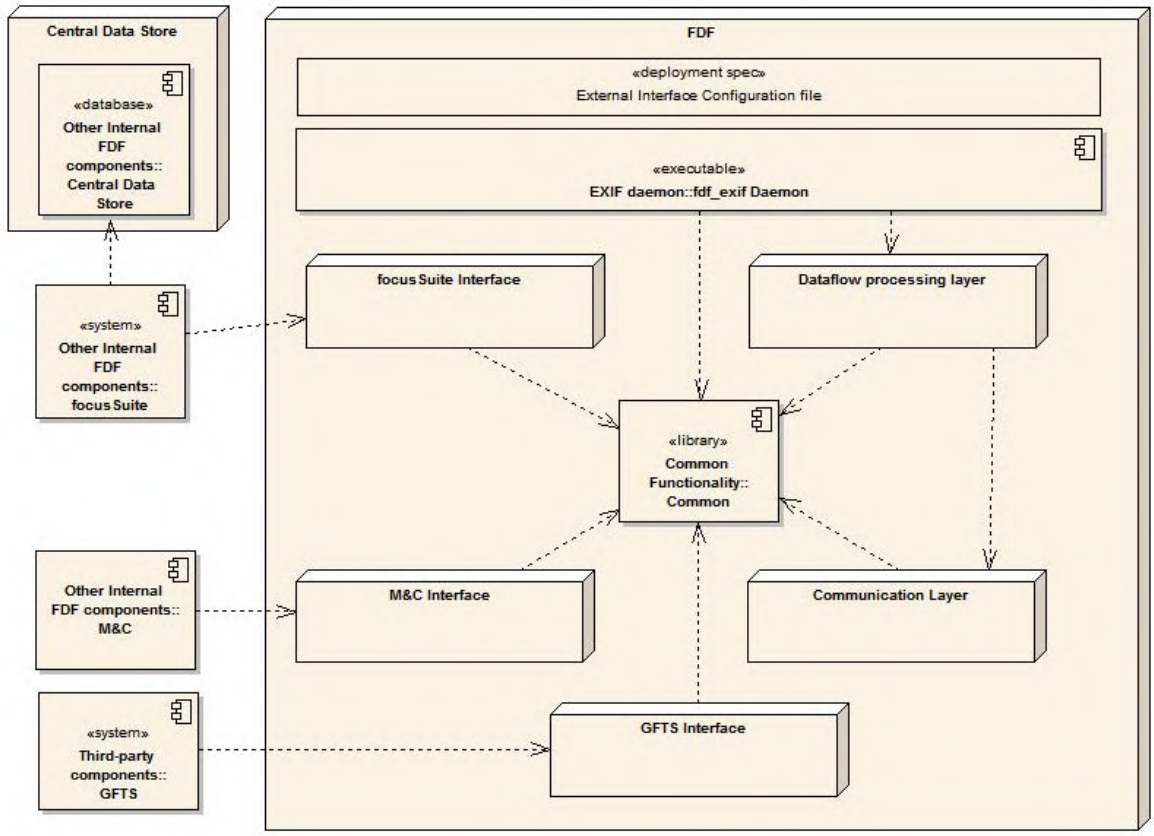



*Figura 5: Arquitectura do EXIF*

O *EXIF* encontra-se dividido em 4 grandes componentes:

1. *EXIF Daemon* – Este componente será o ponto de entrada para toda a funcionalidade do *EXIF*.
2. *Dataflow Processing Layer* – Esta camada será responsável por todo o processamento relativo a todos os fluxos de dados trocados entre o elemento FDF e todos os outros elementos do GALILEO. Visto que o *EXIF* tem de gerir várias ligações e fluxos de dados de diversos elementos, como o CMCF, ESCC (External Agencies), GMS, OPF, SCCF, SCPF e TTCF, foi necessário desenhar vários módulos internos separados para tratar individualmente os fluxos de dados de cada elemento externo.
3. *Communication Layer* – Esta camada vai disponibilizar os protocolos de comunicação necessários para a troca de todos os fluxos de dados processados pela *Dataflow Processing Layer.*
4. *EXIF Interface Applications* – Estas aplicações vão fornecer mecanismos essenciais para a comunicação entre outros componentes dentro do FDF e o *EXIF.* As aplicações serão as seguintes:
   - 4.1. M&C Interface
   - 4.2. GFTS Interface
   - 4.3. FocusSuite Interface

Cada camada poderá ser composta por várias bibliotecas e/ou executáveis, sendo que cada biblioteca/executável será responsável por implementar um conjunto de funcionalidades que serão acessíveis através de uma interface especifica.

Como se pode ver pela Figura 5 o *EXIF Daemon* vai usar os componentes que se encontram inseridos na *Data Processing Layer*. Contudo não irá usar nenhum dos componentes presentes na *Communication Layer*, sendo assim completamente transparente quais os protocolos de comunicação a usar para a troca de fluxos de dados entre o *EXIF* e os outros elementos.

Os componentes da *Data Processing Layer* vão tirar proveito dos componentes de comunicação, dependendo do protocolo necessário para a troca de dados que cada interface necessita para interagir com o elemento do GALILEO que a interface implementa. A Tabela 4 indica todos os elementos com os quais o *EXIF* terá de interagir:

| Elemento | Funcionalidades |
|---|---|
| **CMCF**<br>**(Central Monitoring and Control Facility)** | O CMCF é o elemento central de monitorização e controlo dentro do GCS. Todos os elementos do GCS interagem com o CMCF de modo a permitir operações de monitorização e controlo das instalações terrestres.<br>O CMCF monitoriza e controla todos os aspectos dentro do GCS, incluindo computadores e estações terrestres.<br>De modo a interagir com o CMCF o *EXIF* vai usar a M&C Interface. O protocolo a usar será o SNMP. |
| **ESCC**<br>**(External Satellite Control Centre)** | Este elemento será responsável por efectuar todas as operações necessárias para colocar os satélites da constelação GALILEO em estado operacional após o seu lançamento.<br>Para interagir com este elemento o *EXIF* terá de estabelecer uma comunicação segura por HTTPS. |
| **GMS**<br>**(Ground Mission Segment)** | Como já foi dito anteriormente o GMS é responsável por monitorizar os sinais de navegação provenientes de todos os satélites.<br>Para interagir com este elemento o *EXIF* terá de suportar o protocolo FTP. |
| **OPF**<br>**(Operations Preparation Facility)** | O OPF contém uma série de ferramentas que são necessárias para o desenvolvimento e manutenção de dados de configuração (*Configuration Items)* usados pelo GCS.<br>Desse modo o *EXIF* terá de interagir com o OPF de modo a efectuar operações sobre os *Configuration Items* (Reservar CI/Substituir CI/Obter CI/etc.).<br>Para isso o *EXIF* irá usar um sistema de troca de ficheiros proprietário da ESA, o GFTS. |
| **SCCF**<br>**(Spacecraft & Constellation Control Facility)** | O SCCF vai disponibilizar um mecanismo em tempo de real para controlar e monitorizar a constelação de satélites através de telecomandos e telemetria.<br>O *EXIF* irá usar um sistema de troca de ficheiros proprietário da ESA (para a troca de dados em modo *offline*) e um serviço baseado CORBA (para a troca de dados em modo *real-time*) para interagir com o SCCF. |
| **SCPF**<br>**(Spacecraft Constellation Planning Facility)** | O GCS tem incluído na sua arquitectura o SCPF de modo a efectuar o planeamento das operações de todos os satélites. Desse modo o *EXIF* irá trocar dados com informação sobre eventos de órbitas (eclipse, etc.), alturas em que os satélites ficam visíveis a cada estação terrestre e dados de planeamento de operações dos satélites com o SCPF.<br>Para isso o *EXIF* irá usar um sistema de troca de ficheiros proprietário da ESA. |

| | |
|---|---|
| **TTCF**<br>**(Tracking Telemetry and Command Facility)** | O TTCF fornece meios para a transmissão e recepção de dados para o controlo da constelação de satélites Galileo.<br>O *EXIF* irá usar um sistema de troca de ficheiros proprietário da ESA para trocar ficheiros com o TTCF. O tipo de ficheiros e os seus conteúdos já estão previamente definidos. |

*Tabela 4: Interfaces do EXIF*

De referir que as *EXIF Interface Applications* são encaradas como componentes externos aos *EXIF* mas internos ao FDF.

Na Figura 5 é possível verificar que todos os principais componentes do *EXIF* estão dependentes de uma biblioteca, a *Common Library*. Esta biblioteca oferece um conjunto de funcionalidades que irão ser usadas por todos os outros componentes internos ao *FDF*. As funcionalidades oferecidas são as seguintes:

- Interface para lidar com eventos internos – Os componentes internos vão comunicar entre si através de eventos. Quando o componente é inicializado é possível registar diferentes componentes internos noutros componentes para desse modo serem notificados de eventos criados pelos componentes onde se registaram.
- Interface para lidar com eventos externos (*EXIF Interface Application*) – Para suportar a *focusSuite* GUI, o sistema GFTS e a função de *Monitoring and Control*, ou seja, aplicações externas, o *EXIF* vai ter de receber eventos destas aplicações. Desse modo necessita interagir com as mesmas através de CORBA e esta interface fornece os meios para que tal aconteça.
- Acesso à base de dados – Como já foi dito anteriormente o *EXIF* tem por função primordial a capacidade de guardar os dados recebidos numa base de dados. Esta base de dados é essencial de modo a garantir que no futuro esses mesmos dados possam ser enviados para diferentes elementos. A *Common Library* vai fornecer o acesso à base de dados e a todas as funcionalidades necessárias para que o *EXIF* cumpra esta função.
- Controlo do ficheiro de Log – O *EXIF* terá de ter um log para efeitos de *debugging*.
- Configuração do *EXIF* – Visto que o *EXIF* lida com vários elementos, cada um dos quais através de protocolos diferentes, é necessário que cada uma das interfaces esteja correctamente configurada. O *EXIF* quando iniciado irá fazer o *parsing* de um ficheiro de configuração onde estarão todas as configurações necessárias para o seu correcto funcionamento e a *Common Library* oferece essa funcionalidade.
- Controlo de execução e monitorização do estado do *EXIF* – O estado global do *EXIF* necessita de ser regularmente monitorizado. Esta funcionalidade encontra-se também presente na *Common Library*.
- Interface de segurança – Para a comunicação entre as aplicações externas ao *EXIF* e o próprio EXIF é preciso garantir que as mesmas são seguras. Para isso a *Common Library* fornece função de cifra, decifra e de *hash.*

  Não será mostrada mais informação relativa à arquitectura do *EXIF* devido às mesmas serem confidenciais.

## 4.2. Consolidação do Desenho Detalhado

### 4.2.1 Abordagem

A abordagem para a consolidação do Desenho Detalhado do componente *EXIF* foi em tudo semelhante à descrita na secção 4.1.1.

Devido a alguma falta de experiência do autor deste documento e também responsável pela consolidação da fase de desenho nas tecnologias usadas neste componente, foi necessário recorrer ao desenvolvimento de alguns protótipos de modo tentar tirar proveito de todas as potencialidades da tecnologia envolvida.

### 4.2.2 Desenvolvimento

Definidos os componentes (ver secção 4.1.2) é preciso definir a maneira como os mesmo serão implementados. Para isto foi necessária a criação de diagramas de sequência, de estados e de classes que representam não só as classes necessárias para a implementação do *EXIF* mas também o funcionamento de cada uma dessas classes, nomeadamente a sequência das operações que determinado fluxo de dados deverá ter quando processado por determinado componente.

As classes definidas são assim uma implementação dos componentes definidos na altura da consolidação da arquitectura do EXIF.

Como enunciado na secção 4.1 a arquitectura do *EXIF* oferece uma série de componentes de modo a ser o mais modelar possível. Mais concretamente a arquitectura específica que o *EXIF* será um *daemon* que será responsável por carregar um conjunto de bibliotecas que oferecem funcionalidades especificas. Desse modo as bibliotecas podem ser classificadas em 2 categorias:

1. Uma categoria referente ao processamento e manuseamento de dados de que fazem parte os seguintes componentes: CMCFMngr, ExtAgencyMngr, GMSMngr, OPFMngr, SCCFMngr, SCPFMngr e TTCFMngr;
2. Uma categoria referente à camada de comunicação de que fazem parte os seguintes componentes: CORBAFramework, SNMPDrv, FTPDrv, HTTPSDrv e o driver referente à implementação do protocolo de comunicação proprietário da ESA.

Não será mostrada mais informação específica sobre o desenho detalhado do *EXIF* devido a questões de confidencialidade.

De seguida ficam alguns dos conceitos importantes usados na especificação do *EXIF.*

#### 4.2.2.1 Design Patterns

Ao longo desta fase foi necessário consolidar alguns conceitos considerados essenciais para o desenho da aplicação desenvolvida.

O objectivo dos padrões de desenho é fornecer uma solução para problemas relativos ao desenho de componentes que são recorrentes quando se desenvolve software. Existem vários padrões de desenho e o mais importante é, logo à partida, identificar o problema em questão e posteriormente verificar se o mesmo se enquadra em algum padrão de desenho. O recurso a padrões de desenho é bastante usual uma vez que os mesmos já provaram, através do seu uso, que resolvem paradigmas de desenho usuais.

Os padrões de desenho disponibilizam soluções genéricas que podem ser adaptadas às circunstâncias de cada projecto.

### Singleton Pattern

Este padrão de desenho é um dos mais utilizados e oferece uma maior facilidade no que diz respeito a aplicações que necessitam ser *thread safe.* Uma classe ao utilizar este padrão de desenho tem garantidamente uma única instância, tendo de oferecer um método que permita obter essa mesma instância única. Em C++ para criar uma classe que siga este padrão de desenho é necessário que o seu construtor seja *private ([14]*) e que, como se observa na Figura 6, ofereça um método (*getInstance()*) que retorne a instância única (que será um atributo membro) dessa classe*:*

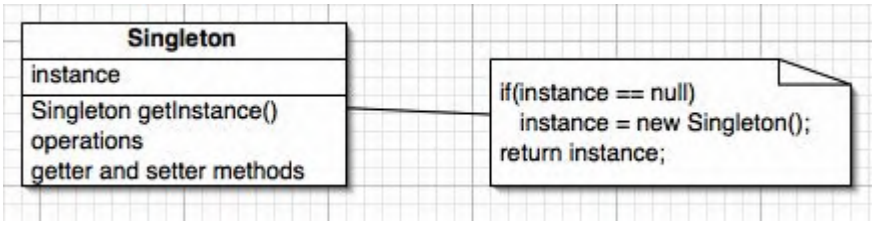



*Figura 6: Singleton Pattern*

### Observer Pattern

Este padrão de desenho pode ser descrito de uma maneira muito simples: um mecanismo de eventos que permite observar o estado de um objecto. Este padrão é bastante usado, especialmente quando existe uma dependência de um para muitos. Quando um objecto muda o seu estado, todos os que dele dependem serão notificados e actualizados automaticamente ([15]).

Como exemplo e utilizando o *EXIF:* quando um novo ficheiro é colocado numa directoria controlada pelo *driver* FTP, este envia eventos a notificar a recepção desse ficheiro a todas os objectos que nele se registaram, de modo a que os objectos a quem esse ficheiro interessa possam tomar as acções necessárias. A Figura 7 apresenta uma descrição geral da implementação deste padrão de desenho:

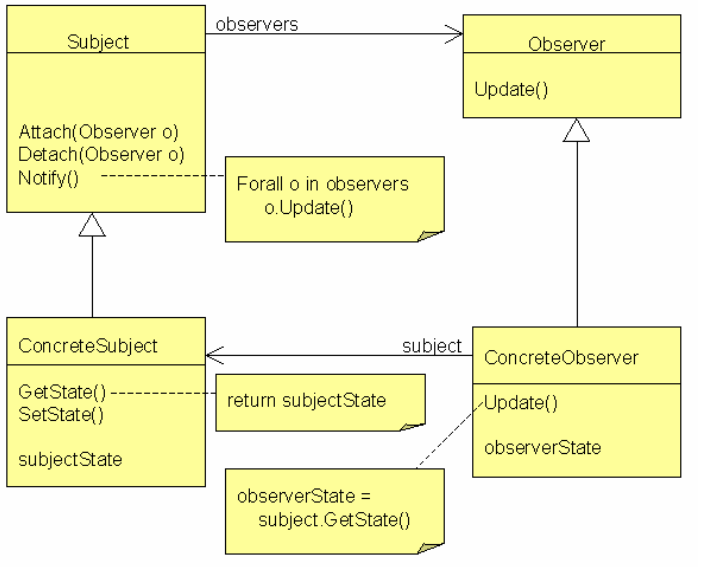



*Figura 7: Observer pattern*

Como se pode observar na figura existem 4 actores:

1. *Subject* – Objecto que conhece os *observer's* e que disponibiliza uma interface para registar e anular o registo desses mesmos *observer's*.

2. *Observer* – Objecto que define uma interface de actualização para os objectos que devem ser notificados das alterações ocorridas no *subject*.
3. *ConcreteSubject* – Envia uma notificação aos *observer's* quando o seu estado muda.
4. *ConcreteObserver* – Mantém uma referência do *ConcreteSubject* e implementa a interface *update* para manter o seu estado coerente com o dos *subject's*.

Este padrão de desenho foi implementado no componente do *EXIF* que lida com os eventos internos e, para lidar com a dificuldade em saber qual o *subject* que enviou a notificação/evento, todos os eventos no *EXIF* têm um identificador único, permitindo assim controlar este aspecto.

#### *4.2.2.2 Sincronização de threads*

Numa aplicação deste género é necessário ter em conta que muitos dos recursos são partilhados. Assim é necessário, logo na fase de desenho, pensar em soluções que permitam que esses recursos estejam a ser usados apenas por uma instância de cada vez, fazendo com que todo o funcionamento da aplicação continue estável ao longo do tempo.

A necessidade de sincronização de *threads* no *EXIF* é uma realidade tendo em conta que irão existir várias *threads*, *threads* essas que partilham memória. Sendo o conceito de sincronização simples, a necessidade de uma correcta implementação é crítica e o código relativo à sincronização é muitas vezes considerado um dos maiores pontos de falha nas aplicações.

Para atingir este fim foi usado no *EXIF mutual exclusion lock's (mutexes)* ([16]). Este método funciona assegurando que apenas uma thread de cada vez está a executar uma secção de código considerado crítico. Isto é conseguido através de operações de *Lock* e *Unlock*. Assim que se efectuar um *Lock* a um *mutex*, outra *thread* só irá conseguir aceder à memória partilhada quando a *thread* responsável pelo *Lock* efectuar o respectivo *Unlock.* O uso correcto do *Lock* e *Unlock* é por isso essencial para evitar a todo o custo situações de *deadlock,* ou seja, um *Lock* sem posteriormente exista o *Unlock*.

Para evitar ao máximo esta situação foi implementada uma solução semelhante à descrita de seguida em pseudo-código ([17] e [18]):

1. O seguinte pseudo-código tem uma secção crítica e por isso são usados *mutexes*:

```
void foo ()
{
       getLock(mutex);
       .... //secção critica
       releaseLock(mutex);
}
```

Sendo o pseudo-código exemplo mostrado neste ponto bastante simples, pode ainda assim ter um ponto de falha. Se a função for muito complexa pode haver vários pontos de saída da mesma, levando a que o *releaseLock (mutex)* nunca seja chamado.

2. Uma das soluções seria:

```
void foo()
{
       getLock(mutex);
       ....
       if (...){
              releaseLock(mutex);
       return;
```

```
        }
        if (...){
                releaseLock(mutex);
                throw exception;
        }
        ...
        releaseLock(mutex);
}
```

Esta solução contudo tem algumas desvantagens: o código torna-se mais extenso e de mais difícil compreensão. Código desta natureza é também propício ao erro humano uma vez que fica dependente da atenção do programador chamar o releaseLock (mutex).

3. Uma solução mais eficaz que se poderia implementar seria definir uma classe chamada por exemplo, *ScopeMutex*, que não tem operações, só um construtor e um destrutor. O construtor chama o *getLock* e o destrutor *releaseLock*. Desse modo a classe *ScopeMutex* seria implementada da seguinte forma:

```
class ScopeMutex
{
        public:
                ScopeMutex(Mutex& mutex) : m_mutex(mutex)
                {
                        getLock(m_mutex);
                }
        ~ScopeMutex()
        {
                        releaseLock(m_mutex);
        }
private:
        Mutex & m_mutex;
};
```

E seria utilizada assim:

```
void foo()
{
        ScopeMutex scopedLock(mutex);
        ....
        if (...) { releaseLock(mutex); }
        if (...) { throw exception; }
}
```

O que acontece neste caso é que ao sair do âmbito da função o destrutor da classe *ScopedMutex* é invocado automaticamente, garantindo assim que as situações de d*eadlock* não aconteçam. Deste modo evita-se a invocação directa do *getLock()* ou *releaseLock()*.

#### 4.2.2.3 CORBA – Common Object Request Broker Arquitecture

De modo a que seja possível implementar as *EXIF Interface Applications* foi necessário recorrer a um serviço disponibilizado pelo CORBA, o serviço de eventos. Com este serviço fica possível o estabelecimento das comunicações entre essas aplicações e o *EXIF ([19]*).

O conceito de serviço de eventos é bastante simples, baseando-se no conceito de *event channel*, ou seja, um objecto que permite a vários "produtores" comunicarem assincronamente com vários consumidores. O *event channel* é então um objecto CORBA que actua como produtor e consumidor de eventos ([20] e [21]).

Para implementar um serviço de eventos 2 modelos foram tidos em conta, o modelo *Push* e o modelo *Pull*. A Figura 8 ilustra de uma maneira clara as diferenças entre ambos os modelos.

No modelo *pull* o produtor é um produtor a partir do qual o *event channel* "puxa" eventos. Neste modelo o consumidor é um consumidor que "puxa" eventos de um *event channel.*

No modelo *push* o produtor "empurra" eventos para o *event channel.* Um consumidor *push* é um consumidor que recebe eventos "empurrados" pelo *event channel.*

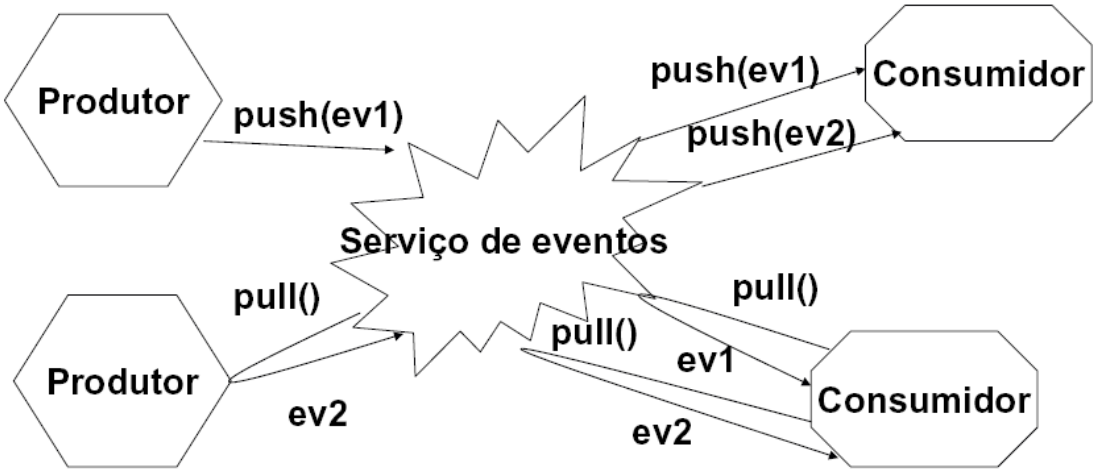



*Figura 8: Modelos de serviço de eventos CORBA*

Para as *EXIF Interface Applications* foi seleccionado o modelo *push* ([22]). Este modelo é o mais indicado uma vez que é o produtor, ou seja, a aplicação que faz a análise sintáctica dos ficheiros, o responsável por enviar os eventos. São as aplicações (produtores) que definem a altura para enviar os eventos (quando um operador assim o entender) ao invés de ser o consumidor (*EXIF Daemon* e os componentes da camada de *Data processing*), que no caso do *EXIF,* não sabem os momentos em que devem ir ao serviço de eventos "puxar" eventos.

A figura seguinte ilustra a abordagem seguida para o desenvolvimento das *EXIF Interface Applications*:

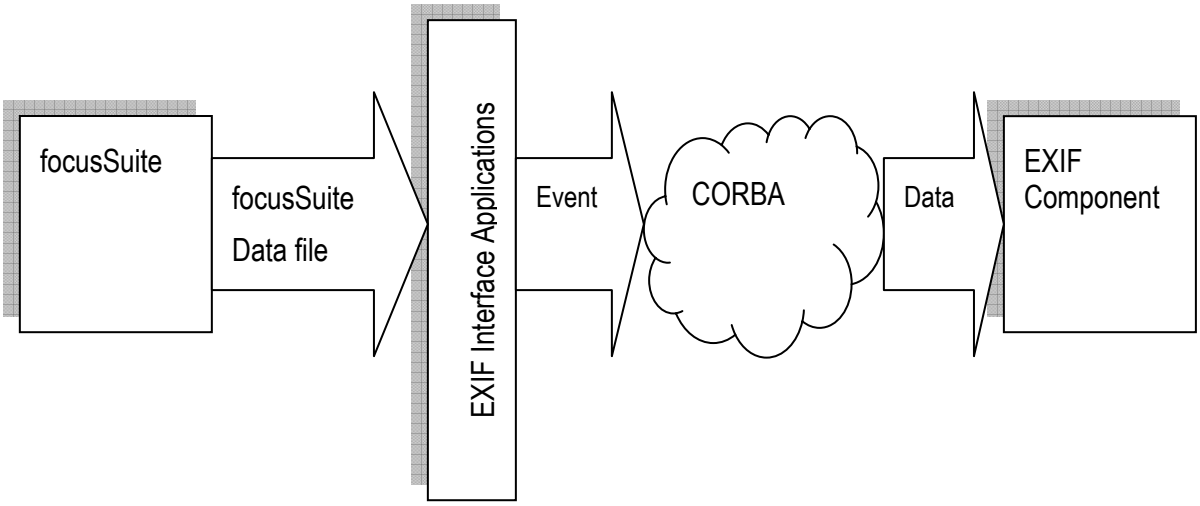



*Figura 9: EXIF Interface Applications*

O funcionamento destas aplicações será bastante simples. Cada aplicação será responsável por fazer o *parsing* do ficheiro de configuração gerado pela GUI, correspondente ao elemento a que a GUI diz respeito. Posteriormente irá criar um conjunto de eventos que depois serão enviados através de um canal CORBA específico (um por cada elemento) que depois será recebido pela interface correspondente do lado do EXIF, ou seja, uma das interfaces que compõem a camada *Dataflow Processing ([23])* ou o próprio *daemon*.

## 4.3. Sumário

Devido à complexidade inerente ao desenvolvimento de software é necessário trabalhar com níveis de abstracção que consigam lidar com essa mesma complexidade. O trabalho descrito neste capítulo refere-se à definição desses níveis de complexidade. No capítulo anterior foi definido "o que fazer", na fase descrita neste capítulo foi definido o "como fazer".

Todo o trabalho realizado nesta fase deu origem a dois documentos. O primeiro documento, com um nível de abstracção mais elevado, traduz os requisitos de software em componentes que implementam a funcionalidade descrita nesses mesmos requisitos. O segundo documento, já com um nível de detalhe considerável, pretende preparar o projecto para a fase de implementação. Este documento apresenta já detalhe suficiente para implementar a aplicação, detalhe esse que pode ser observado através de diagramas de classes, de estados e de sequência.

# Capítulo 5

# Verificação & Validação

Depois de concluída a fase de desenho, e ainda antes de passar à fase de implementação, é necessário definir logo à partida um conjunto de testes que terão de ser obrigatoriamente realizados durante e após a fase de implementação estar concluída.

Esta fase consistiu essencialmente na elaboração de 3 documentos de testes:

1. Secção 5.1, Acceptance Test Plan – Documento com a definição dos testes de aceitação/sistema, orientados à verificação da implementação dos requisitos de software definidos na fase de análise, Capítulo 3;
2. Secção 5.2, Integration Test Plan – Documento com a definição dos testes de integração, ou seja, testes que foram definidos com o intuito de validar a integração entre os vários componentes do *EXIF*;
3. Secção 5.3, Unit Test Plan – Documento com a definição dos testes unitários. Os testes definidos neste último documento seriam realizados em paralelo com a fase de implementação. Estes testes são orientados à funcionalidade específica que determinada interface oferece.

## 5.1. Acceptance Test Plan

### 5.1.1 Abordagem

Embora a execução dos testes de aceitação constitua uma das etapas finais do ciclo de vida de um produto, o seu envolvimento ao longo de todo o processo de desenvolvimento é essencial. É assim necessário elaborar um plano de testes que ofereça aos utilizadores finais do sistema a oportunidade de o testarem eficazmente, antes da sua entrada em funcionamento.

Enquanto que os programadores executam testes em relação à especificação do sistema, à documentação técnica e verificam se estas se adequam aos objectivos, os técnicos de testes de aceitação testam em relação aos requisitos do negócio, verificando, também, a sua adequação aos objectivos. É bastante semelhante a um exame final de um curso universitário, um exame que determina uma nota para passar ou reprovar. A atitude dos técnicos de teste deverá reflectir esta ideia: a hora de correcções de erros já passou, agora é necessário aceitar ou rejeitar o resultado.

Visto que será o cliente a efectuar estes testes é necessário desenvolver um documento formal sobre os processos de teste que garanta que o sistema está a funcionar correctamente e a cumprir com os objectivos. Este documento certifica que as pessoas responsáveis pela execução dos testes sabem o que está a acontecer em todos os momentos do plano de teste.

No documento realizado poderá encontrar-se, para além da definição dos testes de aceitação, uma descrição de como os mesmos deverão ser executados, as ferramentas usadas e a aproximação usada nos testes definidos.

### 5.1.2 Desenvolvimento

Os testes especificados para o *EXIF* serão executados utilizando um de quatro métodos de verificação (todos os métodos são compostos por vários passos):

- Teste – Um procedimento é definido e executado;
- Inspecção – Neste caso as pessoas responsáveis pela execução do teste devem inspeccionar o sistema à procura de evidências que comprovem que o mesmo está a cumprir o requisito que determinado teste está a validar;
- Análise – Uma análise é efectuada para determinar se um requisito está ou não a ser cumprido;
- Revisão do desenho – É feita uma revisão para garantir que o desenho do componente está de acordo com o especificado pelos requisitos de desenho.

Cada teste terá tantos critérios de validação quantos requisitos estiver a testar. O teste só tem sucesso se todos esses critérios forem observados. Caso falhe um só critério o teste falha/chumba. Além dos critérios de validação todos os testes terão os campos presentes na Tabela 5:

| Campo | Definição |
|---|---|
| **Identificação e Nome** | Cada teste terá uma identificação única e um nome sugestivo. A identificação fornece um modo rápido de identificar o teste e o nome uma forma rápida de perceber o propósito do mesmo. |
| **Descrição geral** | Descrição detalhada do propósito do teste. |
| **Ambiente de execução** | Cada teste terá um ambiente de execução que pretende recriar o ambiente e as condições onde o *EXIF* será executado. |
| **Duração** | Deverá ser especificado para cada teste uma duração expectável para o tempo de execução do mesmo. |
| **Dados de entrada** | Para os testes que necessitem de dados específicos para a sua execução, esses mesmo dados necessitam de estar especificados. |
| **Método de verificação** | É preciso especificar qual o método de verificação que o teste usa (Teste/Inspecção/Análise/Revisão de desenho). |
| **Critérios de validação** | Todos os testes terão obrigatoriamente de ter tantos critérios de validação quantos requisitos o testes pretende validar. |

*Tabela 5: Especificação dos testes de sistema*

## 5.2. Integration Test Plan

### 5.2.1 Abordagem

O objectivo dos testes de integração é o de assegurar que todos os componentes distintos de uma aplicação podem interagir correctamente de acordo com os requisitos existentes. Estes testes são desenvolvidos com o propósito principal de exercitar a interactividade entre todos os componentes.

Nos testes de integração, testes a módulos individuais são combinados e reunidos, formando um grupo. Os testes de integração sucedem aos testes unitários e precedem os testes de sistema.

Para os testes de integração foram tidas em conta duas abordagens ([24]):

5. *Top-Down* – É uma abordagem incremental. Começa com os componentes de alto nível de um sistema, e a integração dá-se de cima para baixo numa hierarquia de componentes. Componentes individuais num nível mais baixo na hierarquia são representados por *stubs*. A Figura 10 ilustra o funcionamento desta abordagem:

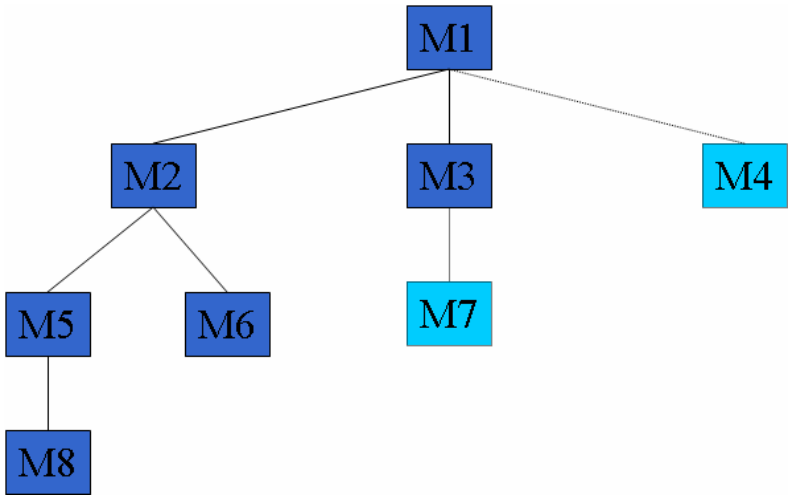



*Figura 10: Estratégia Top-Down*

6. *Bottom-Up* – Inicia a construção e os testes com modelos atómicos. Usa a seguinte estratégia:
   6.1. Módulos de baixo nível são combinados em *clusters*;
   6.2. Um *driver* é escrito para coordenar a entrada e a saída do caso de teste;
   6.3. O *cluster* é testado;
   6.4. Os *drivers* são removidos e os *clusters* são combinados ao dirigirem-se para cima na estrutura de programa.

A Figura 11 exemplifica a estratégia apresentada:

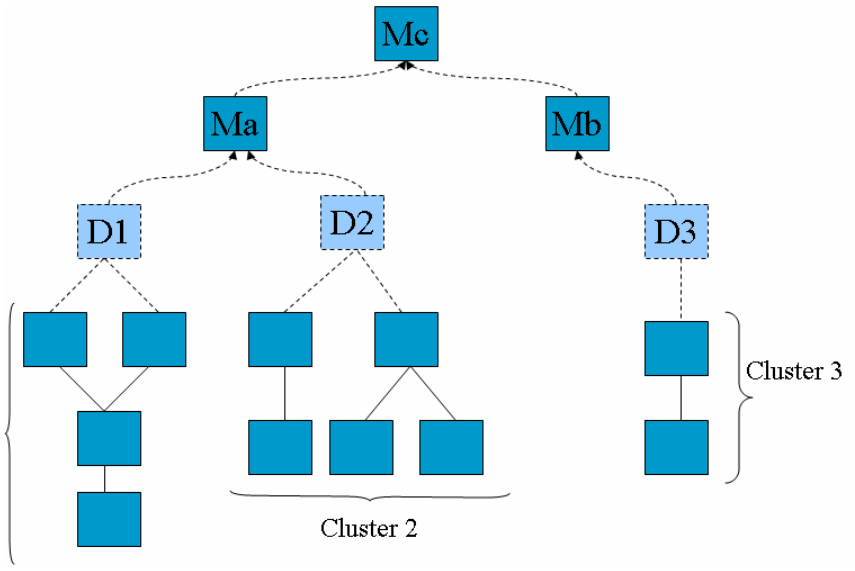



*Figura 11: Estratégia Bottom-Up*

No trabalho realizado foram usadas as duas estratégias de modo a poder aproveitar as vantagens que ambas oferecem.

### 5.2.2 Desenvolvimento

O trabalho desenvolvido relativamente aos testes de integração foi em tudo semelhante ao descrito em 5.1.2. Contudo em vez de ser direccionado aos requisitos foi direccionado aos componentes descritos em 4.1. Devido ao facto dos testes de integração testarem a interacção entre os diferentes componentes, todos os testes planeados têm por método de verificação o método de teste. É bastante difícil, se não mesmo impossível, determinar se os componentes se encontram bem integrados através de uma análise/inspecção ou revisão do desenho. Estes métodos podem em último caso complementar a utilidade dos testes de integração efectuados com o método de teste.

O facto dos testes de integração serem direccionados aos componentes em nada altera os critérios de validação. Os componentes implementam determinados requisitos de software e os testes de integração terão, no máximo, tantos critérios de validação quantos requisitos de software implementarem. É dito no máximo porque é normal que o teste não valide todos os requisitos que cada componente tem a si associado e por isso é feito uma filtragem a cada teste de integração de maneira a ficar bem claro que requisitos estão a ser validados e por conseguinte quais os componentes.

## 5.3. Unit Test Plan

### 5.3.1 Abordagem

O plano de testes unitários é essencial no desenvolvimento de qualquer aplicação. Esta metodologia sugere que para cada método haja um ou mais testes associados, que testam todas as possibilidades de execução. O objectivo é que o código ganhe mais qualidade.

Um dado interessante é que os testes funcionam como uma especificação do sistema que estamos a desenvolver. Se temos um teste que diz que *a+b=c*, então sem dúvidas que aquele será o comportamento do sistema. É uma forma que retrata as especificações da forma mais concreta possível.

No caso específico do *EXIF* foi utilizada uma ferramenta direccionada para a execução de testes unitários em C++: o *CPPUnit*.

O *CPPUnit* permitiu que a seguinte abordagem fosse implementada ([25]):

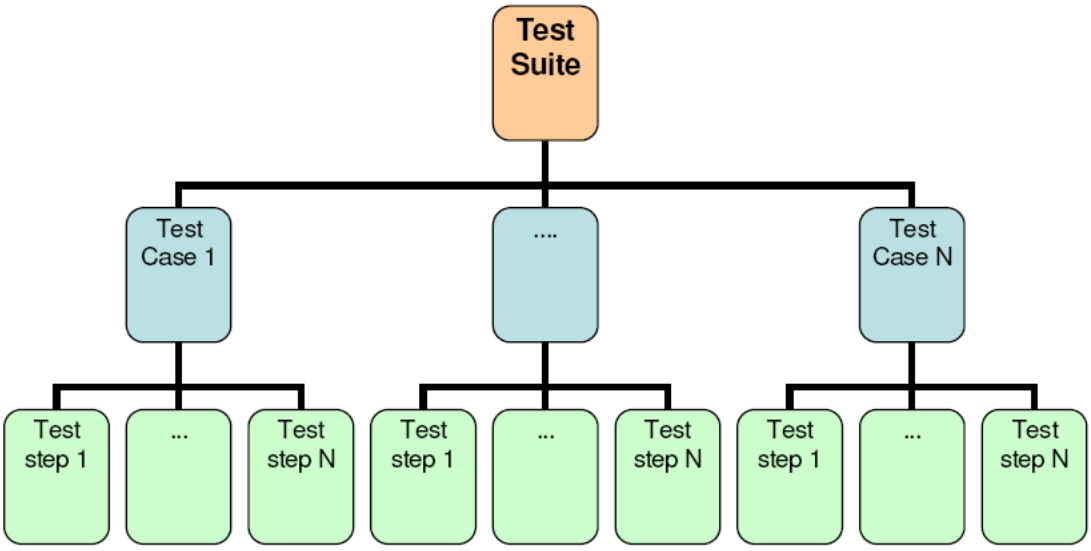


*Figura 12: Organização dos Testes Unitários*

Na Figura 12 'Test Suite' representa um conjunto de 'Test Cases' que estão afectos a um componente do FDF. A 'Test Suite' pode depois ser executada através do 'Test Driver' relacionado com componente que se está a testar.

Neste projecto em específico, a fase de testes é essencial uma vez que a cobertura de todos os componentes do sistema através da execução de testes unitários garante que todos os componentes e seus constituintes estão a funcionar de acordo com o pretendido, prevenindo assim que no futuro eventuais erros, possivelmente de difícil detecção, possam ocorrer.

A cobertura pretendida é de 100%.

### 5.3.2 Desenvolvimento

A especificação dos testes unitários é em tudo semelhante ao que se encontra descrito em 5.1.2, exceptuando o facto dos testes unitários não necessitarem de ambiente de execução devido à particularidade de testarem apenas uma funcionalidade muito especifica, em que o ambiente externo é desprezado.

Na altura de definir os testes unitários deparou-se com um problema: como obter *feedback* da execução de um teste quando apenas o estado interno da classe testada é alterado e não oferecendo a classe *interfaces* para verificar essa alteração? Não podendo adoptar uma solução em que código seria inserido apenas para testes adoptou-se a solução de tornar as classes de teste *friend* das classes que pretendiam testar. Deste modo, e usando uma macro especifica no comando de compilação, foi possível contornar o problema e simplificar os testes unitários. O pedaço de código seguinte foi então colocado no '.h' das classes que sofriam deste problema:

```
#ifdef UNITTEST
      friend class Class_friend;
#endif
```

## 5.4. Sumário

O objectivo primário deste capítulo é definir os testes que servem para validar e verificar a aplicação desenvolvida.

O processo de verificação está directamente relacionado com os testes de integração e com os testes unitários. Este processo pretende garantir que a aplicação cumpre com o especificado e se está bem construída.

O processo de validação consiste em analisar se o produto construído é bom, cumprindo com os requisitos estabelecidos inicialmente. Para este processo serão usados os testes de sistema/aceitação.

Os testes unitários referem-se a testes direccionados aos métodos da classe. Os testes de integração são testes que verificam a integração entre todo o software desenvolvido. Por fim os testes de sistema/aceitação são realizados na aplicação completa sendo feitos com o objectivo de validar a aceitabilidade do projecto.

# Capítulo 6

# GUI e Manual do Utilizador

A aplicação desenvolvida, como qualquer outra, é orientada a um fim e será utilizada por pessoas que podem ter ou não formação específica para o seu uso.

Estando definido nos requisitos que a aplicação teria de ter obrigatoriamente uma GUI, a mesma foi então desenhada tendo em conta todo o trabalho desenvolvido quer na fase de análise, quer na fase de desenho. Desse modo e após a definição da GUI foi também necessário elaborar um documento que demonstrasse de um modo claro todas as potencialidades da mesma e que tentasse cobrir as várias situações com as quais um operador se poderia deparar aquando da sua utilização.

## 6.1. GUI do EXIF

### 6.1.1 Abordagem

Foi desenvolvida uma GUI para o *EXIF* com uma ferramenta proprietária da ESA. A GUI desenvolvida corre como uma aplicação independente da aplicação *EXIF*, sendo que esta irá ler os ficheiros de dados gerados pela GUI. Com a GUI desenvolvida é possível controlar todos os fluxos de dados trocados entre o *EXIF* e os componentes internos ou externos ao FDF. Toda a validação sobre os valores inseridos será feita na GUI, ficando o *EXIF* sem qualquer tipo de preocupação sobre os valores recebidos.

### 6.1.2 Desenvolvimento

Como referido no Capítulo 4 o *EXIF* interage com 3 aplicações internas ao elemento FDF. A aplicação *focusSuite Interface Applications* representa todos os executáveis que serão instanciados quando alterações forem produzidas na GUI de maneira a fazer o parsing do ficheiro com os dados introduzidos.

Para cada elemento com o qual o *EXIF* tem de interagir, irá existir uma *tab* na GUI que irá permitir fazer não só os pedidos a cada elemento como também vai permitir o visionamento dos fluxos de dados trocados previamente entre cada elemento externo e o *EXIF.*

A Figura 13 apresenta o ecrã inicial que será mostrado depois da GUI ser inicializada.

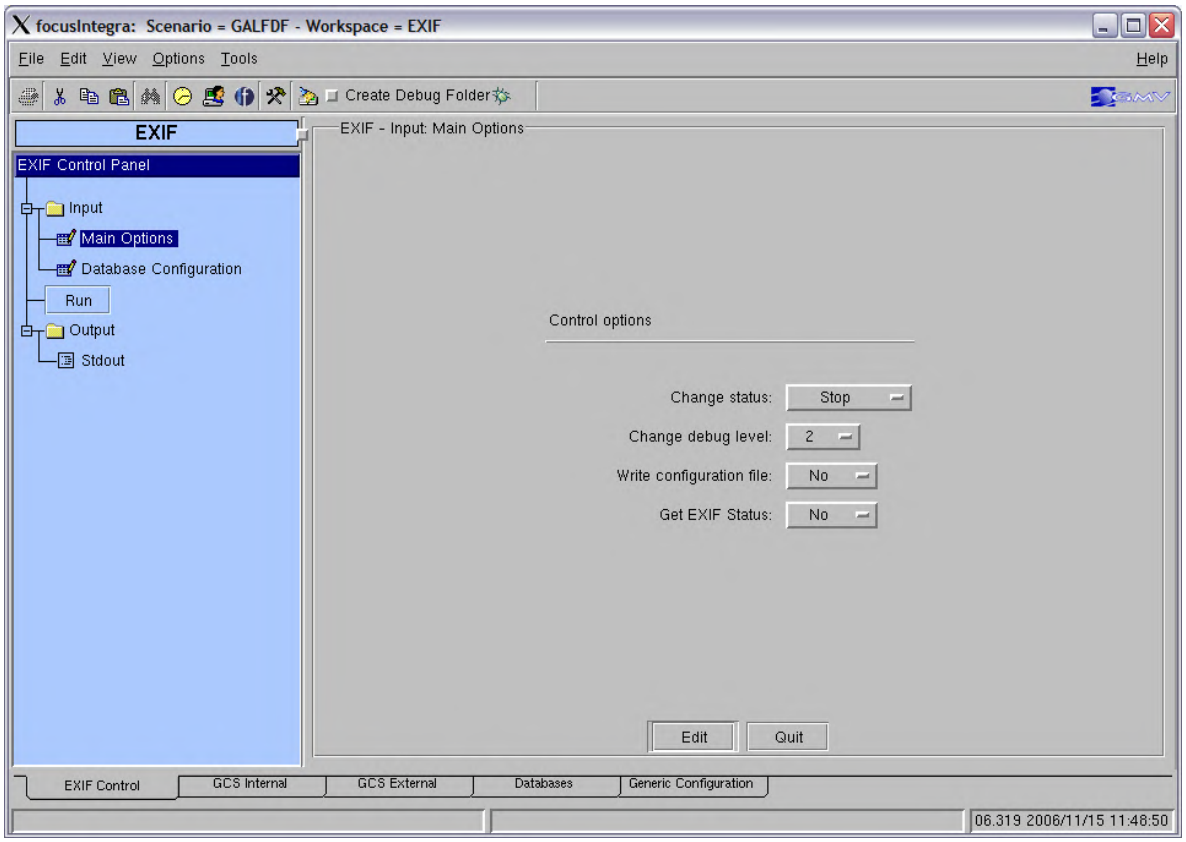



*Figura 13: Ecrã inicial da GUI do External Interfaces*

Sempre que se pretender editar alguma das opções que se encontram em qualquer das janelas da GUI é necessário pressionar o botão 'Edit'. A cor de fundo da aplicação muda de cor, indicando que os campos estão editáveis e o utilizador pode assim inserir/alterar os valores que pretender.

Depois de um utilizador efectuar todas as alterações pretendidas e depois de guardar as mesmas deverá pressionar o botão 'Run', visível na Figura 13 no lado esquerdo. Esse mesmo botão estará "ligado" à aplicação que corresponde à *tab* onde as alterações foram efectuadas. A aplicação irá então fazer o processamento dos dados introduzidos na GUI, lendo para isso os ficheiros com os dados correspondentes e enviará, por CORBA, eventos para o *EXIF* de modo a que estes sejam devidamente processados.

A GUI foi desenvolvida através de TKFORMS (Tcl/tk) de acordo com o que era exigido pelo sistema focusSuite.

## 6.2. Manual da aplicação

O objectivo do manual desenvolvido foi descrever como configurar e usar a GUI do *EXIF*. O manual oferece também informação sobre a capacidade do software desenvolvido e uma lista de erros e operações mais comuns. Procedimentos sobre como instalar o software também se encontram presentes no documento.

É essencial que os futuros utilizadores desta aplicação tenham um documento onde possam tirar as suas dúvidas, quer seja para garantir o normal funcionamento da aplicação, quer na resolução de possíveis erros que possam advir da sua utilização. O manual apresenta um conjunto de passos a seguir em caso de erro e descrições "passo a passo" de como efectuar todas as operações possíveis de realizar com a GUI. São fornecidos também esclarecimentos sobre como utilizar a aplicação na linha de comandos, sendo que neste caso, o seu uso é bastante restrito.

## 6.3. Sumário

Neste capítulo foi abordado o trabalho desenvolvido no sentido de disponibilizar uma interface entre o componente *EXIF* e o utilizador final, o operador da aplicação. Esta GUI foi implementada usando uma plataforma disponibilizada pelo cliente. A GUI é assim disponibilizada por uma aplicação independente do *EXIF*, oferecendo a possibilidade de definir os fluxos de dados a enviar e/ou receber de/para um determinado elemento com o qual o *EXIF* interage.

Sendo disponibilizada uma GUI é necessário fornecer também um manual de modo a que alguém com pouca experiência na utilização da ferramenta consiga entender todas as potencialidades e meios para obter os resultados desejados.

# Capítulo 7

# Implementação

Em termos de processos de engenharia está deverá ser a fase onde menos problemas deverão surgir uma vez que esta só tem o seu início depois das fases de análise, desenho e validação estarem concluídas ou muito perto de se considerarem concluídas. Não obstante, a mesma é necessária para que exista um produto final resultante da interpretação de todas as fases mencionadas.

Ao longo do estágio foi possível comprovar que 90% do trabalho relativo ao desenvolvimento de software é feito nas fases enunciadas nos capítulos anteriores. Esta fase depende bastante do que foi produzido anteriormente e, ainda que seja normal surgirem alterações, todo o trabalho já realizado apresenta um grau de consistência elevado que permitirá uma evolução da implementação do componente de uma forma controlada/planeada. Esta consistência advém essencialmente do trabalho realizado no Capítulo 4.

## 7.1. Implementação do EXIF

### 7.1.1 Abordagem

Esta fase do projecto, sendo crucial no desenvolvimento do software, conta com suportes bastante importantes: a arquitectura, o desenho detalhado e o documento de testes unitários. É com base nestes documentos que toda a implementação é feita, ficando por enfrentar "apenas" problemas relativos ao acto de programar, sendo que todo o processo que permitia definir o que usar e o como fazer já se encontra feito.

Nesta fase o maior obstáculo foi sem dúvida a falta de experiência nas tecnologias usadas, tendo de recorrer a documentação relativa a CORBA ([19], [20], [21] e [22]), sincronização de *threads* ([16], [17] e [18]), etc., uma vez que, nos documentos já abordados nesta secção, tudo o resto se encontra especificado/explicado.

O acto de desenvolver software passa em grande parte pela criação de documentação, não achando por isso que, em termos de processo de engenharia, a fase de implementação venha trazer muito mais conhecimento que aquele que se obteve nas tarefas mencionadas nas secções anteriores.

### 7.1.2 Desenvolvimento

Embora no planeamento esteja referenciado que esta fase teria início em Janeiro e se prolongaria até fins de Abril, a verdade é que a mesma só teve o seu início em Março. Desse modo o trabalho desenvolvido, ainda que considerável, não contempla uma aplicação totalmente funcional. O atraso ocorrido provém do facto do estágio realizado se encontrar inserido num projecto Europeu onde nem sempre é possível contornar problemas que estão além das responsabilidades da Critical Software.

Mesmo tendo em conta estes atrasos creio que o objectivo principal foi atingido, ou seja, ter contacto com as tecnologias, obter experiência no acto de programar e entender as implicações que a documentação tem na fase de implementação.

No que se refere a tecnologias usadas no *EXIF* foi necessário ter contacto com as seguintes:

- CORBA
- FTP
- HTTPS
- SNMP
- XML
- Bases de dados relacionais (MySQL)

No que se refere a linguagens utilizadas o *EXIF* foi maioritariamente desenvolvido em C++ e a parte gráfica em TCL/TK. Algumas das bibliotecas utilizadas eram desenvolvidas em C obrigando assim a que algum código C fosse integrado em código C++.

Todo o *EXIF* foi desenvolvido usando o IDE Eclipse com o *plugin* CDT, de forma a poder interpretar código C++, fazendo a integração com o compilador usado, GCC. Houve uma preocupação constante desde o início do desenvolvimento para que o ambiente de desenvolvimento fosse o mesmo onde o *EXIF* virá a ser executado no futuro, ou seja, numa máquina com o sistema operativo SuSE 9.0.

O facto do estágio estar inserido num projecto Europeu levou a que certos conceitos fossem interiorizados. Só assim se podia garantir que esta fase corresse conforme o planeado. Foi necessário, desde o início, conhecer os *standards* e as *guiding lines* ([6], [7], [8], [9] e [10]) para desse modo produzir um produto com a melhor qualidade possível, evitando assim problemas que poderiam ter consequências desastrosas num futuro próximo.

A Critical Software, com o intuito de obter uma qualidade acima da média nas aplicações desenvolvidas, promove sessões de inspecção de código onde se obtém um conhecimento extraordinário visto que nas mesmas participam todos os elementos do projecto, nomeadamente pessoas com larga experiência neste ramo.

## 7.2. Métricas

A Tabela 6 apresenta as métricas do componente em que estive envolvido até ao momento da conclusão do estágio. Todas estas LOCS (qualquer linha do programa que não seja comentário ou linha em branco, independente do número de sentenças (lógicas ou operações) que estão presentes nessa linha) são o resultado não só da minha contribuição como a de outros elementos que compõem a equipa de desenvolvimento do componente *EXIF* (os componentes nos quais tive envolvido encontram-se marcados com '*****').

| Componente EXIF |
|---|
| **Common (100% concluído) (*)**<br>1663 LOCS em que 75% se encontram testadas através de testes unitários.<br>Contribuição de 440 LOCS. |
| **Dataflow Processing Layer (0 % concluído)**<br>275 LOCS – Apenas esqueleto das classes. |
| **Communication Layer**<br>1358 LOCS. Estas LOCS dizem respeito às seguintes bibliotecas:<br>- FTP Driver (*): 274 LOCS com 67% testadas através de testes unitários.<br>- CORBA Framework (*): 531 LOCS com 80% testadas através de testes unitários.<br>- HTTPS Driver: 553 LOCS com 86% testadas através de testes unitários. |
| **FDF Exif Daemon (*)**<br>677 LOCS<br>Desenvolvido um protótipo inicial que depois foi modificado por outro elemento da equipa do projecto. |
| **EXIF Interface Applications (100% concluído) (*)**<br>956 LOCS em que 93% se encontram testadas através de testes unitários. |

*Tabela 6: Métricas relativas a código efectuado*

Todos estes valores foram obtidos com o recurso à ferramenta LCOV (http://ltp.sourceforge.net/coverage/lcov.php). Esta ferramenta permite extrair do código não só as LOCS apresentadas mas também a percentagem dessas mesmas LOC'S que são efectivamente executadas após a execução dos testes unitários, permitindo assim controlar a *coverage* pretendida nos testes. Toda esta informação poderá, com o recurso a *scripts perl* disponibilizados pela ferramenta LCOV, ser convertida em páginas HTML, tornando assim a leitura e compreensão destas métricas bastante fácil.

O anexo A.1 contém informação mais detalhada sobre as métricas do projecto.

## 7.3. Sumário

Este capítulo apresenta um breve resumo de todo o trabalho realizado relativamente à implementação, apresentando as tecnologias usadas e enumerando todo o processo que existe na Critical Software para garantir a qualidade do trabalho desenvolvido.

Ao longo da fase de implementação revisões de código foram promovidas no sentido de encontrar possíveis erros e também no sentido de promover uma fácil aprendizagem no que às tecnologias envolvidas diz respeito.

Devido à falta de experiência foi recorrente o uso de pequenos protótipos que implementavam pequenas funcionalidades, em tudo semelhantes às funcionalidades que o *EXIF* teria de oferecer.

Este capítulo apresenta também as métricas actuais relativas à implementação efectuada durante o estágio.

# Capítulo 8

## Apreciação Crítica do Trabalho Desenvolvido

Ao longo dos 9 meses de estágio foram vários os desafios e estes foram mencionados ao longo de todo o documento (*baseline* instável, necessidade de adiamentos, introdução de novas tecnologias, etc.). Estes desafios foram apenas e só um entrave momentâneo ao correcto desenrolar do projecto. Ao conseguir ultrapassar estes desafios foi possível adquirir não só um melhor conhecimento das tecnologias envolvidas mas também uma experiência notável no que ao desenvolvimento de software diz respeito.

O que se passou na Critical Software acontece, de certeza, numa grande maioria de empresas que desenvolvem software. Não sendo a Critical Software a empresa *prime* no projecto Flight Dynamics Facility – External Interfaces, aquela que detém o poder de tomar todas as decisões "finais", é por vezes impossível controlar o impacto dos riscos, ainda que estes estejam devidamente identificados.

Todo o trabalho realizado está de acordo com o inicialmente delineado ao nível da documentação o que revela que, como referido ao longo do documento, as fases de análise e modelação são as mais importantes quando se desenvolve software. Na fase de implementação foram detectados alguns erros, na sua grande maioria devido à falta de conhecimento. A falta de experiência nas tecnologias usadas pelo *EXIF* fez com que alguns erros tivessem introduzido algum atraso no planeamento, contudo esse tempo tem vindo a ser recuperado.

Sendo o *EXIF* um componente crítico no FDF, todo o trabalho realizado terá ainda de atravessar uma fase de verificação e validação, algo que tem vindo a ser feito gradualmente ao longo da fase de implementação.

Este projecto revelou-se um autêntico desafio e também uma oportunidade única de aprendizagem.

## 8.1. Trabalho Futuro

O rumo a seguir após a conclusão do estágio é, numa primeira fase, terminar a implementação do *EXIF* para de seguida se proceder à fase de validação, onde será necessário realizar e documentar o resultado de todos os testes delineados nos documentos de testes.

Assim que todo o processo de verificação e validação estiver concluído será necessário efectuar a instalação da aplicação nas instalações do cliente, tendo sempre em mente que essa instalação poderá estar sujeita a acontecimentos não planeados.

Em termos de funcionalidades a adicionar não há muito a fazer uma vez a liberdade para o fazer é quase nula. O facto de não haver funcionalidades a adicionar é, no contexto deste projecto, algo positivo uma vez que indicia que os requisitos se encontram estáveis e que não será introduzido mais *overhead* ao projecto.

## 8.2. Apreciação do Estágio

Em género de balanço final do estágio este tem de ser considerado como uma experiência bastante positiva, ultrapassando as expectativas iniciais. Um dos receios que surgiram, ainda antes de iniciar o estágio, foi o de que ao ter pouca experiência as opiniões expressadas não seriam tidas em linha

de conta. Na Critical Software aconteceu o oposto, sendo várias as vezes em que se mostraram interessados em ouvir a minha opinião, sentindo a todo o momento que era realmente uma mais valia para o projecto.

O facto do estágio estar inserido num projecto europeu obrigou a que fosse estabelecido contacto com pessoas de outro país, mais concretamente Espanha, em que toda a comunicação teve de ser feita em Inglês. Além disso, para este projecto, e devido ao seu contexto (Galileo), foi necessário adquirir muitas metodologias e competências, quer de código quer de documentação, que irão ser de certeza uma mais valia na vida futura como profissional do ramo da Informática.

Sendo a Critical Software uma empresa com certificações de alto nível, que dão crédito não só à empresa como também aos seus trabalhadores, foi ainda necessário obter formação nos processos internos da Critical Software, não só para cumprir com as certificações da Critical Software como também para conhecer o processo de engenharia adoptado pela Critical Software para o desenvolvimento de software.

A Critical Software ofereceu também a possibilidade de aprender algo que não é possível aprender na faculdade: noções relativas à gestão de um projecto, todas as etapas por que um projecto tem de passar, etc.

Sobre o *background* universitário a Licenciatura em Engenharia Informática obtida foi a adequada, essencialmente pela mentalidade imposta aos alunos desde o início, ou seja, o que se aprende na faculdade são os conceitos e as bases para que no futuro se possa evoluir de um modo gradual e sem dificuldades. Muitas das metodologias e conceitos aprendidos na faculdade revelaram-se uma mais valia para conseguir ultrapassar todos os desafios que surgiram ao longo de 9 meses de trabalho.

De salientar ainda assim algumas lacunas no curso que, olhando para trás, se revelariam também uma mais valia:

1. Em primeiro lugar o curso peca pela falta de cadeiras que tenham em linha de conta todo o processo de desenvolvimento de software. Grande parte das cadeiras com projecto, do início ao fim do curso, focam-se no resultado final do projecto, na aplicação, se compila, se corre, etc., quando no fundo o mais importante de todo o desenvolvimento diz respeito à análise e desenho da aplicação e não tanto à implementação.
2. Outra das lacunas tem a ver directamente com a validação dessas mesmas aplicações, do produto final. Ainda que exista no curso uma ou outra disciplina que fale nesse aspecto é de salientar que, se calhar, 90% das cadeiras com projecto não avaliam em qualquer altura a viabilidade e robustez da aplicação desenvolvida, não incutindo nos alunos uma mentalidade de responsabilização sobre o trabalho desenvolvido.

# Bibliografia


[1] M. Richardson, Ground Control Segment: Directory of Acronyms and Abbreviations, 2005, Issue 2.

[2] M. Gameiro, N. Silva, L. Teixeira, N. Duro, J. Freitas, Technical Proposal: Galileo GCS - Flight Dynamics Facility, 2005.

[3] F. Moreira, Management and Administrative Proposal: Galileo GCS - Flight Dynamics Facility, 2005.

[4] Energy And Transport – Galileo: http://ec.europa.eu/dgs/energy_transport/galileo/index_en.htm

[5] R.S. Thompson, I.R. Brighton and C.B. Payne, GALILEO Constellation Control System.

[6] GSWS Study Team, Galileo Software Standard, 2004, Issue 7.

[7] G. Montalto, Guidelines For Applying The Galileo Software Standard, 2005, Issue 1.

[8] Luis Teixeira, Project Software Development Plan, 2007, Issue 3.

[9] Luis Teixeira, Configuration Management Plan, 2007, Issue 3.

[10] Luis Teixeira, Software Product Assurance Plan, 2007, Issue 3.

[11] Donn Le Vie, Jr., Writing Software Requirements Specifications, http://www.techwr-l.com/techwhirl/magazine/writing/softwarerequirementspecs.html, 2007

[12] Robert Japenga, How to write a software requirements specification, http://www.microtoolsinc.com/Howsrs.php.

[13] Enterprise Architect: http://www.sparxsystems.com/products/ea.html.

[14] David Geary, Simply Singleton, JavaWorld.com, 2003

[15] Analysis & Design Patterns: http://www.cs.ualberta.ca/~hoover/

[16] W. Richard Stevens, Bill Fenner, Andrew M. Rudoff, UNIX Network Programming Volume 1, Third Edition: The Sockets Networking API, Addison-Westley, 2003, ISBN: 0131411551.

[17] D. Marshall, Further Threads Programming: Synchronization, http://www.cs.cf.ac.uk/Dave/C/node31.html, 1999.

[18] Ciaran McHale, Generic Synchronization Policies in C++, http://www.ciaranmchale.com/, 2006

[19] M. Henning and S. Vinoski, Advanced CORBA Programming with C++, Addison-Westley, 1999, ISBN: 0201379279.

[20] J. Orvalho, L. Figueiredo e F. Boavida, Extensões ao CORBA Event Service para Comunicação Confiável Multicast, CISUC – Centro de Informática e Sistemas da Universidade de Coimbra.

[21] Paulo Sérgio Almeida, Sistemas de Notificação de Eventos, Grupo de Sistemas Distribuídos do Departamento de informática da Universidade do Minho, 2007.

[22] H. Miranda, M. Correia, Common Object Request Broker Architecture, Programação em Sistemas Distribuídos - DI-FCUL, 2000.

[23] FDF Team – GMV, SW Interface Control Document, 2006, Issue 3.

[24] Integration Testing: http://hissa.nist.gov/HHRFdata/Artifacts/ITLdoc/235/chapter7.htm

[25] Unit Test Techniques: http://c2.com/cgi/wiki?UnitTestTechniques

# Índice Remissivo

Anexo
A

# Anexo A.

## A.1. Métricas do projecto

| Métricas Gerais | |
|---|---|
| Requisitos de sistema afectos ao EXIF | **341** |
| Requisitos de software | **152** |
| Componentes de arquitectura | **24** |
| Classes | **61** |
| Testes de aceitação | **90** |
| Testes de integração | **43** |
| Testes unitários | **206** |
| **Mapeamento dos requisitos de software** | |
| Requisitos de software mapeados a componentes de arquitectura | **152** |
| Requisitos de software mapeados a componentes do desenho detalhado | **152** |
| **Mapeamento dos componentes de arquitectura** | |
| Componentes de arquitectura mapeados a classes do desenho detalhado | **24** |
| **Mapeamento dos testes** | |
| Requisitos de software mapeados a testes de aceitação | **152** |
| Classe de interface mapeadas a testes de integração | **20** |
| Classes mapeadas a testes unitários | **55** |
| **Linhas de código** | |
| Total de linhas de código já implementadas (previstas cerca de 9500) | **4900** |
| Total de linhas de código já testadas | **2800** |