Framework para Ambientes de Trabalho na EFACEC Sistemas de Electrónica S.A.

António Augusto Botelho Granjo Pinto Lisboa

Relatório do Estágio Curricular da LEIC 5º ano

Orientador na FEUP: Prof. Ademar Manuel Teixeira de Aguiar

Orientador na EFACEC Sistemas de Electrónica: Eng. Rogério Dias Paulo

Faculdade de Engenharia da Universidade do Porto

Licenciatura em Engenharia Informática e Computação

Aos meus pais, ao meu irmão e à Susana

Resumo

O projecto *Framework para Ambientes de Trabalho* surge na EFACEC Sistemas de Electrónica, S.A. como um projecto de teste de modo a apreender a viabilidade da criação de raiz de uma *framework* para configuração de dispositivos de subestações de energia.

Os principais objectivos deste projecto foram a criação de ferramentas de gestão de persistência de informação, gestão de módulos e, numa segunda fase, de inclusão de componentes adicionais genéricos de suporte e componentes específicos de visualização. Numa fase posterior, que foi para além dos objectivos deste estágio, foi criado um módulo de teste de modo a melhor testar as funcionalidades criadas.

O projecto pode-se dividir em 3 fases. Uma primeira de criação de estrutura de gestão de persistência de dados e de gestão de módulos. Uma segunda fase de criação de componentes de visualização. E uma terceira fase de criação de um módulo de teste.

Durante cada uma das fases deste projecto foi seguida uma abordagem decorrente da engenharia de software. Foram reunidos requisitos, após os quais foram realizadas definições de arquitectura e posteriormente a implementação do desenho criado. Finalmente, também foram elaborados documentos que descrevem todos os processos realizados.

Este projecto teve um desafio acrescido que consistiu no uso de tecnologias e ferramentas em estado Beta (.NET 2.0 e Visual Studio 2005 beta), algo que tornou o projecto mais interessante mas também resultou na presença de problemas que em versões finais não estariam presentes.

Concluindo, todos os objectivos do projecto foram atingidos, tendo sido criada uma *framework* capaz de reconhecer módulos e os integrar. Foi também bem sucedida a implementação de um módulo de teste que permite a navegação na Internet.

Agradecimentos

Em primeiro lugar gostaria de agradecer à EFACEC Sistemas de Electrónica S.A., sem a qual não teria tido este estágio.

Gostaria de agradecer também ao Engenheiro Rogério Dias Paulo, meu orientador na EFACEC Sistemas de Electrónica, por toda a assistência e orientação. Agradeço também pela confiança, paciência e boa disposição que mostraram ser indispensáveis durante o meu estágio.

À FEUP devo agradecer por me ter proporcionado uma vida académica repleta de experiências que me formaram e me mudaram bastante.

Também gostaria de agradecer ao Professor Ademar Aguiar que, ao ser meu orientador, teve a paciência e vontade de orientar mais um aluno apesar do pouco tempo que dispunha. E também por valiosos conselhos que ajudaram-me durante este estágio.

Ao Professor Augusto Sousa, que ao longo do curso me marcou e apoiou bastante. A sua disponibilidade, orientação, conselhos e boa disposição foram essenciais, principalmente durante o meu período de Erasmus.

Agradeço também ao Professor Raul Vidal cujo esforço tem levado o curso da LEIC e seus alunos a um nível de excelência ao longo dos anos.

Acima de tudo gostaria de agradecer à minha família, aos meus pais e irmão, que sempre, ao longo da minha vida, estiveram comigo e sempre procuraram orientar-me e ajudar-me da melhor maneira. Espero que todo este caminho que tenho percorrido vá de encontro às expectativas deles.

Por fim quero agradecer aos meus amigos, que também tiveram um papel muito importante. À Susana por toda a atenção e animo que me deu. Ao Tiago e ao Sérgio por prezarem a minha amizade mesmo nas alturas mais peculiares. À Cristina cuja companhia e conselhos sempre apreciei. Ao Gaspar pela sua altivez e bom exemplo do que um amigo deve ser. E a todos os outros que me ajudaram a levantar após cada queda.

A todos, obrigado.

Índice de Conteúdos

Índice de Figuras

Índice de Tabelas

1 Introdução

1.1 Apresentação da EFACEC

A EFACEC foi fundada em 1948 iniciando então a sua actividade no fabrico de pequenos motores eléctricos e de transformadores de distribuição.

O Grupo EFACEC está centralizado numa sociedade holding "EFACEC Capital, SGPS", tem participações em empresas afectas a diversas áreas de negócio quer na fabricação de uma vasta gama de equipamentos eléctricos e electrónicos quer na realização de sistemas eléctricos e electromecânicos, de automação, de logística e de telecomunicações.

O suporte industrial da EFACEC é constituído por 4 pólos nos arredores da cidade do Porto, por duas fábricas no Extremo Oriente, uma em Macau e outra na Republica Popular da China em joint-venture com parceiros chineses e ainda por uma fábrica na Argentina em joint-venture com um parceiro local.

As novas fábricas, em Portugal, onde são produzidas as máquinas e os equipamentos eléctricos, estão equipadas com os mais modernos meios de produção e de gestão integrada por computador, segundo os padrões da tecnologia C.I.M.

Estas novas fábricas, consideradas como as mais modernas da Europa no seu ramo, foram inteiramente concebidas e projectadas com recurso às competências das empresas do Grupo, nomeadamente nas áreas da electrónica, gestão, informática, automação e robótica.

O Grupo EFACEC emprega, em Portugal, 2800 pessoas entre as quais 500 licenciados.

Tecnologia, I&D, Qualidade

Os primeiros equipamentos fabricados pela EFACEC envolviam, na sua época, tecnologias avançadas cedidas temporariamente por contratos de licença.

Essas tecnologias foram sucessivamente assimiladas e depois desenvolvidas pelos seus próprios recursos humanos afectos que à produção quer às equipas de Investigação e Desenvolvimento.

No fim da década de 80, a EFACEC dispunha da sua própria tecnologia, tendo concretizado o seu primeiro Contrato de Cedência de Tecnologia com dois fabricantes de transformadores na República Popular da China.

De forma a manter os seus equipamentos e sistemas no *State of the Art* a nível mundial e procurar novos produtos de tecnologia avançada, cada empresa do Grupo desenvolve o seu I&D e subscreve os acordos de cooperação mais convenientes, em que intervêm parceiros nacionais ou internacionais de âmbito institucional e empresarial.

Para manter a competitividade dos seus equipamentos e dos seus sistemas na generalidade dos mercados internacionais, a EFACEC assegura uma qualidade acima da ISO 9001.

Em particular as suas fábricas permitem a realização de ensaios segundo a generalidade das normas internacionais, dispondo para o efeito de laboratórios equipados com a mais moderna aparelhagem que permite a aquisição automática de dados, o seu tratamento e a elaboração por computador dos respectivos certificados de ensaios.

Internacionalização

A partir dos anos 70, a EFACEC começou a exportar motores eléctricos e transformadores.

No início dos anos 90, com os primeiros indícios da globalização mundial da economia, a EFACEC pôs em prática uma estratégia de internacionalização com o objectivo de criar um rede comercial de Agentes, Delegações e Filiais para a venda dos seus produtos e sistemas na Europa, Africa, Américas, Médio e Extremo Oriente, cobrindo mais de 50 países.

No seu essencial, esta tarefa foi considerada terminada em 1995 e a sua operacionalidade confirmada por um crescimento muito rápido e sustentado das exportações.

Porém a consolidação das posições adquiridas não era mais possível com toda a produção concentrada em Portugal.

Para penetrar mais eficazmente nos mercados considerados estratégicos, a EFACEC decidiu então criar competências nesses mercados mediante a constituição de joint-venture's com parceiros locais quer para a fabricação total ou parcial de alguns dos seus equipamentos quer para a realização de certos trabalhos envolvidos no fornecimento dos seus sistemas.

Na área dos sistemas, a contribuição local também foi implementada nas Delegações e Filiais já existentes e em outras a constituir, dotando-as dos meios necessários à realização de uma parte de engenharia de obre e de configuração, bem como do SW específico.

Em resultado desta estratégia foram criadas e estão em fase de criação novas joint-venture's, Delegações e Filiais no Extremo e Médio Oriente, em África e na América Latina.

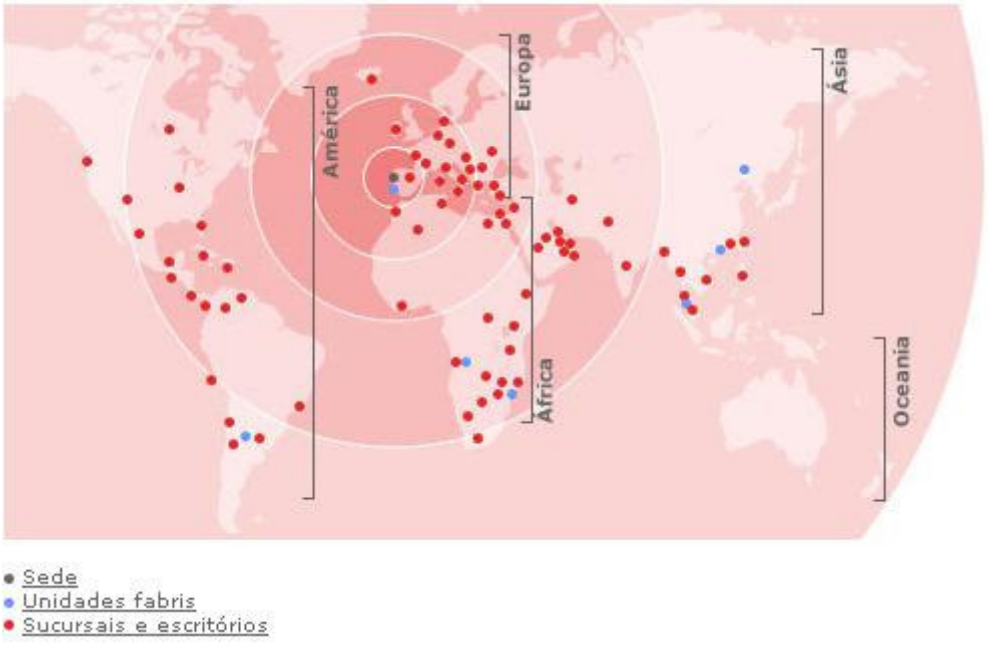

Figura 1 – Distribuição mundial da EFACEC.

1.2 Apresentação da EFACEC Sistemas de Electrónica, S.A.

A EFACEC Sistemas de Electrónica é uma empresa do Grupo EFACEC, vocacionada para as tecnologias de informação e electrónica, mais concretamente na área de automação de sistemas de energia. Fornece soluções de apoio à gestão e automação de redes de energia.

A EFACEC Sistemas de Electrónica é composta por várias unidades as quais procuram atender a diferentes necessidades do mercado. Os recursos humanos da empresa distribuem-se do seguinte modo.

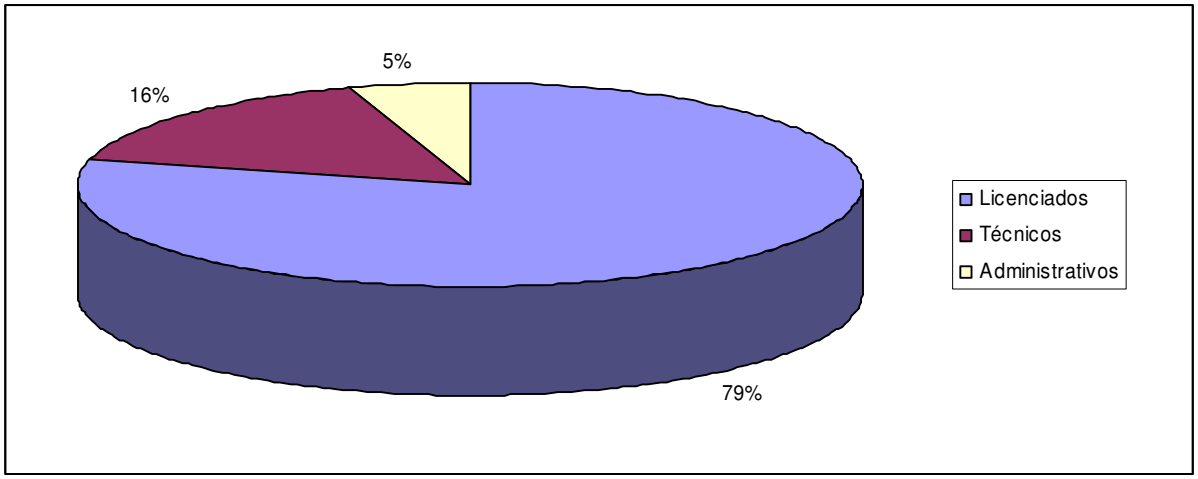

Figura 2 - Recursos humanos da EFACEC Sistemas de Electrónica.

Além disso, de modo a alcançar os seus objectivos foi estabelecida uma política de qualidade, ambiente, segurança e saúde no trabalho que procura fidelizar os clientes, cumprir com requisitos legais e apostar na melhoria e inovação. A complementar esta política é realizado um contínuo investimento em I&D, ponto essencial para manter os seus produtos e sistemas *state of the art*.

As várias unidades que compõem a EFACEC SE estão organizadas da seguinte forma.

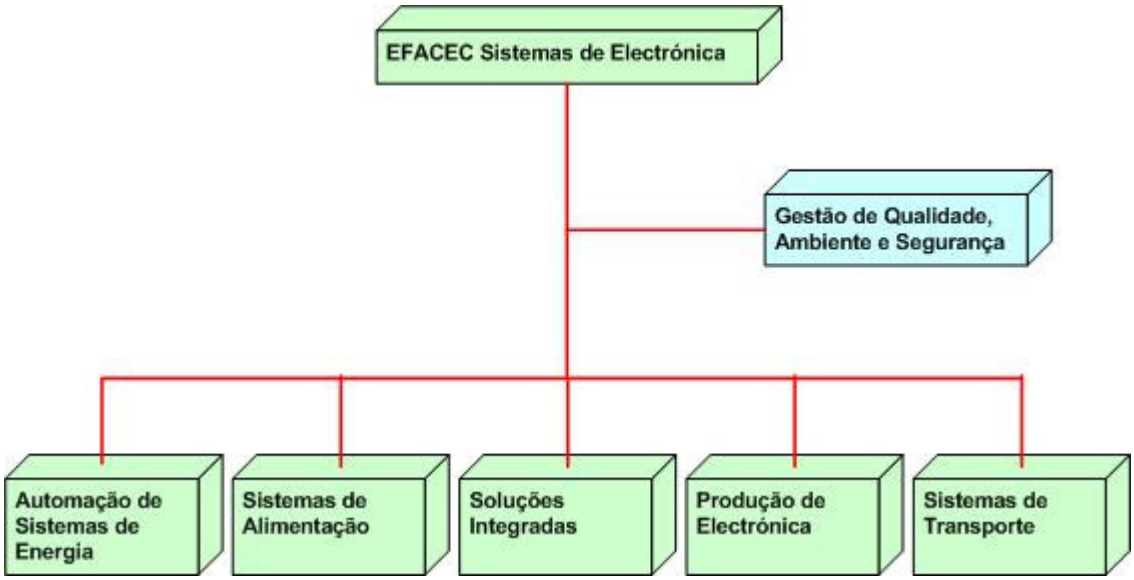

Figura 3 – Estrutura organizacional da EFACEC Sistemas de Electrónica S.A.

A unidade de Automação de Sistemas de Energia fornece sistemas e presta serviços na área da gestão de redes de energia e da automação de subestações.

A unidade de sistemas de alimentação desenvolve uma actividade integral que inclui concepção e fabrico, assim como comercialização e pós-venda de sistemas de energia.

A unidade de soluções integradas está vocacionada para a gestão de projectos de instalação, incluindo o fornecimento de infra-estruturas, nos sectores das telecomunicações, dos transpordes e da energia, dando igualmente o seu apoio às actividades das outras unidades da empresa no domínio das instalações eléctricas.

Na produção electrónica é realizada a assemblagem e teste de cartas electrónicas.

Por fim, a unidade de sistemas de transporte participa em todos os domínios dos sistemas de transporte, criando sistemas para sinalização ferroviária, localização de veículos e outros.

A EFACEC SE cria vários produtos como:

- SCATE X, sistema de gestão de redes de energia
- CLP 500, plataforma distribuída de supervisão e controlo
- µURR, plataforma compacta de supervisão e controlo
- Outros.

1.3 Protótipo *Framework para Ambientes de Trabalho*

A EFACEC tem competências em várias áreas, como transportes, telecomunicações, energia e outras. Em cada uma dessas áreas fornece uma grande gama de produtos e serviços.

Uns dos produtos na área da energia são as subestações de energia. Uma subestação faz parte de um sistema de distribuição de energia e é tradicionalmente responsável pela transformação de altas para baixas tensões, mais apropriadas a distribuição local.

Cada uma destas subestações é gerida por um sistema de comando de controlo, distribuído, de tempo-real e autónomo, que se denomina sistema de automação. Este tipo de sistemas é composto por inúmeros componentes e dispositivos, podendo chegar a ter até um par de centenas de dispositivos com significativo poder de processamento. Cada um desses dispositivos tem que ser configurado e é perante essa necessidade que surge a ideia de criação de uma *framework* que centralize todo este processo.

Com isso em mente surgiu a ideia da criação de uma *framework* expansível que permitisse, através da integração de módulos, configurar todos esses dispositivos. Essa *framework* deveria ter uma estrutura modular cujos módulos pudessem facilmente ser adicionados e removidos, e que ao mesmo tempo fosse possível interagir entre eles. A *framework* deveria igualmente permitir a existência de interfaces ricas que facilitassem a sua usabilidade.

Ficou definido então que se iria desenvolver um protótipo que servisse de base a futuros módulos. Essa *framework* deveria carregar módulos dinamicamente, gerir configurações e criar interfaces dinamicamente.

Com esta base seria depois possível desenvolver módulos que teriam as funcionalidades pretendidas no âmbito das subestações de energia.

1.4 Organização e Temas Abordados no Presente Relatório

O primeiro capítulo deste relatório consiste numa introdução, na qual é descrita a empresa de estágio, é também descrito sumariamente o projecto elaborado.

No segundo capítulo é enquadrado o projecto de estágio na EFACEC SE, de onde surgiu e o que visa resolver.

O terceiro capítulo contém as opções e escolhas feitas a nível tecnológico. Isto reflecte linguagens e ferramentas que foram escolhidas para o desenvolvimento do projecto.

O quarto capítulo contém a especificação do projecto a um nível mais profundo. São referidos requisitos, arquitecturas adoptadas e várias fases do projecto.

O quinto capítulo descreve a implementação do projecto, tal como problemas encontrados e respectivas soluções.

Por fim o sexto capítulo contém as conclusões do projecto, desde o projecto que foi desenvolvido a futuros desenvolvimentos.

2 Enquadramento do Projecto na Empresa

A EFACEC tem vindo ao longo dos anos fornecido vários produtos e serviços. Essa gama tem vindo a crescer e actualmente são fornecidas soluções em variadas áreas como:

- Transportes
- Logística
- Ambiente
- Energia
- Indústria e Edifícios
- Telecomunicações
- Serviços

Relativamente à área de Energia há várias subáreas que pertencem igualmente às competências da EFACEC, mais precisamente:

- Produção de Energia
- Transmissão e Distribuição de Energia
- Automação e Telecontrolo
- Sistemas de Alimentação de Energia
- Assistência e Manutenção

Focando o ponto da transmissão e distribuição de energia um dos produtos que a EFACEC fornece são subestações de energia a partir das quais surge a ideia da realização do protótipo *Framework para Ambientes de Trabalho*.

2.1 Subestações de Energia

Uma subestação de energia é um importante componente num sistema de transmissão e distribuição de energia. Geralmente o propósito de uma subestação é a transformação de altas tensões para baixas, fazendo uso de transformadores. Isto porque é mais eficaz transmitir electricidade a longas distâncias a altas tensões. No entanto para a sua distribuição local as tensões deverão ser baixas.

Figura 4 – Subestação de energia.

As subestações trabalham uma determinada voltagem, os valores dessas voltagens podem ir desde os 110V até aos 765kV. Consequentemente existem diferentes tipos de subestações, as quais podem ser construídas à superfície, podem ser subterrâneas (em edifícios próprios para o efeito) ou aéreas (montadas em postes de electricidade). Podem também ser móveis, sendo para isso montadas em reboques.

Figura 5 – Subestação de energia móvel.

Cada subestação é composta por variados componentes, como por exemplo transformadores, os quais deverão ser correctamente configurados. Muitos dos dispositivos a configurar são sistemas hardware/software com capacidades computacionais equivalentes a um PC e cada um destes é tendencialmente configurado independentemente. Surge então a necessidade de criar um sistema centralizado de configuração que permita gerir todo este processo de uma forma facilitada.

2.2 Projecto *Framework para Ambientes de Trabalho*

Para criar uma aplicação capaz de configurar um grande número de dispositivos definiu-se que se desenvolveria um protótipo de uma aplicação modular cuja base permitisse a integração dinâmica de módulos de *software* e que permitisse igualmente a criação de interfaces para controlar esses mesmos módulos.

O desenvolvimento do protótipo foi dividido em três fases:

- Desenvolvimento de componentes de gestão de módulos e de gestão de configurações.
- Desenvolvimento de gestão de componentes de visualização
- Elaboração do relatório de estágio.

Além do protótipo desenvolvido foi também implementado um módulo de teste de modo a procurar verificar o bom funcionamento da aplicação. Apesar de não fazer parte dos objectivos do estágio, e como houve tempo para isso, julgou-se ser um importante complemento à *framework* desenvolvida.

3 Opções Tecnológicas de Desenvolvimento

Para o desenvolvimento deste projecto foram ponderadas várias abordagens quer várias plataformas e linguagens, as quais estão descritas neste capítulo. Isto porque, apesar de se ter o objectivo de uma aplicação que integrasse diferentes módulos, essa mesma aplicação poderia ser ela mesma um módulo de uma outra aplicação.

3.1 Abordagens Possíveis

O problema inicial era a criação de uma plataforma que permitisse configurar os dispositivos de uma subestação de energia. Para isso foram ponderadas duas possibilidades. A criação de um módulo para uma *framework* já existente ou a criação de uma *framework* de raiz que integraria por sua vez módulos.

3.1.1 Criação de um módulo

Para criar um módulo, a melhor plataforma no momento é o Eclipse. O Eclipse tem vindo a ser desenvolvido e tem actualmente uma base robusta, poderosa e muito versátil. Vai na sua 3ª versão e conta já com inúmeros módulos que lhe conferem uma vasta gama de funcionalidades acrescidas. É baseado na linguagem JAVA [Java 2005], que embora seja uma poderosa linguagem, o que restringe a escolha da linguagem de programação utilizada.

Esta abordagem implica que haja um bom conhecimento da estrutura do Eclipse e das regras e limitações cujos módulos devem ter. Além disso, dada a complexidade e potencialidades inerentes ao Eclipse, a criação de um módulo pode requerer a criação de mais funcionalidades que as desejadas. No entanto seria uma boa escolha já que, como os próprios criadores do Eclipse referem, a versão 2.0 é um *IDE* para tudo e para nada em particular, sendo a versão 3.0 uma aplicação Java genérica com uma arquitectura modular e flexível [Eclipse 2005].

Ao criar um novo módulo este iria estar ligado à plataforma Eclipse como mostra a seguinte figura. Sendo os módulos posteriores ligados ao módulo desenvolvido.

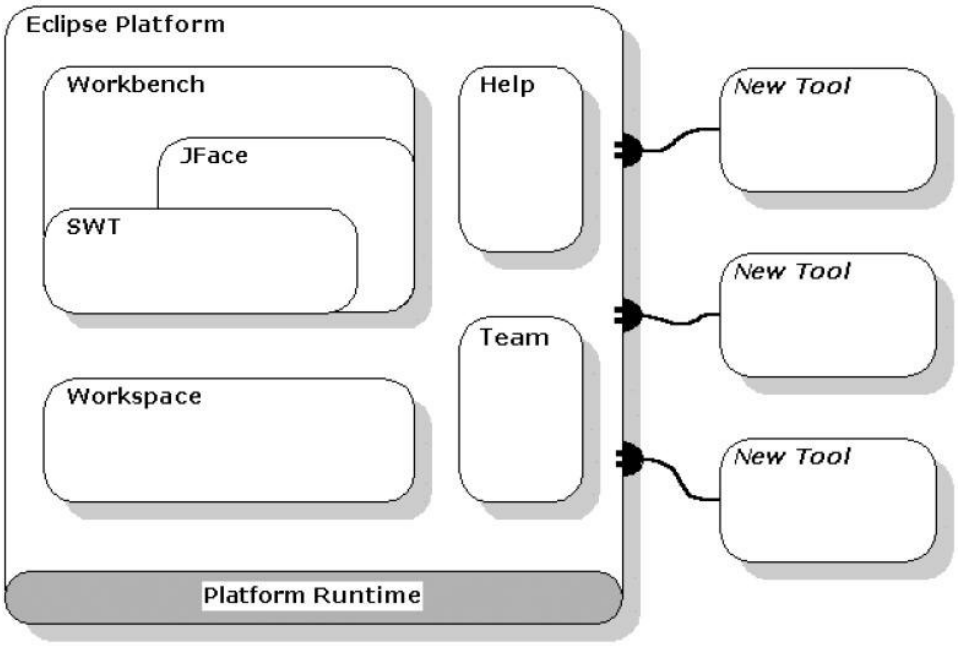

Figura 6 – Arquitectura da plataforma Eclipse.

3.1.2 Criação de uma framework de raiz

A outra possibilidade é a criação de uma *framework* de raiz, processo esse que só por si é bastante complexo. Por outro lado permite criar uma aplicação que encaixe perfeitamente nas necessidades requeridas. Para criar uma *framework* de raiz é necessário definir o modo como está estruturada e como são realizadas as ligações entre cada um dos seus componentes. Isto requer, entre outras coisas, o estudo de *framework*'s existentes, de modo a recolher boas e más ideias, e adaptá-las ao caso em questão.

Em suma, é necessário definir uma estrutura da *framework*, passando pelo modo como a informação é guardada, como os módulos devem ser e como toda essa informação é tratada, transformada e guardada. Toda estas definições arquitecturais podem culminar numa solução mais trabalhosa do que a criação de um módulo para Eclipse, isto já que, para além de ser necessário conhecer a estrutura de uma *framework* é necessário adaptá-la às necessidades em questão [CP 2005].

3.2 Tecnologias Utilizáveis

Para além das abordagens, e embora a escolha de uma determinada abordagem possa condicionar a tecnologia utilizada, é necessário escolher as tecnologias utilizadas no desenvolvimento. Neste caso a escolha recai sobre a utilização da linguagem Java ou C#. Além da linguagem será utilizado um ambiente de trabalho apropriado, como descrito nos seguintes pontos.

3.2.1 JAVA

A linguagem Java, criada pela Sun com o objectivo de substituir o C++, é utilizada desde 1996, é uma linguagem orientada a objectos, sendo já bastante madura. Corre em qualquer plataforma e é de distribuição livre. Facilitando bastante o processo de distribuição e simultaneamente reduzindo os custos.

Relativamente a características da linguagem Java, esta tem gestão automática de memória, presença de interfaces e classes. É uma linguagem que procura primar pela segurança e tem um modelo de compilação *just-in-time*, conferido pela *Java Virtual Machine*. Além disso tem uma API que permite facilmente procurar métodos e funcionalidades já existentes.

Figura 7 – Logótipo da linguagem Java.

3.2.2 C#

A linguagem C# é bem mais recente que a linguagem Java. Foi criada pela Microsoft procurando tirar o melhor de outras linguagens, como a facilidade de programação do Visual Basic e o poder e rapidez do C++. Tal como o Java também tem gestão automática de memória, compilação *just-in-time*, serviços de segurança e uma extensa API [MSDN 2005].

No entanto não partilha com o Java a compatibilidade de plataformas. Já que requer a plataforma .NET e consequentemente um ambiente Windows. Esta limitação pode não ser tão vincada uma vez que existem projectos, como é o caso do projecto Mono [Mono 2005], que visam permitir que programas .NET possam correr em ambientes Linux, OSX, BSD e Solaris ou em alguns tipos de sistema Unix. No entanto são ferramentas desenvolvidas por terceiros e que padecem da falta de apoio da Microsoft.

Pode não ser uma linguagem tão experiente como o Java mas tem no entanto uma visão mais actualizada do mundo da programação, podendo estar mais adaptada aos desafios com que os programadores se deparam actualmente.

Além disso, a versão 2.0 do .NET, que ainda está a ser desenvolvida, mas já disponível em versão beta, tem novas funcionalidades, especialmente no que toca a interfaces. Dando uma nova facilidade na criação de programas com aspecto profissional sem grandes dificuldades.

3.3 Ambientes de Desenvolvimento

3.3.1 Eclipse

O Eclipse é um ambiente de desenvolvimento que tem vindo a evoluir bastante. Passou por ser uma alternativa não muito poderosa para desenvolvimento em Java para a plataforma de eleição. Tem contínuos projectos a serem desenvolvidos que lhe conferem enormes potencialidades.

Como pontos centrais tem o desenvolvimento em Java, o controlo de desenvolvimento em equipa (fazendo uso de um sistema de CVS) e um ambiente de desenvolvimento de plug-ins (denominado PDE – *Plug-in Development Environment*).

Estes pontos tornam o Eclipse numa plataforma de desenvolvimento muito atraente e que tem todas as funcionalidades necessárias para o desenvolvimento de plug-ins.

Figura 8 – Logótipo da Plataforma Eclipse.

3.3.2 Visual Studio .NET 2003

O Visual Studio .NET 2003 é a versão corrente e estável disponibilizada pela Microsoft para desenvolvimento em .NET.

Permite desenvolver aplicações em C# e outras linguagens (como C++ ou Visual Basic). É um IDE poderoso que tem como pontos fortes a facilidade de criar aplicação para Internet, serviços de suporte de XML e procura tirar ao programador o máximo de trabalho possível, através da automatização de várias tarefas. O Visual Studio está pensado para criar aplicações para Windows estando bastante optimizado para isso. Está no entanto limitado não estando disponível para outros sistemas operativos. E embora hajam projectos que procuram remover esta limitação, estes projectos são no entanto desenvolvidos por terceiros e não é garantida a sua estabilidade.

Figura 9 – Logótipo do Visual Studio .NET.

3.3.3 Visual Studio .NET 2005 beta 1 e beta 2

O caso do Visual Studio .NET 2005 [VSBE 2005] é diferente os dois ambientes de desenvolvimento apresentados. Representa a próxima versão do Visual Studio e ainda se encontra em desenvolvimento. Na altura do início do estágio a versão disponível era a beta 1. Esta versão algo estável ainda sofria de bastantes problemas. *Bugs* eram encontrados com facilidade e havia opções que ainda se encontravam em desenvolvimento.

Em finais de Abril foi lançada a versão beta 2, já mais estável e com um aspecto mais finalizado (estima-se que seja a ultima versão antes da final).

Apesar de ser uma versão beta já estão presentes novas funcionalidades e importantes aspectos que dão ao Visual Studio .NET 2005 algumas vantagens em relação às outras plataformas.

Um dos fortes aspectos nesta versão do Visual Studio passa pela presença de testes unitários integrados e a existência de novos componentes gráficos relativos à plataforma .NET 2.0 [Smart 2005].

A integração dos testes unitários facilita a realização de testes, já que no caso do Eclipse ou da versão anterior do Visual Studio era necessário utilizar ferramentas auxiliares, como o Junit ou o Nunit (para Java e .NET respectivamente). Com esta nova ferramenta pode-se realizar um desenvolvimento seguindo a metodologia de *Test Driven Development* [TDD 2005] com muita mais facilidade.

Além disso o Visual Studio também tem ferramentas como o *Team Share* que permite o trabalho conjunto numa solução (podendo existir arquitectos, analistas e programadores que têm diferentes papéis no desenvolvimento). Outra ferramenta é o Virtual Safe que permite salvaguardar todo o trabalho feito. Semelhante a um sistema de CVS.

3.4 Escolhas Realizadas

Apesar de terem sido apresentadas todas estas possibilidades as escolhas realizadas foram tomadas inteiramente por parte da empresa, que escolheu criar uma *framework* de raiz utilizando C# e o Visual Studio .NET 2005 beta.

Relativamente à escolha de criar uma *framework* de raiz esta foi feita já que como o que se desenvolveu seria um protótipo procurou-se testar duas ideias em conjunto, o desenvolvimento de uma *framework* de raiz e o desenvolvimento da aplicação desejada.

Por outro lado como se desejava criar um protótipo para ambiente Windows a escolha recaiu para a linguagem C#. Embora a linguagem JAVA corra sem quaisquer problemas em Windows dado que o C# é desenvolvido pela Microsoft torna-o numa escolha mais propícia para Windows. Além disso como se desejava já utilizar as novas funcionalidades do Visual Studio .NET 2005 beta, como os testes unitários integrados e os novos componentes visuais, a escolha foi facilitada.

4 Especificação do Projecto

Os requisitos podem ser agrupados em três secções, respeitantes a cada uma das fases do projecto. A primeira fase é respeitante á persistência de informação e gestão dos módulos existentes. Esta é a fase mais crucial já que representa a base de qualquer *framework*, uma vez que deve permitir que informação como, recursos existêntes, módulos presentes, e estado da aplicação estejam devidamente organizados que possam ao mesmo tempo ser convenientemente manipulados.

Numa segunda fase é necessária a criação dinâmica de componentes de visualização, tal como os menus e butões de um determinado módulo, e a ligação desses objectos gráficos com as funcionalidades de cada módulo.

Finalmente numa terceira fase, a qual já se extende para lá os objectivos iniciais do estágio, é a criação de um módulo de teste que permita verificar o correcto funcionamento da aplicação criada.

4.1 A Framework para Ambientes de Trabalho

Principais Funcionalidades

A principal funcionalidade da *framework* é de servir de base e agente centralizador de módulos cujos objectivos são de configurar dispositivos. Será necessário fazer a gestão de configurações assim como a gestão de módulos. Além disso é também importante a gestão de componentes de visualização. A criação de módulos deverá seguir regras próprias as quais deverão ser definidas.

Utilização Prevista

A utilização prevista da *framework* é por um único utilizador, que será um técnico especializado da EFACEC, que para além de ter conhecimentos respeitantes aos dispositivos a configurar, possui também avançados conhecimentos de informática.

Deste modo não é necessária a implementação de funcionalidades que distingam diferentes utilizadores.

Assunções e Dependências

Um dos requisitos da *framework* é que esta seja versátil, sendo possível corrê-la em qualquer máquina e sem grandes transtornos. Os únicos requisitos serão uma máquina com sistema operativo Windows e a *Framework .NET 2.0* (que ainda se encontra em estado beta).

No entanto módulos que venham a ser desenvolvidos podem requerer outros requisitos.

4.2 Tipos de Requisitos

Os requisitos das várias fases podem dividir-se em 3 categorias demonstrando diferentes níveis de prioridade.

P0 → Essencial (nível de prioridade mais elevada). Os requisitos com este nível de prioridade devem ser validados na primeira versão do produto a entregar ao cliente.

P1 → Muito desejável (segundo nível de prioridade). Estes requisitos devem ser contemplados numa segunda fase de implementação do produto, não é necessário que sejam contemplados na primeira versão, mas **deverão obrigatoriamente ser implementados**. A versão final do produto deve contemplar estes requisitos que devem, nessa altura ser validados.

P2 → Desejável (terceiro nível de prioridade). São requisitos que têm ou podem ter interesse para o cliente, mas cuja implementação não é absolutamente indispensável. Se forem implementados, devem ser validados.

4.3 Fase 1 – Gestão de Configurações e Gestão de Módulos

Esta fase do projecto teve várias componentes que foram desenvolvidas.

4.3.1 Requisitos de Configurações

As configurações consistem em todos os dados que são necessários estar presentes quando a aplicação corre, como as variáveis de controlo, etc. Deve ser possível guardar qualquer parâmetro e que haja, se necessário, hierarquias entre eles.

Assim sendo tem-se os seguintes parâmetros.

Ref.	Descrição	Prioridade
S1	*Deverá ser possível guardar qualquer tipo de parâmetro*	*P0*
S2	*Um parâmetro é constituído por um nome e um valor*	*P0*
S3	*Deverão estar presentes funções de set e get e outras necessárias para manipular os parâmetros*	*P0*
S4	*Os parâmetros deverão estar hierarquizados*	*P0*
S5	*Os parâmetros serão guardados num ficheiro XML*	*P0*
S6	*Deverão existir funções para comparar e fundir diferentes ficheiros de configurações*	*P0*
S7	*Todas as configurações deverão estar centralizadas numa única estrutura.*	*P0*

Tabela 1 – Requisitos do gestor de configurações (Settings).

4.3.2 Requisitos dos Módulos

Os módulos são outra importante parte desenvolvida. Um módulo confere uma expansibilidade de funcionalidades, algo que se tem em mente quando da criação de uma *framework*.

Para o gestor de módulos devem ser respeitados os seguintes requisitos.

Ref.	Descrição	Prioridade
P1	*A framework deverá carregar os módulos no arranque, não sendo necessário que isto seja realizado em tempo real*	*P0*
P2	*Cada módulo terá uma lista das suas dependências em relação a outros módulos*	*P0*
P3	*A framework deverá carregar os módulos tendo em conta as suas dependências.*	*P0*
P4	*Cada módulo deverá estar contido numa directoria específica com uma estrutura standard a definir*	*P0*
P5	*Cada módulo deverá ter um ficheiro de configuração em XML*	*P0*
P6	*A directoria de um módulo deverá ter várias subdirectorias standard que dividam os diferentes tipos de ficheiros, como imagens, bibliotecas ou definições*	*P0*
P7	*Deverá ser possível desactivar módulos com um método simples, tal como colocar um ficheiro "disable" na sua directoria*	*P1*
P8	*A remoção de plug-ins deverá ser simples, bastará apagar o seu directório*	*P0*
P9	*Os módulos deverão ser configuráveis através do gestor de configurações*	*P0*

Tabela 2 – Requisitos do gestor de módulos (Plug-Ins).

4.3.3 Requisitos do Gestor de Recursos

Algo que também tem que estar presente são recursos, estes podem ser imagens, strings ou outros elementos utilizados por módulos nas suas várias funcionalidades. Estes devem estar centralizados facilitando o seu acesso e manuseamento.

Os seguintes requisitos devem ser respeitados.

Ref.	Descrição	Prioridade
R1	*A framework deverá permitir que facilmente se altere a língua, cursores ou imagens do seu interface*	*P0*
R2	*Os recursos que dor módulos quer da framework deverão estar guardadas no sistemas em estruturas de directorias standards*	*P0*
R3	*Todos os recursos deverão estar centralizados numa única estrutura a qual deverá ser acessível quer pela framework quer pelos módulos*	*P0*
R4	*A utilização do gestor de recursos deve ser o mais simples e flexível possível, fazendo-se uso de métodos set, get, remove e análogos.*	*P1*

Tabela 3 – Requisitos do gestor de recursos (Resources).

4.3.4 Requisitos do Gestor de Logs

Dado que há muitas operações envolvidas é necessário guardar a ocorrência de problemas para tal é necessária a criação de um sistema de *log's* que permita manusear essa mesma informação.

Ref.	Descrição	Prioridade
L1	*O sistema deverá permitir guardar mensagens provenientes da framework*	*P0*
L2	*As mensagens deverão ficar guardadas num ficheiro*	*P0*
L3	*O gestor deverá estruturado de tal modo que permita a sua expansão futura para, por exemplo, outro tipos de log (ecrã, mail, etc.)*	*P0*
L4	*O gestor deverá correr numa thread distinta da restante framework*	*P0*
L5	*Para facilitar o processo de comunicação entre a framework e o logger poderá haver uma pilha de mensagens.*	*P1*
L6	*As mensagens deverão ser acompanhadas de informação temporal (data ou hora, ou ambos)*	*P0*
L7	*Deverá ser possível activar e desactivar os diferentes tipos de logging (guardar ou não para ficheiro, etc.)*	*P1*
L8	*O logger deverá ser construído de tal maneira que facilite a futura criação de novos tipos de loggers*	*P1*

Tabela 4 – Requisitos do gestor de logs (Logger).

4.3.5 Outros Requisitos

Outros requisitos que não cabem em nenhuma das anteriores categorias estão listados abaixo.

Ref.	Descrição	Prioridade
O1	*Deverão ser implementados testes unitários para testar as funcionalidades que vão sendo desenvolvidas nesta primeira fase. Os testes deverão ser realizados fazendo uso das ferramentas presentes no Visual Studio .NET 2005*	*P1*

Tabela 5 – Outros requisitos.

4.4 Fase 2 – Criação de Componentes de Visualização

4.4.1 Requisitos Gerais de Componentes de Visualização

Ref.	Descrição	Prioridade
A1	*A aplicação deverá permitir a presença de vários ambientes de trabalho*	*P0*
A2	*Deverão ser guardadas as informações respeitantes à aplicação, como, por exemplo, tamanhos de janelas e suas posições*	*P0*
A3	*Em cada momento deverá haver um único ambiente de trabalho activo*	*P0*
A4	*Após uma alteração a um ambiente de trabalho todas as informações alteradas deverão ser actualizadas*	*P0*
A5	*A informação relativa ao ambiente de trabalho deve ser carregada a partir de um ficheiro XML*	*P0*
A6	*Deverá existir em cada ambiente de trabalho uma barra de menus, barras de ferramentas e uma barra de estado*	*P0*
A7	*As barras de ferramentas e de estado poderão ser ou não visíveis*	*P0*
A8	*Deve ser também possível guardar a informação da aplicação num ficheiro XML*	*P0*

Tabela 6 – Requisitos gerais de componentes de visualização.

A segunda fase do projecto consiste na criação dos componentes de visualização da *framework*, ou seja, o que o utilizador vai ver e com o qual vai interagir. Estes componentes devem ser dinâmicos e devem seguir os seguintes requisitos.

4.4.2 Requisitos de Barras de Ferramentas

As barras de ferramentas, de menus e de estado representam uma importante componente da interface criado. Assim sendo há vários pontos que devem respeitar.

Ref.	Descrição	Prioridade
B1	*A barra de menus poderá ser composta por vários menus*	*P0*
B2	*Cada menu poderá ter vários sub menus e assim sucessivamente*	*P0*
B3	*Os itens dos menus poderão ter imagens associadas*	*P0*
B4	*A posição e visibilidade de cada barra de ferramentas deverão ser guardadas e repostas quando o ambiente de trabalho é carregado*	*P0*
B5	*Cara barra de ferramentas poderá ter vários itens*	*P0*
B6	*As barras de menus e de estado terão uma posição fixa no topo da janela e no fundo, respectivamente*	*P0*
B7	*Os itens da barra de menus e de ferramentas deverão permitir a presença de um nome e um texto de ajuda*	*P1*
B8	*A criação dos itens para as diferentes barras é da responsabilidade dos módulos presentes, a framework só os colocará no interface conforme o solicitado pelos módulos*	*P0*
B9	*Os módulos deverão utilizar tipos base de elementos de interface, os quais fazem parte da framework*	*P0*

Tabela 7 – Requisitos de barras de ferramentas e menus.

4.4.3 Requisitos de Painéis

Por fim a criação de componentes de visualização contempla a criação de painéis onde será visualizada a maior parte da informação. Estes painéis, inseridos em áreas, devem seguir as seguintes linhas.

Ref.	Descrição	Prioridade
P1	*A área de trabalho poderá estar dividida em áreas laterais e uma área central*	*P0*
P2	*A área central deverá permitir a presença de vários painéis (diferentes tabs)*	*P0*
P3	*As áreas laterais poderão ser compostas por um conjunto de vários painéis ou por duas subáreas*	*P0*
P4	*As subáreas da areal lateral poderão ser compostas por outras áreas ou por um painel*	*P0*
P5	*Os painéis serão definidos pelos módulos sendo a framework responsável pela sua visualização*	*P0*
P6	*Deverá existir uma estrutura centralizada com todos os tipos de painéis. A partir desses tipos é que deverão ser criadas novas instâncias sempre que se deseja ter um novo painel*	

Tabela 8 – Requisitos de painéis de visualização

4.5 Fase 3 – Módulo de Teste (Web Browser)

O módulo de teste desenvolvido é um web browser simples que permite navegar na Internet, deve no entanto respeitar os seguintes pontos:

Ref.	Descrição	Prioridade
M1	*O módulo criado deve ser compatível com a aplicação já desenvolvida, devendo seguir as linhas já definidas para o desenvolvimento de módulos*	*P0*
M2	*O módulo deverá ser um web browser simplificado que permita a navegação na Internet*	*P0*
M3	*O módulo deverá ter recursos a serem carregados*	*P0*
M4	*O módulo deverá procurar utilizar as várias funcionalidades disponibilizadas pela framework*	*P0*
M5	*Deverão estar presentes um menu e uma barra de ferramentas*	*P0*
M6	*Deverão estar presentes funcionalidades como "back", "forward", "refresh", "stop", "go" e "home". Semelhantes às existentes noutros web browsers*	*P0*

Tabela 9 – Requisitos do módulo de teste.

5 Desenvolvimento do Projecto

5.1 Fase 1 – Persistência de Dados e Gestão de Módulos

5.1.1 Perspectiva Global

A parte de persistência de dados e gestão de módulos representa a primeira fase do projecto e é a base para as outras componentes a serem desenvolvidas. Para esta fase foi necessário criar uma estrutura de dados que fosse poderosa o suficiente para conter toda a informação directamente ligada à aplicação. Além disso foi também necessário criar uma sistema de carregamento de módulos, os quais têm dependências que devem ser respeitadas, tendo em atenção problemas como dependências circulares.

Esta fase foi acompanhada com a realização de testes unitários de modo a garantir o máximo de qualidade possível, permitindo dar à *framework* criada uma base segura e de confiança, graças a cerca de duas centenas de testes unitários implementados.

5.1.2 Arquitectura lógica

O sistema desenvolvido será dividido em camadas de modo a facilitar quer a sua implementação quer a sua manutenção futura. Uma estruturação em camadas facilita também a criação e integração de novas funcionalidades.

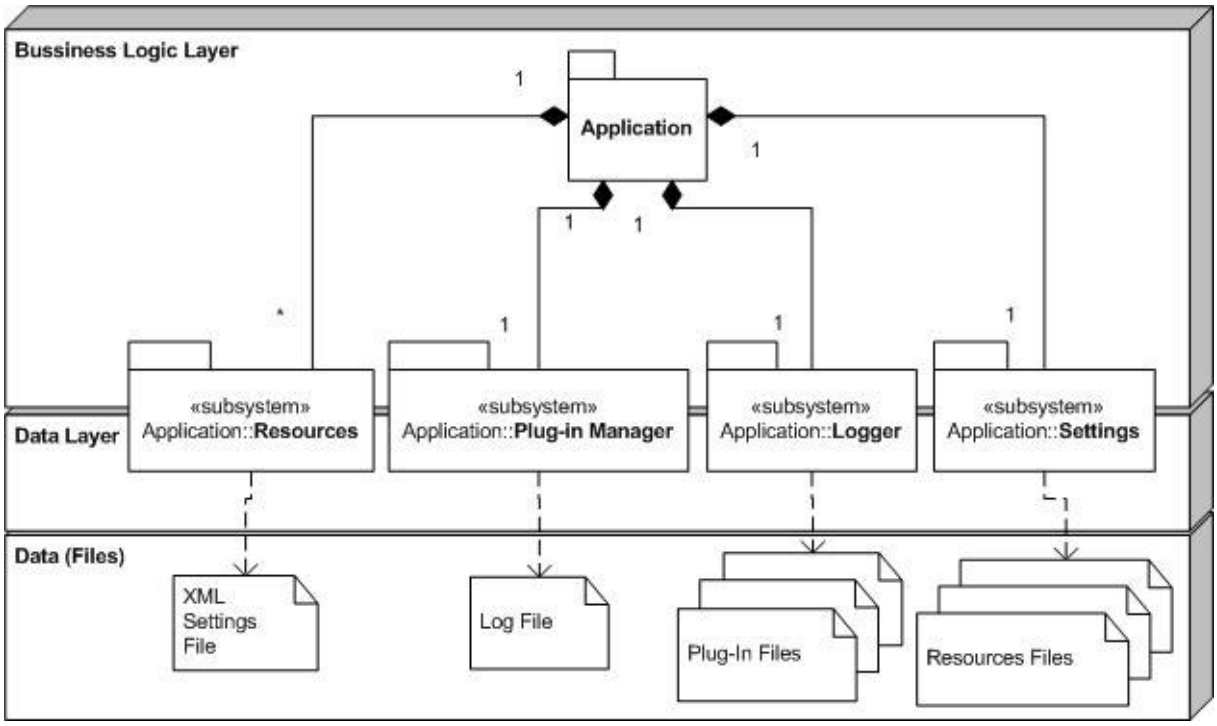

Figura 10 – Arquitectura lógica.

Camada de Lógica de Negócio

A camada de lógica de negócio (*bussiness logic layer*) é a responsável pelo grosso do processamento da aplicação. É aqui que a informação proveniente da camada de dados e da (futura) camada de interface é tratada e processada, nesse ponto de vista é também uma ponte dessas duas camadas.

Camada de Dados

A camada de dados (*data layer*) é que realiza a ponte entre os dados contidos em ficheiros e a informação existente nas restantes camadas. Esta camada está integrada nos subsistemas da *framework* permitindo que o tratamento dessa informação seja mais centralizado.

Cada um dos componentes (*Resources, Plug-in Manager, Logger e Settings*) serão descritos no ponto seguinte. Estes componentes estão numa situação intermédia já que para além de realizarem processamento da camada de dados têm também funcionalidades pertencentes à camada de lógica de negócio.

Dados – Ficheiros

Os dados utilizados pela a aplicação estão maioritariamente contidos em ficheiros. Embora estes ficheiros não representem uma camada em si, estão aqui presentes para facilitar a compreensão da estrutura da aplicação.

5.1.3 Diagrama de Classes

O cerne da *framework* tem a seguinte estrutura relacional.

Neste diagrama de classes podem-se ver os subsistemas referidos na divisão de camadas da arquitectura lógica. Aqui deverá referir-se que as classes *Settings* pertence ao subsistema *Settings*. Além disso a classe *Logger* e *FileLogger* estão incluídas no subsistema *Logger*.

As classes *PlugIn*, *PlugInDescriptor*, *PlugInId*, *StandardPlugIn* e *PlugInManager* pertencem ao subsistema *Plug-Ins*.

Por fim a classe *Resources* corresponde ao subsistema *Resources*.

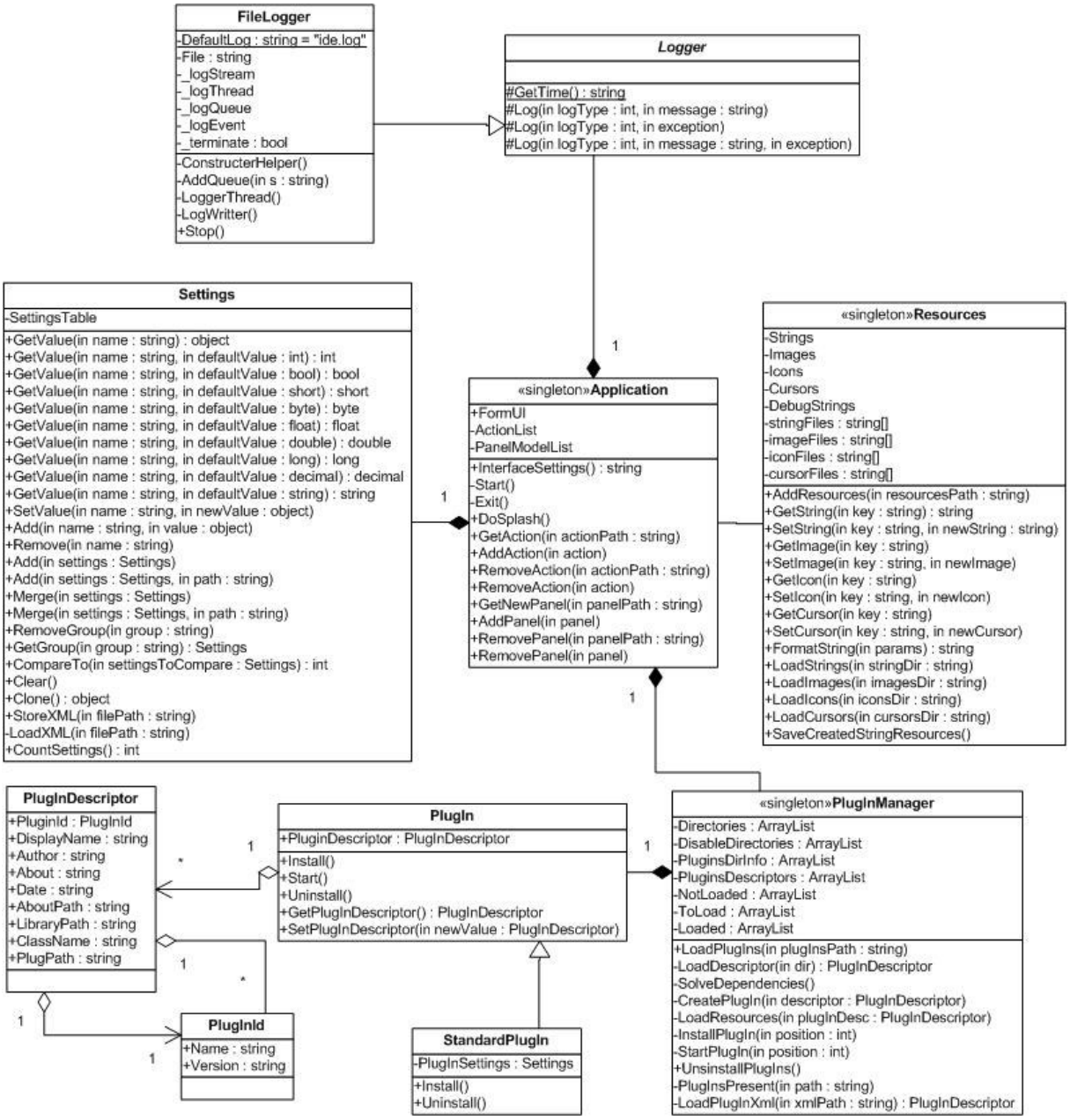

Figura 11 – Diagrama de classes

Application

Esta é a classe principal, tendo em conta que é a partir desta que todo a *framework* arranca. É composta por vários componentes, tal como *Settings* ou *FileLogger*.

Settings

Esta classe é responsável pelo tratamento de informação relevante a configurações. É aqui que estão presentes os métodos que permitem aceder a configurações, alterá-las ou remove-las.

PlugInManager

Esta é a classe responsável pela gestão dos *plug-ins*. É ela responsável por verificar os *plug-ins* presentes, os que deverão ser inicializados e os que não poderão, algo que dependerá destes estarem desactivados ou suas dependências não estarem presentes.

PlugIn

A classe *PlugIn* define-se como a super classe de todos os plug-ins, isto é, um *plug-in* terá de ter presentes métodos referidos nesta classe para poder ser invocado, como a presença de métodos *Install* ou *Uninstall*.

StandardPlugIn

A classe *StandardPlugIn* estende a classe *PlugIn* com o intuito de permitir que um *plug-in* tenha presentes as suas próprias definições, que serão posteriormente adicionadas às definições da aplicação.

PlugInDescriptor

PlugInDescriptor permite guardar toda a informação respeitante a um *plug-in*. Esta informação passa pela informação de autor, dependências, localização da biblioteca de arranque, entre outras. A única informação respeitante a um *plug-in* que não é aqui guardada é o nome (identificador) e a versão que estão guardados noutra classe.

PlugInId

A classe *PlugInId* guarda o identificador do *plug-in*, ou seja, guarda o seu nome e a sua versão. Com este par qualquer *plug-in* pode ser identificado, dado que o nome será único. A versão está presente dado que caso contrário podiam-se requerer o uso de funcionalidades ainda/já não presentes em diferentes versões do mesmo *plug-in*.

Logger

Esta interface serve de molde aos vários tipos de *loggers*. Cada *logger* alterará os métodos aqui presentes (polimorficamente) para servir os propósitos necessários.

FileLogger

Esta classe trata de realizar o *logging* de mensagens gravando as mensagens num ficheiro de texto.

Resources

A classe *Resources* serve para manipular os diferentes recursos presentes na *framework*. Aqui estão disponibilizadas ferramentas para manipular esses mesmos recursos.

5.1.4 Processos importantes

Carregamento de Módulos (*Plug-ins*)

Este é dos processos mais complexos existentes na plataforma. Envolve pesquisa dos plug-ins existentes e seu carregamento ordenado, tendo em conta as suas dependências. Este processo pode ser visto com maior pormenor na imagem seguinte. Pode ser também dividido em sub-processos que serão descritos nos pontos seguintes.

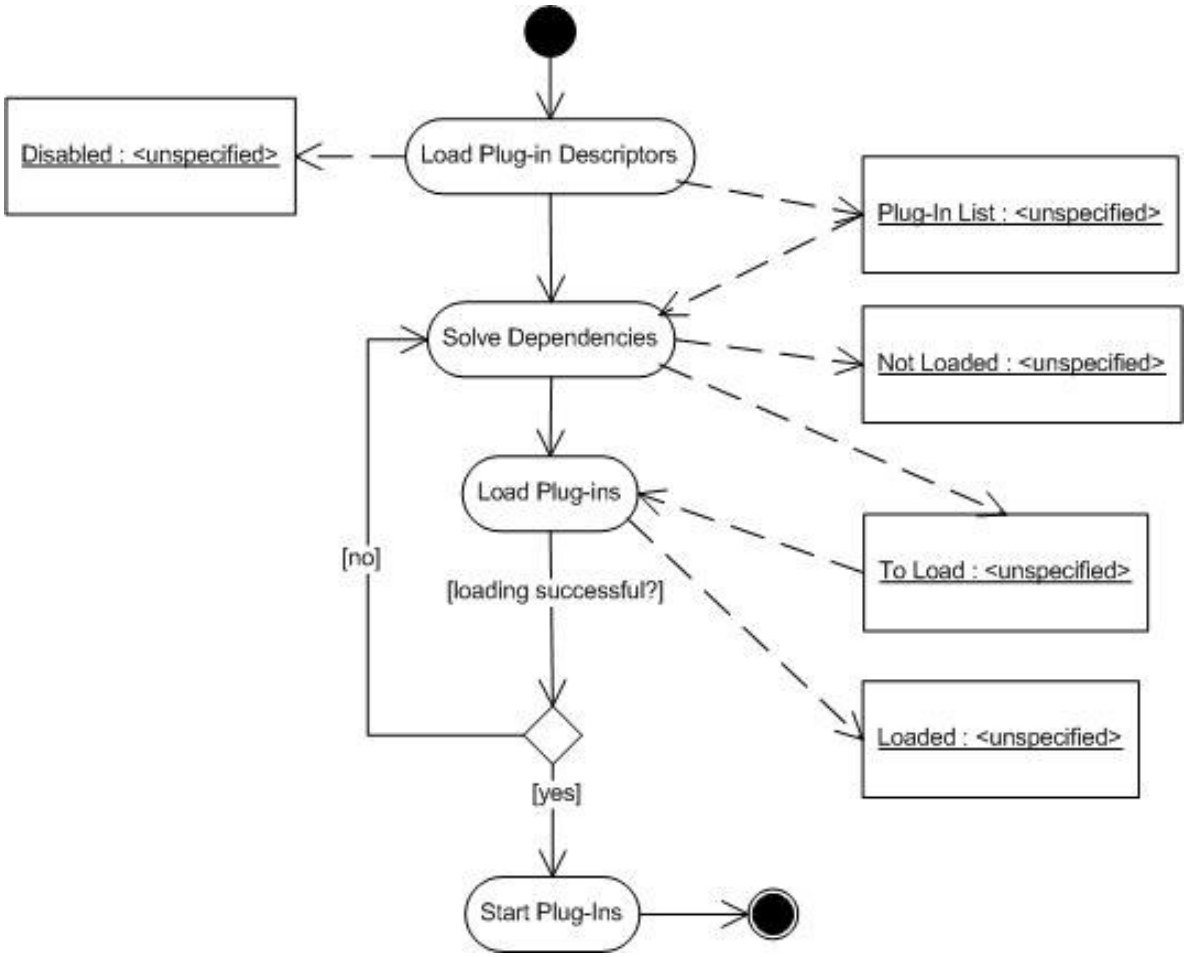

Figura 12 – Carregamento de módulos (Plug-ins).

Carregamento de Descritores

Os descritores contêm toda a informação respeitante a um módulo. Logo, para os carregar ordenadamente é necessário saber a descrição e necessidades do mesmo antes de se proceder a esse carregamento. O processo em si tem o aspecto da Figura 13.

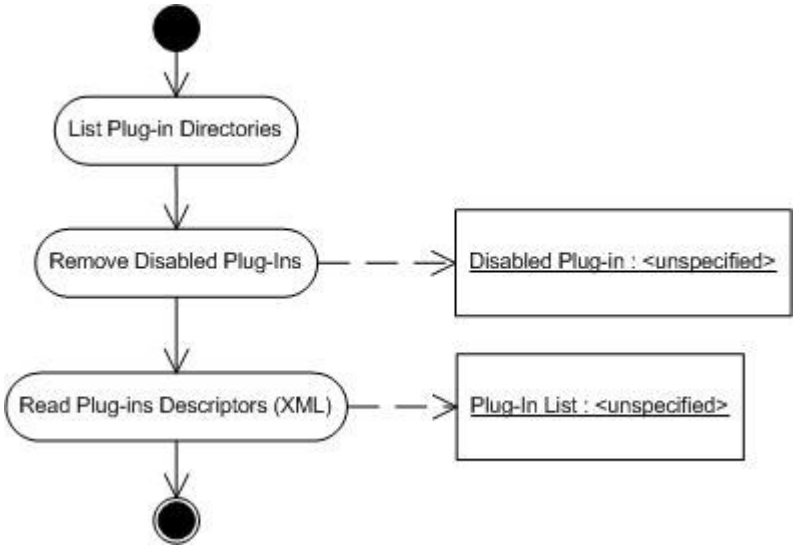

Figura 13 – Carregamento de descritores.

Verificação de Dependências

Após o carregamento dos descritores dos módulos (plug-ins) é necessário verificar as suas dependências, se estas existem ou não e tratar de as ordenar de modo a que não ocorram problemas. Tendo isso em mente o processo de verificação de dependências deve ser o seguinte.

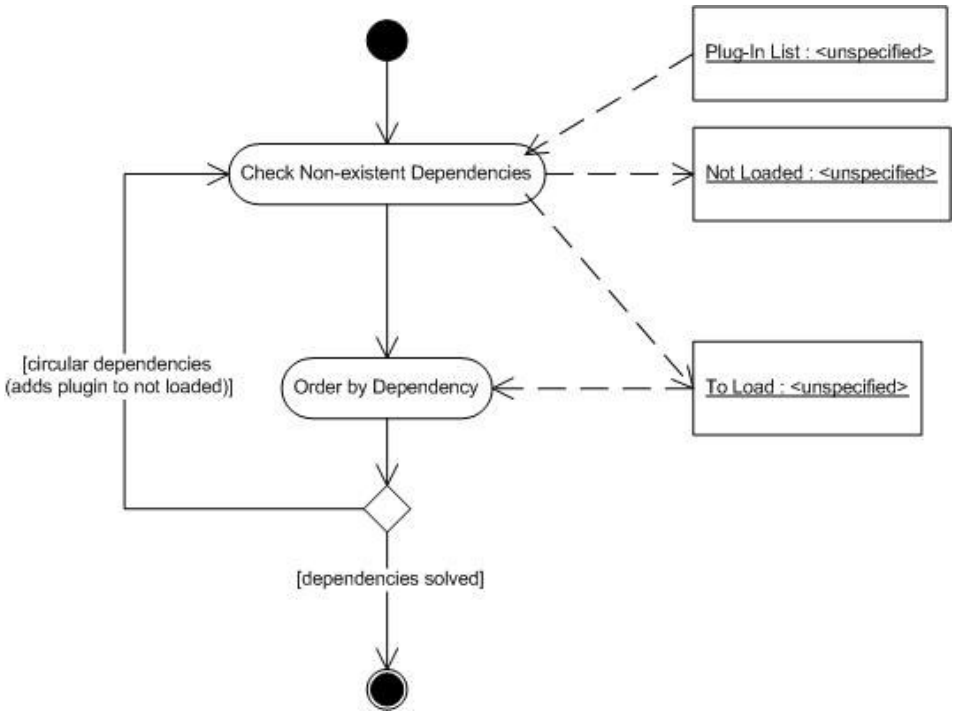

Figura 14 – Verificação de dependências.

Carregamento de Bibliotecas de Módulos (*Plug-ins*)

Neste passo é que é realizado o carregamento de módulos. É o único processo que tem duas saídas. Isto acontece por um motivo simples, caso um módulo não seja carregado com sucesso é necessário voltar a verificar dependências dado que podem haver outros módulos a depender do que não foi carregado com sucesso. Isso leva a que este processo tome o seguinte aspecto.

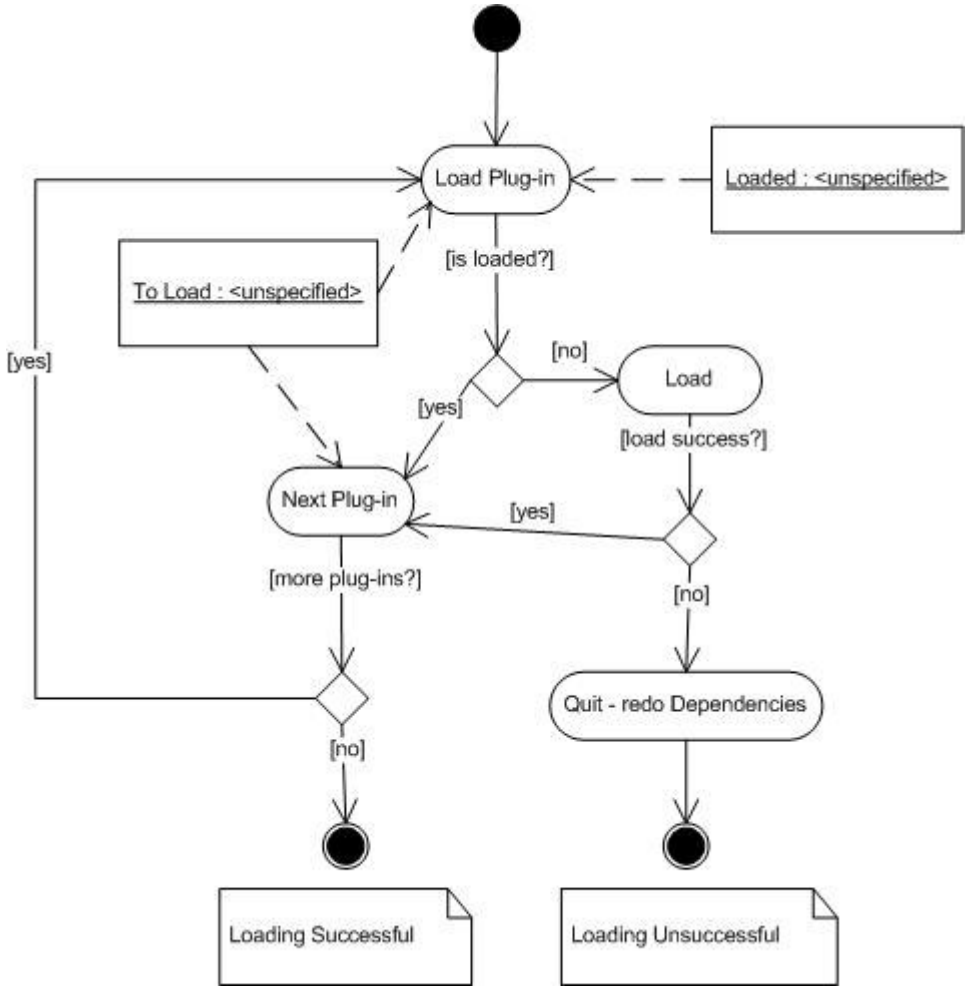

Figura 15 – Carregamento de bibliotecas de módulos (Plug-ins).

Arranque de Módulos (*Plug-ins*)

Por fim, e após os módulos serem carregados, é necessário arrancá-los, algo que é realizado neste processo.

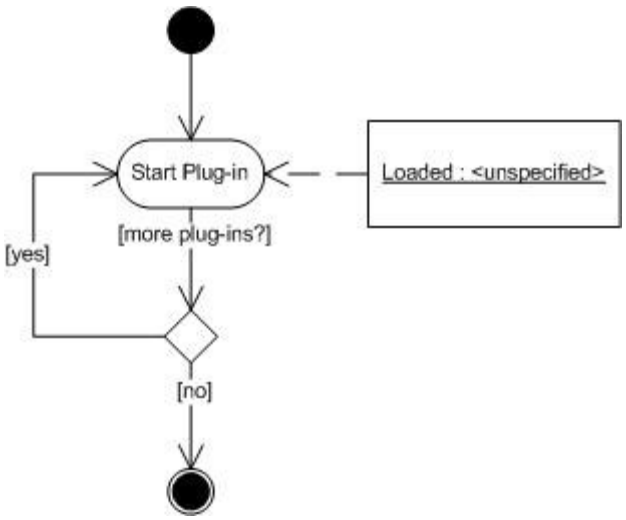

Figura 16 – Arranque de módulos (Plug-ins).

5.2 Fase 2 – Criação de Componentes de Visualização

5.2.1 Perspectiva Global

A segunda fase do projecto tinha em vista a criação de componentes de visualização, isto é, para cada módulo carregado deveria ser também criados os vários componentes gráficos a ele ligados. Estes componentes podem ser menus ou barras de ferramentas com respectivos botões ou outras funcionalidades. Como a presença de vários ambientes de trabalho teve de ser contemplada a criação uma estrutura de dados que contivesse a informação de cada ambiente de trabalho para facilitar comutações entre estes. Os vários elementos de visualização são criados quando um determinado ambiente de trabalho é carregado, fazendo-se uso da estrutura existente para o caso.

5.2.2 Diagrama de Classes

A estrutura do ambiente gráfico foi dividida em duas partes. Uma delas representa a estrutura de componentes de visualização e a outra os componentes que estão a ser visualizados num determinado momento. Esta divisão foi realizada por um simples motivo. Como os vários componentes de visualização podem ser em número considerável há que melhorar a performance. Para isso evitou-se ter em memória todos os componentes existentes, tendo unicamente os que estão a ser visualizados. Os que não estão, são guardados em objectos mais simples, como simples strings, que na altura de mudança dos componentes a ser visualizados são construídos e destruídos conforme as necessidades.

5.2.3 Estrutura de Dados

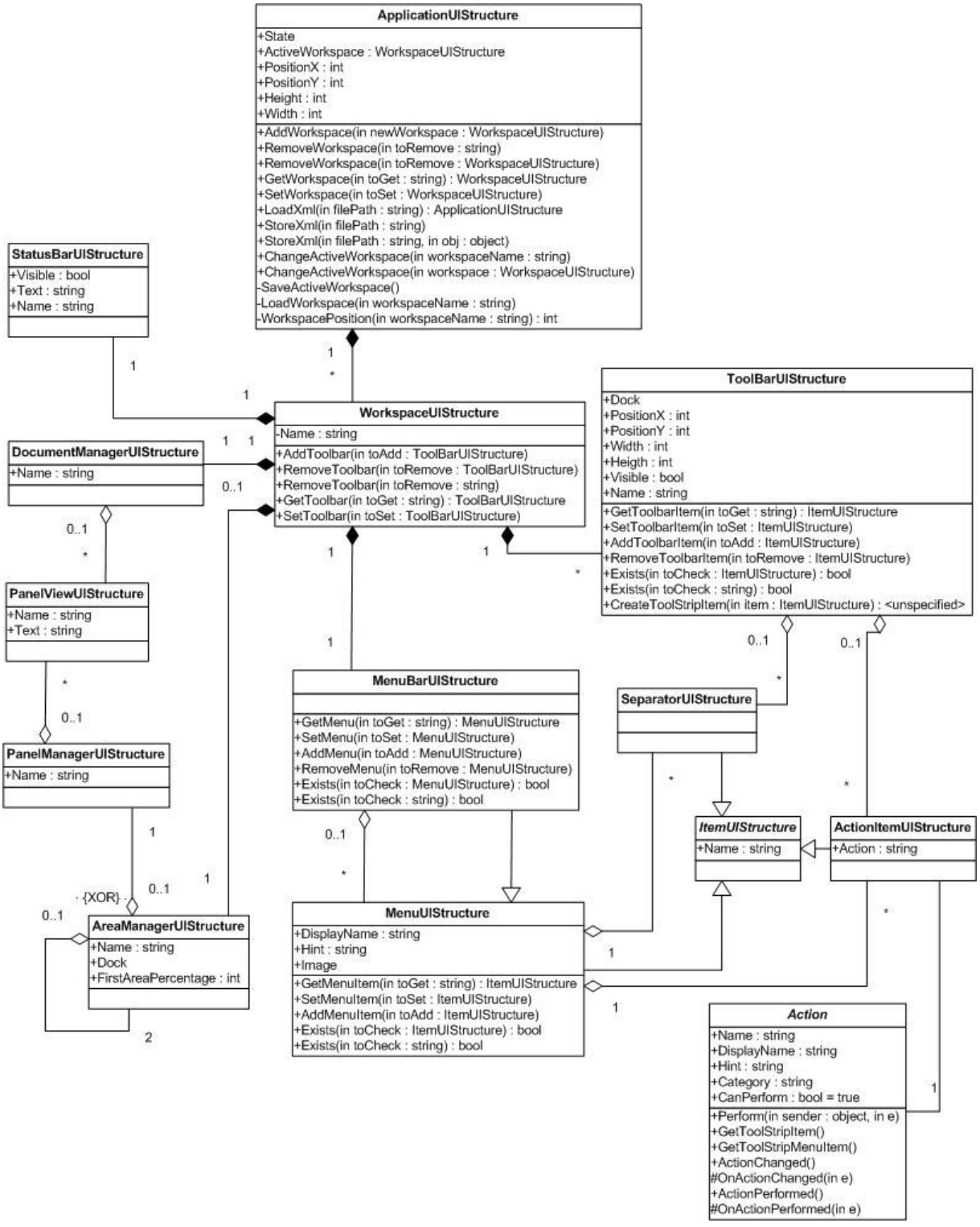

Figura 17 – Diagrama de classes da estrutura de dados.

ApplicationUIStructure

Esta classe contém a informação da aplicação relativa aos componentes de visualização. É composta por vários elementos como ambientes de trabalho existentes, o corrente ou posições da janela.

WorkspaceUIStructure

Esta classe equivale a um ambiente de trabalho. Um *WorkspaceUIStructure* tem presentes a barra de menus, barras de ferramentas e barra de estado, assim como painéis de visualização. Estes vários elementos podem variar bastante tendo em conta o ambiente de trabalho.

MenuBarUIStructure

A barra de menus é representada por esta classe, a qual define uma barra semelhante às presentes na maioria dos programas. Esta barra pode ser composta por vários menus.

MenuUIStructure

Um *MenuUIStructure* define um menu, componente de uma barra de menus. Estes menus são análogos aos que normalmente estão presentes na maioria das aplicações. Um menu pode ser composto por menus ou por acções. Um menu poderá ter textos de ajuda e uma imagens associados.

ToolBarUIStructure

Esta classe contém a informação de uma barra de ferramentas. Essa informação passa pela sua posição, visibilidade e itens que a compõem.

ItemUIStructure

ItemUIStructure é uma classe abstracta que serve de base para estruturas como menus, separadores ou acções. Com esta classe procura-se reunir propriedades semelhantes a todas as suas subclasses para facilitar o manuseamento das componentes de visualização.

SeparatorUIStructure

Com esta classe representa-se um separador, estes separadores podem ser utilizados quer em menus que em barras de ferramentas.

ActionItemUIStructure

Esta classe contém uma ligação para uma acção, descrita seguidamente. Estas acções provêem dos módulos presentes e fornecem variadas funcionalidades. Esta estrutura representa item que podem fazer parte quer de menus quer de barras de ferramentas.

Action

Uma acção (action) é uma classe abstracta que é estendida nos módulos sempre que se deseja ter um nova funcionalidade. Uma acção tem métodos que permitam criar novos objectos para colocar em menus e barras de ferramentas e ao mesmo tempo deverá ter um método *Perform* que é chamado sempre que esta é chamada. Cada um dos objectos *Action* é único (*Singleton*) e está contido numa estrutura própria fazendo parte da classe *Application*, e é carregado no arranque da *framework*.

StatusBarUIStructure

A classe *StatusBarUI* representa a barra de estado, esta pode estar visível ou não e durante a execução da aplicação permite a visualização de mensagens.

AreaManagerUIStructure

Esta classe representa uma área que será visualizada na zona central da aplicação. Pode-se comparar a zonas laterais presentes em aplicações como o *Windows Explorer*, na qual é

visualizada informação complementar. Esta área pode ser composta por subáreas (e essas por outras subáreas e assim sucessivamente) ou por painéis.

PanelManagerUIStructure

Esta classe representa um conjunto de painéis, é comparável a um gestor de *tabs* como os presentes em muitas aplicações.

PanelViewUIStructure

Esta estrutura representa um painel, que por sua vez representa uma área que será definida pelos módulos. Esta estrutura contém uma referência a um tipo de painel. Os tipos existentes estão contidos numa estrutura própria contida na classe *Application*, cada um desses tipos é acrescentado no arranque da aplicação por cada um dos módulos presentes.

DocumentViewManagerUIStructure

Esta estrutura um conjunto de painéis, sendo comparável a um gestor de *tabs*. É semelhante à classe *PanelManagerUIStructure*, no entanto esta faz parte da área central da aplicação enquanto o *PanelManagerUIStructure* faz parte das áreas laterais. Existem duas classes distintas pois podem haver diferenças no modo como cada uma delas são visualizadas, sendo uma funcionalidade a pensar no futuro da *framework*.

5.2.4 Estrutura de Componentes

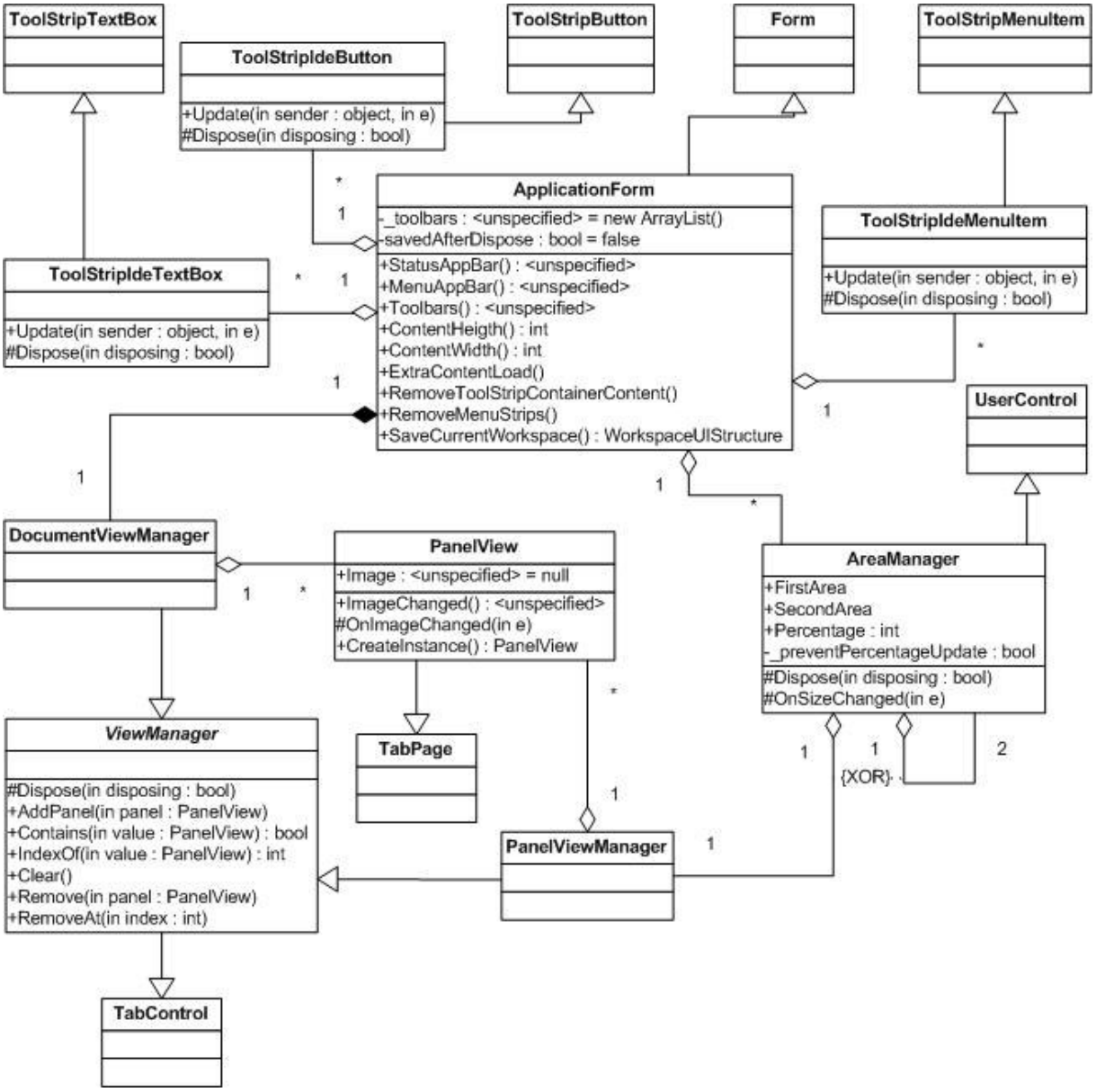

Figura 18 – Diagrama de classes dos componentes de interface

ApplicationForm

Esta classe é o *Form* que é visualizado pelo utilizador, estende a classe *Form* (System.Windows.Forms.Form). Esta interface contém todos os itens a ser visualizados, como barras de ferramentas, menus e painéis. Tem presentes métodos que permitem alterar a interface actual utilizando a informação contida na estrutura de interfaces descrita anteriormente.

ToolStripIdeButton

Esta classe representa um botão que pode ser adicionado a uma barra de ferramentas, sendo uma extensão da classe *ToolStripButton* (System.Windows.Forms.ToolStripButton).

ToolStripIdeMenuItem

Esta classe representa um item de um menu que pode ser adicionado a um menu, sendo uma extensão da classe *ToolStripMenuItem* (System.Windows.Forms.ToolStripMenuItem).

ToolStripIdeTextBox

Esta classe representa uma caixa de texto que pode ser adicionada a uma barra de ferramentas, sendo uma extensão da classe *ToolStripTextBox* (System.Windows.Forms.ToolStripTextBox).

Estas 3 classes são adaptações das classes já existentes na *Framework .NET 2.0*. Caso se deseje acrescentar a possibilidade de novos itens à *framework* bastará criar uma classe com estrutura semelhante a estas ultimas.

PanelView

A classe *PanelView* representa um painel. Esta classe estende a classe *TabPage* (System.Windows.Forms.TabPage), consistindo numa *tab* que pode ser acrescentada à interface.

ViewManager

A classe *ViewManager* estende a classe *TabControl* (System.Windows.Form.TabControl), representando um gestor de *tabs* que pode ser utilizado na *framework*. É no entanto uma classe abstrata que agrupa as funcionalidades usadas quer pela classe *PanelManager*, quer na classe *DocumentViewManager*.

PanelManager

Esta classe estende *ViewManager*, apesar de nesta altura não acrescentar mais funcionalidade ou parâmetros foi criada de modo a permitir futuras expansões. Consiste um gestor de *tabs* que pode ser utilizado em áreas laterais.

DocumentViewManager

Esta classe também estende *ViewManager* e é semelhante à classe *PanelManager*. Foi criada a pensar em futuras expansões da *framework*.

AreaManager

Por fim a classe *AreaManager* representa uma área lateral na interface, algo semelhante ao que se vê em muitas aplicações. Esta pode ser composta por duas subáreas ou um *PanelManager*. Deste modo confere-se bastante liberdade para a criação de interfaces.

5.3 Fase 3 – Módulo de Teste (Web Browser)

5.3.1 Perspectiva Global

O módulo de teste implementado para testar a aplicação desenvolvida foi um web browser simplificado mas que tivesse presente as várias componentes que são manuseadas durante a execução da aplicação. Com este módulo procura-se testar, em primeiro lugar as componentes de persistência de dados e de carregamento de módulos e em segundo lugar a criação dos elementos de visualização do módulo de teste. A sua estrutura é semelhante ao que será um módulo típico, ou seja, tem um núcleo que utiliza painéis e acções próprios, como visível na figura.

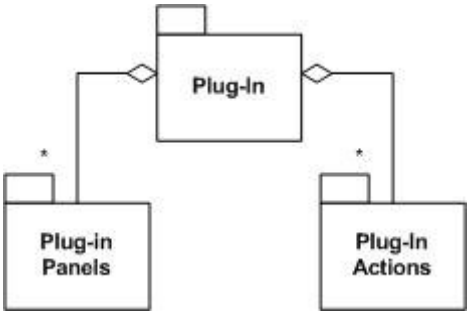

Figura 19 – Estrutura típica de um módulo

5.3.2 Painéis e Acções Presentes

O módulo de teste tem presentes as seguintes acções e painel.

- ActionBack – Navega para a página anterior.
- ActionForward – Navega para a página seguinte.
- ActionGo – Navega para a página indicada.
- ActionHelp – Mostra a ajuda do módulo.
- ActionHome – Navega para a página inicial.
- ActionRefresh – Realiza um *resfresh* à página actual.
- ActionStop – Para a navegação.
- ActionWebPath – Caixa de texto contendo os URL's
- WebBrowserPanel – Painel contendo o *web browser* propriamente dito.

6 Conclusões

6.1 Trabalho Realizado

Relativamente ao projecto pode-se dizer que este foi de encontro aos objectivos, superando-os já que foi inclusive desenvolvido um módulo para integrar a *framework* o qual não estava inicialmente planeado.

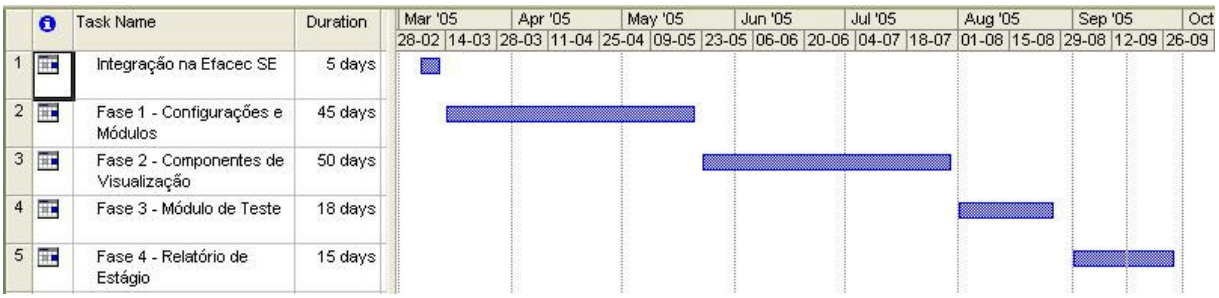

Figura 20 – Diagrama de Gantt do projecto

O projecto foi em quatro fase principais:

- **Fase 1 – Persistência de dados e gestão de módulos**

Durante esta fase foram criadas ferramentas para a gestão de módulos e configurações.

- **Fase 2 – Criação de componentes de visualização**

Esta fase resultou na criação de ferramentas de gestão de interfaces.

- **Fase 3 – Criação de módulo de teste**

O resultado desta fase foi um módulo de teste que permite a navegação na Internet.

- **Fase 4 – Relatório de estágio**

Finalmente foi elaborado o relatório de estágio e restante documentação.

6.2 Problemas Encontrados

Durante a realização deste projectos houve, como seria de esperar, alguns problemas que dificultaram, uns mais e outros menos, todo o desenvolvimento.

Um dos problemas resultou do uso do Visual Studio .NET 2005 Beta. A versão utilizada no início do estágio foi a beta 1 que padecia da presença de inúmeros bugs, o que dificultou o trabalho com esse ambiente de desenvolvimento. Essas dificuldades surgiram dada a presença de *bugs* e funcionalidades por desenvolver, ou parcialmente desenvolvidas, a erros, como era o caso de sempre que se fechava o Visual Studio aparecia uma mensagem de erro. Além disso dado que se trata de uma ferramenta beta a documentação existente é escassa, sendo difícil por vezes descobrir como trabalhar com certas funcionalidades.

6.3 Experiência Pessoal

Ao longo do período do estágio foram adquiridos conhecimentos e experiências enriquecedoras e que dificilmente se têm em meio académico.

O caso mais díspar do meio académico foi sem dúvida os horários. Apesar de na EFACEC SE haver flexibilidade de horários foi cumprido o horário normal de trabalho. Esta diferença entre horários impostos e horários à medida foi definitivamente sentida. No entanto, apesar desta imposição pode-se dizer, que ao criar hábitos de trabalho a produtividade aumenta, já que com os horários indefinidos do meio académico é mais fácil perderem-se os objectivos e a produtividade dar lugar a períodos de grande inactividade e letargia.

Outra vantagem que o ambiente empresarial teve foi o de melhorar outros pontos como a capacidade de relacionamento entre colegas de trabalho, relações essas que para além de fornecerem importantes conhecimentos técnicos deram também a conhecer outros pontos de vista e um maior alargamento de horizontes.

6.4 Perspectivas para Trabalho Futuro

Apesar do projecto ter atingido todos os seus objectivos não se pode dizer que é um projecto concluído, pelo contrário, é um projecto que está agora a começar, isto porque a *framework* desenvolvida é só a base para a ferramenta que se planeia ter, a qual permitirá configurar uma série de dispositivos de uma subestação.

No entanto, não referindo os módulos que serão desenvolvidos, ainda há alguns pontos que podem ser melhorados.

Um protótipo pode ter alguns problemas e funcionalidades por desenvolver já que o seu objectivo é provar um conceito. No entanto um produto já implica um nível superior de qualidade o qual requer estabilidade e uma gama completa de funcionalidades. Assim um trabalho futuro seria a transformação da *framework* criada de um protótipo para um produto. Neste ponto está incluída a implementação de mais testes unitários para testar toda a aplicação, e a criação de novas funcionalidades não presentes na versão actual.

Para além da *framework* há obviamente o interesse de desenvolver módulos que realizem as funções desejadas.

Referências e Bibliografia

[Sommerville 2000] I. Sommerville, Software Engineering, 6th Edition, Addison-Wesley (2000)

[Robinson 2003] S. Robinson, Professional C#, 2nd Edition, Wrox (2003)

[TDD 2005] Test Driven Development (Junho 2005), site: http://www.testdriven.com/

[CP 2005] Code Project (Setembro 2005), site: http://www.codeproject.com/

[W3S 2005] W3 Schools (Setembro 2005), site: http://www.w3schools.com/

[VSBE 2005] Visual Studio 2005 Beta Experience (Agosto 2005), site: http://www.microsoft.com/emea/msdn/betaexperience/

[VSMSDN 2005] Visual Studio 2005 – Microsoft Developers Network (Setembro 2005), site: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/vs2005anchor.asp

[MSDN 2005] Microsoft Developers Network (Setembro 2005), site: http://www.msdn.com

[Smart 2005] Smart Client Developer Center: Features (Julho 2005), site: http://msdn.microsoft.com/smartclient/understanding/windowsforms/2.0/features/default.aspx

[Eclipse 2005] Eclise (Abril 2005), site: http://www.eclipse.org/

[Java 2005] Java Technology (Abril 2005), site: http://www.sun.com/java/

[Mono 2005] Mono Project (Abril 2005), site: http://www.mono-project.com/

Glossário e Lista de Acrónimos

Windows – Sistema Operativo desenvolvido pela Microsoft.

C# – Linguagem de programação orientada ao objecto desenvolvida pela Microsoft como parte da *Framework .NET*.

Java – Linguagem de programação orientada ao objecto desenvolvida pela Sun MicroSystems.

IDE – Integrated Development Environment

Eclipse – Plataforma de desenvolvimento de software Open Source para desenvolvimento de aplicações ricas, tipicamente para desenvolvimento em Java.

Visual Studio – Plataforma de desenvolvimento criada pela Microsoft para desenvolvimento de aplicações em linguagens da *Framework .NET*, tal como C++, C#, J# ou Visual Basic e outras.

Bug – Comportamento indesejado e uma aplicação informática, também visto como uma falha ou defeito.

Junit – *Framework* para realização de testes unitários realizados em linguagem Java.

Nunit – *Framework* para realização de testes unitários realizados em linguagens .NET.

Framework .NET – Plataforma de desenvolvimento criada pela Microsoft tendo em vista o desenvolvimento rápido e poderoso de aplicações.

Beta (versão) – Indica uma aplicação que ainda está em fase de desenvolvimento. As principais funcionalidades estão desenvolvidas, tendo como objectivo a pesquisa e remoção de bugs.

Modulo – Componente que confere funcionalidades acrescidas a uma *framework* base. Deve seguir linhas orientadoras de modo a estar em conformidade com a plataforma com que se vai integrar.

Test Driven Development – Técnica de programação baseada em *Extreme Programming*. Essencialmente consiste em criar um teste e posteriormente implementar o código de modo a passar no teste criado.

Anexo A – Regras relativas a componente criados (de modo a serem compatíveis)

Componentes

Um componente representa um conjunto de funcionalidades que são desenvolvidas, por sua vez cada componente poderá ser constituído por um ou vários módulos. Um componente representa a nível de programação uma solução, enquanto um módulo é um projecto. Cara um destes módulos após a compilação representa uma biblioteca.

Procedimento

O desenvolvimento de um componente deve ser realizado numa metodologia de programação de *Test Driven Development*, isto é, ao longo do desenvolvimento devem ser realizados testes unitários. Os testes unitários representam um módulo de desenvolvimento de um componente, e como tal têm o seu próprio projecto e consequentemente a sua própria biblioteca. Programaticamente os ficheiros fonte de teste devem ter o nome da classe sobre a qual realizam os testes seguido de *Test*, isto é, [*nome da classe*]Test.

O processo formal de programação deve seguir os seguintes passos:

- Fazer o esqueleto do método
- Criar teste (a predefinição é ser um teste inconclusivo)
- Alterar o teste (de modo a realizar o desejado)
- Correr o teste novamente (que deve falhar, já que já está alterado)
- Implementar a função
- Repetir o teste (que deve passar, caso contrário deve-se corrigir a função)

A compilação bem sucedida de um componente é constituída por 3 passos:

- Compilação do código sem erros
- Execução dos testes unitários sem erros
- Cópia dos ficheiros de output para as respectivas directorias

Estrutura de Directórios

As directorias relativas ao desenvolvimento de componentes devem ter o seguinte aspecto. Há duas directorias principais, *components*, que contém o código que está a ser desenvolvido, e *output*, que contém as componentes após compilação.

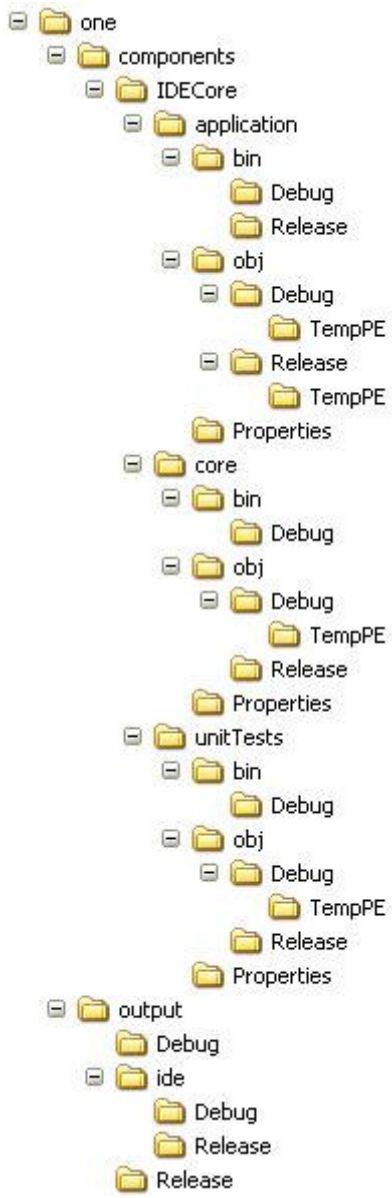

Figura 21 - Estrutura de directorias.

A directoria *components* contém as várias directorias respeitantes a cada componente. Dentro dessa directoria existe uma subdirectoria por cada módulo a ser desenvolvido. Além dos vários módulos que se desejam desenvolver deverá existir um módulo que tem o nome *unitTests* que contém os testes unitários do componente. As subdirectorias de cada módulo são as standards criadas pelo Visual Studio .NET 2005 (como se pode ver pela figura).

A directoria output contém 3 directorias, *Debug*, *Release* e *IDE*. As duas primeiras contêm todas as bibliotecas correctamente compiladas, tendo em conta a sua configuração (*Debug* ou *Release*), ou seja uma colecção de ficheiros *dll* e *exe*. A directoria IDE contém a plataforma One com a sua core e plug-ins existentes a qual será explicada na secção de plug-ins.

Opcções do Visual Studio .NET 2005

Este ponto serve para descrever como os pontos já referidos se adaptam ao Visual Studio .NET 2005.

Testes

Os testes podem ser realizados utilizando a ferramenta para esse fim embutida no Visual Studio .NET 2005. Para isso deve-se aceder ao menu Tests.

Compilação

A compilação deve incluir a cópia dos ficheiros de output para a directoria de output, como já referido, para isso devem-se utilizar passos pós compilação (post-build steps) que deverão sempre que possível utilizar as macros disponíveis e também caminhos relativos. Caminhos absolutos nunca devem ser usados. Um exemplo destes tipo de passos pode ser:

```
copy $(TarguetFileName) ..\..\..\..\..\output\Debug
```

Programação de Plug-ins na Plataforma One

Plug-ins

Um plug-in é um componente particular, pelo que no seu desenvolvimento deverão ser observadas as regras definidas no para os componentes.

Procedimento

O procedimento é semelhante ao procedimento descrito para os componentes.

Estrutura de Directórios

O esquema de directorias é o mesmo do referido para os componentes, com elementos adicionais. Dado que um plug-in compreende um conjunto adicional de elementos (como recursos) estes devem estar presentes numa subdirectoria da directoria base do componente, denominada *pulgIn*.

Além disso a directoria output\ide contém uma versão de teste da *framework*, isto é, contém o core da *framework* e deverá conter os plug-ins desenvolvidos. A estrutura interna dos directórios que deve ser usada está também documentada no documento acima referido.

Quando se realizar a compilação de um plug-in, para além dos passos tomados para um componente, dever-se-á copiar os ficheiros de output e de recursos para a directoria *output\ide\debug* ou *release* (conforme a configuração) seguido a estrutura já definida.

Opções do Visual Studio .NET 2005

Os procedimentos descritos no ponto anteriormente devem ser seguidos, acrescento além disso passos pós compilação para colocar os ficheiros necessários na directoria *output\ide*. A colocação de ficheiros deve ser feita em 3 passos:

- Apagar directoria de destino (eliminando os ficheiros antigos)
- Criar a directoria
- Copiar todos os ficheiros para a directoria e subdirectorias correspondentes

Regras de Nomes

Os módulos criados devem ser nomeados Efacec.[*nome do componente*].[*nome do módulo*].dll (ou exe). Caso o componente tenha um único módulo o nome deverá ser Efacec.[*nome do componente*].dll ou exe.

Para o caso dos plug-ins os nome devem ser Efacec.[*nome do plug-in*].[*nome do módulo*].PlugIn.dll ou exe. Caso o plug-in tenha um único módulo o nome deverá ser Efacec.[*nome do plug-in*].PlugIn.dll ou exe.

Ligação Entre Estruturas

As estruturas referidas neste relatório, componentes, módulos, plug-ins e plataforma One estão todas interligadas. O diagrama de classes UML da figura seguinte representa o modelo conceptual apresentado neste documento.

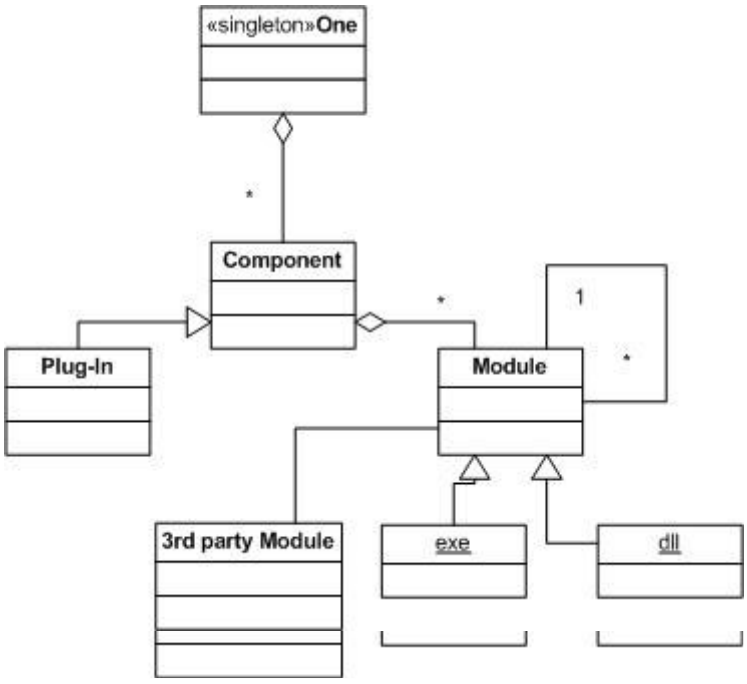

Figura 22 - Estruturas na framework.

Por outro lado a ligação entre directorias, componentes desenvolvidas e conceitos da plataforma One é apresentada no diagrama de classes UML da figura seguinte:

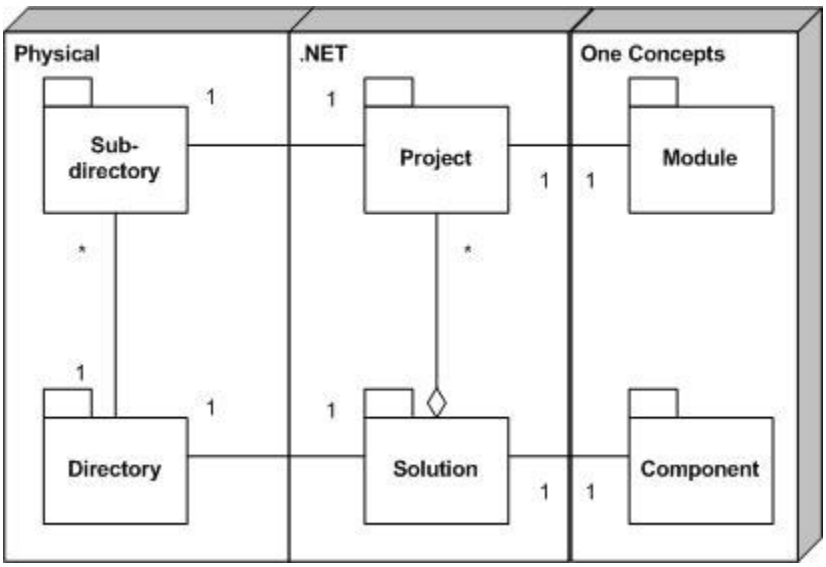

Figura 23 - Ligação entre Dimensões

Anexo B – Outros documentos

Durante o período de estágio foi produzida alguma documentação de modo a melhor definir o que foi produzido. Estes documentos não estão anexados a este relatório já que a maior para da informação que contém está também aqui presente, tornando-os redundantes. Seguidamente é apresentada a lista desses documentos.

- Especificação de Requisitos da Framework para Ambientes de Projecto
- Desenho de Alto Nível para Framework
- Procedimento para Desenvolvimento de Componentes

Anexo C – Web Browser de Teste

De modo a testar e verificar o modo de funcionamento das novas funcionalidades na *Framework .NET 2.0* foi criado um web browser de teste, algo que não era necessário, mas para além de permitir conhecer o que de novo havia na *Framework* permitiu ter uma simples e funcional alternativa a browsers como o Internet Explorer ou o Firefox.

O foco dos testes passou principalmente pelas barras de ferramentas. Este teste de funcionalidades resultou numa aplicação simples e funcional como se pode ver pela figura seguinte.

Figura 24 – Web browser de teste.

Anexo D – Interfaces

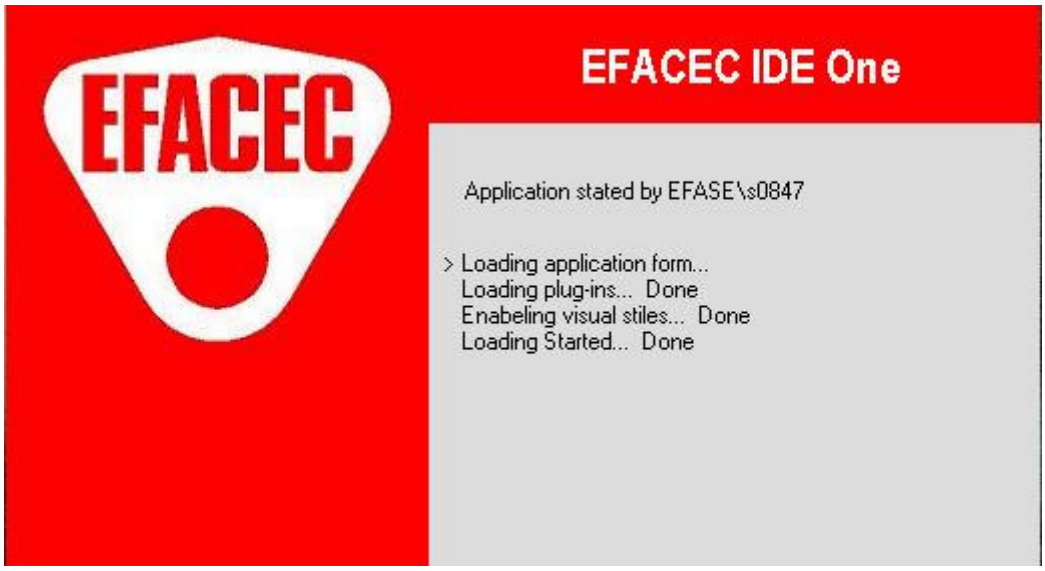

Figura 25 – Splash screen da Framework

Figura 26 – Interface criado pela framework a partir do módulo de teste