

Faculdade de Engenharia da Universidade do Porto



FEUP

Implementação de ferramenta genérica para suporte de diversas bases de dados de medicamentos

Diogo Pedro Pinto Coelho

Relatório de Projecto realizado no Âmbito do
Mestrado Integrado em Engenharia Informática

Orientador: Prof. António Fernando Vasconcelos Cunha Castro Coelho

Julho de 2008

Diogo Pedro Pinto Coelho©

Implementação de ferramenta genérica para suporte de diversas bases de dados de medicamentos

Diogo Pedro Pinto Coelho

Relatório de Projecto realizado(a) no Âmbito do
Mestrado Integrado em Engenharia Informática

Aprovado em provas públicas pelo Júri:

Presidente: Maria Cristina de Carvalho Alves Ribeiro (Prof.)

Arguente: Álvaro Rocha (Prof.)

Vogal: António Fernando Vasconcelos Cunha Castro Coelho (Prof.)

31 de Julho de 2008

Aos meus pais...

Termos Legais

Nos termos do protocolo de estágio e do acordo de confidencialidade celebrado com a ALERT Life Sciences Computing, S.A. ("ALERT"), o presente relatório é confidencial e poderá conter referências a invenções, *know-how*, desenhos, programas de computador, segredos comerciais, produtos, fórmulas, métodos, planos, especificações, projectos, dados ou obras abrangidos por direitos de propriedade industrial e/ou intelectual da ALERT. Este relatório só poderá ser utilizado para efeitos de investigação e de ensino. Qualquer outro tipo de utilização está sujeita a autorização prévia e por escrito da ALERT.

Resumo

O trabalho apresentado surge no contexto de um cenário real e bastante frequente em diversas empresas de desenvolvimento de software. Quando se fala em aplicações de larga escala que envolvem várias tecnologias e camadas de desenvolvimento, há que considerar os problemas que podem resultar da programação paralela das várias camadas. Em muitos casos, a má distribuição de tarefas, ou o desequilíbrio de carga de trabalho entre diferentes camadas, pode originar tempos de espera derivados das dependências. É precisamente esta falha, que está na base deste trabalho.

Transportando este cenário para o caso de estudo, considere-se a ALERT Life Sciences Computing, S.A. A Alert é uma empresa de desenvolvimento de software clínico, que apresenta uma boa variedade de tecnologias e uma arquitectura composta por 3 camadas: *user interface*, *java interface* ou *application/transaction* e dados ou *business logic*, sendo que a segunda camada funciona como ponte de comunicação entre as outras duas. No contexto de uma das equipas da empresa, a equipa de medicação/prescrição, o problema aplica-se devido a uma forte sobrecarga das equipas de bases de dados, que se traduz em tempos alargados de espera pela parte da camada de *user interface*, resultando em falhas de eficiência.

Assim sendo, o projecto apresentado foca primariamente na optimização de todo um processo de desenvolvimento com o objectivo de promover uma certa independência entre as camadas, reduzindo tempos de espera e perdas de eficiência. Nesse sentido é procurada a solução mais adaptada ao contexto apresentado, embora a metodologia genérica possa ser aplicada em cenários de natureza semelhante, com óbvias adaptações a nível da implementação.

O trabalho encontra-se assim dividido em duas etapas essenciais. Uma primeira fase em que o autor descreve a problemática em si, procurando uma solução para o problema e uma segunda fase que corresponde à aplicação prática desta mesma solução, segundo uma metodologia detalhada mais adiante, sobre o caso de estudo apresentado, no âmbito de uma empresa real. Como solução, foi concebido um pequeno sistema de simulação/teste que, substituindo o acesso à base de dados por chamadas a ficheiros XML, garante uma maior autonomia às equipas de desenvolvimento de *user interface*. Em adição, esta ferramenta possibilita ainda a geração automática de documentação a partir dos ficheiros XML, como documentos de especificação de interface entre as camadas de *user interface* e dados.

Como é descrito mais adiante, esta solução fornece um serviço simples, transparente e bastante cómodo, por ser uma solução altamente integrada.

Abstract

This project comes with a real very usual problem among many of the software development companies around the world. When it comes to large-scale applications, involving many technologies, it's important to consider tiered architectures and the problems that may result from the peer programming. In many cases, the workload difference between these tiers is strong and considering the functional dependence that bind them, this may lead to low effectiveness. The presented project will precisely work on this problematic.

Considering this, let's focus in ALERT Life Sciences Computing, SA. Alert is a software development company, which presents a good variety of technologies and a 3 tiered architecture: user interface, java interface or application / transaction or business logic and data, where the second layer serves as the communication gate between the other two. In the medication / prescription team, this problem occurs due to a work overload in the data/business logic tier, which leads to extended periods of waiting for the *user interface* developers. This leads to a low effectiveness that may affect the final result.

Therefore, the project focuses primarily on the optimization of a whole development process with the aim of promoting the independence between the development tiers, reducing waiting times and loss of effectiveness. So, the goal is to find a solution that fits this study case.

This work is divided into two essential steps. A first step describes the problem itself, looking for a solution that fits this case, and a second phase refers to the practical application of this solution, according to a previously specified methodology. The result was a small simulation / test system, which replaces the remote access to the database for XML files, ensuring greater autonomy for the *user interface* developers. In addition, this tool also allow the automatic document generation, using the same XML files, in order to specify the interface between the of *user interface* and *data/business logic* tier.

As described below, this solution provides a simple and transparent service, due to its high level of integration.

Agradecimentos

A realização deste trabalho revestiu-se de um significado muito relevante para mim, não só pela sua natureza e por marcar o final de uma etapa decisiva da minha vida como, também, por assinalar o início da minha vida profissional. Se alcancei este patamar foi porque tive sempre quem me apoiasse, pois a caminhada da vida de um Homem não pode ser feita sozinha.

Devo expressar um agradecimento muito especial ao Professor António Coelho, orientador dedicado e atento, que me apontou sempre a direcção certa a seguir, por ter sabido motivar o que de melhor há em mim.

À ALERT agradeço reconhecidamente a oportunidade que me facultou para concretizar este projecto.

Aos colegas da ALERT, que tão bem me acolheram, deixo aqui um sincero obrigado pela permanente solicitude e prontidão para me ajudarem sempre que necessário. Foi um prazer trabalhar durante estas 20 semanas num ambiente tão agradável, onde todos os dias se batalha arduamente com um sorriso nos lábios.

Agradeço ao Orlando por todo o esforço, apoio e dedicação, que foram determinantes para o sucesso deste trabalho. Foi um líder de equipa exemplar, optando sempre por soluções de excelência. Agradeço, também, aos meus colegas de equipa Patrícia, Mauro e Rafael por toda a ajuda e pelo apoio prestados e, em especial, ao Filipe e ao Ricardo, com quem muito aprendi, por me facilitarem a integração.

Quero agradecer, igualmente, a todos os meus colegas estagiários, com um enfase particular para os meus amigos Fábio e Nelson, por todo o encorajamento que me deram. Este trabalho não poderia ter sido levado a bom porto sem o contributo das pessoas que sempre estiveram a meu lado durante todo o meu percurso escolar e académico. Devo, por isso, manifestar o meu agradecimento aos meus pais pela educação que me deram e por me terem facultado as condições necessárias ao desenvolvimento do meu esforço. Sem eles, nada teria sido possível. Agradeço, naturalmente, à minha irmã Sara com quem sei que posso sempre contar.

À minha namorada, Ana Sofia, obrigado por ter estado sempre ao meu lado e pelo amor, carinho, amizade e suporte que constituíram um pilar para enfrentar todos os desafios.

Quero agradecer, ainda, a toda a minha vasta família. Estão todos no meu coração.

Estendo, por fim, os agradecimentos a todos os demais meus amigos que sempre acreditaram em mim.

A todos um MUITO OBRIGADO

Conteúdo

1. Introdução	1
1.1 O Projecto	1
1.1.1 Motivação	1
1.1.2 Objectivos	2
1.2 A empresa	3
1.2.1 As 3 camadas de desenvolvimento	3
1.2.2 Organização Interna – A equipa de medicação/prescrição	4
1.3 Planeamento do projecto	5
1.4 Estrutura do documento	6
2. Revisão Bibliográfica	7
2.1 Estado da arte: Estudos no domínio	7
2.1.1 Testes unitários e Mock Objects	7
2.1.2 Conceitos na área da saúde: Prescrição Médica	9
2.2 Revisão Tecnológica	11
2.2.1 Testes unitários: Frameworks existentes	11
2.2.2 Armazenamento de dados	14
2.2.3 Geração de documentos	17
2.2.4 Flash VS Flex	18
2.2.5 XML e Flash: As limitações do Flash	20
3. Ferramenta Genérica para Suporte de Diversas Bases de Dados de Medicamentos	21
3.1 Análise do problema	21
3.1.1 Processo de análise de requisitos	23
3.2 Requisitos	31
3.3 Solução tecnológica	34
3.3.1 Arquitectura	37
3.4 Metodologia a aplicar	39
4. Implementação da solução	41
4.1 Estudo do sistema e impacto da solução	41
4.2 Definição da estrutura de dados	45
4.2.1 Tipos de dados	45
4.2.2 Estrutura dos ficheiros XML	46
4.3 Geração de documentos	48
4.4 Aplicação da metodologia ao desenvolvimento de funcionalidades internas	
ALERT Prescription 2.4.3	49
4.4.1 Especificação das funcional	49
4.4.2 Implementação da funcionalidade Co-sign	50
4.4.3 Adaptação dos ecrãs de medicação e de prescrição	51
4.4.4 Desenvolvimento do novo componente de pesquisa	55
5. Conclusões	59
5.1 Avaliação dos resultados	59
5.2 Conclusões	60
5.3 Perspectivas de trabalho futuro	60

5.4 Comentários sobre o desenvolvimento do projecto	62
Referências e Bibliografia	64
ANEXO A: Diagrama do Planeamento	67
ANEXO B: Data Abstraction Doc.....	69
ANEXO C: Exemplo XML Data Abstraction.....	70
ANEXO D: Data Abstraction Schema	72

Lista de Figuras

Figura 1: As 3 camadas de desenvolvimento	3
Figura 2: Enquadramento da equipa de medicação/prescrição	4
Figura 3: Planeamento do projecto	5
Figura 4: Problema da falta de autonomia.....	22
Figura 5: Problema da falta de sincronismo	23
Figura 4: Análise da arquitectura ALERT.....	25
Figura 5: Processo de desenvolvimento ALERT	29
Figura 6: Arquitectura Data Abstraction	37
Figura 7: Diagrama representativo da metodologia	40
Figura 8: Comunicação Remota	41
Figura 9: Chamadas remotas	42
Figura 10: Inicialização da classe <i>Connector</i>	43
Figura 11: Funcionamento do Data Abstraction.....	44
Figura 12: Implementação visual do co-sign.....	50
Figura 13: Aspecto antigo do ecrã principal de medicação.....	51
Figura 14: Novo aspecto do ecrã principal de medicação	52
Figura 15: Novo aspecto do ecrã principal de medicação (2)	53
Figura 16: Aplicação do Data Abstraction ao ecrã de medicação.....	54
Figura 17: componente de pesquisa - novo e antigo	55
Figura 18: Diagrama de estados - antes.....	56
Figura 19: Diagrama de estados - depois.....	57
Figura 20: Proposta de melhoramento.....	61

Lista de Tabelas

Tabela 1: Comparação de frameworks	14
Tabela 2: XML vs Ficheiros não estruturados.....	16
Tabela 3: XML vs Bases de Dados	16
Tabela 4: Flash vs Flex	19
Tabela 5: Parte I - Stakeholders.....	24
Tabela 6: Ferramentas usadas na ALERT	27
Tabela 7: Parte I - Requisitos funcionais.....	32
Tabela 8: Parte I: Requisitos não funcionais	32
Tabela 9 - Comparação de frameworks.....	35
Tabela 10: Geração de documentos - Comparação	36
Tabela 11: Valores de saída na BD (output).....	45
Tabela 12: Valores de entrada da BD (input).....	45
Tabela 13 – Componente pesquisa (antes)	56
Tabela 14 - Componente Pesquisa (depois)	57

Capítulo 1

Introdução

1.1 O Projecto

O projecto apresentado surge no âmbito de uma problemática actual, cada vez mais frequente nos dias que correm, em empresas de desenvolvimento de software. Considerando sistemas informáticos de médias/largas dimensões, é muito frequente recorrer a várias tecnologias, o que resulta numa divisão em várias camadas aplicacionais. Isto tem como consequência o desenvolvimento em paralelo e dependências entre as várias camadas, requerendo uma grande capacidade de organização interna, mas podendo originar também problemas derivados destas mesmas dependências. Por exemplo, a má distribuição de tarefas ou sobrecarga de uma das camadas pode comprometer a produtividade de outra, devido à fraca capacidade de autonomia que cada uma delas possui. Isto acontece frequentemente quando se em bases de dados e adaptações de novas necessidades de informação. Estas adaptações traduzem um forte impacto não só a nível das bases de dados, mas sim por todas as outras camadas de que dela dependem, pelo que é importante garantir alguma autonomia de desenvolvimento, sem perder a capacidade de sincronização, de modo a facilitar o processo.

Assim sendo procura-se essencialmente desenvolver uma solução que actue sobretudo ao nível da camada de *user interface*, fornecendo toda a autonomia que esta precisa de forma a ultrapassar esta falha.

Tendo em conta este problema, o trabalho aqui apresentado visa combater estas falhas de eficácia, aplicando a solução encontrada a um caso de estudo específico, no âmbito da ALERT Life Sciences Computing, S.A.

1.1.1 Motivação

O problema apresentado anteriormente é cada vez mais uma realidade presente em muitas das empresas actuais. O crescimento destas empresas leva muitas vezes à necessidade de adaptações ao nível da camada de dados, resultando nas consequências referidas. Face a este panorama, é importante conceber uma solução para o problema, principalmente focando no caso de estudo apresentado, e considerando as suas necessidades específicas.

No contexto empresarial, foi pedido ao autor do documento que procedesse a novos desenvolvimentos de forma a adaptar a interface gráfica do ALERT às novas funcionalidades para o mercado EUA. A chegada da empresa a este mercado tem um impacto natural no que toca a novas necessidades de informação.

Focando especialmente na equipa de trabalho onde o projecto se realiza (equipa de medicação/prescrição), este impacto manifesta-se de forma notável, sobretudo quando falamos de mercados como é o caso de EUA, que nos introduz conceitos completamente alheios àquilo que se passa a nível nacional, como é o caso do *co-sign*, conceito este que está

relacionado com a capacidade de delegação da capacidade de prescrição, com base em aspectos legais. Estes conceitos levam à necessidade de proceder a um conjunto de adaptações funcionais e de estrutura de dados. Se por um lado a maioria das alterações consequentes afectam sobretudo a camada de dados, por requerer adaptação de modelos e novas funcionalidades a este nível, por outro há que considerar que a camada de *user interface* sofre não só o impacto das novas alterações a implementar mas também alguns atrasos de desenvolvimento resultantes do desequilíbrio de tarefas entre as camadas de dados e *user interface* desta equipa. Este desequilíbrio traduz-se num desenvolvimento de software assíncrono e atrasos desnecessários, reduzindo a produtividade da equipa.

Considerando os prazos cada vez mais apertados provenientes da necessidade de responder a vários mercados espalhados pelo mundo, torna-se importante otimizar todo o processo de desenvolvimento de software, neste caso particular, da equipa de medicação/prescrição.

A nível empresarial, a solução enquadra-se como ferramenta de apoio aos novos desenvolvimentos para a nova versão dos produtos ALERT: a versão 2.4.3.

1.1.2 Objectivos

Tendo em consideração o problema apresentado no âmbito deste trabalho, o principal objectivo é procurar uma solução para o problema da dependência entre camadas de desenvolvimento. Esta solução deve actuar ao nível da camada de *user interface*, permitindo obter uma maior autonomia. Procura-se também que seja a mais genérica possível e aplicável todas as equipas de desenvolvimento. No entanto, de forma a poder aplicar esta mesma solução ao caso de estudo, é também considerado o desenvolvimento de algumas das funcionalidades 2.4.3, no âmbito da equipa de medicação/prescrição.

Enumerando, são considerados os seguintes objectivos para o este trabalho:

- Resolver os problemas resultantes da dependência entre as camadas de dados e *user interface* durante a fase de desenvolvimento. A solução deve promover a independência entre as duas camadas de forma a evitar tempos de espera, contudo, sem perder o sincronismo necessário para a produção de bons resultados;
- Deve focar essencialmente na camada de *user interface*;
- Deve ser uma solução relativamente simples e rápida, praticável no ambiente em que se enquadra, e deve ser possível actuar e testar durante o tempo disponível para o desenvolvimento do projecto;
- Adaptar-se às necessidades de todo o pessoal de desenvolvimento da empresa, no que diz respeito à camada de *user interface*;
- Deve ser devidamente apresentada, documentada e partilhada no âmbito da empresa;
- Não entrar em conflito com os procedimentos internos da empresa onde é aplicada esta solução;
- Devem cumprir todas as especificações desenhadas e documentadas;
- Deve ser implementado o conceito de *Co-sign*, de acordo com o que é especificado internamente, tendo em conta as permissões e tipos de utilizador da aplicação;
- Devem ser efectuadas as alterações visuais necessárias sobre o ecrã de acordo com as novas regras de interacção;
- Deve ser desenvolvido um novo componente de pesquisa a incluir em vários ecrãs ALERT, com maior impacto sobre os ecrãs de medicação;

1.2 A empresa

A *Alert Life Sciences Computing S.A.* é uma empresa dedicada ao desenvolvimento de software clínico que tem crescido cerca de 100% por ano, tanto em colaboradores como em lucros. O seu crescimento tem-na levado não só a conquistar vários hospitais a nível nacional, mas também a outros mercados espalhados pelo mundo, como é o caso dos EUA, Brasil e Singapura. É uma empresa jovem, com uma média de idades entre os 30 anos, aproximadamente. Devido à natureza crítica dos seus produtos, tem apostado sobretudo na qualidade e performance, procurando manter boas práticas de desenvolvimento de modo a favorecer o seu crescimento contínuo.

1.2.1 As 3 camadas de desenvolvimento

Foram já referidas pelo menos duas camadas de desenvolvimento distintas, que estão na base do problema deste projecto.

Deste modo, torna-se relevante apresentar diante o autor, de uma forma brev e simplista, as 3 camadas de desenvolvimento dos produtos ALERT, cuja descrição detalhada surge mais adiante, na secção de análise do problema. Para já, considere-se o esquema da Figura 1

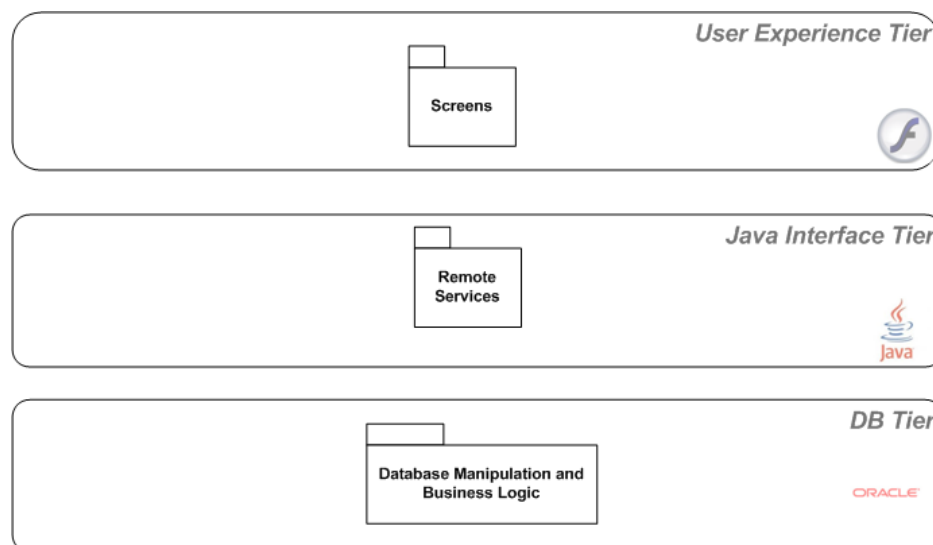


Figura 1: As 3 camadas de desenvolvimento

User Experience Tier ou User Interface Tier

Camada responsável por desenvolver a interface gráfica dos produtos ALERT toda ela implementada em *Flash/Actionscript*. Nesta camada são tratados todos os aspectos gráficos e de interacção com o utilizador.

É nesta camada de desenvolvimento que se insere o autor do documento, na equipa de prescrição/medicação.

Java Interface Tier

Responsável estabelecer interface entre ALERT e outras aplicações do domínio. No âmbito do projecto, surge como facilitador da comunicação entre a *user interface* e a camada de dados, por motivos explicados mais à frente, no capítulo relativo à análise do problema. No entanto, serve ainda outros propósitos, como é o caso da adaptação do ALERT a outras aplicações externas, de modo a torna-lo num serviço completo e integrado.

DB Tier

Camada de dados responsável por gerir e manipular toda a informação dos produtos ALERT. Aqui é concentrada toda a lógica de negócio e por isso, grande parte dos aspectos funcionais dos produtos. Todo o trabalho de processamento de dados é feito a nível desta camada, deixando o *user interface* tratar apenas do aspecto visual.

1.2.2 Organização Interna – A equipa de medicação/prescrição

Internamente a ALERT encontra-se dividida em vários departamentos, sendo importante destacar o departamento de desenvolvimento de software. Este departamento é bastante abrangente e complexo, por sua vez composto por sub-departamentos de forma a facilitar o desenvolvimento de software e aumentar a organização destes processos. Estes sub-departamentos separam as principais funcionalidades ALERT pela sua natureza: por exemplo, o “HOSPITAL” é relativo aos produtos de apoio clínico nas tarefas “práticas”, como é o caso do serviço de urgência (ALERT EDIS); já o sub-departamento “ADW” é relativo ao produto ALERT ADW, sendo este especialmente desenvolvido para tarefas de gestão hospitalar. Entre os vários sub-departamentos de desenvolvimento é de destacar o “CORE FEATURES”, que tal como o nome indica, é relativo ao desenvolvimento de funcionalidades “nucleares”. Entenda-se por funcionalidades nucleares, todas aquelas que são transversais aos produtos ALERT e portanto, de grande impacto nas várias aplicações.

A equipa de prescrição é uma das equipas “CORE”, por se tratar de uma das maiores funcionalidades transversais dos produtos ALERT. Como demonstra a Figura 2, os resultados desta equipa tem impacto sobre 6 produtos diferentes. Tal como as restantes equipas deste sub-departamento, a equipa de prescrição é composta por programadores *Oracle* e *Flash/Actionscript*. O trabalho é feito em paralelo entre os vários membros, e muitas das vezes o grande problema resulta precisamente do sincronismo existente entre estas duas camadas de desenvolvimento, problema esse que deve ser resolvido com a solução deste projecto.



Figura 2: Enquadramento da equipa de medicação/prescrição

1.3 Planeamento do projecto

Este trabalho encontra-se dividido em 2 fases essenciais. Uma primeira fase, denominada “*Data Abstraction*” corresponde ao estudo e desenvolvimento da solução para o problema deste trabalho. A segunda, consiste na aplicação dessa mesma solução no âmbito da empresa, tendo como base os desenvolvimentos necessários para a nova versão do produto.

No entanto, devido a factores exteriores ao âmbito do projecto mas necessários para a sua realização, foram ainda acrescentadas as fases de formação e de documentação. Enquanto a primeira fase diz respeito à integração no ambiente da empresa, a última refere-se ao tempo inteiramente dedicado à conclusão de toda a documentação e entrega dos deliverables. A Figura 3 ilustra as diferentes etapas e sub-etapas deste projecto. Para mais informação relativa a este tópico, recomenda-se que analise o diagrama inserido no ANEXO A: Diagrama do Planeamento, com a calendarização detalhada, e ainda o *logbook* apresentado na *wiki* do projecto.

Planeamento	
Formação	3
Alert	1
Flash	1
Framework	1
Desenvolvimento do Projecto	14
Estudo e Análise	3
Análise do problema	2
<i>Identificação e descrição do problema</i>	
<i>Análise do Estado da Arte</i>	
Definição da Metodologia	1
Parte I: <i>Data Abstraction</i>	6
Conceptualização	1
<i>Desenho e definição da arquitectura</i>	
<i>Estratégia de integração da solução no sistema</i>	
Implementação	4
<i>Desenvolvimento da aplicação</i>	
<i>Integração no ALERT (local)</i>	
<i>Testes e estudo do comportamento da aplicação</i>	
Documentação (<i>Data Abstraction</i>)	1
Parte II: Aplicação da metodologia	4
Análise das necessidades de informação	1
<i>Análise das necessidades de informação para os ecrãs 2.4.3</i>	
<i>Desenvolvimento da documentação de especificação de interface</i>	
Implementação	3
<i>Desenvolvimento das novas funcionalidades 2.4.3</i>	
<i>Testes e estudo do comportamento das novas funcionalidades e documentação</i>	
Fase de Testes e Avaliação da metodologia (estudo de performance)	1
Documentação Final - Deliverables	3
Wiki (conclusão)	
Relatório do Projecto (conclusão)	

Figura 3: Planeamento do projecto

1.4 Estrutura do documento

Este documento encontra-se dividido em 6 capítulos. A primeira parte do documento onde se enquadra esta secção é reservada ao breve enquadramento e contextualização do leitor no âmbito do projecto e a sua envolvente. O capítulo 2 descreve sobretudo o estudo do estado da arte do projecto, que tanto diz respeito aos estudos feitos no domínio como também às tecnologias existentes, na medida em que constituem a base de potenciais soluções para o problema enfrentado. No terceiro capítulo é feita uma descrição mais detalhada do projecto em si, sendo também descrita solução encontrada. Esta descrição inclui as principais decisões baseadas nos estudos referidos no capítulo 2, e em aspectos internos à empresa. O capítulo 4 é reservado aos detalhes de implementação, descrevendo todo o trabalho envolvido ao longo do projecto e as técnicas/ferramentas utilizadas. No capítulo 5 são abordadas as questões relativas aos resultados da implementação, impacto na empresa, cumprimento de objectivos e aspectos a melhorar. Finalmente são apresentadas as conclusões e perspectivas de trabalho futuro no capítulo 6.

Capítulo 2

Revisão Bibliográfica

Antes de tomar qualquer decisão relativa à implementação de uma solução para o problema apresentado, um dos pontos essenciais deste projecto passou pela pesquisa e investigação do estado da arte existente de modo a procurar as melhores adaptações tecnológicas e também as melhores soluções no domínio. Neste caso, o autor encontra-se perante um problema que pode ser dividido em duas partes. A primeira está nitidamente relacionada com metodologias de desenvolvimento de software, procurando uma optimização no contexto do ambiente de trabalho anteriormente descrito. O seu domínio está totalmente direccionado para a área de engenharia de software, que é abrangente e variado. Tendo em conta os seus principais objectivos, terá todo o interesse discutir sumariamente algumas metodologias de desenvolvimento de software utilizadas na actualidade, medindo os pros e os contras de cada uma. Como será comprovado adiante, o autor nota uma maior orientação do projecto para a área de testes unitários. A segunda parte é mais orientada para o domínio das interfaces gráficas, pelo que também será interessante fazer um estudo neste domínio.

Para além disso, é também relevante fazer uma revisão tecnológica nos domínios do projecto de forma a verificar as melhores soluções tecnológicas a aplicar.

2.1 Estado da arte: Estudos no domínio

De modo a conhecer melhor as técnicas e metodologias aplicadas na actualidade, este projecto baseia-se em alguns artigos como referência. Neste capítulo é dada a conhecer toda a investigação feita no sentido de aprender sobre algoritmos, técnicas e/ou boas práticas utilizadas na actualidade, nos domínios do problema. Há essencialmente dois domínios a considerar: Engenharia de software – metodologias e técnicas de desenvolvimento de software; e Desenvolvimento de interfaces gráficas, mais especificamente em Flash.

2.1.1 Testes unitários e Mock Objects

Um problema no âmbito do desenvolvimento de software que sempre assombrou programadores de todo o mundo é a questão do controlo de qualidade. Não só porque determinadas aplicações requerem níveis de segurança bastante rígidos e elevados como também porque muitas vezes é difícil detectar falhas. Apesar de muita gente evitar esta fase durante o desenvolvimento dos seus produtos, a luta por desenvolver novas metodologias para combater possíveis falhas é tem sido uma realidade que tem levado a uma evolução neste campo. Ao longo desta evolução surgiram conceitos muito utilizados actualmente, como é o caso dos **testes unitários**. [wik2]

Um teste unitário parte do princípio que as funcionalidades devem ser isoladas de modo a melhor detectar possíveis falhas ou avarias de um sistema, ao aplicar testes simples de

I/O (*input/output*) a pequenas secções do código. É considerada unidade uma parte reduzida da aplicação, a nível “atómico”, como por exemplo métodos ou funções simples. A ideia é aplicar uma estratégia destes testes atómicos que comprovam o funcionamento de cada secção de um sistema, isolando e identificando melhor as falhas que possam surgir.

Com o aparecimento e utilização cada vez mais frequente da programação orientada a objectos, o conceito de testes unitários evolui para uma nova etapa, dando origem aos chamados *mock objects* [Tim00]. Antes de explicar o que são estes *mock objects*, será melhor explicar o conceito de *fake object*. Um *fake object* é um objecto que retorna valores pré-definidos à entidade que o chama independentemente dos valores de entrada, de forma a garantir a independência. Partindo deste conceito, um *mock object* faz o mesmo que o *fake*, mas testa os valores de entrada antes de retornar valores pré-definidos, testando assim a chamada ao próprio objecto. Podemos ver os *mock objects* como *crash test dummies* do software: são concebidos para simular o comportamento do objecto real sem no entanto ser o verdadeiro objecto servindo apenas para testar alguns cenários. Ou seja, trabalhar como um objecto deste tipo será como jogar ténis contra uma parede: quando mandamos a bola como deve ser, esta volta para nós como desejado, caso contrário a bola faz ricochete para longe (erro), apenas temos de nos preocupar com a nossa maneira de jogar porque se a bola vem mal, é porque a enviamos mal, a parede não tem culpa. Quando enviamos os valores de entrada perfeitamente válidos, o objecto retorna um valor pré-definido por e completamente previsível por nós, caso contrário irá enviar uma mensagem de erro que não nos interessa.

Estes objectos são hoje em dia muito utilizados no âmbito dos testes unitários, uma vez que parte deste mesmo conceito, transportado para a realidade da programação orientada a objectos. No problema em causa, estes conceitos podem ser de extrema utilidade, mesmo que não aplicados directamente.

XUnit

Após o aparecimento do conceito de testes unitários, explicado anteriormente, foram surgindo algumas *frameworks* de teste para várias linguagens de programação, como é o caso do *java*, *c++* e até mesmo *Actionscript*. Muitas das *frameworks* utilizadas hoje em dia tiveram origem no design inicialmente concebido para o SUnit (*framework* de testes para *SmallTalk*).

Hoje em dia o conjunto formado por todas estas “Unit’s é denominado *XUnit*, sendo praticamente um standard no que toca ao desenho e concepção de ferramentas para testes unitários. Apesar de ser um modelo bastante utilizado como é possível verificar a partir do exemplo do *JUnit* não implica necessariamente que é superior às outras implementações de *frameworks*, como será provado mais adiante, no entanto é um ponto a favor por respeitar um conjunto de regras (vantagens de recorrer ao *standard*). [Cod1]

2.1.2 Conceitos na área da saúde: Prescrição Médica

Durante a realização deste trabalho foi necessário proceder à implementação de algumas funcionalidades como membro da equipa de medicação/prescrição. Estas alterações não só exigiram conhecimentos a nível técnico (*Actionscript*) mas também conhecimentos na área da saúde. De forma a entender totalmente o trabalho em mãos foi feito um estudo prévio neste domínio. Antes de especificar a área da saúde há alguns conceitos básicos do domínio que devem ter-se em conta. Um conceito importante no que diz respeito ao tratamento clínico é o conceito de episódio.

Episódio

Por episódio clínico, considere-se a passagem de um doente pela instituição de saúde, desde o momento em que é tido como paciente, até ser dada alta. Estes episódios tanto podem ser vistos no âmbito do internamento como de consulta externa, e podem ser compostos por várias visitas. Este conceito é várias vezes utilizado no âmbito do tratamento de um determinado paciente, e como será visto adiante, faz parte dos conceitos a ter em conta na área da prescrição ALERT.

Prescrição médica

Prescrever um medicamento é o acto de autorizar/recomendar um paciente sempre que necessário. Esta actividade é normalmente exercida pelos médicos e é de uma responsabilidade extrema, pois há imensos factores a ter em conta: por exemplo, a idade, sexo, actividade, hábitos alimentares, alergias, etc.

Manipulados

Medicamentos manipulados, como o nome indica, são soluções manipuladas à responsabilidade exclusiva dos serviços hospitalares ou farmaceuticos, sob responsabilidade do farmacêutico em ambos os casos. Esta manipulação pode seguir duas fórmulas: magistrais, preparado a partir de uma receita médica; ou officinais, não sujeitos a receitas médicas e disponibilizados directamente pelo serviço de farmácia. [Inf]

Dietéticos

Dietéticos como o próprio nome indica estão relacionados com a dietas do paciente, sendo sobretudo compostos por suplementos alimentares que visam auxiliar o tratamento de um dado paciente. Não podem ser considerados propriamente drogas semelhantes às restantes, uma vez que não estão destinados a tratamentos específicos, mas podem ser vistos como químicos auxiliares.

Soros

Genericamente, soros são soluções parentéricas administradas de forma contínua tendo algumas variantes conforme as necessidades dos pacientes. Visam a hidratação e reposição de electrólitos. É mais frequente recorrer ao uso do soro fisiológico, composto essencialmente por cloreto de sódio.

Posologia

Posologia está relacionada com o modo de administração de um dado fármaco. Especifica a quantidade, dose e outros detalhes que os profissionais de saúde devem ter em conta quando estão a administrar um medicamento a um paciente.

Prescrição na ALERT

A nível funcional, esta equipa desenvolveu a maioria das funcionalidades básicas a considerar no domínio, para o mercado português, estando a trabalhar de momento em adaptações a outros mercados, de forma a acompanhar o crescimento geográfico da empresa. Se parte do projecto passou pela integração na equipa de medicação/prescrição, e pela alteração de alguns aspectos nos ecrãs existentes, algo a ter em conta é o estado inicial.

Quando se fala de uma alteração, apenas fará sentido se for conhecido o ponto de partida e de chegada. Antes de poder implementar alterações nestes ecrãs, foi necessário compreender o conteúdo no seu ponto de partida.

Tipos de prescrição/medicação

Falar em funcionalidades prescrição dos produtos ALERT implica ter em conta o próprio conceito de prescrição médica, como ideia e como funcionalidade.

Como é sabido, o acto de prescrever um medicamento consiste em passar uma receita de um medicamento a um paciente, podendo ser visto como uma permissão para que um dado paciente possa tomar essa medicação. No âmbito do tratamento clínico, este conceito surge em diferentes contextos, desde a receita para o paciente ir buscar a sua medicação à farmácia, até à administração de urgência, feita no próprio hospital ou clínica, pelo enfermeiro.

No contexto dos produtos ALERT, são considerados os seguintes cenários:

- Administrar no local: consiste em administrar um determinado medicamento a um paciente na própria instituição, como acontece muitas vezes no contexto de internamento;
- Receita para o exterior: quando um paciente tem alta de um internamento, pode estar ainda dependente de medicação. Num caso destes ou semelhante, como numa consulta externa, o médico passa uma receita para o doente poder comprar a medicação numa farmácia.
- Medicação anterior (relato de medicação): o facto de um paciente deixar de tomar uma determinada medicação não retira a sua importância, muito antes pelo contrário. Deve constar no seu registo na aplicação, de forma a que o médico possa ter consciencia do histórico do paciente.
- Soros: devido à natureza específica deste tipo de medicação, os soros são considerados à parte em relação a outros tipos. No entanto, pode e deve ser considerada medicação administrada no local.
- Manipulados: manipulados são tipos de medicação complexa e normalmente utilizada em contexto de internamento.

- Dietéticos: dietéticos não são exactamente medicamentos mas sim produtos nutricionais usados em contexto clínico.

Em todos os cenários apresentados, é de destacar o facto de que é sempre o médico a entidade responsável pela prescrição, independentemente da situação ou contexto. Isto é relevante na medida em que reflecte o que se passa no mercado português.

O mercado EUA e a funcionalidade *Co-sign*

No âmbito da proposta para este projecto e do seu objectivo inicial, tornou-se necessário fazer um estudo das novas funcionalidades relativas aos mercados estrangeiros, nomeadamente dos EUA. Entre outras funcionalidades específicas é de destacar aquela que de facto foi atribuída à responsabilidade do autor do presente documento, no âmbito do projecto: o *co-sign*.

O *Co-sign* (em português, “abaixo-assinado”) [wik1] traduzido à letra, consiste em responsabilizar por um acto realizado por outra pessoa. Por exemplo, o professor responsabilizar-se pela intervenção que um estudante de medicina faz. É um conceito usado na área da prescrição que está sobretudo relacionado com aspectos de permissões legais que afectam os profissionais da saúde. Fazendo um paralelo com a realidade do mercado português, considere-se um cenário de internamento. O paciente está sob efeito de medicação prescrita pelo médico responsável, no entanto, surge a necessidade de fazer uma nova prescrição, devido a uma manifestação nervosa pela parte do paciente. Apesar de estar em estado alterado, o enfermeiro de serviço não poderá administrar um ansiolítico, a não ser que tenha sido prescrito pelo médico.

Aqui entra a realidade americana, que esta funcionalidade ALERT pretende cobrir. Numa situação semelhante à apresentada, um enfermeiro americano poderá fazer a própria prescrição, desde que um médico assuma essa mesma responsabilidade. De certa forma, pode ser visto como uma permissão/responsabilização pela parte do médico da acção do enfermeiro, sendo esta documentada devidamente (*co-sign* ou “abaixo-assinado”). O enfermeiro poderia então prescrever e administrar o paciente sob responsabilidade do médico.

No âmbito da informática da saúde, nos EUA, é já contemplada esta funcionalidade à luz da lei americana [Cos].

2.2 Revisão Tecnológica

Se por um lado é interessante observar as técnicas e teoria relacionada com os domínios do problema, não terá menos interesse fazer uma revisão tecnológica sobre os temas, de modo estar ciente de possíveis soluções para o problema, ou aspectos relacionados com tópicos de carácter mais técnico que possam auxiliar na escolha de uma solução adequada. Desta forma, foi feita uma investigação daquilo que o autor considera relevante no âmbito do projecto.

2.2.1 Testes unitários: Frameworks existentes

Flexunit - visual flexunit (Flex)

A *FlexUnit* foi a primeira *framework* de testes unitários para *actionsript*. Actualmente suporta testes mesmo em *flex* e *actionsript 3* e integra algumas funcionalidades

comuns ao *JUnit* apesar de não pertencer à “família”. Dispõe também de uma interface gráfica para correr os testes apenas. Está no entanto a ser desenvolvida uma alternativa, a *Visual FlexUnit* toda ela baseada na anterior, mas desta vez mais orientada a testes de **Componentes Flash**. Esta Framework ainda em construção vai dispor de novas formas de realizar testes de maneiras mais intuitivas, a partir de uma interface gráfica e ficheiros **Ant**.
[Fle]

Reflex unit (Flex)

Surge como uma resposta ao *FlexUnit* apresentando novas soluções alternativas que defendem a realização de testes de formas mais simples e poderosas. Pode ser vista como uma versão bastante melhorada do *FlexUnit*, ainda em desenvolvimento. O nome provém da sua característica principal e estratégia utilizada para facilitar a implementação dos testes por código: **a reflexão**. Tirando proveito desta capacidade, a API do *ReflexUnit* permite declarar métodos estáticos sem ter de os implementar e preocupar-se com extensões de classes.

Além desta sua característica, o *ReflexUnit* vai permitir a execução de testes múltiplos e em simultâneo e vai oferecer ainda 3 formas de outputs bastante eficazes:

- FlexViewer: Semelhante ao *Xray* utilizado pelas equipas de desenvolvimento da ALERT;
- CruiseControlLogger: Ferramenta utilizada para construção (build) programada e assistida de projectos em desenvolvimento. Baseia-se em ferramentas de controlo de versões (como o *SVN*) e em ferramentas de “*Project-building*” (como o *Apache ANT*) para garantir uma construção de software de uma forma mais eficaz e organizada, multi-plataforma e automática. Faz ainda registos (Logs) em ficheiros XML após terminar o processo;
- ConsoleViewer: O próprio nome diz tudo. É uma consola/ linha de comandos onde são apresentados os dados de saída.

[Ref]

Funit (Flex)

É uma implementação *XUnit* que surge também como um avanço relativamente ao *FlexUnit*, apresentado anteriormente.

Comparando com o *FlexUnit*, o *FUnit*:

- Não precisa de estender a *TestCase Class*;
- Não usa prefixos para chamar os métodos, recorrendo antes a um sistema de *tag's* como [Test] ou [ExpectedError];
- Suporta métodos característicos do *XUnit* (“*SetUp*”, “*TearDown*”...) ao implementar a sua arquitectura.

[Fle]

As2lib (Actionscript)

Parte do princípio que a programação em actionscript não é tão estruturada como devia mas que o uso de testes unitários pode ter uma influência positiva neste aspecto. Oferece principalmente um conjunto de bibliotecas com funcionalidades “core” como tratamento de erros e eventos que aliadas às capacidades de reflexão (já referida anteriormente) e *Logging* (feito através de uma interface concebida para o efeito) se traduzem numa melhor experiência de desenvolvimento em *actionscript*.

[AsL]

Asunit (Actionscript) (Não confundir com o da applescript)

Framework de testes unitários desenhada para *actionscript*, que funciona nas versões *Flash* 7, 8, 9 e *Actionscript* 2 e 3. O seu lema é “simplificar o processo de testes” e apesar do seu nome sugestivo, não faz parte da “família” *XUnit*. No entanto mas é uma ferramenta poderosa que oferece:

- XUL UI: Uma interface gráfica para a implementação de testes de uma forma mais simples, intuitiva e transparente
- SWF UI: Interface de demonstração de *outputs* semelhante ao **Xray**.

[AsU]

Comparação das Alternativas:

De modo a comparar as várias alternativas e verificar aquelas mais adequadas ao problema em questão, é importante saber aquilo que se procura. Tendo em conta o trabalho em questão, há que considerar fundamentalmente 4 aspectos:

- Flex/AS: se a ferramenta foi desenvolvida para *Flex* ou *Actionscript*. Relevante na medida em que a empresa utiliza *Actionscript*;
- Disponibilidade: um aspecto crítico é saber se realmente é possível aceder à solução, caso contrário não poder ser admitida como uma hipótese válida;
- Usabilidade: É procurada uma solução simples e fácil de usar, capaz de captivar o utilizador logo num primeiro contacto com a ferramenta.

Tabela 1: Comparação de frameworks

	Flex/AS	Disponibilidade	Usabilidade
FlexUnit	Flex	Sim (<i>Opensource</i>)	Um pouco rudimentar
Visual FlexUnit	Flex	Não	Sim, possuirá modo de interacção visual
ReFlexUnit	Flex	Sim (Beta)	Bom controlo de outputs, mas exige modificação por código para implementar testes
FUnit	Flex	Sim (Gratuito)	Cópia do JUnit adaptada à realidade <i>flex</i>
AS2Lib	AS	Sim (Gratuito)	Pode ser comparada a um ReFlexUnit transportado para <i>Actionscript</i>
ASUnit	AS	Sim (Gratuito)	Facilidades gráficas fornecem claras vantagens sobre o AS2Lib

No ponto de vista de usabilidade é de destacar o Visual FlexUnit, o ReFlexUnit e ainda o ASUnit, que facilitam o processo de testes. No entanto, considerando ainda a disponibilidade das hipóteses, o autor ve-se obrigado a descartar o VisualFlexUnit por ser uma solução ainda não disponível. Finalmente, considerando que o ASUnit foi desenvolvido para Actionscript é de realçar, descando-o como a hipótese mais favorável. [Wik] [lul]

2.2.2 Armazenamento de dados

Tendo em vista o desenvolvimento de uma nova ferramenta de testes como alternativa às *frameworks* estudadas neste capítulo, um dos pontos a ter em conta será o tratamento alternativo da informação. Se com esta nova funcionalidade pretende-se substituir a chamada

à base de dados, como alternativa, deverá haver um suporte para armazenamento e pesquisa de dados eficaz e fácil de manipular que permita a esta funcionalidade ter uma boa performance que não comprometa o funcionamento do sistema ALERT. Deve sobretudo ir de encontro aos seguintes pontos:

- Simplicidade: quando mais complexo for, mais problemas vai trazer. Primeiro aspecto é sem dúvida o facto de estarmos a falar de uma aplicação que pode vir a afectar a forma como é desenvolvido software numa empresa. Isto leva-nos a pensar na questão da integração da aplicação que será mais difícil quanto mais complexa e custosa for.
- Eficácia: Deve apresentar resultados nunca piores aos já existentes, isto é, se o acesso aos dados apresentar tempos de pesquisa muito piores que os anteriores e isso for significativo, os programadores vão sentir-se relutantes a utilizar uma ferramenta destas.
- Independência da rede: Como se pretende substituir o acesso à base de dados ALERT, não faria sentido ter um repositório de dados centralizado na rede, uma vez que estaria dependente do funcionamento da rede e pretende-se minimizar as dependências de factores externos. Assim sendo os dados devem ser mantidos a nível local, uma cópia por máquina.
- Estabilidade: Esta nova solução não deve trazer novos problemas, deve facilitar na resolução dos outros que já existem

Estudo das alternativas

Ficheiros XML

Antes do aparecimento das bases de dados e ainda hoje em dia, sempre que é necessário armazenar pequenas quantidades de informação que não justifiquem o uso de um modelo de dados complexo, recorre-se ao uso de ficheiros. Embora primitiva, é a forma mais simples de guardar dados, ocupando menos espaço em memória devido à fraca ou inexistente quantidade de informação relativa ao formato dos mesmos. Carecendo de qualquer tipo de controlo ou validação de formatos, os ficheiros de texto não estruturados têm vindo a perder terreno face à utilização de standards actuais como é o caso do XML.

Não se tratando de uma linguagem de programação nem de um formato específico, o XML é uma metalinguagem flexível, expansível e sobretudo simples, que a torna numa solução bastante eficaz para o armazenamento e pesquisa de informação. O seu crescimento levou ao uso de XML no contexto da bases de dados que por sua vez permitiu atingir níveis de abstracção mais elevados, como é o caso das Ontologias RDF.

Em muitos casos, vários programadores optam pelo armazenamento dos dados em ficheiros XML principalmente nos cenários em que a quantidade de informação a guardar é relativamente pequena. [vel] [sty]

Tabela **Erro! Indicador não definido.:** XML vs Bases de Dados

XML	Bases de Dados	Comentários
Devido à sua simplicidade, ficheiros XML são estáticos e não necessitam de manutenção ou qualquer trabalho extra.	Requer um maior trabalho de desenvolvimento e manutenção.	A questão da manutenção é realmente um ponto a favor, no entanto a solução que se pretende desenvolver trabalhará com valores sobretudo temporários e quanto mais simples for, melhor
Método de pesquisa menos eficaz, mas com tempos perfeitamente aceitáveis quando se tratam de pequenas/médias quantidades de informação e bons resultados devido ao uso da estrutura do XML	O uso de índices permite atingir melhores tempos de acesso durante uma pesquisa.	As bases de dados aqui ganham, no entanto, devido ao acesso directo aos ficheiros sem recorrer a funções de bases de dados e o facto de se tratar de informação local permite compensar essa falha
Necessidade de ter um <i>Parser</i> (interpretador) pela parte da aplicação. No entanto, já várias linguagens de programação possuem ferramentas para tratamento destes ficheiros (entre as quais o <i>Actionscript</i> , que possui as classes <i>XML</i> e <i>XMLNode</i>)	Gestão de informação automática e independente. Dados são devolvidos a partir de <i>queries</i> que devolvem os dados organizados em tabelas, mais fáceis de tratar na aplicação.	A única validação sobre os XML externa à aplicação seria feita pelo XSD que controla apenas a forma do documento, e isso realmente obriga o programador a ter o cuidado de integrar novas funcionalidades de tratamento de dados na aplicação
Protecção apenas da forma que é garantida pelo <i>XML Schema</i> (<i>xsd</i>)	Protecção do conteúdo e das regras do modelo de dados a partir de restrições e <i>transactions</i>	Na ferramenta de testes que se pretende construir não será crítica a questão do conteúdo, uma vez que será apenas para efeitos de teste, no entanto a forma será importante para efeitos de correcta análise dos dados por parte da aplicação
Uso simples e directo. Apenas necessita ficheiros XML e um ficheiro XSD para validação, sem necessidade de instalações ou configurações	Necessidade de instalação do SGBD, configuração e alojamento (disco) que requer mais recursos.	Relativamente a este ponto é importante salientar a dependência da rede, o acesso aos dados deve ser local para evitar dependências, incluindo

		da rede da empresa. Uma base de dados central não seria um problema, mas torna-se complicado quando se quer ter uma base de dados por máquina, exigindo mais instalações e configurações
--	--	--

Após analisar as diferentes alternativas, tendo em conta os pontos mencionados no início desta secção, conclui-se que a melhor solução passará pela utilização de ficheiros XML, por se tratar da hipótese mais apta face às necessidades existentes.

2.2.3 Geração de documentos

Considerando a importância dos documentos de especificação de interfaces que se adivinha para este projecto, devido à sua natureza que foca essencialmente na questão da comunicação entre camadas de desenvolvimento, pode ter alguma relevância estudar soluções tecnológicas que permitam gerar documentação de uma forma automática. Considerando as várias alternativas anteriores, o autor do documento vê-se numa problemática que relaciona a adaptação de uma solução existente ao facto de ter de acrescentar uma nova funcionalidade. Após fazer alguma pesquisa foi possível traçar uma imagem do panorama actual e até mesmo destacar algumas soluções tecnológicas no âmbito da conversão de XML em documentos perfeitamente formatados, considerando nesse caso uma solução de raiz baseada em XML

XML ->WORD: *WordprocessingML Transform Inference Tool* (Microsoft Office 2003)

Apesar de não ser muito conhecido, este é um *plugin* de extrema utilidade num cenário como o actual, sendo este desenvolvido pela própria *Microsoft* para o *Office 2003*. Apesar de ter sido desenvolvido inicialmente para aquela versão, é também compatível versão 2007.

A ideia deste *plugin* é permitir que um utilizador possa criar um ficheiro XSL para transformações de XML -> WordML formatado, sem ter de se preocupar de facto com o código. Na realidade, *WordML* não se trata propriamente de uma nova estrutura de dados, mas sim variante do XML que resulta da sua integração no do *Microsoft Word*. Na realidade, tal é a importância do XML que esta foi a forma encontrada pela *Microsoft* de o enquadrar nas suas soluções de escritório (*Office*). Aproveitando este “*pseudo-formato*” e o estado da arte relativo à utilização de XML no *Word*, foi apenas um pequeno passo para a geração desta funcionalidade, visto que as transformações XSL já eram possíveis de efectuar a partir do *Word*, o único desconforto era ter de desenvolver um XSL para o efeito. Com este *plugin*, o utilizador apenas tem de se preocupar em editar uma instância XML de modo a definir um template genérico que queira aplicar a qualquer instância de acordo com aquele XML *schema*. A partir desse template, a *WordprocessingML Transform Inference Tool* gera de forma automática o ficheiro responsável por converter as instâncias XML num documento formatado, que em nada se assemelha com a sua forma original.

[Cod] [Mic]

XML -> PDF: Criação de XSL-FO utilizando as facilidades ALTOVA

Uma alternativa será recorrer às ferramentas ALTOVA, já conhecidas pelo conjunto de funcionalidades que oferece no domínio de tratamento de XML. Uma das suas funcionalidades adicionais ao pacote básico consiste precisamente numa ferramenta auxiliar ao desenvolvimento de ficheiros XSLT para criação de documentos a partir de XML.

Considerando os vários formatos de documentação existentes, os mais favoráveis a um ambiente empresarial são essencialmente dois: documento Word, ou PDF. Relativamente à primeira hipótese é bastante complexa, pois é um domínio muito restrito, apenas acessível pela solução anterior. Desse modo é importante considerar que a utilização desta tecnologia seria essencialmente para geração de PDF. É bastante intuitivo e poderoso, produzindo resultados de qualidade. Apesar da sua utilidade, o seu acesso é condicionado pelo facto desta ferramenta ser apenas disponível para demonstração, com acesso completo para pagantes.

[Alt]

XML->PDF: Ecrion XFDesigner

Esta ferramenta desenvolvida especificamente para a geração de PDF é sem dúvida uma das mais intuitivas e fáceis de utilizar, sendo uma solução poderosa. O seu grau de transparência permite ao utilizador definir um template “final”, sem ter de se preocupar demasiado com o código, à semelhança da primeira proposta apresentada. No entanto, há que ter em conta o facto de ser uma ferramenta de acesso restrito a pagantes, condicionando assim a escolha desta tecnologia e pondo em causa a sua integração no projecto. No entanto, não deixa de ser uma referência de valor.

[Ecr]

XML NOTEPAD 2007: Criação manual de XSL-FO (Pdf) ou XML- HTML

Como último recurso e para manipulação básica de XML é apresentada esta ferramenta, que ganha pela simplicidade.

- É simples e "leve", não exigindo muito tempo de instalação nem configurações complexas;
- É gratuita e de fácil acesso;

Se as alternativas para geração de PDF são restritas, dificultando o acesso a estas, há sempre a hipótese a considerar de criar manualmente o XSLT a partir do desenvolvimento de código num editor simples e *freeware*.

No entanto, esta opção entra em conflito com o planeamento do projecto, por se tratar de uma alternativa demorosa e relativamente complexa, quando comparada com as anteriores.

[XSL] [W3s]

2.2.4 Flash VS Flex

A tecnologia Flash nasceu como uma ferramenta de *design*, orientada para desenho e animação de objectos vectoriais. A sua utilização assemelhava-se ao uso de uma aplicação tipo *moviemaker* ou *Photoshop*, baseando-se em ícones e o código quase sempre era oculto ou inexistente. Com o passar dos anos foi evoluindo e ganhando cada vez mais adeptos no que toca ao desenvolvimento de aplicações multimédia. Por um lado, devido à codificação simples, o *actionscript* que estava ao acesso da maioria dos programadores. Por outro lado,

devido à sua forte componente de design que fornece uma capacidade visual distinta e à capacidade de integrar o resultado final num browser, transportando estas aplicações para o domínio Web.

Na ALERT, a aposta no componente visual dos produtos é um ponto evidente, contando com uma equipa de designers e programadores responsáveis por garantir essa mesma qualidade. Por este motivo e por outros explicados mais adiante, a empresa escolheu a ferramenta Flash para desenvolver a camada de interfaces gráficas.

Flash (no sentido de Flash IDE) e *Flex* não são a mesma coisa, pelo menos actualmente. Para começar, o IDE de Flash (como o Flash 8, por exemplo) é originalmente da *Macromedia*, se bem que actualmente foi comprada pela Adobe, à qual pertence também o *Flex*. Apesar de ambos se basearem em código *Actionscript*, apresentam diferenças na forma de interacção que afectam a experiência de desenvolvimento pela parte do utilizador.

A tabela seguinte apresenta sumariamente algumas das principais diferenças entre Flash e *Flex*.

Tabela 2: Flash vs Flex

Flash IDE	Flex
Essencialmente orientado para o design	Mais virado para o desenvolvimento de código
Desenvolvimento é mais demorado, mas por outro lado há um maior controlo dos vários componentes visuais do filme (vantagem da timeline).	Maior rapidez e facilidade de desenvolvimento, menor controlo ao nível do design uma vez que é tratado a partir de CSS.
.fla são ficheiros binários.	.swf são ficheiros de texto.
Separação do código por “camadas” menos evidente (tudo misturado).	Boa organização e divisão do código: CSS para estilos, HTML para publicação, etc.
Número de componentes mais limitado. Ferramenta mais “fechada” e limitada no que toca a desenvolvimento.	Maior variedade de componentes Flash.

Basicamente podemos considerar que o Flash IDE é uma ferramenta mais apta para o design e trabalho criativo (artístico) não exigindo grandes conhecimentos de programação. Por outro lado para quem já for mais experiente na área, o *Flex* apresenta melhores soluções de desenvolvimento, sendo uma ferramenta apta para o efeito.

Actualmente na ALERT utiliza-se o Flash 8 da *Macromedia*, principalmente devido à forte interacção existente entre as equipas de desenvolvimento e design. Utilizar *Flex* seria um obstáculo consideravelmente grande para os designers da empresa, que iriam ver-se forçados a trabalhar mais como programadores e iriam ver a sua criatividade limitada pelas suas capacidades de codificação. Uma vez que os produtos ALERT têm uma forte componente visual, esta aposta no design faz todo o sentido.

[Fla] [Fla1]

Então porquê Flash na ALERT?

Toda a camada de interfaces gráficas na ALERT foi e continua a ser implementada em *Flash*. Actualmente é utilizada a versão 8 da *Macromedia* e o código está escrito em *actionscript* 2.0, se bem que isto poderá vir a sofrer alterações com a chegada da versão 3.0 à ALERT. O uso desta tecnologia permite uma boa combinação do design criativo produzido internamente com a capacidade de programação orientada a objectos. Apesar de nem sempre ter sido assim, a grande maioria do código encontra-se actualmente organizado em classes e

devidamente estruturado de forma a possibilitar uma maior organização e reutilização do mesmo.

Se por um lado a programação em *actionscript* é mais “solta” ou “livre”, as equipas de desenvolvimento da ALERT apostam em ferramentas de *debugging* de forma a garantir um maior controlo sobre o código que é produzido. Além disso é feito um controlo de versões de forma a garantir um desenvolvimento seguro, o que é sem dúvida uma medida necessária para uma empresa que se encontra em crescimento.

2.2.5 XML e Flash: As limitações do Flash

Actualmente, o tratamento de XML é relativamente simples, por se tratar de um standard e isso levar à existência de *API's* capazes de suportar a leitura destes ficheiros. A linguagem *Actionscript* não é excepção à regra, e apresenta duas classes (XML e *XMLNode*) que facilitam a tarefa de leitura e interpretação, convertendo a informação para objectos *Actionscript* que são armazenados em memória da aplicação e tratados posteriormente (pelo programador).

A leitura e interpretação não constituem um problema, no entanto o mesmo não se pode afirmar relativamente à escrita de ficheiros de texto: quer estes sejam XML ou não. O facto da tecnologia Flash correr num sistema “fechado” torna-a segura e bastante favorável à utilização em ambiente Web, no entanto, traz desvantagens no que diz respeito ao acesso a recursos exteriores. Prevendo um cenário em que seja necessário desenvolver uma funcionalidade capaz de gerar XML, isto constitui um obstáculo elevado.

Actualmente existe uma solução informática para combater essa “falha”: o ***SWFKit Pro***. O objectivo inicial desta ferramenta foi o de converter as aplicações SWF para ficheiros executáveis, controlando o acesso aos recursos. Isto significa que com esta aplicação é possível ter aplicações com privilégios que de outra forma não seriam acessíveis. À partida transparece que tal operação seria sob pena de torna a aplicação mais “desprotegida”, uma vez que ficaria à responsabilidade do utilizador definir exactamente as operações “ilegais” a realizar. Apesar de parecer arriscado, na verdade esta ferramenta fornece um conjunto vasto de funcionalidades, protegendo totalmente os recursos e deixando assim os utilizadores completamente descansados. Como consequência, esta ferramenta pode ser ainda favorável ao desenvolvimento de grandes GUI's como sendo aplicações *Windows*.

[SWF]

Capítulo 3

Ferramenta Genérica para Suporte de Diversas Bases de Dados de Medicamentos

Como já foi referido na introdução deste documento, o principal objectivo deste trabalho é resolver a problemática resultante do desenvolvimento paralelo entre diferentes camadas aplicacionais, especificamente as camadas de *user interface* e de dados. Apesar de serem camadas bastantes distintas a nível tecnológico, existe uma forte dependência funcional que obriga a ter um nível de sincronismo elevado de modo a produzir resultados satisfatórios.

Como é sabido pela maioria dos leitores familiarizados com o domínio, as duas camadas encontram-se encadeadas de tal forma que os valores de entrada de uma são os valores de saída de outra, e vice-versa. Desta forma, não é possível garantir independência entre as duas camadas. Quando muito, pode ser garantida alguma autonomia das duas camadas, reduzindo a dependência ao fornecer ferramentas de teste e simulação, que permitam reproduzir determinadas funcionalidades. Considere-se a camada de *user interface* como foi especificado nos objectivos para este trabalho. Das duas, é a camada “dependente”. Se for necessário correr um determinado ecrã, o utilizador é obrigado a simular os dados de forma “*hard coded*” (implementação bruta dos dados no próprio código), caso a base de dados não disponibilize as funções necessárias;

A questão passa então por encontrar uma solução adequada que fornece um maior nível de autonomia a esta camada, permitindo testar e simular interfaces gráficas sem depender totalmente das funcionalidades de base de dados. No entanto, deve ter-se ainda em consideração a necessidade de manter o sincronismo entre as duas camadas. Caso contrário, quando for feita a integração das funcionalidades, muito provavelmente será um processo complicado e vulnerável a falhas.

Antes de passar à solução encontrada pelo autor, é feita primeiro a análise ao problema em questão.

3.1 Análise do problema

O problema apresentado surge no domínio da engenharia de software, mas focando na essência da problemática, essa, tem origem na forma como as camadas de software comunicam entre si. Na verdade esta comunicação é a palavra-chave e o ponto de partida para a análise desta questão. Tendo em conta o que foi dito anteriormente a respeito do problema em questão, considere-se o seguinte cenário representado na Figura 4. Acontece que os programadores da camada *user interface* pretendem implementar uma determinada funcionalidade B que por um determinado motivo ainda não se encontra a funcionar

devidamente, podendo até nem existir ainda. Neste cenário, aquela funcionalidade fica em *stand by* até que seja implementada. Ainda assim, essa funcionalidade pode conter erros que serão dificilmente detectados, pois levantam uma outra questão: no caso de erro, de quem é a culpa? Se não for possível oferecer uma certa autonomia às duas camadas de desenvolvimento, complica-se o processo de *debug* e leva a tempos de espera que comprometem a eficácia do desenvolvimento. Para não fugir aos objectivos deste trabalho, chama-se especial atenção para a camada de *user interface*.

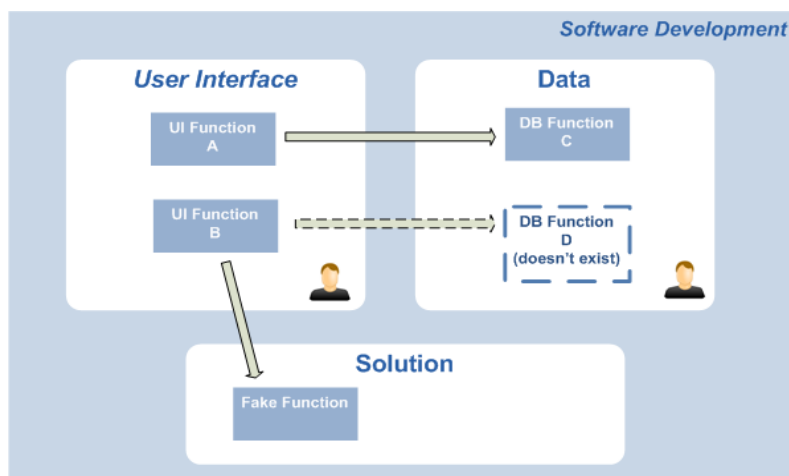


Figura 4: Problema da falta de autonomia

A melhor medida a tomar perante este cenário será garantir alguma autonomia à camada de *user interface* de modo a poder testar as suas funcionalidades de forma independente da camada de dados. No entanto, isto levanta outra questão: como garantir que as duas camadas sejam autónomas e ao mesmo tempo cumprem as mesmas especificações, estabelecendo uma interface que as ligue de forma coerente?

Este panorama levanta assim dois pontos essenciais: **sincronismo** e **autonomia**.

Autonomia

A autonomia em si pode ser dividida em duas questões fundamentais: por um lado, a garantia que a interface pode correr mesmo sem o funcionamento da base de dados; por outro, a capacidade de testar isoladamente as chamadas remotas como se fosse no resultado final de modo a melhorar o processo de *debug*. Isto resume-se essencialmente em duas funcionalidades: capacidade de teste e de simulação. Felizmente, o estado da arte relativo a testes de software é bastante rico, fornecendo bastantes conceitos, ideias e soluções neste sentido, disponíveis em várias tecnologias. Como foi apresentado no capítulo do estado da arte, há várias formas de combater esta falha. No entanto necessário ter em conta aspectos técnicos de integração antes de escolher uma solução adequada.

No que toca a simuladores, não se pode dizer que seja um conceito tão polémico, tratando-se na realidade de um tópico específico e que depende muito do contexto em que se insere. No entanto, as capacidades de teste e simulação podem ser facilmente entrelaçadas.

Sincronismo

O que se pretende obter neste ponto é garantir que duas equipas autónomas sejam capazes de produzir trabalho complementar, para que este “encaixe” no final. Imagine-se a construção de um carro. Cada máquina constrói uma peça, mas se não forem feitas a pensar nas outras, no final não vai dar bom resultado. No fundo este problema é semelhante, logo a

solução a aplicar poderá ser vista da mesma forma. Considerando o que é feito no estado da arte actual, a ideia é estabelecer uma documentação prévia que contenha todas as regras a cumprir pelas duas camadas, de forma a garantir que o produto é visto como um só e que no final tudo funcionará como planeado. Estes documentos são utilizados de modo a fornecer um maior controlo sobre as funcionalidades a implementar e para evitar “surpresas” desagradáveis.

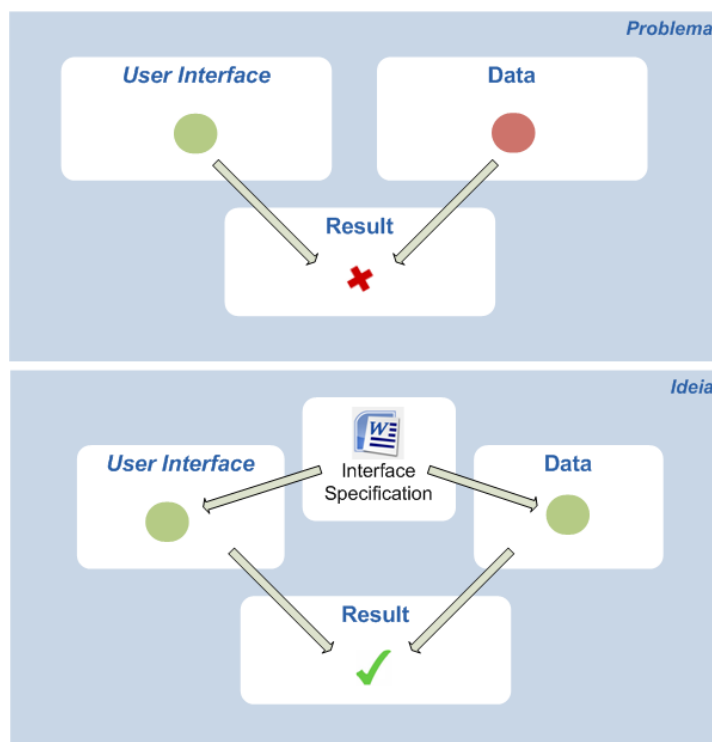


Figura 5: Problema da falta de sincronismo

Considerando o âmbito no qual deve ser aplicada a solução e partindo para uma análise mais detalhada e focada em soluções específicas foram considerados dois factores principais: por um lado os requisitos da empresa, que foram retirados a partir do estudo do ambiente interno; por outro lado, a revisão tecnológica e bibliográfica que foi apresentada no capítulo do estado da arte deste mesmo documento, de forma a procurar uma solução adaptável ao caso de estudo prático.

3.1.1 Processo de análise de requisitos

Os principais aspectos a ter em conta na concepção de uma solução para este problema foram os requisitos estabelecidos pela empresa. O processo de identificação e levantamento destes requisitos não foi directo nem pré-estabelecido de uma forma acessível, mas sim progressivo e obtido a partir do estudo dos hábitos de trabalho da empresa e de opiniões formadas dos *stakeholders* de maior peso. De facto, para este projecto foram consideradas as seguintes técnicas para o processo de levantamento de requisitos:

- Entrevistas e reuniões brainstorming;
- Observação de hábitos de trabalho;
- Estudo do ambiente interno (arquitetura *Flash*).

Será justo afirmar que grande parte dos requisitos básicos e intangíveis foram retirados a partir do estudo do ambiente e hábitos de trabalho, enquanto as entrevistas serviram essencialmente para uma compreensão mais detalhada do problema e das suas necessidades.

Stakeholders

O primeiro passo desta etapa foi a identificação de todos os *stakeholders*. Considerando um cenário optimista e atentando à descrição do problema, o melhor impacto que esta solução poderá ter será ao nível de todo o desenvolvimento CORE, e possivelmente outras equipas de desenvolvimento, nas camadas *Flash/Actionscript*. Há portanto que ter em conta os seguintes *stakeholders*, devidamente categorizados:

Tabela 3: Parte I - Stakeholders

Categoria	Stakeholders
Primários	Equipa de prescrição/medicação
	Chefe da equipa de prescrição: Orlando Antunes;
	Responsável pela arquitectura Flash da empresa: Rui Dias;
	Responsável pela camada de <i>Flash/Actionscript</i> na empresa: Rui Silva;
	Em última instância, todos os programadores Flash na ALERT.
Secundários	Restantes Membros da ALERT: Programadores, não programadores, dentro e fora dos escritórios.
Terciários	Empresas concorrentes à ALERT, como por exemplo a Siemens ou Microsoft;
	Clientes dos produtos ALERT: administradores de hospitais, médicos e outros profissionais da área da saúde.

Para o desenvolvimento deste projecto foram considerados todos os *stakeholders*, no entanto, foi dada uma maior importância aos primários, por vários motivos: maior influência no projecto, grau de importância, disponibilidade e acessibilidade.

Arquitectura ALERT

Uma vez que este problema foca nos aspectos de comunicação entre base de dados e *user interface*, faz todo o sentido, ao longo desta etapa do projecto, fazer uma breve análise da arquitectura do sistema em causa. Tirando algumas pequenas excepções que fogem um pouco ao âmbito do projecto, a maior parte dos produtos ALERT funciona como um só produto cuja arquitectura que pode ser descrita pelo diagrama da Figura 6.

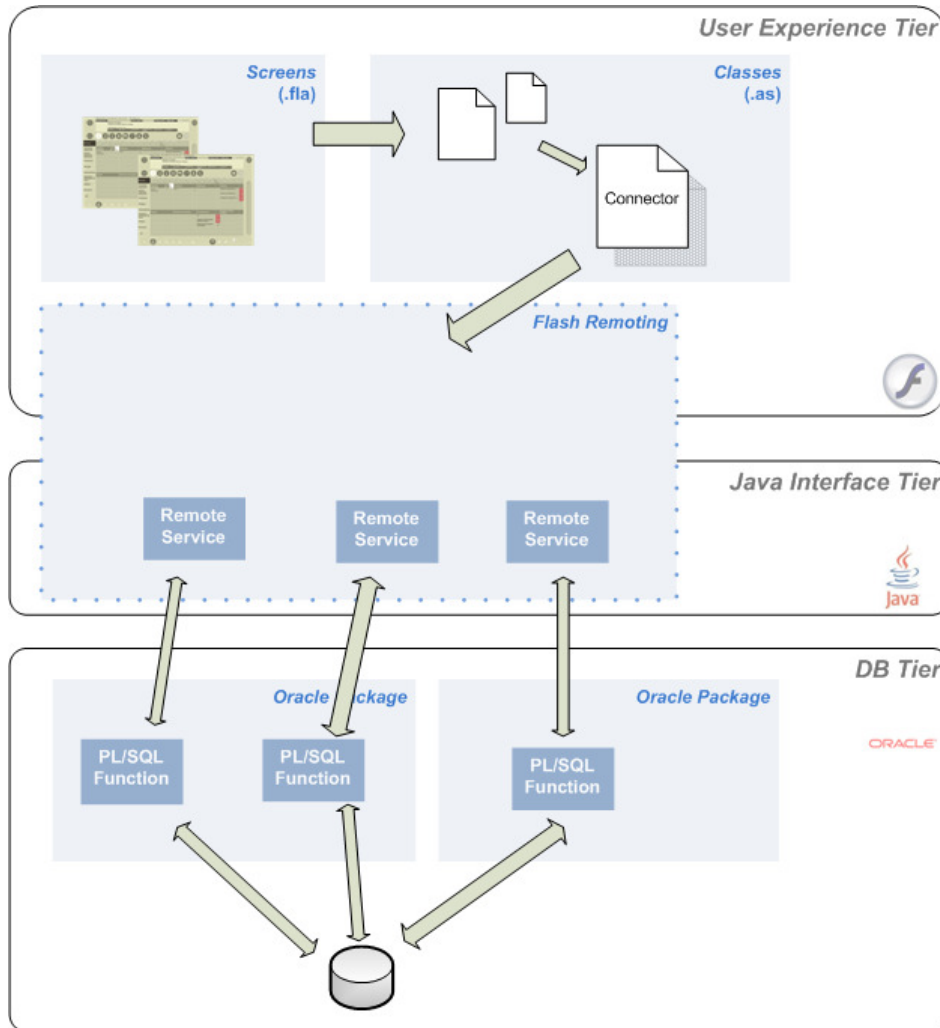


Figura 6: Análise da arquitectura ALERT

Como é possível reparar analisando o diagrama, cada produto ALERT está dividido em 3 camadas de software ligadas entre si.

User Interface/ User Experience

Esta é a camada onde o projecto se foca, no entanto, devido à necessidade de trabalhar numa temática que envolve comunicação entre camadas, há que ter em conta as restantes camadas referidas posteriormente. A interface gráfica do ALERT é dividida em ecrãs (.fla) e as classes *Actionscript* correspondentes(.as), havendo casos em que um ecrã envolve mais do que uma classe, naturalmente. Algumas classes fazem chamadas a dados externos ao Flash, exigindo assim que classe responsável por estas comunicações, a *Connector.as* entre em funcionamento. Aqui entra o conceito de *Flash Remoting*, que é utilizado para chamar os dados remotamente, que deverão chegar ao ecrã de uma forma perfeitamente manipulável.

Interfaces (Java Interface)

Esta camada desenvolvida em Java é principal responsável pela adaptação de vários modelos de dados, resultantes da ligação dos produtos ALERT com outros produtos externos, como é o caso do SAM, por exemplo. Como é sabido, os produtos ALERT surgem num período em que a informática na saúde tem já um estado relativamente evoluído, tendo de assumir assim uma postura responsável e consciente da existência de outras tecnologias. Em vez de trabalhar concorrentemente, a ALERT procura integrar várias soluções existentes, ligando-os aos produtos ALERT.

Uma vez que os produtos ALERT estão preparados para trabalhar segundo um modelo fixo, embora facilmente adaptável a novas soluções, esta camada desempenha um papel de bastante relevo no que toca à integração e ligação a novos sistemas.

No entanto, e focando essencialmente no problema em análise, esta camada é o ponto central da comunicação entre as camada de dados e de *user interface*. Isto acontece principalmente devido às limitações impostas pela tecnologia Flash, que trabalha num ambiente “fechado”, alias, como é referido anteriormente no capítulo relativo à revisão tecnológica. Através do *Flash Remoting*, estabelece-se ligação entre as classes *Actionscript* e os serviços remotos em Java. Estes serviços responsabilizam-se por chamar funções da camada de dados, e retornar valores serializados para a camada de *user interface*, uma vez que seria impossível fazê-lo directamente a partir do Flash.

Futuramente a parte de comunicação poderá ser feita também por *WebServices*.

Data (DB)

Esta camada é responsável por toda a manipulação e gestão de dados dos produtos ALERT. Por efeitos de segurança, encontra-se replicada em vários ambientes: *Development*, *Quality Control* (para efeito de testes) e ainda *Demos* e *Pre-Produção*, para apresentações e simulação de ambiente hospitalar. Para o problema em questão apenas será focado o ambiente *Development*, sobre o qual é feito todo o desenvolvimento de código.

As várias funções PL-SQL são organizadas por pacotes, que vão estar associados aos serviços remotos em Java.

Ambiente interno: Ferramentas de trabalho na empresa

Ao longo deste projecto foram utilizadas uma série de ferramentas de trabalho que são apresentadas de seguida. Durante esta etapa, torna-se relevante observar as ferramentas utilizadas de forma a arranjar uma melhor compatibilidade com a solução que se procura.

Tabela 4: Ferramentas usadas na ALERT

Categoria	Nome da Ferramenta	Descrição
Debug	XRAY	Ferramenta utilizada frequentemente pelas equipas de desenvolvimento <i>Flash/Actionscript</i> que permite consultar alguns valores em tempo real: estados, variáveis e outputs produzidos por uma aplicação <i>Flash</i> . Permite ainda interagir com essa mesma aplicação ao mudar o valor de alguns parâmetros, como por exemplo, o nível de transparência de um cabeçalho. É de extrema utilidade devido à dificuldade que por vezes existe em encontrar um erro no código <i>Actionscript</i> (muita coisa passa como sendo válida quando não deveria ser).
	SERVICE CAPTURE	É utilizado para consultar todo o acesso remoto feito pela aplicação ALERT apresentado todas as chamadas à base de dados bem como valores de entrada e saída, permitindo detectar possíveis erros de comunicação ou a origem de um erro (interface ou base de dados).
Desenvolvimento	FLASH e ECLIPSE	Toda a camada de interfaces gráficas na ALERT foi e continua a ser implementada em Flash (e não em FLEX). Actualmente é utilizada a versão CS3 da Adobe e o código está escrito em <i>Actionscript 2.0</i> , se bem que isto poderá vir a sofrer alterações com a chegada da versão 3.0 à ALERT. O uso desta tecnologia permite uma boa combinação do design criativo produzido internamente com a capacidade de programação orientada a objectos. Apesar de nem sempre ter sido assim, a grande maioria do código encontra-se actualmente organizado em classes e devidamente estruturado de forma a possibilitar uma maior organização e reutilização do mesmo. Se por um lado a programação em <i>Actionscript</i> é mais “solta” ou “livre”, as equipas de desenvolvimento da ALERT apostam em ferramentas de <i>debugging</i> de

		forma a garantir um maior controlo sobre o código que é produzido. Além disso é feito um controlo de versões de forma a garantir um desenvolvimento seguro, o que é sem dúvida uma medida necessária para uma empresa que se encontra em crescimento. Devido às diferenças entre Flash e Flex (apresentadas em anexo), conclui-se que o ambiente <i>Macromedia</i> não é o mais indicado para programação, principalmente quando se trata de aplicações complexas e de grande dimensão. Para contornar esta situação, as equipas de desenvolvimento de interfaces gráficas na ALERT recorrem ao Eclipse para desenvolvimento de código <i>Actionscript</i> . Todo este código é chamado pelo ecrã (.fla) correspondente no <i>Flash IDE</i> . Para a metodologia actual de desenvolvimento são também necessários os seguintes <i>plugins</i> do Eclipse:
	Plugin: SubEclipse	Basicamente é o SVN integrado no Eclipse, portanto, utilizado para controlo de versões de código.
	Plugin: FDT (Flash Development Tool)	<i>Plugin</i> necessário para o desenvolvimento do código <i>Actionscript</i> (As) no Eclipse. Actualmente a versão 3 já suporta não só As2 como As3 também.

Ambiente interno: processo de desenvolvimento de software.

Uma vez que este projecto pretende otimizar todo um processo de desenvolvimento, torna-se imperativo observar e analisar esse próprio processo, de modo a identificar onde é que pode haver melhoramentos. Por questões de organização e para melhorar a qualidade do trabalho, há uma série de procedimentos internos que são obrigatórios para qualquer trabalhador. Ultimamente a importância do cumprimento destes procedimentos tem-se tornado bastante importante, chegando a ser vital para o bom funcionamento da ALERT ¹. Apesar de estes procedimentos afectarem todas as áreas da empresa, este documento vai focar essencialmente em alguns procedimentos mais importantes para desenvolvimento. A Figura 7 indica como se desenrola todo o processo, que é válido para as várias camadas que constituem a arquitectura.

De forma a facilitar a compreensão deste diagrama, este foi dividido 4 partes: Análise e conceptualização, Arquitectura, Implementação e Testes.

¹ De facto, ainda durante o calendário do projecto realizou-se um retiro “anual” (foi excepcionalmente o segundo do ano) durante o qual se discutiram precisamente questões relacionadas com os procedimentos internos e a importância/impacto que podem ter. Esses dias estão contemplados no diagrama GANT no ANEXO A: Diagrama do Planeamento.

Os círculos amarelos assinalam as fases do projecto em que a solução pode de facto intervir e causar algum impact, isto sem ir contra os procedimentos apresentados. De forma a clarificar a ideia que o diagrama representa, são descritas as várias fases do processo:

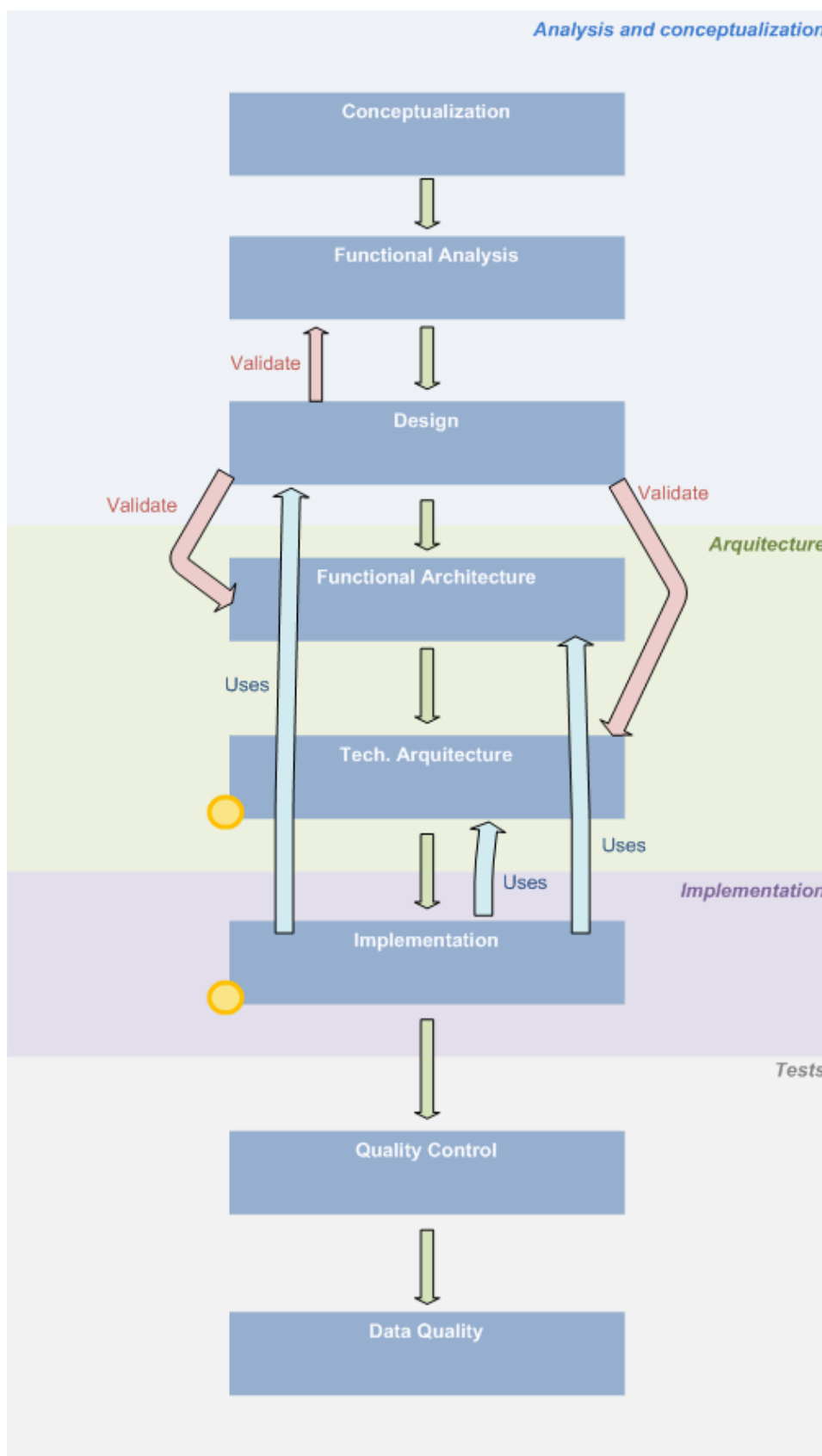


Figura 7: Processo de desenvolvimento ALERT

Conceptualização

Todo o processo de desenvolvimento começa com uma análise das necessidades reais. Este trabalho é realizado pelos principais responsáveis da área do desenvolvimento de software, composta por elementos altamente qualificados dentro da empresa. Estes são responsáveis por apurar as necessidades reais dos clientes e passa-las para uma solução virtual, traçando uma espécie de mapa conceptual da ideia. Nesta fase não se tem em conta detalhes técnicos nem integrações no produto, apenas são traçados os requisitos.

Análise Funcional

Quando os requisitos de uma nova versão chegam às equipas de análise, é feita a análise funcional, altura em que as ideias tomam forma e é visto de que forma é que determinadas necessidades podem ser convertidas em funções ALERT, ainda pondo de parte grande parte dos pormenores técnicos, mas no entanto, evitando entrar em conflito com funcionalidades já existentes nos produtos.

Design

As equipas de design dão forma às novas funcionalidades, tendo por base a estilos já definidos como standard na empresa. Daqui saem desenhos com aspecto final dos produtos ALERT, quando são devidamente aprovados. Apesar de ser da inteira responsabilidade da equipa de design, os desenhos só são válidos assim que forem devidamente aprovados pelos responsáveis da arquitectura funcional, arquitectura técnica e pelo responsável pela equipa de design.

Arquitectura Funcional

Esta actividade é da inteira responsabilidade das equipas de arquitectura, destacadas para o efeito. No entanto, a validação e aprovação dos documentos resultantes deve passar pelas mãos da equipa acima mencionada. A arquitectura técnica é um cenário identico, no que toca à realização e validação dos documentos.

Nesta fase são escritos documentos de arquitectura onde as funcionalidades “conceptuais” são convertidas em funcionalidades devidamente fundamentadas, justificadas, e adaptadas à realidade dos produtos. Muitas decisões a este campo podem afectar desenhos feitos na fase anterior, que resultam na sua reprovação e modificação.

Arquitectura Técnica

Após terem sido definidos todas as novas funcionalidades a implementar e com os desenhos aprovados “na mão”, é produzido um novo documento onde são especificadas todas as necessidades técnicas, sejam estas tabelas de dados, novos componentes visuais (Flash) ou outras. Esta é uma das fases do processo em que a solução apresentada pode trazer vantagens, como será possível verificar mais adianta, neste mesmo documento, no ponto relativo à apresentação da metodologia.

Implementação

Com todas as fases anteriores concluídas é lançada a fase de codificação e implementação, onde inicialmente surgem os problemas de sincronização e onde a solução

pretende actuar de forma mais directa. Seguindo a documentação produzida anteriormente e os desenhos validados, o processo de implementação deve decorrer sem grandes problemas.

Controlo de Qualidade

Após serem concluídos todos os desenvolvimentos, os vários produtos ALERT são submetidos como uma nova versão para o ambiente de *Quality Control*, onde são submetidos a testes rigorosos por uma equipa inteiramente dedicada a essa tarefa. Apenas são considerados os aspectos funcionais, e é verificado se os ecrãs cumprem todas as especificações documentadas ao longo do processo.

Qualidade dos dados

Uma vez que os produtos ALERT actuam numa área cujo vocabolário e estado da arte são bastante complexos, torna-se necessário proceder à validação de todo o conteúdo dos vários ecrãs. Para o SSP a empresa conta com uma equipa dedicada a essa validação, de forma a garantir que quando este ciclo termina, o produto está pronto a ir para o mercado e ser usado em ambientes reais.

Tendo em conta este processo, é possível salientar facilmente a fase de implementação onde o problema realmente se abate, pois é onde a comunicação Flash/BD falha. Como se pode ver no diagrama assinalado a amarelo e como foi referido, há basicamente duas fases onde a solução pode intervir e otimizar o processo de implementação: a própria faz e de implementação e a arquitectura técnica. Neste último caso, a única intervenção relevante será no sentido da documentação de especificação de interfaces e tudo o resto terá impacto na outra hipótese, podendo assim assumir que a verdadeira fase de intervenção seja a fase de implementação.

3.2 Requisitos

Considerando os factores anteriormente referidos, foram atingidos os seguintes requisitos, devidamente categorizados e descritos. Para cada um dos requisitos, é indicada a sua prioridade, tendo em conta a seguinte legenda:

Alta: Bastante importante para satisfazer os objectivos da empresa. Deve ser feito até ao final do projecto imperativamente.

Média: Importante, no entanto se por algum motivo estes requisitos não forem cumpridos parcialmente ou na totalidade, apenas pioram a condição da solução, mas não comprometem o seu funcionamento.

Baixa: De baixa importância. São aspectos engraçados a considerar, no entanto, são apenas extras a considerar no caso de haver tempo disponível para isso.

Os requisitos foram divididos ainda em duas categorias principais: funcionais e não funcionais. Como o nome indica, a primeira categoria está relacionada com aqueles requisitos ligados a funcionalidades, enquanto os restantes se encontram na segunda categoria. No problema em questão, o número de funcionalidades é reduzido, concentrando-se sobretudo em dois grandes aspectos: Simulação e Testes.

Tabela 5: Parte I - Requisitos funcionais

Requisito	Descrição	Prioridade
Capacidade de Simulação	Deve permitir ao utilizador correr os seus ecrãs ALERT como se de facto tivesse ligado a uma função da camada de base de dados. Deve ter em conta todos os dados enviados pelos serviços remotos de modo a simular valores semelhantes.	Alta
Capacidade de Teste	Deve ser capaz de enunciar de forma clara os vários erros da simulação, possibilitando ao utilizador desenvolver o seu ecrã. Deve ter em conta a validação dos tipos de dados de entrada na base de dados.	Alta
Configuração Local	Cada utilizador deve poder fazer as simulações localmente sem ter de depender das condições da rede da empresa.	Alta
Documentação	Deve fornecer apoio no que toca a documentação de interfaces, de modo a facilitar o processo de desenvolvimento. Ou seja, no que toca à solução, esta deve fornecer algum mecanismo para este efeito ou então facilitar a sua ligação com mecanismos externos que permitam obter esta funcionalidade.	Alta
Interacção com o <i>XRay</i>	Se possível, deve apresentar os erros encontrados no <i>XRay</i> , onde costumam ser apresentados.	Média
Interface gráfica visual para configuração de todo o sistema	Apesar de ser uma coisa “bonita” de se ver, com todas as configurações facilmente acessíveis, é algo que consome imenso tempo e não compensa, quando comparado com soluções mais simples.	Baixa

Tabela 6: Parte I: Requisitos não funcionais

Requisito	Descrição	Prioridade
Acessibilidade	O sistema a desenvolver deve ser de acesso fácil para todos os seus utilizadores (empresa), gratuita e preferencialmente disponível a partir da internet ou sistema de partilha interno, de modo a chegar facilmente a qualquer um que queira utilizar.	Alta
Flexibilidade	Deve ser adaptável aos mais diversos ecrãs, de modo a poder integrar novas funcionalidades sem qualquer problema.	Alta
Segurança	Este sistema deve ser seguro ao ponto de evitar ser confundido com o ambiente final do produto, por se tratar apenas de uma ferramenta de desenvolvimento. Desta forma, não deve comprometer os produtos	Alta

	ALERT noutros ambientes nem prejudicar a empresa de forma alguma.	
Transparência e Facilidade de utilização	De modo a reduzir a aceitação pela parte dos utilizadores, este sistema não deve, de forma alguma, dar demasiado trabalho ou ser demasiado complicado, caso contrário, não fará sentido, pois vai apenas causar perda de tempo. Desta forma a solução deve ser <i>user friendly</i> e fácil de utilizar por qualquer utilizador da empresa e deve também fornecer a transparência suficiente para não ser demasiado intrusivo para o utilizador com os conceitos que envolvem este processo de simulação. Para isso deve também evitar grandes alterações no código das classes existentes. Procura-se aqui que qualquer utilizador possa usufruir deste serviço sem grande esforço.	Alta
Formato de documentação: <i>Standard</i>	A solução relativa à documentação apresentada para este projecto deve apresentar os dados num formato standard, de modo a ser facilmente utilizado pela empresa, preferencialmente Word ou PDF. No entanto, a empresa está aberta a outros formatos, desde que sejam compatíveis com Windows (por exemplo HTML).	Alta
Baixos custos de Instalação, Configuração e Manutenção	O impacto na empresa não deve ser demasiado acentuado, evitando grandes alterações dos processos de desenvolvimento. Por um lado iria requerer validações pela parte da empresa que poderiam não ser aprovadas, criando assim maior resistência à aceitação desta nova solução. A instalação, configuração e manutenção devem ser simples e pouco trabalhosas. Tal como os utilizadores finais, o pessoal responsável pela arquitectura Flash da empresa pode não ver com bons olhos uma solução com custos elevados de instalação, configuração e manutenção. Deve assim ter-se em conta os custos não só de instalação deste sistema, mas também de configuração e manutenção (neste último caso devem ser, se possíveis, praticamente nulos).	Média
Formato de documentação: Editável	Se preferível, o formato de documentação usado pela empresa deve ser editável no caso de esta ser gerada automaticamente, de modo a facilitar o processo de edição no caso de ser preciso emendar algum aspecto.	Média
Performance	Não deve afectar, de forma alguma, a performance dos produtos ALERT, mesmo	Média

	que sirva apenas para efeitos de teste e simulação. Deve proporcionar uma experiência o mais parecida possível com o aspecto final, de modo a evitar surpresas e má avaliação de performance dos ecrãs.	
--	---	--

3.3 Solução tecnológica

Pegando no que foi apresentado na revisão tecnológica deste mesmo documento e nos requisitos da empresa, foi feita uma grelha de avaliação de modo a testar a adaptabilidade de cada solução ao problema em análise. Por motivos relacionados com disponibilidade e facilidade de adaptação, o problema foi dividido em dois módulos, correspondendo a dois blocos funcionais distintos. O primeiro diz respeito à capacidade de teste e simulação, enquanto o segundo diz respeito à escolha de uma solução para a geração de documentação.

Solução para Simulação e Testes:

Como foi visto no capítulo do estado da arte do presente documento, foi destacada a ASUnit como framework de testes mais apta para este problema. No entanto, há que considerar ainda a hipótese de desenvolver uma ferramenta de raiz. Dessa forma, foi feita uma avaliação comparativa entre a framework existente, e a suposta solução. Essa avaliação baseia-se num conjunto de parâmetros valorizadores da aplicação, que são apresentados sumariamente:

Custos ICM (Instalação, Configuração e Manutenção): Reforça os aspectos de simplicidade e redução do impacto que a solução possa ter sobre o sistema. Tem em consideração as alterações de código necessárias que devem ser minimizadas, não só porque constituem uma maior perda de tempo como afectam o nível de transparência da funcionalidade.

Capacidade de simulação: Foca sobre um aspecto muito importante que é a capacidade de simulação.

Capacidade de testes: Foca na capacidade de testes da ferramenta, um dos objectivos deste módulo funcional.

Tempo de desenvolvimento/adaptação: Foca sobre um aspecto muito importante que é a capacidade de simulação. Como será visto adiante, não foram encontradas ferramentas que reúnam ambas as capacidades de teste e simulação, constituindo um factor crítico e decisivo.

Tabela 7 - Comparação de frameworks

	Custos ICM	Simulação	Testes	Tempo de Desenvolvimento
ASUnit	Relativamente alto, por se tratar da adaptação a um cenário específico. Exige codificar os testes que se pretende realizar	Não permite simular em ambiente “normal”, quando muito sob forma de teste.	Várias funcionalidades e soluções bem trabalhadas	Baixo, no entanto, os custos de integração ocupam bastante tempo
Solução de raiz	Baixo, por ser uma solução específica e integrada. Pode nem exigir escrita de código.	Pode permitir uma simulação perfeita	Pode permitir testes, mas em princípio serão bem mais simples	Depende da solução, podendo variar entre baixo e elevado

Solução para Geração de documentos:

Este bloco funcional diz essencialmente respeito à documentação de especificação de interfaces. A ideia da solução será gerar toda esta documentação, tendo por base transformações XSLT, aspecto já mencionado na revisão tecnológica deste projecto. Da mesma forma que foi feita uma comparação das alternativas para o primeiro bloco funcional, é agora apresentada a escolha relativamente a este caso. Considerando os requisitos apresentados relativamente à documentação interna, a avaliação foi feita considerando os seguintes factores:

Formato editável: Apesar de ser apenas um requisito de média prioridade, pode ser factor decisivo na escolha da solução.

Facilidade de desenvolvimento: Tempo é dinheiro, e considerando que isto é um bloco funcional de menor dimensão, é importante facilitar o processo de desenvolvimento, evitando grandes obstáculos à realização do gerador.

Acessibilidade: Pelos mesmos motivos apresentados para o tópico anterior. Não adianta escolher uma solução se depois for de difícil acesso ou impossível de utilizar dentro do prazo estabelecido.

Tabela 8: Geração de documentos - Comparação

	Editável?	Fácil?	Acessível?
<i>WordprocessingML Transform Inference Tool</i>	Sim (.doc)	Sim	Sim
<i>ALTOVA</i>	Não (.pdf)	Sim	Não
<i>Ecrion XFDesigner</i>	Não (.pdf)	Sim	Não
<i>XML NOTEPAD 2007</i>	Não (.pdf/.html)	Não	Sim

Como se pode concluir facilmente a partir da tabela de avaliação, a hipótese mais vantajosa é a primeira, sendo facilmente adaptável à solução apresentada. No entanto, é ainda relevante considerar a última hipótese na tabela, não para o fim especificado à partida, mas sim para a realização de operações simples, como por exemplo a validação das instâncias XML com base no XML *Schema* desenvolvido para o efeito, como será detalhado no capítulo relativo à implementação.

3.3.1 Arquitectura

O sistema de simulação/testes proposto, denominado internamente *ALERT Data Abstraction* é composto por dois módulos funcionais: o *Data Abstraction* e o *Data Abstraction Doc Generator*, como pode ser visto na Figura 8. O primeiro é uma aplicação desenvolvida de raiz, com todas as especificações detalhadas mais adiante; o segundo, baseia-se na *WordprocessingML transform inference tool*. Como um todo, este sistema acrescenta novas funcionalidades ao ambiente ALERT, permitindo simular, testar e gerando a documentação necessária para efeitos de especificação de interfaces, no âmbito da comunicação entre camadas.

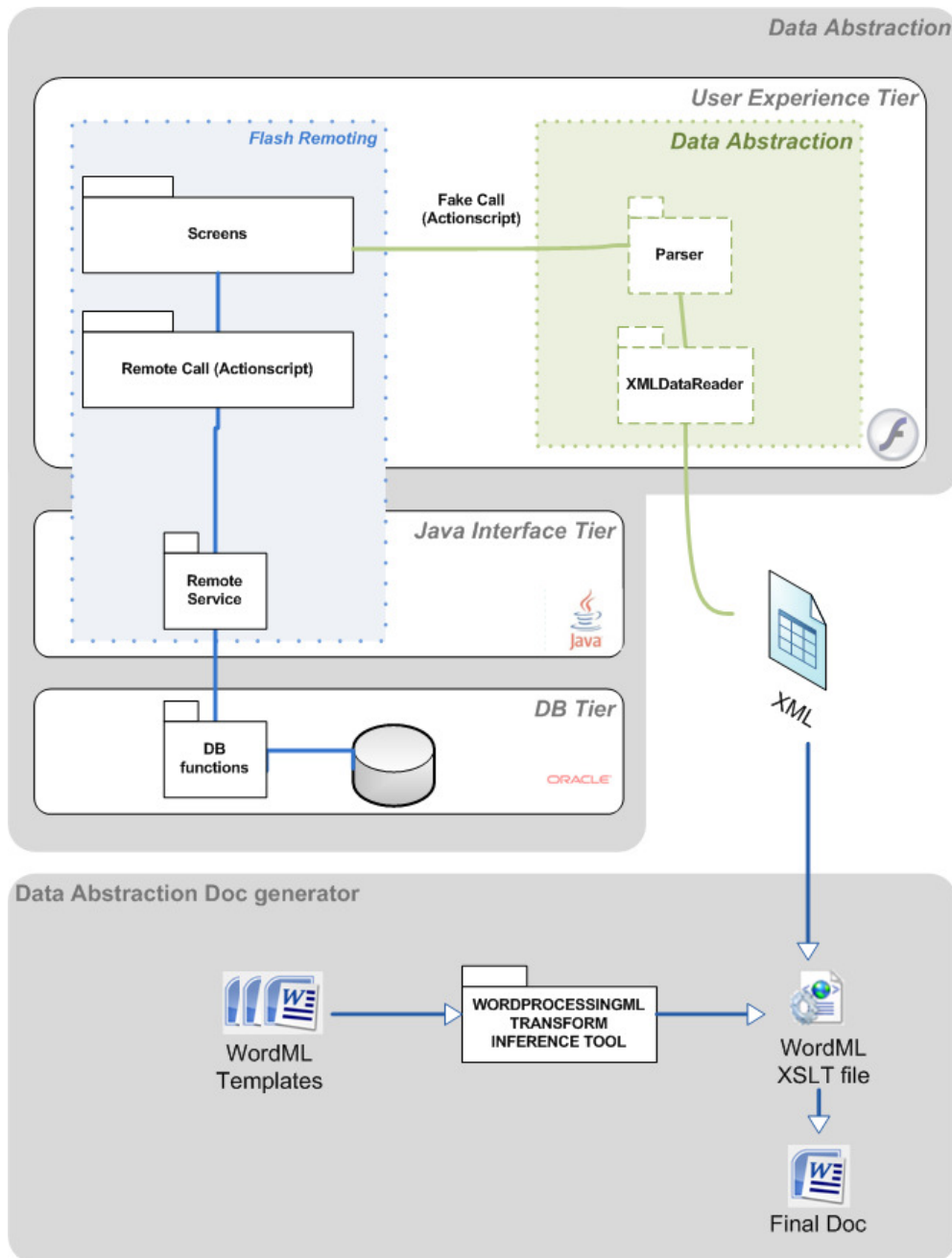


Figura 8: Arquitectura Data Abstraction

XML

A chave encontrada para esta solução, passa pela utilização de ficheiros XML. O motivo da escolha destas estruturas de dados comparativamente a outras alternativas foi já devidamente justificado no capítulo relativo à revisão tecnológica. De facto, e em adição aos motivos anteriores, há que destacar que toda a envolvente ALERT é favorável à utilização desta estrutura. Já outras funcionalidades anteriores, e mesmo operações CORE actuais são implementadas recorrendo a estes ficheiros, tornando-os facilmente acolhidos na empresa, sendo vistos com bons olhos pelos responsáveis pela camada Flash. Para entender melhor o sistema desenvolvido, são apresentados os dois módulos em separado.

Data Abstraction

Este é o módulo mais importante, sendo o principal responsável pela parte de simulação e testes. É o principal responsável por resolver a questão da autonomia entre as camadas. Por se tratar de uma solução de raiz, foi possível atingir um maior nível de integração com o sistema ALERT, chegando a um nível de transparência bastante satisfatório. A ideia do *Data Abstraction*, na verdade, é bastante simples. Actualmente, a comunicação é toda feita a partir do *Flash Remoting* (ou considerando aspectos futuros, podemos ainda ter em conta os *WebServices*) e toda a informação é armazenada em bases de dados. Tendo como referência a Figura 8, no que diz respeito ao *Data Abstraction*, se antes o funcionamento seguia o “caminho” marcado a azul, agora com esta funcionalidade irá seguir o “caminho” verde, evitando as chamadas remotas.

Segundo a mesma linha de raciocínio dos *Mock Objects*, o que verdadeiramente interessa para efeito de testes será validação da chamada remota. Ou seja, não interessa qual é o valor de retorno, assumindo que este é sempre válido. O importante será verificar se os valores de entrada são válidos. No entanto, por uma questão de controlo e considerando a necessidade de simulação, estes valores de retorno devem ser perfeitamente configuráveis pelo utilizador, ao especificarem os seus ficheiros XML.

Na verdade, do ponto de vista de utilização, o utilizador tem de se preocupar pouco ou nada com a simulação do seu ecrã, pois, simulando um serviço interno, o *Data Abstraction* não provoca alterações no código do ecrã, a não ser claro, a especificação de um serviço diferente, de forma a ser feita a chamada correcta às classes responsáveis pela simulação e teste. Assim sendo, a única coisa que o utilizador deve ter verdadeiramente em conta é a especificação de um ficheiro XML válido para futura utilização, tanto a nível de documentação como desenvolvimento.

Para validação destes ficheiros, foi concebido ainda um XML *Schema*, de forma a garantir o funcionamento dos XML após validação correcta.

Data Abstraction Doc Generator

Este módulo é de menor foco e dimensão, no entanto é uma ferramenta poderosa e de grande utilidade no contexto em que se insere, solucionando a questão do sincronismo entre as camadas. Trata-se de uma adaptação de uma tecnologia já existente e apresentada no capítulo do estado da arte deste documento, o *WordprocessingML transform inference tool*.

Como já foi referido, este *plugin* desenvolvido para o *Microsoft Word* permite criar automaticamente ficheiros XSLT a partir de templates em *WordML*. Os motivos da escolha desta tecnologia são os já referidos anteriormente, no capítulo do estado da arte. Apesar desses motivos é importante referir que a utilização de documentos Word na empresa é um ponto a favor.

No contexto da solução para o problema, a ferramenta é utilizada para gerar a documentação de especificação de interfaces a partir das mesmas instâncias que são utilizadas pelo *Data Abstraction*. Ou seja, ao mesmo tempo que o programador cria a estrutura de dados para a simulação do ecrã em desenvolvimento, está também a documentar e especificar todos os parâmetros de entrada e saída de dados, de forma organizada e perceptível. Nesse sentido apenas tem de recorrer ao ficheiro de transformação concebido no âmbito deste projecto. A forma como isso é feito será detalhada mais adiante, no capítulo relativo à implementação da solução.

3.4 Metodologia a aplicar

A solução encontrada para este problema não se resume a adoptar uma ferramenta implementada desde raiz em busca dos requisitos estabelecidos. É importante enquadrar esta mesma solução e adapta-la ao próprio processo de desenvolvimento: essa é a verdadeira solução encontrada. A performance da ferramenta de simulação/teste vai depender da forma como é utilizada na empresa.

Assim sendo, é importante definir uma estrutura adequada e uma arquitectura coerente com o ambiente em que se insere, bem como definir um processo no qual estas novas funcionalidades se enquadrem.

Procurando cumprir os requisitos especificados para o problema, foi delineada uma metodologia para a implementação da solução num caso prático.

Como já foi explicado, o problema deste trabalho foca essencialmente em dois aspectos de comunicação entre as duas equipas, solucionados por módulos funcionais do *ALERT Data Abstraction*. Aplicando na prática, o processo deverá decorrer como apresentado na Figura 9.

Contrariamente ao processo de desenvolvimento que foi analisado anteriormente, mas sem violar nenhum procedimento da empresa, é feita uma intervenção precisamente na fase de implementação. Isto de modo a otimizar e rentabilizar o tempo de implementação. O processo cumpre essencialmente os seguintes passos:

1. Construção e validação dos ficheiros XML;
2. Geração da documentação de especificação de interfaces a partir dos ficheiros XML;
3. Ajuste da documentação numa discussão prévia entre as camadas de *user interface* e de dados;
4. Arranque da fase de desenvolvimento do ecrã, recorrendo ao *ALERT Data Abstraction* de modo a simular os dados recebidos pela base de dados;
5. No final do desenvolvimento de ambas as camadas, fazer as pequenas alterações de modo a ligar as camadas (se os primeiros passos foram realizados correctamente, este último passo deve ser directo e sem qualquer conflito);

O arranque da fase de desenvolvimento é simultâneo para ambas as camadas de desenvolvimento da equipa. No entanto, antes de arrancar com a codificação propriamente dita, será necessário considerar todos os aspectos de interface entre as duas camadas, de modo a garantir sincronismo durante o resto do processo e evitar surpresas desagradáveis.

Recorrendo ao *Data Abstraction Doc generator*, será mais simples elaborar a documentação necessária, no entanto, requer ainda que seja construído o(s) ficheiro(s) XML pelos programadores *Flash*. Após ser devidamente validado pelo XSD Schema, é gerado o documento e ambas as equipas analisam em conjunto de modo a fazer pequenos ajustes, passando então para a fase seguinte do processo. Durante o processo de codificação, focando

mais no desenvolvimento Flash/Actionscript, são primeiro desenvolvidas todas as funcionalidades utilizando os XML elaborados anteriormente, e só depois de serem devidamente testadas pelos autores é que são dadas como prontas, para posteriormente ligarem às funcionalidades correspondentes da camada de dados. Finalmente, ambas as camadas iniciam o processo de debug em conjunto. Sempre que necessário pode-se recorrer ao *Data Abstraction* para fazer testes isolados às funcionalidades Flash, mas só deverá ser necessário em casos externos.

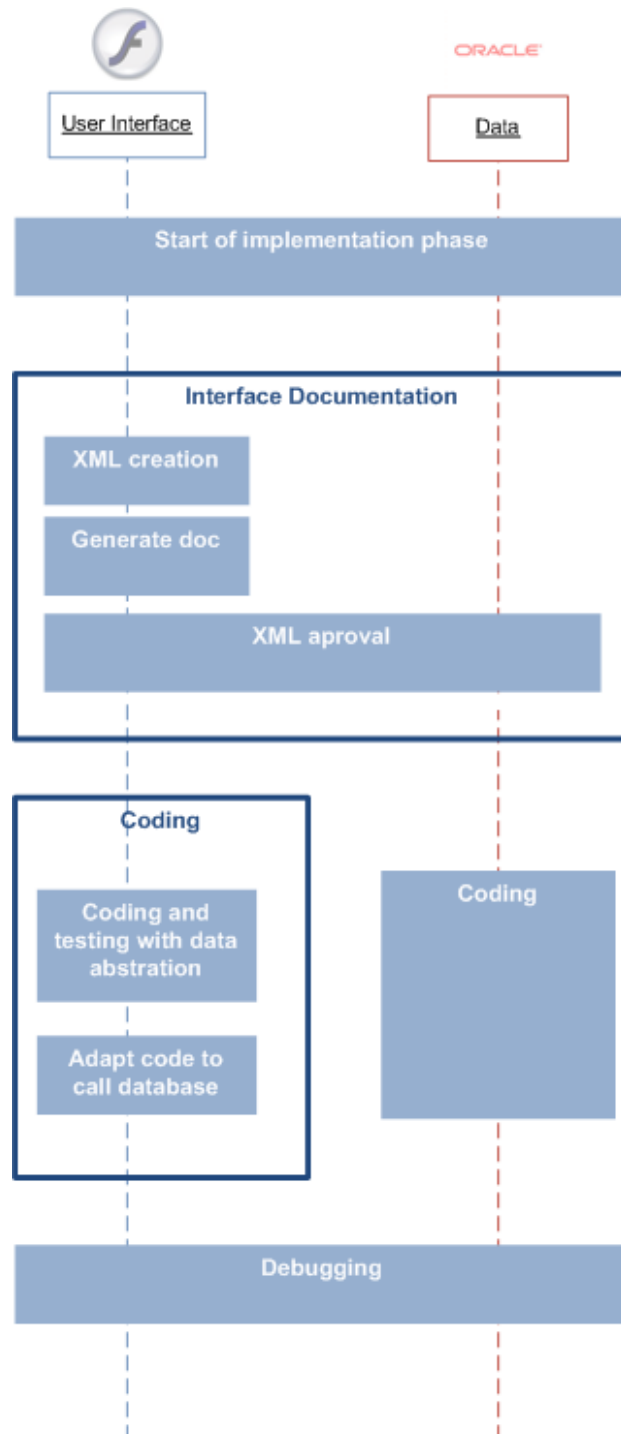


Figura 9: Diagrama representativo da metodologia

Capítulo 4

Implementação da solução

Neste capítulo são descritos os aspectos ligados à fase de implementação e desenvolvimento. Numa primeira abordagem, é focada a solução Data Abstraction, sendo o principal tema deste trabalho. No entanto, seguindo a metodologia de implementação apresentada no capítulo 3 são ainda descritas as implementações das funcionalidades ALERT (versão 2.4.3) como casos de estudo da solução anterior.

4.1 Estudo do sistema e impacto da solução

Quando se fala em desenvolver uma solução integrada num sistema tão completo como o de uma empresa, o primeiro passo não pode ser outro senão a compreensão do próprio sistema e da forma como as coisas estão implementadas. Tecnicamente falando, não adianta partir para desenvolvimento de um *parser* sem saber ao certo quais são os parâmetros a considerar. O processo de integração é crítico para o bom funcionamento desta solução, pelo que teve de ser cuidadosamente estudado, de forma a tirar máximo proveito do *Data Abstraction* com o menor custo de impacto possível.

A primeira coisa a considerar foi a comunicação entre as camadas, neste caso a partir do *Flash Remoting*. Nesse sentido teve-se em conta o diagrama apresentado na Figura 10 e alguns exemplos de implementação utilizados como casos de estudo.

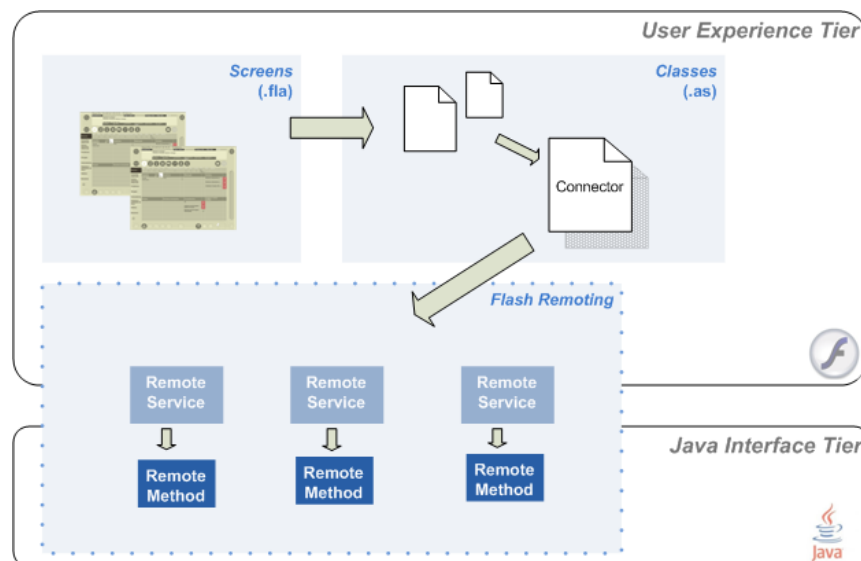


Figura 10: Comunicação Remota

A Figura 10 foca esquematicamente na forma como é feita a comunicação. Basicamente todo o código é remetido para a classe *connector.as* a partir da qual é feito todo o processo de ligação ao *flash remoting*, abstraindo os programadores desse facto. O que o programador deve realmente ter em conta é a chamada ao *connector*. Essa chamada pode ser feita de forma directa, ou indirecta, dando origem a alguns cenários diferentes conforme o ecrã em causa.

Simplificando, há 4 cenários possíveis representados na Figura 11. Todos eles recorrem à chamada directa, no entanto, de forma a reutilizar código e facilitar a construção dos ecrãs, foram desenvolvidas algumas classes auxiliares que aumentam o grau de abstracção no que toca ao acesso remoto. No caso 1, a chamada é feita directamente, no entanto, as outras 3 constituem um cenário diferente, pois recorrem a classes auxiliares criadas pela empresa de modo a facilitar o desenvolvimento. Assim sendo, a solução a desenvolver deve considerar todos estes cenários, sem que o utilizador seja forçado a utilizar um determinado método específico.

A solução apresentada visa ser o mais transparente e integrada possível, e dados os factores apresentados, a única alternativa viável consiste em actuar sobre o *connector*. Como pode ser observado na Figura 11, comprova-se que o primeiro ponto comum entre as várias alternativas é precisamente esta classe de ligação. Deste modo, qualquer alteração a nível do *connector*, será perfeitamente adaptável a todos os cenários, sem que o programador tenha de se preocupar com isso.

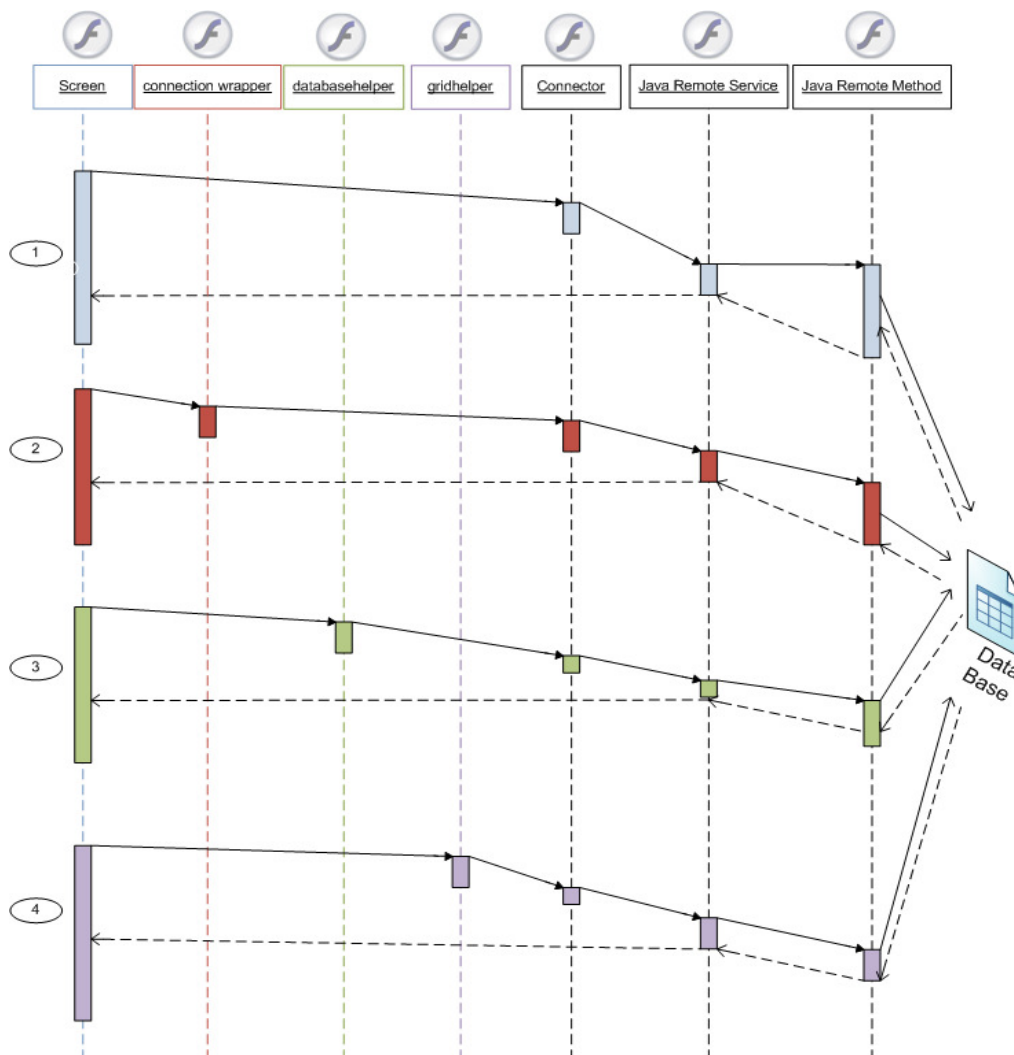


Figura 11: Chamadas remotas

Serviços Remotos

Por se tratar de uma zona crítica, foi necessário minimizar estas alterações. Como mostra na Figura 10, cada chamada do *connector* aponta para um serviço, e dentro desse serviço para um método remoto. Por motivos de organização e capacidade de abertura e crescimento, todos os serviços são definidos nas configurações dos produtos ALERT. Estas configurações são armazenadas em ficheiros XML e são carregadas na inicialização da classe, como representado na Figura 12. Durante este processo, entre outras inicializações irrelevantes para o cenário em questão, é carregada uma lista de serviços remotos armazenada no *CoreServices.xml*. Para além do nome, é também especificado o tipo de serviço, permitindo ao *connector* identificar se deve chamar o *flash remoting*, ou, considerando implementações futuras, um *webservice*. De forma a introduzir a nova funcionalidade, foi criado um novo tipo de serviços que remete para o *Data Abstraction*. Desta forma, as alterações da classe de ligação são mínimas, pois esta apenas terá de fazer uma chamada da classe responsável por iniciar o processo de simulação e teste, devidamente introduzida no sistema ALERT.

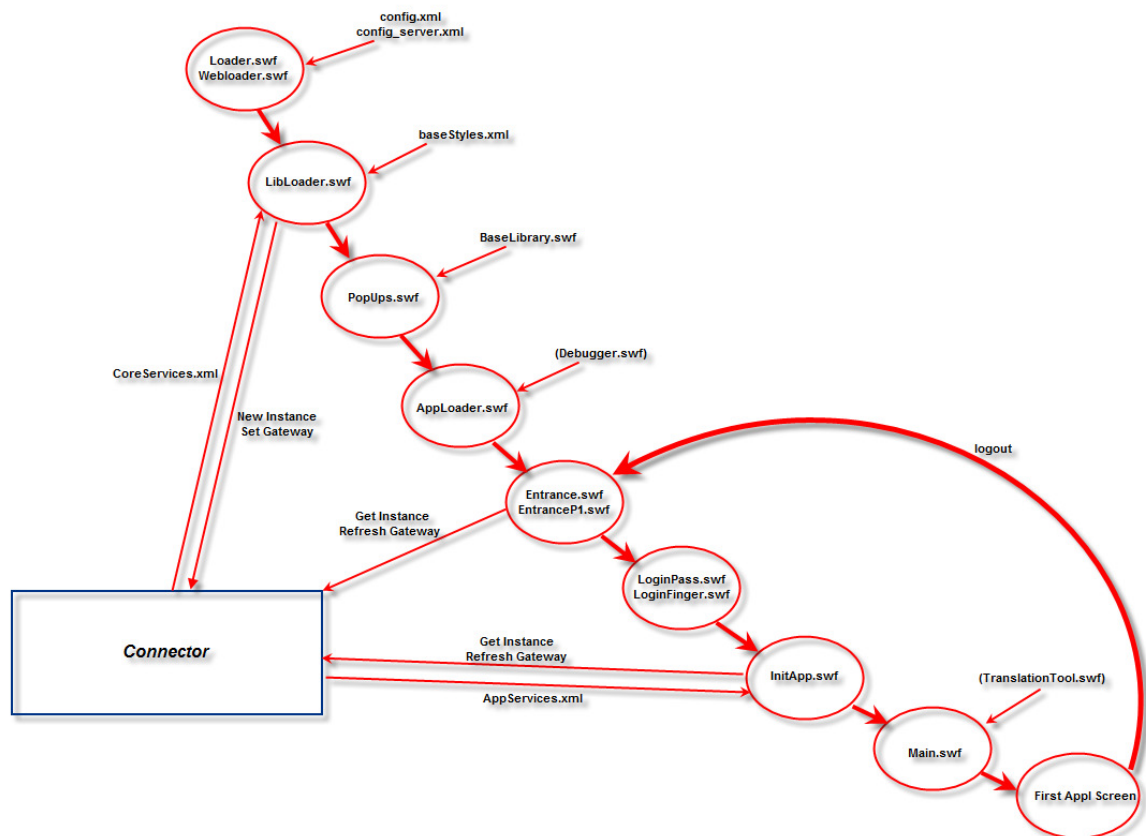


Figura 12: Inicialização da classe *Connector*

Funcionamento Interno do *Data Abstraction*

Tendo por base o que foi dito anteriormente, o *Data Abstraction* é considerado internamente como mais um serviço remoto, com a simples excepção que não é um serviço remoto, e é tudo que o programador precisa de saber. À parte da definição prévia dos ficheiros XML, o utilizador deve considerar que o *Data Abstraction* é apenas mais um serviço, porque no fundo o seu funcionamento é tratado de forma análoga. Em vez de ser chamado um serviço remoto, o *connector* vai interpretar o pedido e vai redirecciona-lo para o *parser Data Abstraction*, responsável por interpretar os ficheiros XML previamente definidos pelo programador. Após ter sido feito o *parsing* do ficheiro de acordo com a chamada executada pelo ecrã, pode acontecer os seguintes cenários:

1. Todos os requisitos de funcionamento são cumpridos. Neste caso o *Data Abstraction* vai criar estruturas de dados semelhantes ao que seria devolvido pelo *Flash Remoting* e chama directamente o ecrã que enviou o pedido;
2. Algumas das validações falham, e nesse caso são enviadas mensagens de erro que são capturadas pelo *XRay*;
3. É detectado que o ambiente de trabalho não é o de desenvolvimento, isto é, que está a correr noutro ambiente como *quality control* ou até mesmo no cliente. Neste caso é enviada uma mensagem de alerta para o ecrã avisando que aquela funcionalidade está a actuar e não devia, por se tratar de uma ferramenta de apoio ao desenvolvimento apenas;
4. O serviço não está especificado no sistema ALERT ou o XML não está devidamente configurado. Nesta situação a aplicação não vai funcionar e é enviada uma mensagem de erro para o *XRay*.

De forma a comprar com o processo anterior representado na Figura 13, é apresentado um esquema semelhante focando no novo processamento interno que substitui as chamadas remotas, permitindo simular as funcionalidades na perfeição e testar o funcionamento do código.

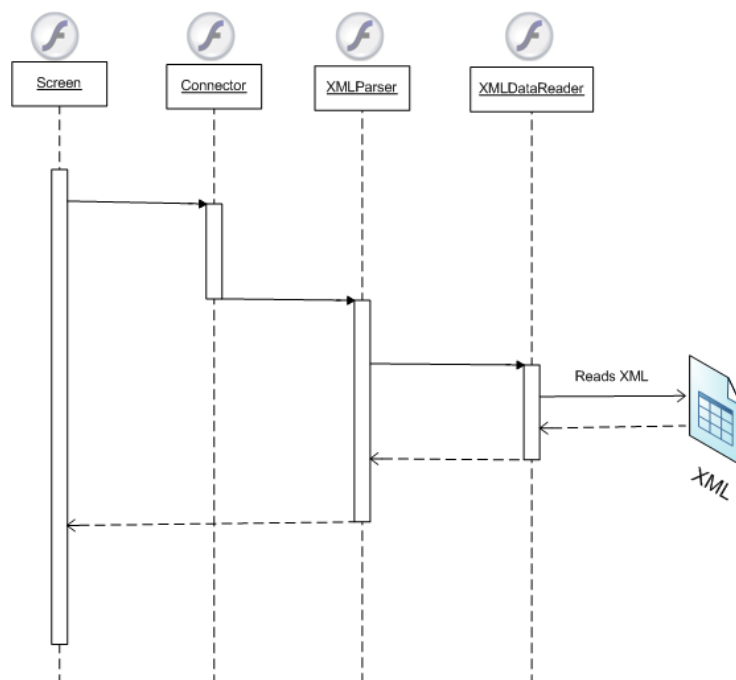


Figura 13: Funcionamento do *Data Abstraction*

4.2 Definição da estrutura de dados

No panorama inicial, as várias funcionalidades *user interface* apenas tinham como fonte de conteúdo a base de dados remota ou implementações “*hard-coded*”. Por “*hard-coded*” entenda-se a implementação de dados massivos no código de forma a forçar que uma dada funcionalidade tenha o retorno de determinados valores. Como é sabido esta solução é bastante trabalhosa e pouco usável. Dessa forma, foi necessário definir uma estrutura alternativa, capaz de simular na perfeição estes mesmos conteúdos. Como foi dito anteriormente ao apresentar o sistema *ALERT Data Abstraction*, a chave de toda esta solução baseia-se na escolha da estrutura de dados: o XML. Devido a motivos já referidos, esta estrutura de dados adapta-se ao problema em questão de uma forma bastante favorável. Em vez de pesquisar uma base de dados remotamente, os ecrãs são remetidos para ficheiros locais, cujo conteúdo é definido pelo próprio programador, reduzindo o tempo de acesso e tornando a aplicação independente da rede, o que é sempre um ponto a favor.

4.2.1 Tipos de dados

Se a solução passa por simular os valores obtidos remotamente, o primeiro passo consiste em identificar estes mesmos valores de forma a poder reproduzir uma cópia deste serviço, “enganando” as aplicações de *user interface*. Nesse sentido foram feitas entrevistas com os técnicos responsáveis pela camada *Flash*, de modo a fazer um levantamento eficaz de todos os tipos usados nestas comunicações. Em adição, foi feito um estudo baseado numa amostra de ecrãs reais dos produtos *ALERT*, recorrendo uma das ferramentas de *debug* já apresentada anteriormente: o *service capture*. Como já foi explicado, esta ferramenta permite interceptar dados de comunicação remota entre a aplicação *Flash* e componentes externos, capturando desta forma as chamadas remotas à base de dados. Tendo em conta este estudo, foram obtidos os seguintes resultados, ordenados de forma por grau de importância (partindo do mais importante).

Tabela 9: Valores de saída na BD (output)

Tipo (Flash)	Tipo (Oracle)
RecordSet	Cursor
String	Nvarchar
Number	Plsinteger/Number
Array	Array
Boolean	Boolean

Tabela 10: Valores de entrada da BD (input)

Tipo (Flash)	Tipo (Oracle)
String	Nvarchar
Number	Plsinteger/Number
Array	Array

Recordset/Cursor

Devido à sua complexidade e importância, este tipo de ficheiros deve ser analisado com especial atenção. O *Recordset* é a estrutura de dados utilizada em *Actionscript* com a capacidade corresponder à estrutura análoga da camada de dados: o cursor. Um cursor, na gíria das bases de dados é das estruturas mais complexas. A ideia deste formato é conseguir referir resultados de *queries* de pesquisa (*selects*), como uma *view*, mas em vez de armazenar a informação, baseia-se em apontadores internos, evitando duplicar dados. Ao transmitir para a camada de *user interface*, a camada de Java trata de armazenar os dados referidos no cursor no *Recordset*, tentando reconstruir a imagem de uma tabela a partir dos conceitos de *array* e *objectos*. Desta forma, o *Recordset* pode ser visto como uma matriz de dados. No entanto, em vez de se utilizar um *array* bidimensional, cuja referência é feita apenas a partir dos índices, é utilizado um *array* de *objectos*, facilitando este processo: é mais intuitivo e livre de erros referir o atributo “nome” na linha 2, do que a posição (2,4) de uma matriz.

String, Number, Boolean e Arrays

Como os próprios nomes indicam, os primeiros 3 formatos de dados são bastante directos e simples, contendo pouca quantidade de informação. Por este motivo, são também usados com bastante frequência, principalmente *Numbers* e *Strings*. Um *Number*, como o nome indica é um formato numérico utilizado pelo *Actionscript* que representa um número real. Uma *String*, é simplesmente uma cadeia de caracteres, análoga ao *Nvarchar*, enquanto o *Boolean* fala por si, tratando-se de um valor booleano (verdadeiro ou falso). Os *arrays* correspondem a vectores de dados, utilizados para transmitir sequências de valores.

4.2.2 Estrutura dos ficheiros XML

Considerando os parâmetros de comunicação referidos, o próximo passo foi construir uma estrutura capaz de representar toda a informação necessária para a simulação de uma determinada funcionalidade. A estrutura resultante, na base do funcionamento do *Data Abstraction* pode ser representada da seguinte forma:

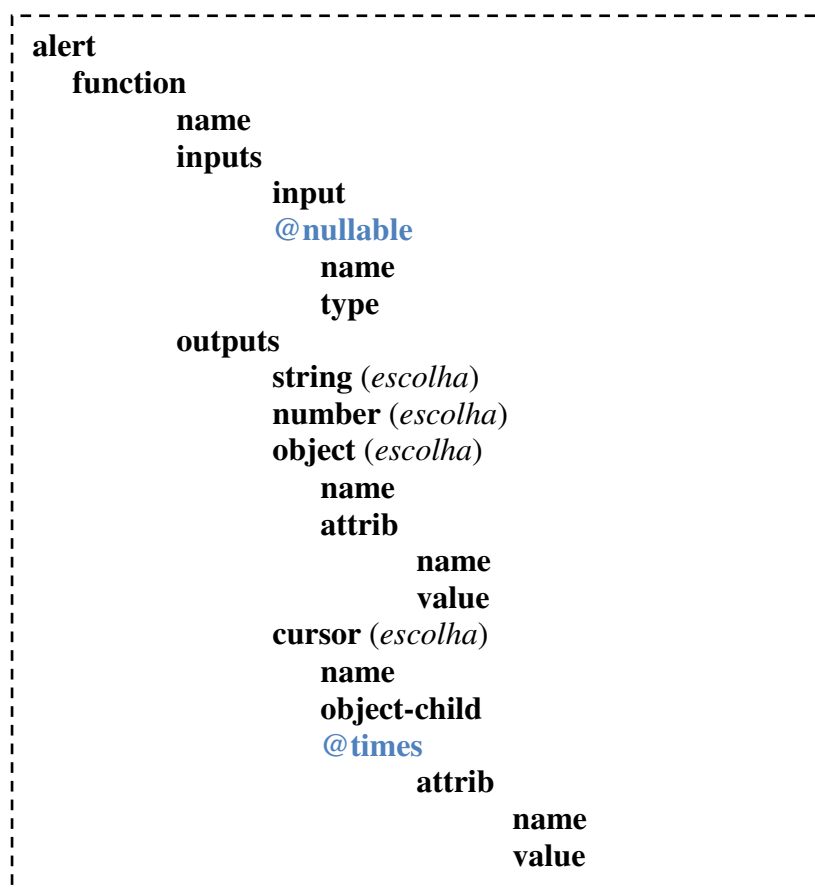


Figura 14 – Estrutura do XML

A primeira decisão a fazer foi escolher o nível adequado para a base da estrutura. De forma a tornar estes ficheiros adaptáveis e flexíveis, não foi imposto qualquer limite de funcionalidades por ficheiro. Na verdade, um ficheiro *data abstraction* tanto pode corresponder a um ecrã inteiro, como a um conjunto deles, ou apenas a algumas funcionalidades, ficando ao critério do utilizador.

Para cada funcionalidade, foi criado o elemento *function*. Dentro deste elemento é que são definidos todos os aspectos relacionados com uma dada chamada remota, desde os seus parâmetros de entrada, de forma a poder testar a validade dos mesmos, como os parâmetros de saída, que a aplicação deve simular e devolver ao ecrã responsável pelo pedido. Foi convencionado que um parâmetro de entrada, é aquele que é enviado do *Flash* para a base de dados, enquanto o de saída corresponde ao sentido inverso. Além destes dois aspectos, representados sob forma dos elementos *inputs* e *outputs*, é necessário ter em conta um terceiro elemento de extrema importância: o *name*. À semelhança do *attrib* e do *value*, o *name* foi inicialmente definido como sendo um atributo, no entanto, para facilitar aspectos ligados ao design na conversão de XML para documentos Word, foi necessário convertê-los para elementos. Como “filho” do elemento *function*, o nome atribuído deve corresponder exactamente ao nome referido no código *Actionscript*, pois apenas deste modo é possível simular uma situação idêntica.

A nível dos *inputs* é apenas considerado relevante armazenar os nomes e tipos de dados de cada parâmetro, que é a informação necessária para se poder testar a validade da chamada remota. O conteúdo destes parâmetros apenas seria relevante em contexto de simulação, que está relacionado com os *outputs* e não com os *inputs*. O atributo *nullable* indica a possibilidade de enviar um determinado parâmetro a nulo ou não, útil para a

validação dos tipos. No que toca aos tipos, são suportados *String*, *Number* e *Object*, em que objecto corresponde a um *Array* (foi necessário proceder à alteração do nome, devido ao reconhecimento de tipos internos do *Actionscript*), tal como foi especificado anteriormente.

A nível dos *outputs* foi criada uma estrutura mais complexa, de modo a poder representar com algum detalhe os conteúdos a simular pela aplicação. Cada output corresponder a 1 de 4 tipos diferentes: *Number*, *String*, *Object* ou *Cursor*, correspondendo aos tipos utilizados para efeitos de simulação. O *Boolean* foi excluído, porque costuma ser utilizado apenas para indicar o correcto funcionamento da comunicação remota, que não se aplica ao caso. Enquanto os três primeiros tipos são relativamente directos e fáceis de entender, o *Cursor* é um bocado complexo, e ao nível de estrutura XML pode ser visto como uma sub árvore contendo vários *object-child*. Este elemento é uma réplica do *object*, com duas diferenças essenciais: a primeira, é a questão da identificação usada para referir os objectos do lado do *Actionscript*, perfeitamente descartável neste caso, uma vez que o cursor já foi devidamente identificado no elemento *name*; a segunda é a capacidade de “replicação” obtida pelo atributo *times*. Este atributo indica à aplicação o número de *object-childrens* semelhantes que quer replicar dentro do cursor, sendo uma funcionalidade particularmente útil para efeitos de população de grelhas.

Toda esta estrutura é validada por um XML *Schema* que é fornecido ao utilizador da aplicação, juntamente com um exemplo de uma instância XML *Data Abstraction*. A garantia de funcionamento é dada, desde que seja feita a validação segundo o especificado.

4.3 Geração de documentos

Aos construir a estrutura dos ficheiros XML foi necessário ter em conta outra funcionalidade como solução que combata o problema da sincronização entre camadas: o *Data Abstraction Doc Generator*. De facto, se for considerada a hipótese de utilizar os ficheiros XML como base da geração dos documentos de especificação de interface, é importante considerar se estes ficheiros contêm toda a informação necessária às duas camadas.

Para especificar uma interface é necessário especificar duas coisas: os parâmetros de entrada e os de saída. No que toca ao detalhe desta informação, não interessa saber o conteúdo, uma vez que este serve para efeitos de simulação. No entanto interessa considerar os tipos de dados e os nomes a utilizar, uma vez que devem corresponder de forma a não levantar problemas durante a comunicação remota. Numa primeira iteração da estrutura dos XML, enquanto os *outputs* eram devidamente identificados, para serem reconhecidos pela aplicação, os *inputs* já não. Assim sendo foi necessário acrescentar esse elemento. Graças às funcionalidades do MSWord, o utilizador pode aplicar um determinado ficheiro de transformação XSL a um ficheiro XML, convertendo num documento devidamente formatado.

O processo de adaptação consiste basicamente na criação deste ficheiro XSL, recorrendo à *WordprocessingML Transform Inference Tool*. Como já foi dito, esta ferramenta permite definir como template um ficheiro XML previamente formatado em word 2003 ou 2007.

Utilizando esta técnica, o autor do documento cria um template simples mas de fácil leitura (exemplo de resultado disponível no

ANEXO B: Data Abstraction Doc). Infelizmente, esta funcionalidade constituiu uma barreira no que toca à representação de atributos, levando a algumas modificações na estrutura inicial do XML mas sem que isso afecte o funcionamento da aplicação: apenas uns poucos atributos convertidos em elemento, nomeadamente os elementos *name*. Outra alteração foi remover a referência para schema, para evitar ter de o referir a partir do word cada vez que é feita a transformação. Além disso, se a amostra utilizada para a definição do template contemplar os cenários possíveis, a ferramenta em causa encarrega-se de construir internamente um schema suficiente para efeitos de conversão.

4.4 Aplicação da metodologia ao desenvolvimento de funcionalidades internas ALERT Prescription 2.4.3

Durante o período especificado para a realização do projecto não foram feitas apenas implementações no sentido de desenvolver o sistema *Data Abstraction*. Na realidade, como membro da equipa de medicação/prescrição, foi necessário implementar algumas alterações para a versão 2.4.3 dos produtos ALERT. Estas implementações encaixam no projecto como casos práticos da aplicação da solução encontrada, ajudando também no que toca à análise de resultados e testes de desempenho. Desta forma, são também apresentadas estas mesmas implementações no âmbito do projecto.

4.4.1 Especificação das funcional

Se por um lado o *Data Abstraction* traz a capacidade de otimizar o processo de desenvolvimento da equipa, a optimização propriamente dita depende da correcta utilização e adaptação a essa funcionalidade. Considerando não só os objectivos iniciais do projecto mas também a necessidade de comprovar as melhorias que se pretende obter, é de toda a relevância considerar casos práticos para aplicação desta ideia.

Enquanto membro da equipa de medicação/prescrição, foi necessário proceder à implementação de novas funcionalidades, adaptando os ecrãs de prescrição à nova realidade do mercado americano e algumas funcionalidades novas para melhorar a interface com o utilizador.

Ao contrário do que se verificou nas implementações anteriores, estas funcionalidades focam-se num âmbito um pouco diferente, nomeadamente no desenvolvimento de interfaces gráficas. Como elemento da equipa de medicação, as tarefas em conta para este problema são:

- Implementação da funcionalidade *Co-sign*, nos ecrãs de medicação;
- Adaptação dos ecrãs de medicação e de prescrição ao novo estilo de interacção 2.4.3;
- Desenvolvimento do novo componente de pesquisa.

Por se tratar de algo que afecta directamente o produto final e que se enquadra perfeitamente no processo de desenvolvimento, foi dada uma menor “liberdade” no que toca a aspectos de implementação. Relembrando o ciclo de desenvolvimento referido anteriormente, é possível identificar facilmente as fontes das especificações necessárias para estas tarefas:

- Documentação de análise funcional;
- Desenhos validados para a versão;
- Discussão com os restantes colegas da equipa, no âmbito do desenvolvimento do documento de arquitectura funcional;

No âmbito da implementação destas funcionalidades, foi necessário fazer um estudo dos ecrãs da equipa, considerando vários cenários e conceitos na área da prescrição, de forma a entender as alterações a realizar. No capítulo 2, são apresentados esses mesmos conceitos, com o detalhe suficiente para enquadrar o leitor.

4.4.2 Implementação da funcionalidade Co-sign

Uma das implementações no âmbito da equipa de medicação/prescrição foi a incorporação de da funcionalidade co-sign. Como já foi explicado no estado da arte deste documento, o co-sign ou abaixo-assinado é uma realidade do mercado EUA. Consiste na capacidade do médico poder delegar a tarefa de prescrição medicamentosa a um enfermeiro, responsabilizando-se pelo acto. Para a implementação deste conceito no produto, foi necessário alterar alguns ecrãs, semelhantes ao que é apresentado na Figura 15.

The screenshot displays a medical application interface. At the top, it shows patient information: 'Internamento', 'Enfª Sousa', 'David Santos', 'Primeira consulta: Saúde de adultos', and 'M / 56'. Below this is a table with columns: 'Processo', 'Episódio', 'Localização', 'Agendamento', 'Efectivação', and 'Em espera'. The 'Localização' column shows 'Enfermaria 2'. Below the table is a row of icons. The main area is a table with columns: 'Tipo', 'Fármaco', 'Instruções', and 'Problemas'. The 'Fármaco' column shows 'Tiamina 100 mg Comp'. The 'Problemas' column shows 'Gravidez 14-Fev-2005', 'Angina de peito 18-Mar-1992', 'Hipertensão 04-Jan-1999', and 'Asma 30-Nov-2004'. A red dashed box highlights a 'DRAGABLE WINDOW' for editing the medication order. Red arrows point to the 'Ordered by*', 'Order time*', and 'Order type*' fields. The 'Order time*' field shows '16:00 09-Nov-2007'. The 'Order type*' field is empty. The 'Notas' field is also empty.

Figura 15: Implementação visual do co-sign

Foi necessário ter em conta verificações ao nível das permissões e linguagem da aplicação, adaptando à *dragable window* (assinalado na Figura 15). Normalmente só é necessário utilizar o campo “instruções” e “Notas”. No entanto, para suportar a nova funcionalidade foram acrescentados 3 novos campos (marcados com setas).

Devido às alterações e ao número elevado de ecrãs afectados por esta funcionalidade, é de salientar a utilidade do *Data Abstraction*, uma vez que permitiu testar eficazmente a

funcionalidade sem ter de esperar pelas modificações da camada de dados que foram mais complexas.

4.4.3 Adaptação dos ecrãs de medicação e de prescrição

Por se tratar de um processo relativamente parecido, foi escolhido um ecrã representativo destas implementações: o ecrã principal da medicação. Este ecrã é bastante completo, contendo muita informação e sujeito a várias alterações. Com o objectivo de detectar as principais alterações a efectuar, é apresentado um estudo comparativo. Anteriormente, o ecrã tinha o seguinte aspecto:

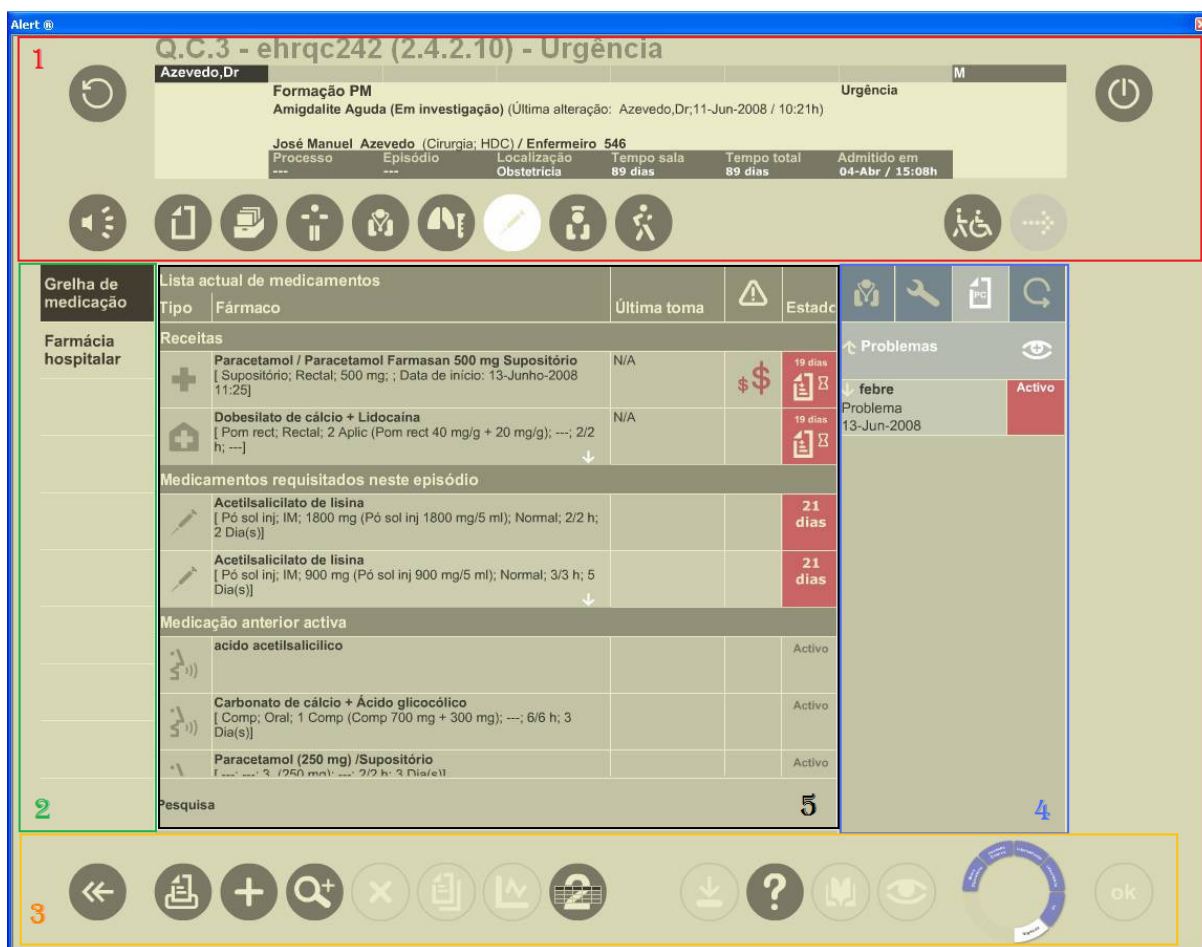


Figura 16: Aspecto antigo do ecrã principal de medicação

O ecrã pode ser essencialmente dividido em 4 secções, indicadas na Figura 16. A primeira corresponde a um cabeçalho fixo, contendo a área de identificação, do utilizador (profissional) e do paciente seleccionado, caso se verifique. Além desta informação essencial para facilitar as tarefas do utilizador o cabeçalho contém ainda os botões de navegação principais, dividindo a aplicação nas suas principais funcionalidades: na Figura 16, está seleccionado o separador “terapêutica” relacionado com tratamentos do paciente: pensos, procedimentos, medicação, dietas...

A segunda secção é uma subárea de navegação, dentro do contexto seleccionado no cabeçalho. Usando como referência o mesmo exemplo, a **Figura 16** indica que o utilizador se encontra nos ecrãs de medicação, dentro da categoria de terapêutica.

A terceira secção é a “*toolbar*”/barra de ferramentas da aplicação. O conjunto de botões utilizados nesta barra é fixo, no entanto nem sempre todos se encontram activos, dependendo do ecrã em questão e das funcionalidades associadas, naturalmente.

A quarta secção tem uma das funcionalidades centrais da ALERT: o *viewer*. Este painel é uma espécie de “*tracker*” que oferece ao utilizador um conjunto de informação e atalhos relacionados com o ecrã em questão, contextualizando e facilitando a navegação na aplicação.

Por exemplo, enquanto o médico passa uma receita, o *viewer* pode disponibilizar toda a informação relacionada com os problemas do doente.

Finalmente, a última secção corresponde ao conteúdo do ecrã propriamente dito, no qual foram feitas as alterações mencionadas no âmbito dos desenvolvimentos para a nova versão do produto.

Nesta nova versão, o ecrã passa a ter o seguinte aspecto visual:



Figura 17: Novo aspecto do ecrã principal de medicação

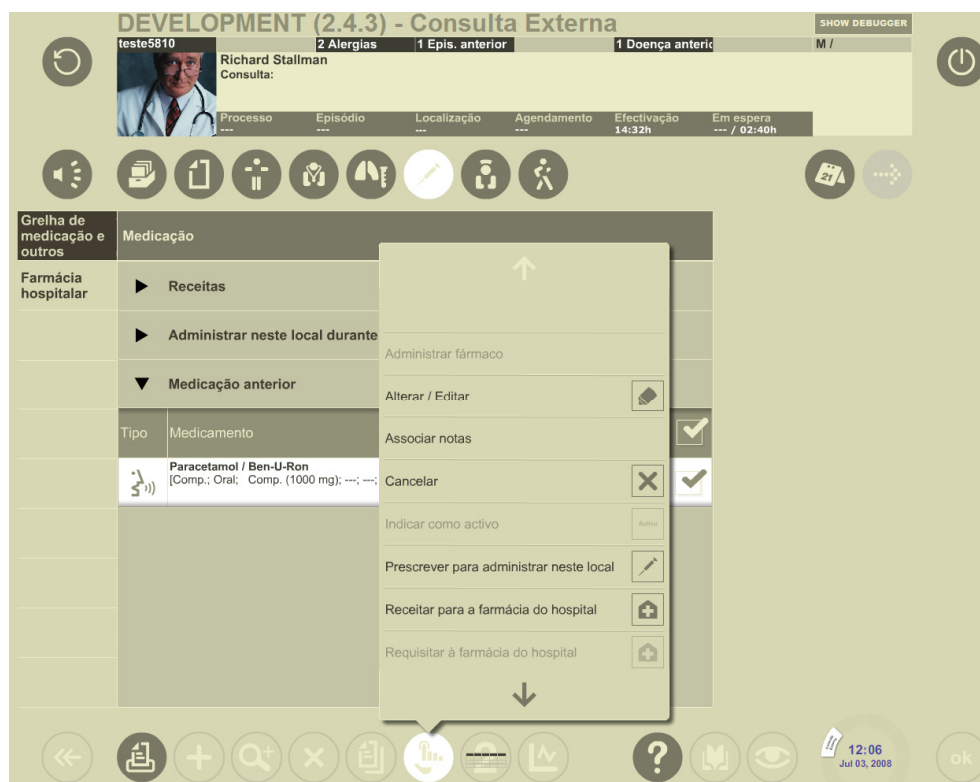


Figura 18: Novo aspecto do ecrã principal de medicação (2)

À primeira vista, o ecrã parece idêntico, no entanto, atentando a alguns detalhes, podem ser detectadas as seguintes alterações:

1. *Novas Grelhas e o novo componente “gaveta”*

Por se tratar do ecrã central da medicação, é representada toda a informação relevante à medicação de um dado paciente. Obviamente isto inclui alguns conceitos na área da prescrição, tornando-os difíceis de analisar por alguém que não pertença à área, motivo pelo qual são introduzidos no estado da arte deste documento. Assim sendo, recomenda-se a leitura desse capítulo para maior contextualização com alguns termos apresentados no ecrã.

Na versão anterior apenas eram apresentadas 3 grelhas principais, separando a medicação em 3 categorias: receitas, medicamentos requisitados no episódio e medicação anterior activa. No que toca a tipos de medicação contemplada no ALERT, a primeira categoria engloba receitas para o exterior, manipulados e dietéticos. A segunda é constituída por soros e administrados no local. A terceira contém os relatos de medicação activos.

Com a chegada da nova versão, surgiu também a necessidade de acrescentar 2 grelhas novas: medicação administrada no bloco durante a visita e medicação do episódio inactiva há mais de 30 dias. No fundo estas duas categorias surgem para refinar a informação relativa aos

medicamentos administrados no local, filtrando aqueles que foram administrados em bloco operatório e aqueles que estão inativos há “demasiado tempo”.

Por questões visuais foi necessário também utilizar um novo componente visual que se assemelha a uma gaveta ou acordeão, cujo nome standard é “closable panel”. Deste modo é possível esconder algumas grelhas de modo a não sobrecarregar o ecrã com informação.

2. Múltipla selecção e o novo botão de acções

Na versão anterior apenas eram aplicadas operações sobre um medicamento de cada vez, o que era pouco prático tendo em vista as necessidades dos utilizadores. Muitas vezes um médico pode querer realizar uma operação sobre vários medicamentos, e desta necessidade surgiu a nova funcionalidade de selecção múltipla. No entanto, por motivos de usabilidade, não basta acrescentar os *check buttons* nas colunas, é também necessário incorporar e configurar o botão de acções no ecrã. Este botão (com o *icon* da mão) surge como um processo alternativo às operações sobre a medicação. Antes bastava pressionar um medicamento e seleccionar a opção desejada numa janela *pop-up*. Agora o utilizador pode seleccionar vários medicamentos e pressionar o botão de acções para abrir um novo menu com as operações disponíveis.

No contexto deste ecrã foi necessário configurar adaptar a esta nova forma de interacção, reconfigurando o menu anterior e incorporando o novo.

3. Nova representação dos estados

Apesar de parecer semelhante, foi necessário alterar a forma dos estados, para facilitar o tratamento dos novos *icons*. Como este processo era sobretudo feito a partir de tratamento de *strings* enviadas pela camada de dados, foi necessário adaptar as grelhas para a recepção destes conteúdos, bem como alterar o formato enviado pela base de dados.

Aplicação do Data Abstraction ao ecrã

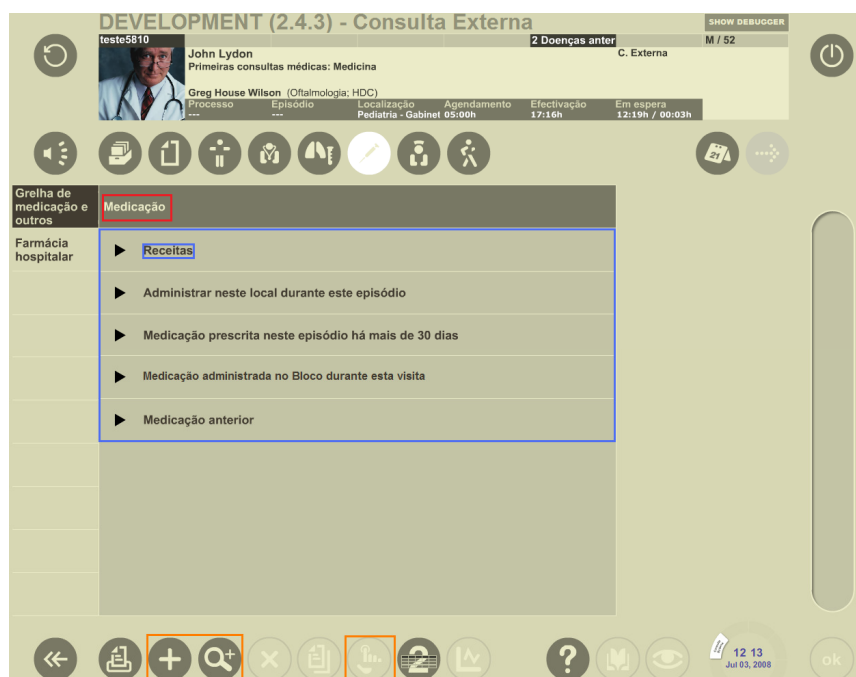


Figura 19: Aplicação do Data Abstraction ao ecrã de medicação

Devido à complexidade deste ecrã e ao conjunto de informação que representa, esta foi a verdadeira “prova de fogo” para o *Data Abstraction*. Na verdade, as chamadas remotas não se ficam simplesmente pela informação visível nas grelhas. Como foi dito anteriormente, a camada responsável por toda a lógica de negócio é a camada de dados, e desse modo toda a informação de menus e conteúdo deve vir da base de dados. Como foi visto nas implementações relativas ao ecrã, foram feitas alterações ao nível das grelhas (estados), menus e cabeçalhos, afectando praticamente todas as chamadas remotas.

Isto significa que para a simulação deste ecrã, é necessário considerar praticamente todo o conteúdo. Enumerando todos estes aspectos, obtêm-se a seguinte lista:

- Conteúdos das grelhas 1 a 5;
- Cabeçalhos para essas mesmas grelhas (informação estática);
- Opções dos menus de criação e de acções;
- Mensagens de ajuda e confirmação;

Toda esta informação originou um ficheiro muito complexo, que por outro lado, o ficheiro facilitou imenso o processo de desenvolvimento, tornando-o mais rápido e robusto.

4.4.4 Desenvolvimento do novo componente de pesquisa

Apesar de não se relacionar directamente com a equipa de medicação por ser uma solução mais genérica, a verdade é que este componente afecta maioritariamente os de prescrição/medicação.

No âmbito da tecnologia *Flash*, um componente pode ser visto como um módulo/função reutilizável, transportado para um ponto de vista mais visual.

Este componente resulta da conversão de uma versão anterior face a alguns aspectos de usabilidade. A ideia do componente é criar uma barra de pesquisa capaz de receber texto e disparar eventos de pesquisa para os vários ecrãs que o incorporem. No componente devem ser definidos todos os aspectos comportamentais, modularizando assim a barra de pesquisa e facilitando a sua integração em diferentes contextos.

Em termos de aspectos visuais, comparando com a versão anterior, o novo componente difere principalmente em termos de interacção com o utilizador. As suas dimensões, cores e forma são permanecem exactamente iguais, no entanto não é reaproveitado código praticamente nenhum, uma vez que o funcionamento difere em alguns aspectos.



Figura 20: componente de pesquisa - novo e antigo

A nível do funcionamento, originalmente a versão anterior comportava-se de acordo com a **Figura 21**.

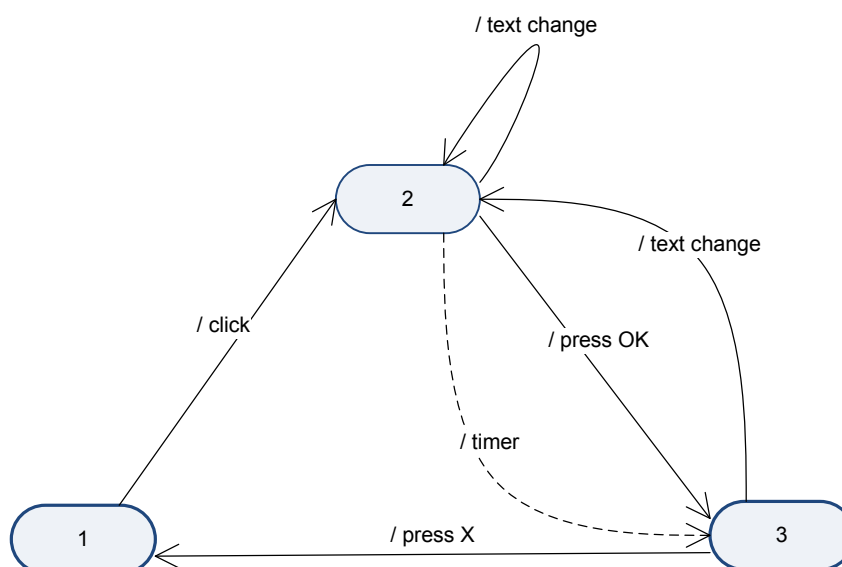


Figura 21: Diagrama de estados - antes

Segundo o diagrama de estados apresentado, a barra de pesquisa pode encontrar-se em 3 estados diferentes:

Tabela 11 – Componente pesquisa (antes)

Estado	Descrição
1- <i>desactivado</i>	A barra encontra-se fechada (cor escura), mostrando a mensagem de texto “pesquisa”, com ambos os botões desactivados e invisíveis para o utilizador. Corresponde à primeira imagem da Figura 20, no lado direito.
2- <i>activado</i>	A barra fica branca e a mensagem “pesquisa” desaparece dando início ao modo de inserção de texto. O botão “ok” fica visível e activo. Corresponde à segunda imagem da Figura 20, no lado direito.
3- <i>final</i>	Semelhante ao estado activado, mas em vez do botão “ok” é apresentado o botão “x” para cancelar o efeito da pesquisa. Corresponde à terceira imagem da Figura 20, no lado direito.

No seu estado inicial, a barra começa no estado desactivado. Clicando sobre esta barra esta transita imediatamente para o estado activado. Durante este estado, se o utilizador escrever alguma informação na barra ao final de algum tempo é accionada a pesquisa automaticamente, mesmo que não carregue no botão “ok”, e transita para o seu estado final. Esta funcionalidade pretendia facilitar o processo de pesquisa, no entanto não teve tão bons resultados como seriam esperados, sendo um dos aspectos a modificar. A partir do estado final só é possível cancelar a pesquisa para voltar ao estado desactivado, fechando o ciclo.

Em comparação, é apresentado agora a nova versão do componente, após a remodelação.

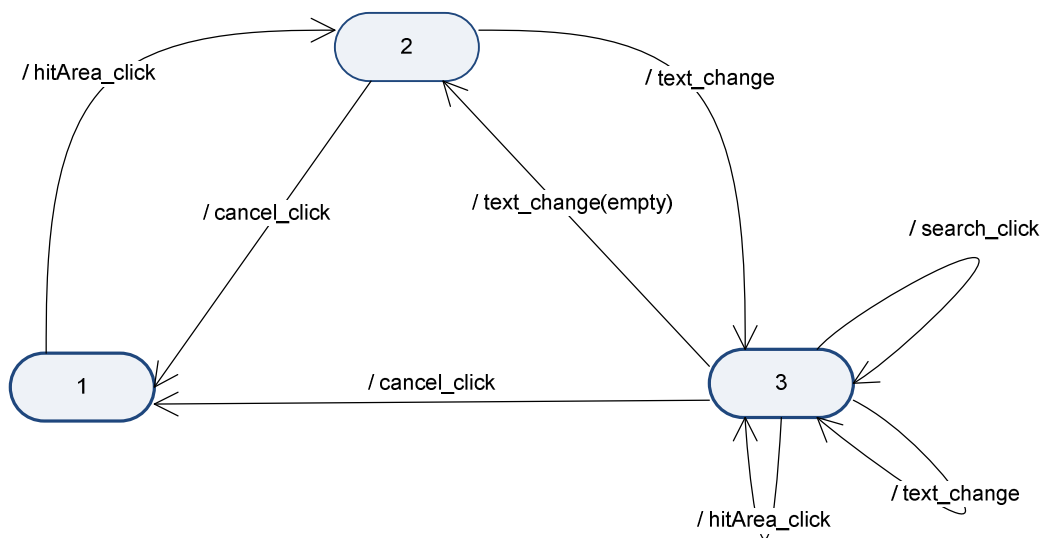


Figura 22: Diagrama de estados - depois

Tabela 12 - Componente Pesquisa (depois)

Estado	Descrição
1- <i>desactivado</i>	A barra encontra-se fechada (cor escura), mostrando a mensagem de texto “pesquisa”, com ambos os botões desactivados. No entanto, ao contrário do caso anterior, os botões estão perfeitamente visíveis, mesmo neste estado. Corresponde à primeira imagem da Figura 20, no lado esquerdo.
2- <i>sublinhado</i>	A barra fica branca e a mensagem “pesquisa” desaparece dando início ao modo de inserção de texto. Neste estado, o botão de cancelamento está activado, enquanto o botão de pesquisa permanece inacessível. Corresponde à segunda imagem da Figura 20, no lado esquerdo.
3- <i>activado (final)</i>	Semelhante ao estado sublinhado, mas com o botão de pesquisa activado. Este ponto é atingido quando a caixa de texto tem alguma informação. Corresponde à terceira imagem da Figura 20, no lado esquerdo.

À semelhança do seu antecessor o componente arranca em estado inactivo. Previamente, só o facto de clicar na barra de pesquisa bastava para desencadear um processo de pesquisa, que era activado ao final de alguns segundos devido à funcionalidade “relógio”. Nesta nova versão o utilizador é obrigado a inserir texto de pesquisa, caso contrário permanecerá no estado sublinhado, tendo apenas acesso ao botão de cancelamento. Este estado traz ainda outra vantagem em relação ao caso anterior, porque agora é possível cancelar uma pesquisa sem ser obrigado a lança-la primeiro, evitando processamento desnecessário.

Sempre que a caixa de texto tem alguma informação é passado para o estado activado, e a partir daí o utilizador pode escolher entre pesquisar ou cancelar.

No que toca à aplicação do *Data Abstraction* sobre esta implementação, não foi feita de forma directa, uma vez que um componente por si só não acede remotamente a informação, mas sim o ecrã que o utiliza. Na verdade, para testar o componente foi criado um ecrã de teste especialmente implementado para o efeito. Por um lado, esse ecrã permitiu testar o funcionamento do componente. Por outro, precisamente por se tratar de um ecrã de teste é o contexto perfeito para a aplicação do *Data Abstraction*, testando simultaneamente o ecrã e o componente.

Capítulo 5

Conclusões

5.1 Avaliação dos resultados

O final do desenvolvimento do *Data Abstraction* foi marcado com uma apresentação a todos os programadores da camada de *user interface* da empresa, onde foi esclarecido o seu propósito e funcionamento. A partir desse momento a solução deste trabalho passou a ser oficialmente uma ferramenta interna da empresa, devidamente documentada e aprovada.

Durante a fase de testes e aplicação prática da metodologia, não só o autor do trabalho utilizou a ferramenta mas também alguns dos programadores da camada de *user interface*.

A partir do feedback recebido tanto da experiência da utilização do *Data Abstraction* como da apresentação referida foi possível retirar alguns aspectos interessantes:

- A ideia foi aprovada, sendo considerada uma mais-valia para muitos dos programadores da camada de *user interface*;
- Verificou-se alguma resistência no que toca à configuração manual dos ficheiros XML, mesmo com a ajuda da validação do Schema.
 - Alguns programadores consideram uma perda de tempo. No entanto, o tempo “perdido” a configurar o XML compensa perfeitamente a espera pela funcionalidade, pois permite avançar com o desenvolvimento das funcionalidades sem necessidade de grandes modificações no final;
 - Uma minoria sente-se desconfortável a trabalhar com XML por falta de experiência na área, constituindo um pequeno obstáculo à utilização destes ficheiros por receio de se meterem em trabalho complicado;
 - Há programadores que consideram pouco *user friendly* ter de escrever directamente no ficheiro, apesar de admitirem que garante uma boa flexibilidade, de modo a construir exactamente a informação que precisam;
- Grande maioria ficou satisfeita com a facilidade de utilização do simulador, e de todos os casos que o autor tomou conhecimento, não houve problemas na mudança para chamadas remotas.
- A identificação dos erros passou a ser facilitada, pois os programadores de flash passaram a poder testar as suas funcionalidades sem a ajuda da base de dados, verificando a origem do mesmo.
- Alguns utilizadores sugerem que seja implementado um sistema de pesquisa em vários XML, integrado no *parser*, formando assim uma base de informação.
- Há testemunhos de casos de pleno sucesso em que o *Data Abstraction* permitiu desenvolver funcionalidades inteiras sem qualquer problema na fase de integração com a camada de dados.

Se por um lado o feedback positivo foi encarado como uma fonte de motivação, por outro lado as críticas foram recebidas de forma construtiva, levando à elaboração das novas propostas apresentadas no terceiro ponto deste capítulo.

De forma geral considera-se que os resultados foram muito satisfatórios. O simples facto da solução ter sido adoptada internamente e entrar em funcionamento só pode ser considerado um êxito. Em adição, alguns testemunhos indicaram melhorias significativas em termos de tempo de implementação, principalmente porque todas as funcionalidades terminaram no tempo previsto sem excepção, sobrando tempo para a realização de testes internos mesmo antes do arranque da fase de testes.

5.2 Conclusões

Como pode ser concluído a partir da leitura deste documento, durante o desenvolvimento paralelo baseado em várias tecnologias, é natural que surjam problemas derivados das dependências das camadas de software. Considerando os resultados deste trabalho, é correcto afirmar que esta questão pode ser facilmente contornada a partir da aplicação pequenas técnicas, que promovam a autonomia entre as camadas. Técnicas estas que têm ainda em consideração a questão do sincronismo do processo de desenvolvimento, necessário para garantir resultados de qualidade.

No problema apresentado foi utilizado um sistema composto por dois módulos funcionais, cada um deles focado num dos principais aspectos de comunicação: autonomia e sincronismo. As escolhas efectuadas revelaram-se adequadas ao cenário em causa, produzindo um bom impacto no ambiente de trabalho, sobretudo devido à simplicidade e transparência na questão da simulação e testes. Na realidade, este impacto resultou na adopção da solução pela parte da empresa, de tal forma que é actualmente utilizada na prática, como uma ferramenta de apoio ao desenvolvimento de software.

A escolha das tecnologias é, sem dúvida, um ponto crucial na resolução do problema. Mas mais importante do que as ferramentas a utilizar é o processo a ter por base. É de realçar a importância da metodologia e da ideia na qual se baseia o Data Abstraction. Enquanto que na ALERT estas escolhas tecnológicas sejam de facto adequadas, nada garante que funcionem igualmente noutro cenário. No entanto, num outro cenário, independentemente das escolhas tecnológicas, poder-se-á recorrer à metodologia no âmbito deste trabalho.

5.3 Perspectivas de trabalho futuroinformática aplicada à saúde

Da avaliação dos resultados, constatou-se que grande parte da resistência encontrada à implementação da solução desenvolvida foca-se sobretudo no tratamento directo com o XML. Considerando este aspecto, o autor apresenta uma proposta para melhorar a experiência do utilizador através do incremento do nível de abstracção.

Proposta de melhoramento: *Data Abstraction Interface*

Na solução apresentada, o sistema *Data Abstraction* contempla os problemas abordados na análise de forma eficaz: o problema do sincronismo e autonomia das camadas. Em adição a estas duas funcionalidades principais e sem entrar em conflito com os desenvolvimentos já efectuados, é considerada uma terceira funcionalidade relacionada com a manipulação do XML, que basicamente pretende atingir dois objectivos:

- Evitar o contacto directo do utilizador com os ficheiros XML e abstrair totalmente da existência destes ficheiros, exceptuando para a geração dos documentos.
- Aumentar o nível da abstracção, facilitando e tornando todo o processo de criação do XML mais rápida e intuitiva.

No esquema da arquitectura da solução, a proposta encaixaria segundo a Figura 15, novamente centrada no XML.

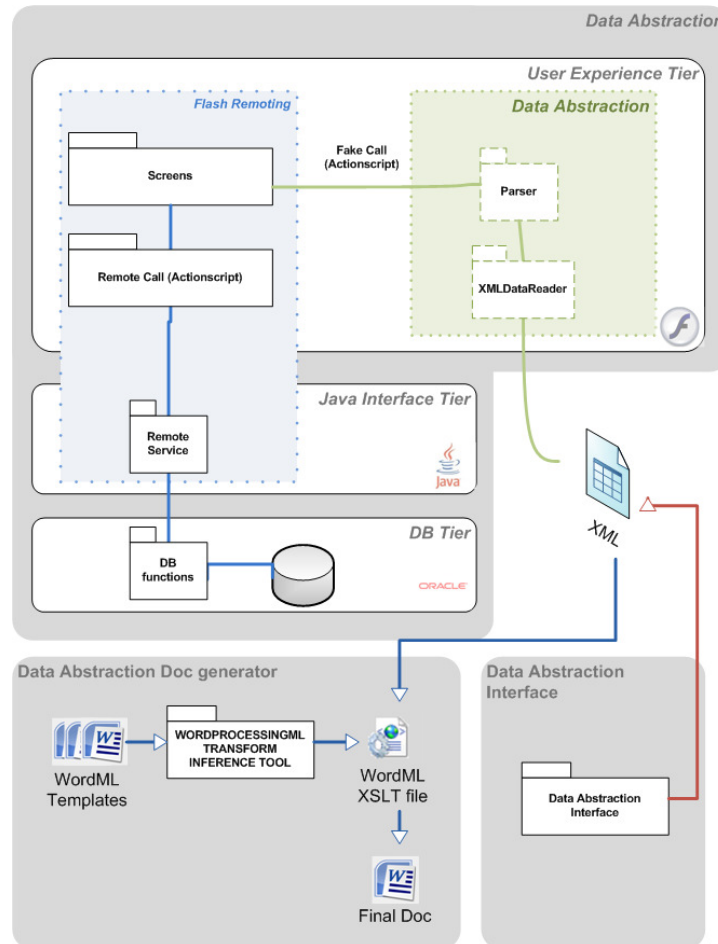


Figura 23: Proposta de melhoramento

Em vez de ter de editar manualmente o ficheiro e proceder ainda à validação do mesmo, esta interface iria permitir a manipulação de uma forma simples e directa, abstraindo os utilizadores do facto de estarem a editar XML. Para o caso do utilizador ainda desejar editar manualmente poderia manter-se o *Schema* de validação mas em cenário “normal” esta validação deixaria de ser necessária, isto porque a geração de documentos vem segundo a estrutura. Além deste aspecto de implementação, a interface poderia ainda possibilitar uma gestão de ficheiros XML, especificando como se deseja dividir a informação.

Uma outra questão interessante que foi levantada foi a forma como é procurada a informação. Na versão actual do *Data Abstraction* a pesquisa é feita directamente no XML especificado para o seu funcionamento. Aumentar o grau de abstracção passa também por ocultar ao utilizador este tipo de questões, passando a ser transparente a gestão dos diversos ficheiros. De forma a suportar essa funcionalidade é também proposto um melhoramento ao *Parser*, de modo a permitir uma pesquisa não num ficheiro XML mas sim num conjunto de

ficheiros, como se fosse de facto uma só fonte de dados. Este conceito já é aplicado actualmente como por exemplo nas bases de dados XML.

Perspectivas de desenvolvimento a longo prazo

Como foi possível verificar o *Data Abstraction* é um sistema com grande capacidade de evolução, havendo sempre pontos a melhorar. A longo prazo será sempre possível estender esta funcionalidade para outros domínios no âmbito do melhoramento do processo de implementação. Uma questão que se considera relevante é a diversidade do código das várias funcionalidades, derivado das diferentes formas de programar dos vários colaboradores. Apesar de o código seguir as regras da boa programação, dois ecrãs iguais implementados por duas pessoas diferentes podem ser bastante diferentes, apesar de fazer e se basearem no mesmo. Considerando a entrada frequente de novos trabalhadores e a adaptação constante do código a novas funcionalidades, quanto mais normalizado este for, melhor. Uma ideia do autor seria combater esta questão recorrendo a processos semelhantes aos que foram utilizados para o *Data Abstraction*, por exemplo, a partir de ficheiros XML que permitissem tal coisa. O objectivo não seria gerar o código final, mas sim o código base, de forma a promover a normalização do mesmo, trazendo óbvias vantagens:

- Maior facilidade na revisão do código
- Maior facilidade de integração de novos elementos e de documentação dos processos de acolhimento
- Maior flexibilidade no que toca a alterações de código: qualquer programador pode facilmente em funcionalidades que sejam da sua autoria com maior rapidez.

Apesar de ser um cenário animador, seria de considerar apenas a longo prazo, por se tratar de uma tarefa cuja importância é de segundo plano face aos objectivos actuais da empresa. No que toca a perspectivas futuras como membro da equipa de prescrição passam pelo desenvolvimento de novas funcionalidades, deixando estes melhoramentos como actividade de segundo plano a ser desenvolvido em médio/longo prazo.

5.4 Comentários sobre o desenvolvimento do projecto

Ao longo das vinte semanas para realização deste trabalho, o autor deste documento passou por uma experiência diferente e bastante positiva. O enquadramento na empresa trouxe bastantes novidades, desafios, mas sobretudo novas opiniões e pontos de vista de ver variados temas que nunca poderiam ser encontrados apenas em ambiente académico. Se por um lado houve trabalho de equipa no seio da equipa de prescrição, também houve trabalho independente para a procura da solução para o problema enunciado. Em qualquer dos cenários, contando sempre com o apoio e conhecimento dos outros.

Tanto a empresa que acolheu este projecto, como o próprio trabalho levaram o autor a adquirir vários conhecimentos, não só na área da programação de interfaces em Actionscript. No âmbito da empresa de medicação, foi necessário tomar conhecimento de vários conceitos clínicos de forma a ficar a par da lógica de negócio.

Além da experiência retirada do dia a dia na empresa, o autor destaca ainda alguns eventos que contribuíram para o enriquecimento pessoal: as formações internas, que permitiram obter conhecimentos na área da saúde até então desconhecidos pelo autor; o retiro sobre procedimentos, que permitiu ficar com uma maior noção do funcionamento geral da ALERT; e ainda pequenas experiências no âmbito de trabalho em equipa, como é o caso da aplicação prática da metodologia SCRUM .

No âmbito do trabalho, a análise de toda a arquitectura de comunicação entre as camadas levou a estudos, reuniões e discussões que alargaram os conhecimentos do autor. No entanto, será sobretudo importante salientar que todos os objectivos foram cumpridos e foi encontrada uma solução. Após toda a experiência retirada deste trabalho, o autor conclui que muitas das vezes os programadores não procuram a melhor solução mas sim uma solução simples e que seja capaz de resolver o problema sem os preocupar muito com isso. Devido ao ritmo de trabalho, o tempo que tem a “perder” é reduzido, logo o que interessa é que os leve facilmente onde querem chegar. Se perderem mais de 10 minutos a analisar uma ferramenta sem a entender, provavelmente nunca a vão utilizar. Assim sendo, aos olhos do autor, não é vitória encontrar uma solução que resolva o problema. Vitória é encontrar uma solução que seja aceite e utilizada na prática, que não realmente sirva o objectivo para o qual foi concebida e que entre nos ouvidos dos utilizadores. E nesse sentido, verificar que o *Data Abstraction* ajudou alguns dos programadores *Flash* da empresa foi gratificante.

Referências e Bibliografia

- [Alt] Altova. Altova XSL-fo solutions.
http://www.altova.com/dev_portal_xslfo.html
- [AsL] AsLib.
<http://www.as2lib.org/>
- [AsU] AsUnit.
<http://www.asunit.org/>
- [Cod] CodeIdol.com. How to work with XSLT Inference Tool
<http://codeidol.com/csharp/c-sharp-in-office/Working-with-XML-in-Word/Importing-an-XML-Data-File-into-the-Mapped-Document/>
- [Cod1] CodePlex. Xunit.
<http://www.codeplex.com/xunit>
- [con] consumidorbrasil.com.
<http://www.consumidorbrasil.com.br/consumidorbrasil/textos/ebomsaber/medicamentos/prescrico>
- [Cos] Cosign- healthcare.
<http://www.freepatentsonline.com/70162302.html>
- [Cru] Cruise Control.
<http://cruisecontrol.sourceforge.net/>
- [Ecr] Ecrion. XFDesigner
<http://www.ecrion.com/Products/XFDesigner/Overview.aspx>
- [Fla] Flash Magazine. Why Flex?
http://www.flashmagazine.com/news/detail/why_flash_developers_should_care_about_flex/
- [Fla1] Flash versus Flex.
<http://flashenabledblog.com/2007/02/23/flex-vs-flash/>
- [Fle] FlexUnit.

<http://code.google.com/p/as3flickrlib/>

[FUn] FUnit.

<http://blog.funit.org/>

[Inf] Infarmed. Medicamentos manipulados.

http://www.infarmed.pt/portal/page/portal/INFARMED/MONITORIZACAO_DO_MERCADO/

[lul] lullabot - Unit testing.

<http://www.lullabot.com/articles/introduction-unit-testing>

[Mic] Microsoft. XSLT Inference Tool

[http://msdn.microsoft.com/en-us/library/aa212886\(office.11\).aspx](http://msdn.microsoft.com/en-us/library/aa212886(office.11).aspx)

[Mic1] Microsoft. Msdn - Unit Testing.

[http://msdn.microsoft.com/en-us/library/aa292197\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/aa292197(VS.71).aspx)

[Ref] Reflex Unit.

<http://code.google.com/p/reflex-unit/>

[sty] stylusstudio - XML vs Flat files.

<http://www.stylusstudio.com/xmldev/200102/post80100.html>

[SWF] SWFkit. What is SWFKit? <http://www.swfkit.com/swfkit.html>

[Tim00] S. F. C. Tim Mackinnon, "Endo-Testing: Unit Testing with Mock Objects," 2000.

[uni] unit testing frameworks. http://www.opensourcetesting.org/unit_misc.php

[vel] velocityreviews - XML vs Flat files.

<http://www.velocityreviews.com/forums/t151822-xml-not-good-for-big-files-vs-flat-files.html>

[W3s] W3schools. W3schools - Xsl-fo tutorial.

http://www.w3schools.com/xslfo/xslfo_intro.asp

[Wik] Wikipedia. Wikipedia - Lista de frameworks de testes unitários

http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

[wik1] wikipedia. wikipedia - cosign.

<http://en.wikipedia.org/wiki/Cosign>

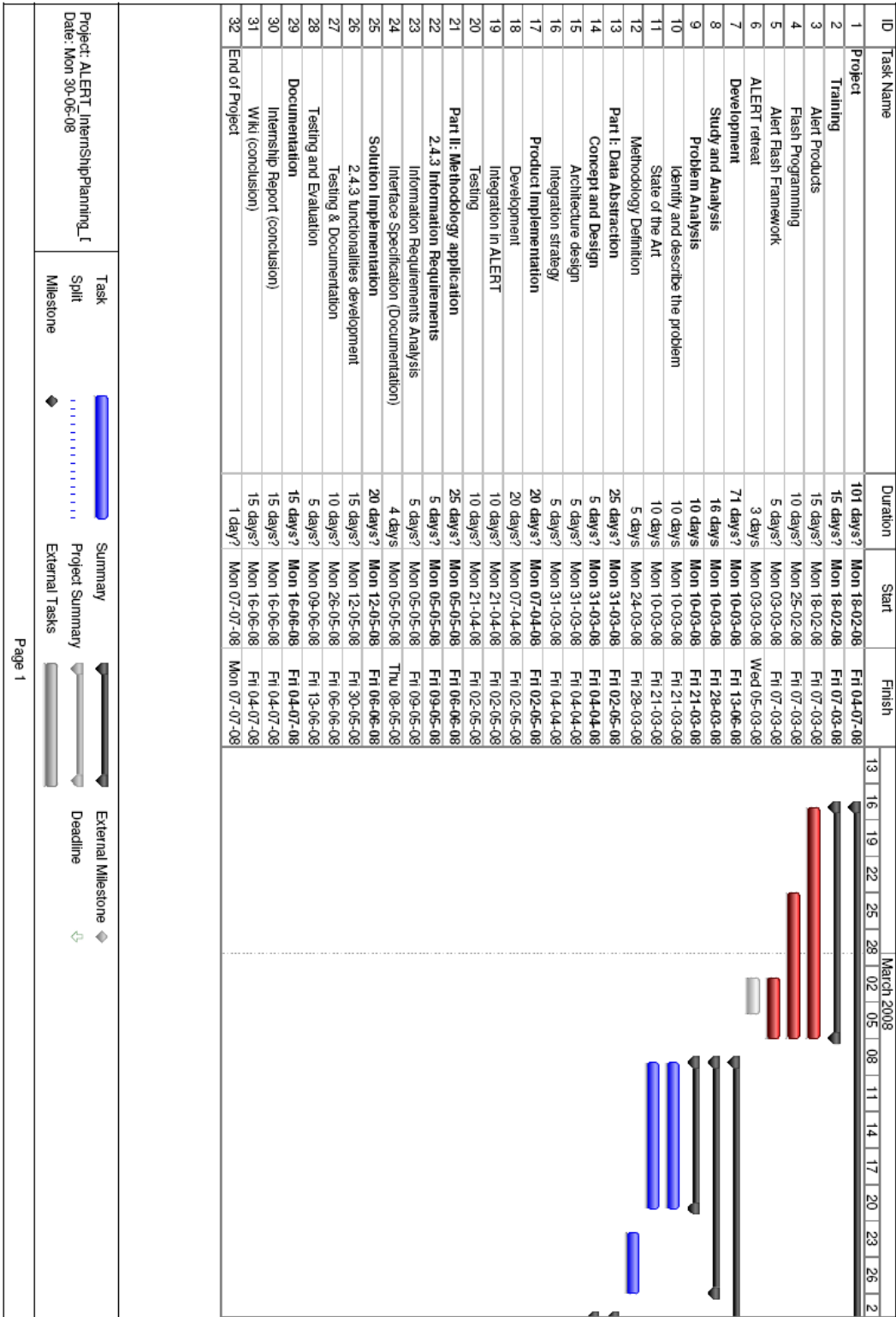
[wik2] wikipedia. Unit Tests

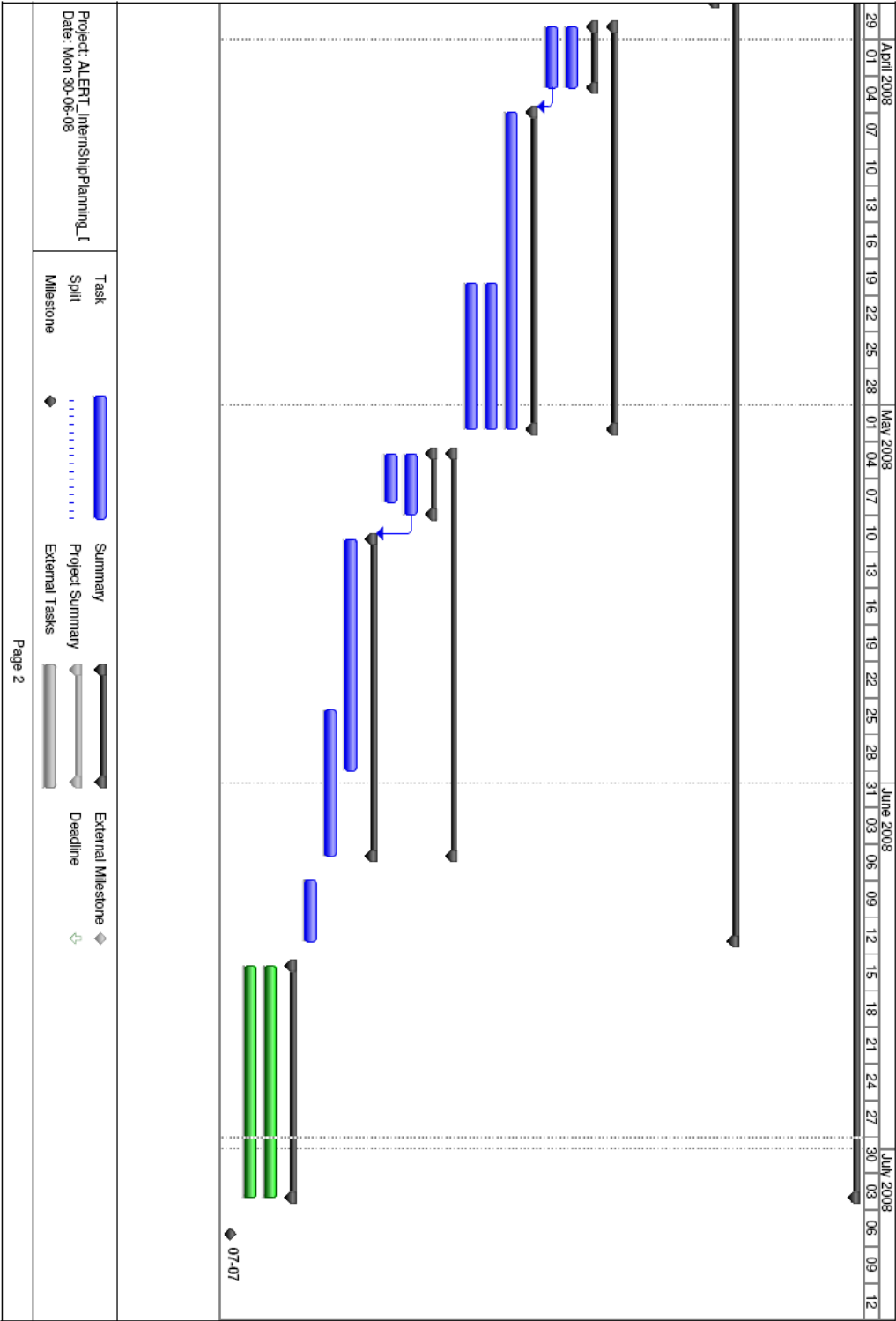
http://pt.wikipedia.org/wiki/Teste_unit%C3%A1rio

[XSL] XSL to convert xml into html.

<http://www.xml.com/pub/a/2000/08/30/xsltandhtml/index.html>

ANEXO A: Diagrama do Planeamento





ANEXO B: Data Abstraction Doc

Function

func_name

Inputs

Name	Type
p1	number
p2	string
p3	string
p4	object
p5	string
p6	string

Outputs

String:
teste

Number:
7

Object
o_object

Name	Value
atrib01	Result
atrib02	I

Cursor
return_cursor

Name	Value
atrib01	val
atrib02	val
atrib03	val

ANEXO C: Exemplo XML Data Abstraction

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<alert xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="U:\dataAbstraction dataAbstraction.xsd">
<function>
  <name>func_name</name>
  <inputs>
    <input>
      <name>p1</name>
      <type>number</type>
    </input>
    <input nullable="true">
      <name>p2</name>
      <type>string</type>
    </input>
    <input nullable="true">
      <name>p3</name>
      <type>string</type>
    </input>
    <input>
      <name>p4</name>
      <type>object</type>
    </input>
    <input>
      <name>p5</name>
      <type>string</type>
    </input>
    <input>
      <name>p6</name>
      <type>string</type>
    </input>
  </inputs>
  <outputs>
    <string>teste</string>
    <number>7</number>
    <object>
      <name>o_object</name>
      <attrib>
        <name>atrib01</name>
        <value>Result</value>
      </attrib>
      <attrib>
        <name>atrib02</name>
        <value>I</value>
      </attrib>
    </object>
  <cursor>

```

```
<name>return_cursor</name>
<object-child>
  <attrib>
    <name>atrib01</name>
    <value>val</value>
  </attrib>
  <attrib>
    <name>atrib02</name>
    <value>val</value>
  </attrib>
  <attrib>
    <name>atrib03</name>
    <value>val</value>
  </attrib>
</object-child>
</cursor>
</outputs>
</function>
</alert>
```

ANEXO D: Data Abstraction Schema

```

<?xml version="1.0" encoding="UTF-8" ?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="alert">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="function" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="attrib">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element ref="name" />
        <xs:element ref="value" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="cursor">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element ref="name"/>
        <xs:element ref="object-child"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:element name="function">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name" />
        <xs:element ref="inputs" />
        <xs:element ref="outputs" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="input">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element ref="name" />
        <xs:element ref="type" />
      </xs:sequence>
      <xs:attribute name="nullable" type="bool" use="optional" />
    </xs:complexType>
  </xs:element>

```

```

</xs:element>

<xs:element name="inputs">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="input" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="name">
  <xs:complexType mixed="true" />
</xs:element>

<xs:element name="number">
  <xs:complexType mixed="true" />
</xs:element>

<xs:element name="object">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name" />
      <xs:element ref="attrib" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="object-child">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="attrib" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="times" type="xs:NMTOKEN" use="optional" />
  </xs:complexType>
</xs:element>

<xs:element name="outputs">
  <xs:complexType>
    <xs:choice maxOccurs = "unbounded">
      <xs:element ref="cursor"/>
      <xs:element ref="number"/>
      <xs:element ref="object"/>
      <xs:element ref="string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name="string">
  <xs:complexType mixed="true" />
</xs:element>

<xs:element name="type">

```

```
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:enumeration value="number"/>
    <xs:enumeration value="string"/>
    <xs:enumeration value="object"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>

<xs:element name="value">
  <xs:complexType mixed="true" />
</xs:element>

<xs:simpleType name="bool">
  <xs:restriction base="xs:string">
    <xs:enumeration value="true"/>
    <xs:enumeration value="false"/>
  </xs:restriction>
</xs:simpleType>

</xs:schema>
```