Faculdade de Engenharia da Universidade do Porto

Mestrado Integrado em Engenharia Informática e Computação

Documentação de Software: Integração de Ferramentas de Modelação e Processamento de Texto

Dissertação MIEIC 2007/2008

Nuno António da Silva Rocha

Orientador na FEUP: Prof. Ademar Aguiar

Responsável pelo acompanhamento na Critical Software, S.A.: Eng. Miguel Barroso

Abril de 2008

Contacto:

Nuno António da Silva Rocha
Faculdade de Engenharia da Universidade do Porto

Rua Dr. Roberto Frias, s/n
4200-465 Porto
PORTUGAL

Tel.: +351 962889107
E-mail: nuno.rocha@fe.up.pt

Nuno António da Silva Rocha
"Documentação de Software: Integração de Ferramentas de Modelação e Processamento de Texto"

A todos os que me apoiaram…

Resumo

O processo de documentação de software é algo que contribui bastante para o sucesso de um projecto. Efectivamente, um sistema bem documentado torna-se mais fácil de perceber por toda a equipa que o desenvolve, além de aumentar a eficácia e eficiência dos elementos da equipa quando é necessário realizar tarefas de manutenção.

Este processo é tipicamente ajudado pela existência de ferramentas capazes de facilitar o trabalho necessário. Entre estas podem-se encontrar as ferramentas de modelação, que permitem o desenvolvimento de diagramas e a modelação dos sistemas de forma intuitiva e tipicamente visual, e as ferramentas de processamento de texto, que ajudam na escrita dos documentos propriamente ditos.

Este trabalho propõe uma abordagem que permita uma integração mais suave entre estas duas classes de ferramentas, de maneira a que se consiga produzir, com menor esforço, documentação com melhor qualidade.

A solução encontrada para este problema passou pela definição de um mecanismo genérico de identificação de determinados pontos em documentos e integração de diversos componentes, tais como um configurador dinâmico de conteúdos e um gerador de documentos, de maneira a conseguir um maior grau de flexibilidade na sua configuração e uma maior facilidade de evolução dos mesmos.

Esta abordagem foi validada através da sua implementação em um caso de estudo concreto, utilizando Enterprise Architect como ferramenta de modelação e MS Word e OpenOffice como ferramentas de processamento de texto, num contexto industrial – a Critical Software, S.A. – e tendo sido obtidos resultados bastante interessantes e aplicáveis no domínio em causa.

Abstract

The software documentation process is a key factor for a project's success. A well documented system is easier to understand by all the elements of its development team and improves efficiency and effectiveness of maintenance tasks.

This process is typically aided by some tools that are capable of easing the needed work. Among these tools we can find the so called modelling tools – that allow the development of diagrams and the modelation of the systems in an intuitive and typically visual form – and the word processing tools, which help the process of producing and writing the corresponding documents.

This project proposes an approach to ease the integration between these two kinds of tools, so that it can be possible to produce, with less effort, better quality documentation.

The solution that was developed for this problem is based on the definition of a generic mechanism of identification of certain places in documents and the integration of several components, such as a dynamic content configurator and a document generator, in order to achieve a higher level of flexibility in the document configuration and a greater possibility of document evolution.

This approach was validated through its implementation in a well defined case study, using Enterprise Architect as the modelling tool and MS Word and OpenOffice Writer as word processing tools. It was developed in an industrial setting - Critical Software, S.A. – and allowed to achieve interesting results applicable to these specific circumstances.

Agradecimentos

Agradeço a todos aqueles que, de uma forma directa ou indirecta, contribuíram para que este trabalho de dissertação se realizasse nas melhores condições e atingisse os objectivos propostos.

Ao Prof. Ademar Aguiar, supervisor deste trabalho, pela atenção e disponibilidade constantemente demonstradas e pelos valiosos conselhos e indicações dados.

Ao Eng. Miguel Barroso, responsável pelo acompanhamento do trabalho na Critical Software, S.A., tendo-se mostrado extremamente dedicado e disponível, apoiando e criticando quando necessário, ajudando assim na obtenção dos resultados pretendidos.

À Critical Software, S.A. por se ter disponibilizado para acolher o desenvolvimento deste projecto, e em particular ao seu Departamento de Qualidade, pelo sempre importante *feedback* fornecido.

À equipa do projecto EBS-XE-MEM – David Sora, Luís Cruz, Pedro Costa e Pedro Reis – pelo excelente companheirismo e apoio demonstrados e que contribuíram de sobremaneira para um excelente ambiente de trabalho.

Aos meus grandes amigos e colegas que sempre me apoiaram e estiveram presentes nos bons e maus momentos.

Por fim, o maior agradecimento de todos vai para os meus pais e irmão, a família que fez de mim o que sou hoje e sem a qual nada teria sido possível.

Índice

Lista de Figuras

Lista de Tabela**s**

Capítulo 1.
Introdução

O processo de documentação de software é algo que contribui bastante para o sucesso de um projecto. Efectivamente, um sistema bem documentado torna-se mais fácil de perceber por toda a equipa que o desenvolve, além de aumentar a eficácia e eficiência dos elementos da equipa quando é necessário realizar tarefas de manutenção.

No entanto, por razões relacionadas com prazos e orçamentos mais curtos, é um processo que frequentemente é desleixado, com consequências por vezes não visíveis a curto prazo mas que a médio/longo prazo se podem tornar graves.

Este processo é tipicamente ajudado pela existência de ferramentas capazes de facilitar o trabalho necessário. Entre estas podem-se encontrar as ferramentas de modelação, que permitem o desenvolvimento de diagramas e a modelação dos sistemas de forma intuitiva e tipicamente visual, e as ferramentas de processamento de texto, que ajudam na escrita dos documentos propriamente ditos.

Este trabalho tem como objectivo procurar uma integração mais eficaz entre estas duas classes de ferramentas, de maneira a que se consiga produzir, com menor esforço, documentação com melhor qualidade.

Neste capítulo são descritos alguns dos principais pontos na área de geração e actualização de documentação técnica e identificados os objectivos de investigação. São ainda descritos os resultados principais em termos globais e é apresentada a estrutura do presente documento.

1.1 Geração e Actualização de Documentação Técnica

Manter a documentação técnica de um projecto de software é uma tarefa que, não sendo demasiado difícil, se torna por vezes impraticável ao longo do dia-a-dia numa organização. Tipicamente os artefactos que compõem uma boa documentação incluem vários tipos de conteúdos, sejam estes texto exemplificativo/explicativo, modelos, imagens ou código fonte, sendo conhecidos por *documentos heterogéneos* [18].

De maneira a aumentar a produtividade e a facilitar o trabalho de todos os elementos de uma equipa, todo o processo de desenvolvimento de software é normalmente auxiliado por ferramentas informáticas que permitem poupar tempo, reduzir esforço e aumentar difusão de conhecimento sobre o projecto, funcionando como repositório de informação e base de trabalho. Estas ferramentas podem ser ambientes de desenvolvimento integrados, ferramentas de modelação ou ferramentas de processamento de texto e são especializadas na função que pretendem desempenhar.

Este facto, em conjunto com a divisão de tarefas normalmente existente numa equipa, faz com que a probabilidade de existência de inconsistências entre os vários artefactos que compõem a documentação de um projecto de software seja relativamente alta.

Também os *standards* de certificação de qualidade intervêm bastante na questão da documentação técnica, uma vez que os conteúdos de uma documentação certificada são bastante definidos e concretos. E, claro, cada vez mais as organizações necessitam de apresentar vantagens competitivas em relação às suas concorrentes, sendo incontornável a questão da Qualidade.

1.2 Objectivos

É inquestionável o papel das ferramentas de modelação e das ferramentas de processamento de texto nos processos de desenvolvimento de software actualmente. Apesar disso, elas continuam a ser utilizadas em separado, cabendo ao próprio utilizador manter a consistência de informação entre o que existe numa e o que existe noutra.

Existindo já algumas soluções desenvolvidas para resolver algumas das questões que são levantadas ao utilizar simultaneamente estes dois tipos de ferramentas, pretende-se procurar expandir estas soluções de forma a que se obtenha algo o mais genérico possível focando algumas destas questões, nomeadamente ao nível de introdução automática de dados e actualização de documentos.

Isto leva ao desenvolvimento de uma nova abordagem, suportada por uma plataforma informática.

Objectivo Principal

O principal objectivo deste trabalho incide, então, sobre a integração de ferramentas de modelação com ferramentas de processamento de texto. Procura, assim, diminuir a possibilidade de ocorrência de inconsistências entre os modelos construídos (e os dados presentes numa ferramenta de modelação) e a documentação técnica que pode (e deve) incluir alguns desses dados, dar ao utilizador uma maior flexibilidade na escolha dos dados presentes na documentação técnica e contribuir para a diminuição do esforço necessário para efectuar a manutenção necessária da documentação.

Objectivos Complementares

Os objectivos complementares passam por aumentar o nível de uniformização de processos e procedimentos. Para tal, definir-se-ão práticas aplicáveis a uma organização como um todo e aplicar-se-ão os conceitos desenvolvidos a um caso concreto, numa organização real, confrontando os resultados obtidos com as abordagens existentes e a abordagem proposta.

1.3 Resultados Esperados

Os resultados principais a alcançar com o trabalho desenvolvido podem ser vistos do aspecto metodológico e do aspecto mais aplicacional.

Em termos metodológicos, podem-se destacar os aspectos da uniformização de processos na integração de dados de ferramentas de modelação com a documentação técnica produzida e da criação de um repositório global para armazenamento dos documentos-tipo relacionados com os dados de ferramentas de modelação.

Em termos aplicacionais, pode-se destacar o desenvolvimento de uma ferramenta *web-based* para geração/actualização de documentos, onde o utilizador tem a liberdade de escolher os dados que pretende efectivamente ver preenchidos. Esta ferramenta surgirá como prova-de-conceito da abordagem desenvolvida aplicada a um domínio concreto, neste caso a integração de dados vindos da ferramenta de modelação Enterprise Architect com documentos em formatos de *suites* de escritório (MS Word e OpenOffice).

1.4 Estrutura do Documento

Este documento está estruturado em sete capítulos, com os conceitos fundamentais relativos à tese apresentada e à abordagem desenvolvida, e ainda um anexo, com informação relativa à forma de utilização da ferramenta desenvolvida para validação de conceitos.

Assim, no capítulo 1, é feita uma introdução ao projecto, sendo mencionados os seus objectivos e resultados principais.

No capítulo 2 apresentam-se as noções mais relevantes na área de documentação de software, os principais desafios e boas práticas a seguir. É ainda apresentada informação sobre ferramentas de modelação e ferramentas de processamento de texto, conceitos também eles envolvidos neste trabalho.

No terceiro capítulo, procede-se a uma descrição do estado da arte, ou seja, é efectuada uma revisão dos conceitos e ferramentas existentes na área da documentação de software e integração de dados de ferramentas de modelação com documentação existente.

No capítulo 4 é dada atenção ao problema em análise, com a apresentação das lacunas existentes nas soluções actuais, justificando-se assim o desenvolvimento desta abordagem.

No quinto capítulo é descrita a solução proposta, sendo identificados os requisitos da mesma e feita uma análise às suas principais características. Após a descrição dos principais papéis envolvidos nesta abordagem é apresentado o modelo de documentação de software sugerido por esta proposta.

No sexto capítulo é descrita com algum pormenor a aplicação informática que serviu de prova-de-conceito para a abordagem proposta, num domínio concreto. Esta é descrita nas suas linhas orientadoras e modelo de funcionamento.

No capítulo 7 procede-se à validação da solução descrita, descrevendo os procedimentos adoptados para a testar num ambiente real e analisando e comparando os resultados obtidos.

Finalmente, no oitavo capítulo são apresentadas as conclusões gerais do trabalho desenvolvido, referindo as principais dificuldades encontradas e perspectivas de trabalho futuro.

Em anexos é incluído o manual de utilização da ferramenta desenvolvida para servir de suporte aos conceitos desenvolvidos e alguma documentação da mesma.

Capítulo 2.
Documentação de Software

Apesar de ser de extrema utilidade e importância, a actividade de documentação no âmbito de um projecto de desenvolvimento de software é, não raras vezes, desleixada, conduzindo ao aumento dos custos do projecto, originados por erros e omissões existentes na documentação.

Quando abordada de forma organizada e sistemática, esta actividade permite produzir artefactos que facilitam a melhoria, extensão e actualização do software a ser desenvolvido. Assim, os programadores e elementos relacionados com o projecto têm uma base comum de conhecimento e informação, garantindo assim a passagem de conhecimento no caso de entrada e saída de colaboradores.

A documentação pode também ser orientada ao utilizador final do software, dando-lhe uma visão da forma como funciona e de como deve ser usado, reduzindo a probabilidade de ocorrência de erros, problemas e frustrações que podem minar uma boa relação comercial.

A produção de documentação para um projecto de software começa antes mesmo do desenvolvimento do produto, com documentos relacionados com a visão e aplicabilidade, especificação de requisitos e de arquitectura do software a desenvolver. Posteriormente, a documentação cobre quer o processo de desenvolvimento, quer o produto desenvolvido (na óptica do utilizador final e na óptica do sistema).

Este processo de produção de documentação é algo que, se efectuado em simultâneo com o desenvolvimento do software, pode ajudar a detectar inúmeras falhas e inconsistências nas diversas fases do projecto, dando *feedback* aos colaboradores para o corrigir e melhorar e evitando possíveis problemas nas fases posteriores.

A importância do conteúdo da documentação é acompanhada pela importância da sua estrutura, nomeadamente em aspectos relacionados com a Qualidade de Software e certificações associadas. Os *standards* propostos pela IEEE identificam os conteúdos típicos

e a sua organização num documento, sendo usados pelas organizações como guias para criarem modelos de documentação mais adequados às suas necessidades.

Os conteúdos presentes na documentação (em especial a de cariz mais técnico) podem ser oriundos de ferramentas específicas. Dentro destas, podem ser referidas as ferramentas de modelação, que facilitam o trabalho de produção de modelos representativos do software a desenvolver em linguagens *standard* como o UML, por exemplo.

Este capítulo apresenta algumas considerações acerca da documentação (mais propriamente a documentação técnica de um projecto de software) e da forma como esta pode ser gerida e usada para aumentar o conhecimento dentro das organizações. Apresenta ainda uma visão geral (o que são e a sua utilidade) sobre os dois tipos de ferramentas abordadas por este trabalho: as ferramentas de modelação e as ferramentas de processamento de texto.

2.1 Documentação de um Projecto

Entende-se por documentação de um projecto de software o conjunto de todos os documentos que acompanham o seu ciclo de vida, desta a análise prévia estratégica até às fases de manutenção. Cobrindo, tipicamente, quer o processo de desenvolvimento, quer o produto desenvolvido, verifica-se a existência de convenções, *guidelines* ou calendarizações para documentar o processo e a separação típica entre documentação para o utilizador e documentação de sistema para o produto.

2.1.1 Documentação para o Utilizador

A documentação para o utilizador descreve o sistema do ponto de vista do utilizador final (possivelmente para vários tipos de utilizadores), isto é, uma vista do tipo caixa preta em que se mostra o que faz o sistema, sem entrar em detalhes técnicos ou específicos, só indicando as entradas e as saídas correspondentes. Inclui também indicações de como deve ser usado o produto [1].

Consiste na construção de manuais para o utilizador final, administradores de sistemas e pessoal de suporte e deve ser preparada tendo em conta as expectativas destes intervenientes. Pode ser apresentada de diferentes formas [2]:

- Em forma de tutorial, guiando o utilizador através de cada passo para cumprir uma determinada tarefa. Considerada a mais útil para um utilizador inexperiente.
- Em forma temática, estando os capítulos e secções concentradas numa área de interesse particular. Considerada mais útil para utilizadores com um nível de conhecimento mais elevado.
- Em forma de manual de referência, organizando as tarefas ou comandos a executar de uma forma alfabética ou agrupada segundo uma lógica pré-determinada e utilizando referências cruzadas para interligar tudo. Esta forma é considerada de grande utilidade para utilizadores avançados, que sabem com exactidão que tipo de informação procuram.

2.1.2 Documentação de Sistema

A documentação de sistema descreve o software do ponto de vista do programador. Tipicamente isto significa uma vista do tipo caixa branca, onde são incluídos todos os detalhes internos do software, de maneira a que estes sejam mais facilmente compreendidos [1]. Explica, assim, como funciona o produto, que processamentos faz, que componentes usa, etc.

Este tipo de documentação inclui documentos relativos a requisitos, arquitectura e *design*, código fonte do software, planos de teste e manutenção. Todos estes contêm informação relativa a uma fase do ciclo de vida do software e revestem-se de essencial importância para a evolução e manutenção dos sistemas (em especial o seu *design*, implementação e teste) já que facilitam a compreensão do produto pelos elementos actuais e futuros da equipa responsável pelo seu desenvolvimento.

Qualquer um dos tipos de documentação acima referidos contribui para a detecção precoce de possíveis erros ou inconsistências que, a serem detectados em fases mais avançadas do desenvolvimento, teriam consequências bastante mais graves quer a nível de custo quer a nível de satisfação do cliente final. É claro que para isso acontecer, é necessário que o esforço de documentação ao longo do projecto seja constante, não caindo no erro de apenas documentar ou actualizar a documentação existente no final de um ciclo ou do desenvolvimento do produto [3].

Existem certos factores que são importante ter em conta para criar uma boa documentação. Dentre eles podem-se destacar os seguintes:

- Completude.
- Clareza.
- Facilidade de acesso.
- Consistência.
- Estruturação.

É necessário que a documentação produzida cubra a totalidade das funcionalidades do sistema ou das questões que possam surgir. Estas tanto podem ser relacionadas com dúvidas de utilização ou configuração (no caso de documentação mais focada para o

utilizador, como os manuais) como relacionadas com esclarecimentos de opções tomadas ou de necessidades efectivas do software (no caso de documentação de sistema, focada para a equipa de desenvolvimento).

A linguagem utilizada deve ser clara e adequada ao público-alvo, de maneira a minimizar a probabilidade de ocorrência de mal-entendidos. Assim, deve ser evitado, por exemplo, o uso de termos bastante específicos em manuais dirigidos ao utilizador final ou, no caso de ser mesmo necessário, incluir um glossário claro e visível.

A documentação deve também ser de fácil acesso, estando disponível em formatos o mais universais possível para evitar que alguém não a possa consultar devido ao facto de não ter um software específico. Uma forma de minimizar este problema é a disponibilização de documentação web, que inclusivamente facilita as actualizações à mesma, dado que estas ficam instantaneamente visíveis, e pode conter mecanismos de pesquisa acelerando desta forma a localização de determinada informação.

Um dos aspectos mais importantes a ter em conta quando se fala em documentação é a consistência da informação nela presente. Por consistência entende-se quer consistência entre o que o software faz (e a forma como faz) e o que está escrito na documentação, quer consistência entre os vários documentos que fazem parte da documentação de um projecto. É importante que os elementos que consultam os documentos tenham uma única visão da realidade de maneira a fomentar a compreensão do sistema, ainda mais quando estes podem ter informação vinda de várias fontes. A consistência é também importante para evitar que quem consulta documentação tenha de "saltar" de documento em documento, procurando informação que pode estar repetida em alguns destes documentos, causando redundância de informação.

Para além do conteúdo, também a forma se revela importante quando se fala em documentação. A forma como um documento é estruturado e apresentado tem grande influência na maneira como este é aceite pelo seu leitor. Uma documentação pouco cuidada leva ao desinteresse, fazendo com que esta não seja aproveitada como devia para esclarecimento de quem a consulta.

Além da apresentação também a correcta organização dos conteúdos é uma característica de uma boa documentação. A instituição IEEE apresenta vários *standards* aplicáveis aos vários tipos de documentos que se podem encontrar num projecto de software [4]. Estes contêm tipicamente os componentes dos documentos, organizados em

questões essenciais e desejáveis e são utilizados pelas organizações como guias. Assim, tendo estes guias por base, as organizações podem criar documentos adaptados às suas necessidades, contendo os conteúdos que consideram necessários para que a qualidade da sua documentação seja reconhecida.

2.2 Gestão Documental

A gestão documental permite o controlo do ciclo de vida dos documentos numa organização – como são criados, revistos, publicados, consumidos, actualizados e até destruídos. De maneira a que possa ser considerada eficiente, a estratégia de gestão documental deve reflectir a cultura da organização onde está inserida. Quer isto dizer que se pode optar por uma abordagem com um controlo mais apertado do ciclo de vida dos documentos ou por uma abordagem com maior flexibilidade. Tudo depende da forma como a empresa está estruturada [5].

Para ajudar à actividade de gestão documental são normalmente usadas plataformas informáticas. Estas, quando bem desenhadas, facilitam a partilha e pesquisa de informação, na medida em que organizam os conteúdos de uma forma lógica e apresentam uma forma comum de criação e apresentação de conteúdos, promovendo a gestão do conhecimento.

Os sistemas de gestão documental apresentam, tipicamente, uma série de funcionalidades, tais como [6]:

- Gestão de metadados dos ficheiros – como o nome do autor, data de criação e descrição do documento;
- Integração – integração com outras aplicações tais como repositórios de informação ou outras aplicações para realizar operações mais específicas (*suites* de escritório ou envio de e-mails);
- Indexação – classificação de documentos a partir de palavras existentes nos mesmos ou nos seus metadados, para permitir recuperação e pesquisa dessa mesma informação;
- Armazenamento de documentos – onde são armazenados, durante quanto tempo, migração de local de armazenamento ou eventual destruição;

- Pesquisa de informação – pesquisa em conteúdo ou metadados de documentos através de termos ou combinação booleana de termos, utilizando as capacidades de indexação;
- Segurança – controlo de acesso a partes do sistema ou até a documentos em particular;
- Colaboração – permitir a vários utilizadores editarem um documento. Tipicamente esta edição não pode ser feita em simultâneo, existindo normalmente um mecanismo de bloqueio/desbloqueio que gere este aspecto;
- Gestão de *workflow* – definição e gestão de *workflow* que certos documentos devem seguir segundo regras de negócio configuradas.

A adopção deste tipo de soluções contribui de uma maneira clara para a existência de aumentos de produtividade nas organizações. Muito do tempo gasto é dispendido a procurar e manipular documentos, sendo que as soluções de gestão documental fornecem funcionalidades de pesquisa e consulta rápida de qualquer pedaço de informação existente na documentação.

Estes ganhos são ainda mais visíveis quando as capacidades acima mencionadas são integradas nos processos das organizações. O nível do serviço prestado aumenta se existir a capacidade de disponibilizar os documentos certos na altura certa, permitindo um eficiente contacto entre as organizações envolvidas em determinado processo. Por outro lado, isto permite também a "poupança" em termos de documentos físicos existentes, diminuindo o custo com a manutenção de espaços físicos necessários para o seu armazenamento e com as questões administrativas decorrentes da sua operação e actualização [7].

2.3 Gestão do Conhecimento

O conhecimento é definido à custa de outros dois conceitos que também são importantes para a documentação de software: o conceito de dados e o conceito de informação.

Dados são factos, afirmações passíveis de serem consideradas verdadeiras ou falsas.

Informação são dados interpretados, factos com significado num determinado contexto ou por via de determinados interesses.

O conhecimento é, assim, obtido a partir da informação através da integração desta com o conhecimento já existente. Caracteriza-se por ser constituído por grandes estruturas duradouras de factos com significado.

O conhecimento pode ser distinguido em duas grandes dimensões: o conhecimento tácito e o conhecimento explícito. O primeiro é tipicamente interno, estando presente de forma inconsciente. Assim, um indivíduo pode ou não aperceber-se daquilo que sabe e como atinge determinados resultados. Já o segundo refere-se ao conhecimento que um indivíduo apresenta de forma consciente e que pode comunicar aos outros por meio escrito [8].

A gestão eficaz do conhecimento está habitualmente ligada a certos objectivos organizacionais tais como melhoria da performance, transferência entre projectos de lições aprendidas ou desenvolvimento geral das práticas de colaboração na organização e ao conceito de aprendizagem organizacional. Isto permite a uma organização evoluir à medida que o conhecimento dos seus colaboradores se vai disseminando, aumentando o seu potencial de desempenho e desenvolvendo assim uma importante vantagem competitiva em relação à concorrência [9].

Como processos constituintes da gestão do conhecimento podem-se considerar os seguintes:

- Criação de conhecimento;
- Codificação/armazenamento do conhecimento;
- Transferência de conhecimento.

Estas actividades são de extrema importância já que apenas quando tomadas todas em consideração se pode obter uma eficaz gestão do conhecimento de uma organização. Criado o conhecimento (por meio de formações, por exemplo) é então muito importante que este seja correctamente armazenado (por meio de escrita de documentos, elaboração de modelos, criação de metodologias, por exemplo) para que a sua transferência seja possível. Esta transferência pode ser feita recorrendo aos artefactos produzidos ou mesmo por meio de partilha de experiências, observação ou *brainstorming* entre vários elementos da organização.

As ferramentas informáticas procuram ajudar todo este processo de gestão. Conhecidas como *Knowledge Management Systems*, apresentam funcionalidades para que potenciam a captura, a definição, o armazenamento, a categorização, a indexação e o relacionamento de conhecimento, além de permitirem pesquisar e subscrever conteúdos relevantes ou apresentar estes conteúdos de uma forma legível e flexível (aplicável em vários contextos). Tipicamente, apoiam ainda a coordenação de equipas de trabalho, fomentando a colaboração entre elementos [8].

Considerando todos os factores acima mencionados, pode-se verificar que tudo o que está presente na documentação de um projecto de software é conhecimento, que precisa de ser gerido de forma eficaz e eficiente. Com efeito, é importante que tudo seja correctamente disseminado pela equipa e pelos *stakeholders* do projecto, sem ambiguidades e incorrecções de qualquer espécie.

2.4 Ferramentas de Modelação

As ferramentas de modelação são aplicações informáticas que ajudam na criação de modelos representativos do software a desenvolver. Estes modelos permitem ter uma visão mais clara e concisa das várias partes e fases de um projecto de software.

Estas ferramentas suportam vários tipos de notação e semântica associadas com representações *standard* aceites na área de desenvolvimento de software. A linguagem *Unified Modeling Language* (UML) é uma das mais usadas e populares para analisar e modelar um sistema de software. Adicionalmente a modelar sistemas de software, UML é também usado para modelar processos de negócio ou representar estruturas organizacionais.

As ferramentas de modelação apresentam genericamente as seguintes funcionalidades [10]:

- Elaboração de diagramas – a criação de diagramas é o centro de uma ferramenta de modelação. Esta tem tipicamente um suporte avançado para as notações pretendidas, permitindo ao utilizador elaborar os diagramas pretendidos com o precioso auxílio da ferramenta, evitando assim alguns erros;
- Engenharia reversa – a engenharia reversa é a capacidade da ferramenta em ler código fonte de um programa e derivando daí os modelos correspondentes.

Esta funcionalidade é de grande utilidade, nomeadamente quando a responsabilidade por um determinado software passa de uma organização para outra, sem ser acompanhado pela respectiva documentação ou com esta desactualizada ou com fraca qualidade.

- Geração de código – capacidade da ferramenta em criar código fonte numa determinada linguagem a partir dos modelos criados pelo utilizador. Esta é a característica base da abordagem *Model Driven Development*, segundo a qual o desenvolvimento deve ser baseado em modelos.
- Transformação de modelos – capacidade de transformar um modelo em outro modelo. Conceito-chave na abordagem *Model Driven Development*, possibilita, por exemplo, a transformação de um modelo de domínio independente da plataforma num modelo específico para Java.
- Geração de documentação – capacidade de gerar algum tipo de artefacto que documente os modelos criados. São tipicamente relatórios descritivos dos modelos que podem ser de alguma forma configurados ao nível de aspecto e conteúdos presentes.

Este tipo de ferramentas é de extrema utilidade no processo de documentação de software. Os documentos mais ricos e completos contêm não só descrições textuais mas também modelos, que facilitam a compreensão visual de um sistema. Estas ferramentas dão o suporte necessário para que sejam elaborados estes modelos com grande qualidade e exactidão.

Existem no mercado várias ferramentas com estas funcionalidades e que são utilizadas extensivamente. De entre elas podem-se referir o Enterprise Architect [11], o Visual Paradigm [12], o Rational Rose [13] ou o Microsoft Visio [14], por exemplo.

2.5 Ferramentas de Processamento de Texto

Ferramentas de processamento de texto são aplicações informáticas usadas para a produção de qualquer tipo de documentação escrita. Por produção, neste contexto, entende-se a composição, edição e formatação de documentos.

Estas ferramentas apresentam uma série de funcionalidades das quais se podem destacar as seguintes [15]:

- Inserção/remoção/edição de texto;
- Copiar/mover secções;
- Definição de tamanho de página e margens;
- Procura e substituição de termos/expressões;
- Especificação de fontes;
- Notas de rodapé e referências cruzadas;
- Suporte para objectos (gráficos, imagens, etc.);
- Definição de cabeçalhos e rodapés;
- Definição de *layout* de documentos;
- Correctores ortográficos;
- Inserção automática de tabelas de conteúdos.

A documentação produzida no âmbito de um projecto de software é, na maioria das vezes, para ser impressa. Assim sendo, são as ferramentas de processamento de texto que dão todo o suporte à sua criação, apresentando-se indispensáveis num processo de documentação mais tradicional (isto é, não baseado na Internet). Estas permitem que o colaborador se possa concentrar mais na tarefa de definição dos conteúdos, dando uma preciosa ajuda nas tarefas acima descritas.

Estes tipos de ferramentas são extremamente comuns e têm lugar em qualquer computador. Aplicações como o Microsoft Office ou o OpenOffice estão plenamente disseminadas, podendo ainda ser referidos o WordPerfect, Lotus Symphony ou o Google Docs (este completamente *on-line*). De destacar ainda o LyX [16], que funciona como *front-end* para documentos criados com base no sistema tipográfico TeX [17] – sistema que procura distanciar o utilizador das preocupações com a apresentação visual da informação, permitindo que se concentre na estrutura desta, trabalhando com conceitos mais lógicos

tais como capítulos, secções ou tabelas. Este sistema é pouco apelativo para um utilizador comum, dado que é algo difícil criar novos modelos de documentos e é baseado em comandos, ao contrário dos outros mencionados que são do tipo WYSIWYG ("*What You See Is What You Get*"). O LyX traz uma interface mais agradável ao T_eX, permitindo ultrapassar as dificuldades enumeradas.

Capítulo 3.
Integração de Artefactos de Software

A construção de documentos a partir de dados provenientes de ferramentas de modelação é um problema de grande complexidade, que tem sido abordado por diferentes prismas.

Verifica-se, então, a existência de abordagens tão distintas como a construção dos documentos recorrendo notações específicas desenvolvidas para permitir a introdução de diversos dados em diferentes pontos do documento ou soluções automatizadas e integradas em certas ferramentas de modelação, que permitem a obtenção de documentos em formatos nativos dos processadores de texto mais disseminados no mercado.

As primeiras são soluções que permitem obter uma grande flexibilidade ao nível de extensão de funcionalidades e configuração dos documentos a produzir, mas são tipicamente menos *user-friendly*, sendo necessários conhecimentos algo profundos para conseguir criar e manter um documento nestes moldes.

As soluções mais automatizadas e integradas já nas ferramentas pretendidas são bastante menos extensíveis (até porque, na maioria das vezes, são soluções proprietárias e adaptadas a um contexto muito específico), mas têm vantagens na facilidade de utilização e efeitos práticos do resultado final.

Este capítulo procura dar uma visão sobre algumas das abordagens existentes para a problemática em questão, descrevendo-as nas suas linhas orientadoras e particularidades consideradas relevantes.

3.1 Documentos Heterogéneos

Sendo uma das características de uma boa documentação o facto de se apresentar completa, permitindo que quem a consulta tenha visíveis múltiplas vistas e diferentes níveis de abstracção sobre o sistema em questão, é conveniente que contenha diversos conteúdos. Estes podem estar apresentados em diferentes notações (texto, código, modelos, imagens) e são tipicamente provenientes de várias fontes (ficheiros de código ou de modelos, especificações efectuadas, etc.). Os artefactos resultantes da combinação destes conteúdos são designados por documentos heterogéneos [18] [19].

Toda esta diversidade traz bastantes inconvenientes ao nível de consistência de informação, já que os diversos conteúdos podem ser desenvolvidos por elementos diferentes da equipa e usando para o efeito diferentes editores, provocando uma troca de ambientes relativamente frequente e levando ao aparecimento de pontos de ineficácia no processo de documentação.

Para procurar diminuir o impacto causado pela diversidade de fontes de informação têm sido propostas algumas abordagens que procuram permitir uma integração mais suave de conteúdos na documentação.

3.1.1 Literate Programming

A técnica conhecida por *Literate Programming*, criada por Donald Knuth no início dos anos 80, tem como filosofia alterar a forma como se constroem programas de computador: em vez de considerar que o objectivo é dizer ao computador aquilo que ele deve fazer, procurar considerar que o objectivo é explicar aos humanos o que se pretende que o computador faça [20].

Assim, ao procurar explicar o pretendido, quem desenvolve um programa recorrendo a esta técnica pode considerar que está a escrever uma obra de literatura. Daí o nome de *Literate Programming*.

Esta técnica consiste em criar um documento, onde esteja a documentação e o código fonte do programa, organizado psicologicamente para que seja compreendido por humanos.

O facto de apresentar a documentação e o código no mesmo documento facilita a manutenção da consistência entre ambos os tipos de conteúdos, embora permitindo que evoluam independentemente.

Uma vez que esta técnica pretende que se explique aos humanos o que o computador deve fazer, é especialmente importante que se construam documentos com uma organização psicológica que facilite esta comunicação, organização esta que não deve ser limitada pela estruturação esperada por um compilador. A organização do documento depende, então, da documentação estando o código apenas incluído onde e quando necessário. Assim é possível não só descrever a função de determinado código mas também apresentar e discutir alternativas ou detalhes prévios considerados relevantes.

Tal como numa obra de literatura, um programa desenvolvido com esta técnica deve procurar garantir a facilidade de leitura e compreensão do código. Para isso deve recorrer a referências cruzadas entre diversas secções ou partes do documento, a índices e a funcionalidades de impressão adequada (*pretty-printing*). O programador deve também incluir imagens ou tabelas que ache necessários para a adequada compreensão do programa.

Sendo que um programa construído com esta técnica se apresenta como um documento organizado de uma forma que o autor considere adequada para a sua compreensão, constituído por partes explicativas e fragmentos de código, é necessário que seja transformado num formato que possa ser compreendido por um compilador (para que seja possível obter o programa em si) e também num formato que possa ser lido facilmente por humanos (a documentação).

As ferramentas de suporte à abordagem *literate programming* (WEB, CWEB ou noweb, por exemplo) fornecem, então, dois tipos de funcionalidades [21]:

- *Tangler* – extrai o código fonte existente no documento, de forma a que possa ser passado como entrada de um compilador;
- *Weaver* – produz documentos passíveis de serem lidos e compreendidos por humanos.

Estas encontram-se representadas de forma gráfica na Figura 1.

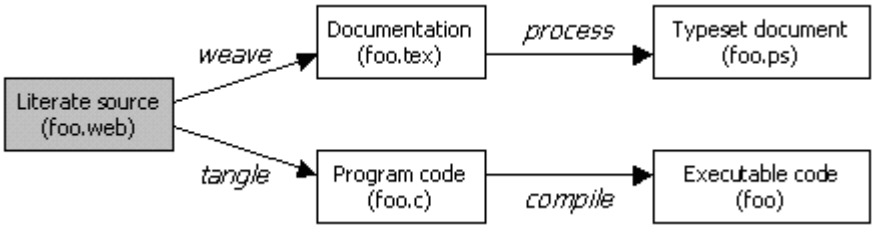

Figura 1 – Esquema Abordagem Literate Programming (extraída de [21])

A escrita de programas usando *Literate Programming* é geralmente baseada na colocação de comandos/*tags* no início das linhas/secções do documento, de maneira a indicar se aquilo que se vai iniciar é considerado documentação ou código. Os fragmentos de código podem ser explicados por qualquer ordem (considera-se tipicamente que um fragmento é explicado pelo texto que o precede imediatamente), estando à responsabilidade do *tangler* reordená-los de uma forma legível para um compilador.

Todas estas características fazem com que esta abordagem seja um incentivo à produção de documentação com qualidade, legível e compreensível, fomentando a preservação da informação ao longo do tempo e a independência em relação aos indivíduos que desenvolveram o programa. Procura-se também diminuir os custos de manutenção, já que haverá muito menos esforço para perceber a arquitectura e funcionalidades do programa. O esforço acrescido em documentar as soluções obtidas leva também ao surgimento de possíveis pontos obscuros na abordagem seguida, fomentando a discussão destes antes de se tornarem problemas mais sérios e também a compreensão do programa como um todo [21].

Apesar de tudo isto, esta abordagem é pouco utilizada em projectos reais devido a certos factores dos quais se podem destacar os seguintes [1]:

- Necessidade do uso combinado de três linguagens: a linguagem de programação, a linguagem de formatação do documento final (tipicamente baseada em sistemas T_eX) e a linguagem que permite a interligação dos fragmentos de código e documentação;
- Formato do documento fonte demasiado complexo, que compromete a legibilidade e compreensão durante o desenvolvimento do programa;

- Demasiado esforço adicional no desenvolvimento de pequenos programas;
- Incompatibilidades com ferramentas de manipulação de código fonte, sejam elas ambientes de desenvolvimento integrados, *debuggers*, geradores de código ou ferramentas de *refactoring* automático, devido ao facto de os documentos estarem escritos com uma organização diferente da esperada por um compilador.

3.1.2 Abordagens de Documentação Single Source e Multiple Source

Algumas abordagens alternativas ao *literate programming* podem ser classificadas em duas categorias: *single source* e *multiple source*.

A abordagem *single source* segue a mesma ideia de apresentar documentação e outros conteúdos em apenas um ficheiro, diminuindo assim as possibilidades de aparecimento de inconsistência, dado que não há repetição de conteúdos por diversos locais.

Os mecanismos Javadoc [22] e Doxygen [23] podem ser vistos como exemplos de aplicação desta abordagem. Estes consistem em documentar funções, métodos ou classes através de uma sintaxe pré-definida de comentários nos próprios ficheiros de código. Estas ferramentas produzem depois documentação, tipicamente no formato HTML, com ligações entre partes que estão relacionadas.

A abordagem *multiple source* é aquela que, tradicionalmente, a documentação usualmente produzida segue. Mantém a documentação e todos os conteúdos que possam nela estar integrados (ou relacionados) em ficheiros separados. A combinação dos dois é feita através de referenciação de linhas ou cópia de conteúdos para a documentação. As inconsistências têm grande probabilidade de acontecer com uma abordagem deste tipo se não for suportada eficazmente por ferramentas que consigam gerir e actualizar as referências existentes. Ferramentas como o DOgMA [24] procuram aliviar este problema, mas verifica-se que a edição de conteúdos (sejam eles código, documentação ou modelos) fora do "seu" ambiente (isto é, editores específicos para a sua manipulação) é um grande entrave à sua aplicação mais global.

3.1.3 XML-based Documentation

A linguagem de anotação XML surge na documentação quase como uma sequência natural das suas características mais visíveis.

A sua estrutura hierárquica, em conjunto com as possibilidades que oferece em termos de pesquisa e extracção de informação e com o facto de ser aberta e facilmente extensível a novas funcionalidades, são factores sem dúvida importantes.

O XML pode ser aplicado até em cima de sistemas já existentes de *literate programming*, por exemplo [25]. Assim, o documento produzido fica bem mais simples de processar, dada a possibilidade de identificação das partes constituintes através das *tags* XML, ao mesmo tempo que facilita a referenciação entre os diferentes fragmentos de códigos espalhados ao longo do documento e a produção de documentação em formatos de documentação baseados em XML, como o DocBook [26], por exemplo (este, por sua vez, pode ser facilmente transformado em formatos como HTML ou PDF para que possa ser lido de uma forma mais simples).

Há já bastante trabalho realizado na área para facilitar a representação de diversos tipos de conteúdos associados a documentação de software em formato XML. Existem dialectos para efectuar representações de código Java (JavaML [27]) ou C++ (cppML [28]) ou de modelos UML [29].

O uso de tecnologias associadas a XML como XQuery ou XSLT contribui para que a pesquisa de informação na documentação saia facilitada e potenciada.

3.1.4 Wiki-based Documentation

Com o passar dos anos a Web foi tendo cada vez mais força em todos os domínios quer relacionados com as tecnologias de informação quer relacionados com o funcionamento da sociedade.

Produzir documentação de software baseada na web passou, então, a ser relativamente comum. Apesar de ser tipicamente mais atractiva, de baixo custo, com possibilidade de incluir conteúdos multimédia e de ser pesquisável, tem também os seus problemas, tais como o facto de ser mais confortável ler documentos fora do ecrã. Outro

dos problemas visíveis, nomeadamente no que à actualização de documentação diz respeito, é o facto de os *browsers* não apresentarem capacidade de edição de páginas [18].

As ferramentas denominadas por Wiki (abreviado de WikiWikiWeb) [30] são aplicações colaborativas que permitem a edição interactiva de documentos, usando apenas um navegador web, sem a existência de necessidade de revisões prévias à sua publicação. As páginas são construídas recorrendo a uma linguagem de anotação simples que suporta formatação de texto e ligadas com um mecanismo denominado por *wiki names*. A maioria dos wikis usa o formato CamelCase [31] para estabelecer as ligações entre páginas. Isto significa que palavras escritas neste formato são automaticamente reconhecidas como possíveis ligações para páginas com o mesmo nome.

Os wikis apresentam também funcionalidades ao nível de controlo de utilizadores de maneira a evitar o uso excessivo e abusivo destes sistemas. Assim sendo, existem mecanismos que exigem o registo de utilizadores para que lhes seja possível editar páginas, sendo estas apenas apresentadas em modo *read-only*. Estes sistemas servem, geralmente, para proteger conteúdos mais sensíveis que não é suposto que sejam visíveis na Internet ou para evitar a edição de conteúdos com informações falsas (como se encontrava a acontecer na Wikipedia, o sistema wiki mais disseminado a nível mundial).

Dadas as suas características, os wikis são usados mais na comunidade de desenvolvimento de software e servem muitas vezes de suporte ao processo de desenvolvimento e documentação, permitindo a colaboração na elaboração de versões preliminares de documentos de requisitos, *design*, arquitectura ou mesmo troca de ideias durante o processo de implementação.

Devido a isto, muitos *wiki engines* apresentam suporte específico para sintaxe em várias linguagens de programação ou desenho de diagramas UML (como o SnipSnap [32], por exemplo - Figura 2). Apesar deste suporte, a consistência entre os diversos artefactos não é ainda uma preocupação geral destes sistemas, havendo no entanto a destacar o XSDoc [33] como excepção neste aspecto.

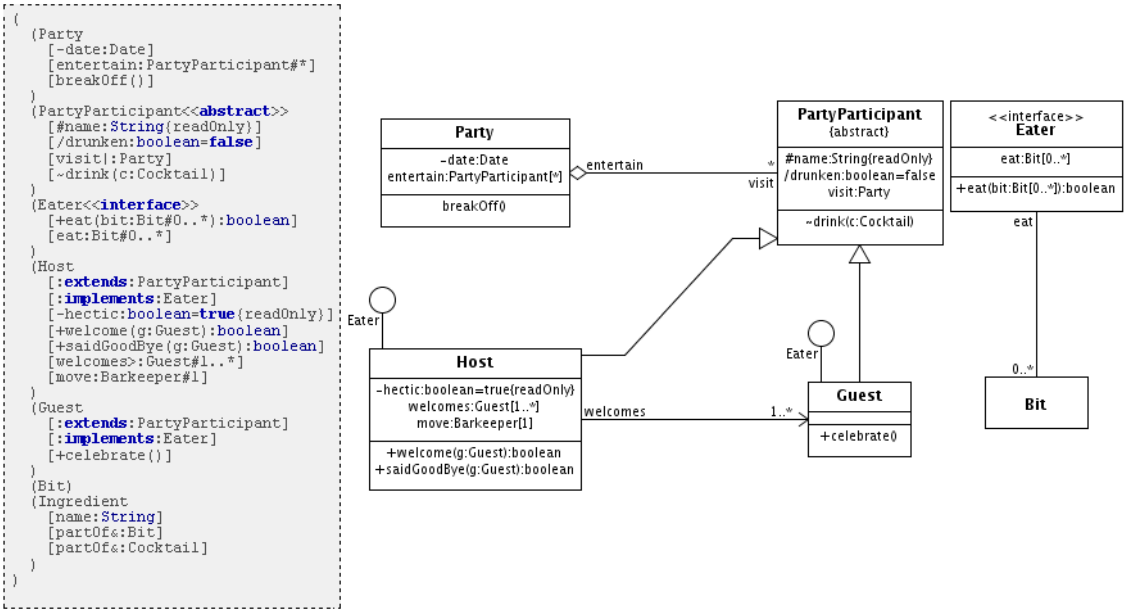

Figura 2 – Desenho de UML no SnipSnap (adaptado de [29])

Com efeito, a generalidade dos wikis limita-se a suportar a edição e formatação das páginas de uma forma colaborativa, sem se preocupar com manter a consistência quando a informação é proveniente de outras ferramentas como ferramentas de modelação.

O projecto Doc-it! estende vários *wiki engines*, no caso o SnipSnap, o MoinMoin e o XSDoc, para produzir uma solução que permita a elaboração de documentação de *frameworks* orientadas a objectos, de uma forma ágil. De destacar a sua característica de referenciação dinâmica de conteúdos, que permite enriquecer a documentação produzida com vários artefactos e procurando ultrapassar o problema da consistência. Assim, foi desenvolvido um mecanismo baseado em *plugins* para inclusão e representação de conteúdos em vários formatos. Desta forma, na *wiki page*, são apenas referenciados os conteúdos pretendidos (através da indicação, por exemplo, do método e da classe pretendidos no caso da linguagem Java) e a ferramenta trata de incluir o código correspondente, na altura da geração da página, directamente obtido do ficheiro fonte. Esta ferramenta tem como principal inconveniente o facto de ser ainda uma prova-de-conceito, o que leva a que, nesta altura, só tenha suporte para conteúdos vindos de ferramentas de modelação se estes se encontrarem num formato gráfico ou num formato facilmente transformável num formato gráfico, tal como o XMI.

3.2 Geração de Documentação a partir de Ferramentas de Modelação

A geração de documentação com conteúdos vindos de ferramentas de modelação é um processo que fica teoricamente mais facilitado se já se apresentar suportado nativamente por esta classe de ferramentas.

Verifica-se que algumas das ferramentas mais utilizadas apresentam já integradas funcionalidades relacionadas com o desenho de *templates* e a geração de documentos com certos conteúdos.

Devido ao elevado número de ferramentas de modelação existentes no mercado e porque seria impensável analisá-las a todas, optou-se por restringir esta análise a três ferramentas: Rational Suite, Visual Paradigm e Enterprise Architect. Estas foram escolhidas tendo em conta o cenário onde iria ser desenvolvido o caso de estudo, a sua aplicabilidade ao domínio da Critical Software, S.A. e as funcionalidades ao nível de geração de documentação (factor em que se centrou a análise efectuada).

3.2.1 Rational Suite

As ferramentas da Rational são uma inevitabilidade quando se fala em soluções de desenho e modelação de software. Apesar de apresentar uma gama bastante variada de produtos, no âmbito deste trabalho interessa analisar em especial as ferramentas Rational Rose e SoDA.

Rational Rose é a denominação comum de uma família de ferramentas que procura auxiliar na modelação e *design* de aplicações informáticas. Tem por base a notação UML e o paradigma da orientação aos objectos para proporcionar uma modelação visual dos componentes e aplicações.

As ferramentas incluídas nesta família de produtos (Enterprise, Modeler, Data Modeler, Technical Developer, Developer for Java, Developer for Unix, Developer for Visual Studio) diferem entre si no suporte específico para determinadas funcionalidades. O Rational Rose Enterprise é a versão mais completa deste software, apresentando funcionalidades tais como [34]:

- Suporte para UML;
- Geração de código;

- Integração com outras ferramentas Rational, nomeadamente controlo de versões (ClearCase) e documentação/*reporting* (SoDA);
- *Add-In* para modelação para aplicações Web-based;
- Suporte para desenho de bases de dados, com possibilidade de representação de integração de dados e requisitos aplicacionais através de diagramas lógicos e físicos;
- Suporte para a criação de DTD's.

Como se pode ser pela lista acima, esta é uma ferramenta com um suporte bastante completo para tarefas de modelação de software.

A possibilidade de gerar documentos com os conteúdos presentes na ferramenta também não ficou esquecida, através da integração com uma outra ferramenta Rational, o SoDA (*Software Documentation Automation*).

Esta solução procura automatizar a criação e manutenção da documentação e relatórios que possam estar associados a um projecto de software. Consegue isto extraindo informação das bases de dados das diversas ferramentas de apoio ao desenvolvimento de software, nomeadamente as da Rational, em relação às quais apresenta uma integração bastante eficaz [35].

Permitindo a geração de documentos nos formatos Word e HTML, utiliza *templates* que podem ser customizados para produzir estes artefactos.

Ao ligar-se directamente às ferramentas pretendidas procura manter a consistência entre a informação original e a presente na documentação, evitando a duplicação de dados já que nos documentos apenas existem referências para as fontes.

O SoDA pode trabalhar com dois tipos de artefactos: relatórios e documentos [36]. A diferença essencial entre estes é que os relatórios são uma visão dos dados num determinado momento, impossíveis de actualizar, enquanto que os documentos podem ser actualizados total ou parcialmente para reflectir evoluções naturais no processo de desenvolvimento de software.

De especial interesse é também a sua capacidade para actualizar apenas uma parte da documentação, aumentando assim a performance quando as alterações não são muito significativas.

O SoDA é, então, uma alternativa bastante viável para a geração de documentação com conteúdos vindos de ferramentas de modelação caso se opte pela utilização das ferramentas da família Rational. O grande inconveniente são os elevados custos que estas comportam para os utilizadores, proibitivos em muitas organizações que não apresentam o poderio financeiro necessário.

3.2.2 Visual Paradigm for UML

Criado pela Visual Paradigm, o Visual Paradigm for UML [12] é uma solução desta empresa para facilitar as organizações no que diz respeito às tarefas de desenho, integração e implementação das suas aplicações e bases de dados associadas. Ao fornecer funcionalidades que permitem efectuar estas tarefas recorrendo a um suporte visual (à base de modelos e diagramas), procura acelerar e maximizar o contributo dos elementos da equipa responsável pelo projecto.

Das funcionalidades disponíveis podem-se referir, para além das tradicionais nesta classe de ferramentas (suporte para UML, modelação de requisitos e bases de dados, geração de código), as seguintes:

- Modelação de processos de negócio – recorrendo à abordagem BPMN (*Business Process Modeling Notation*);
- Mapeamento entre objectos e entidades relacionais correspondentes;
- Modelação colaborativa – através da integração com ferramentas como o CVS ou o SVN;
- Interoperabilidade com outras ferramentas do mesmo género disponíveis no mercado – através da representação de diagramas em formato XMI e também possibilidade de importação de ficheiros nativos de Rational Rose;
- Integração com ambientes de desenvolvimento integrados – Eclipse, NetBeans, IntelliJ IDEA, por exemplo;

- Geração de documentação em vários formatos (PDF, HTML, MS Word).

A funcionalidade de geração de documentação presente nesta ferramenta permite gerar relatórios a partir dos modelos nela construídos (Figura 3).

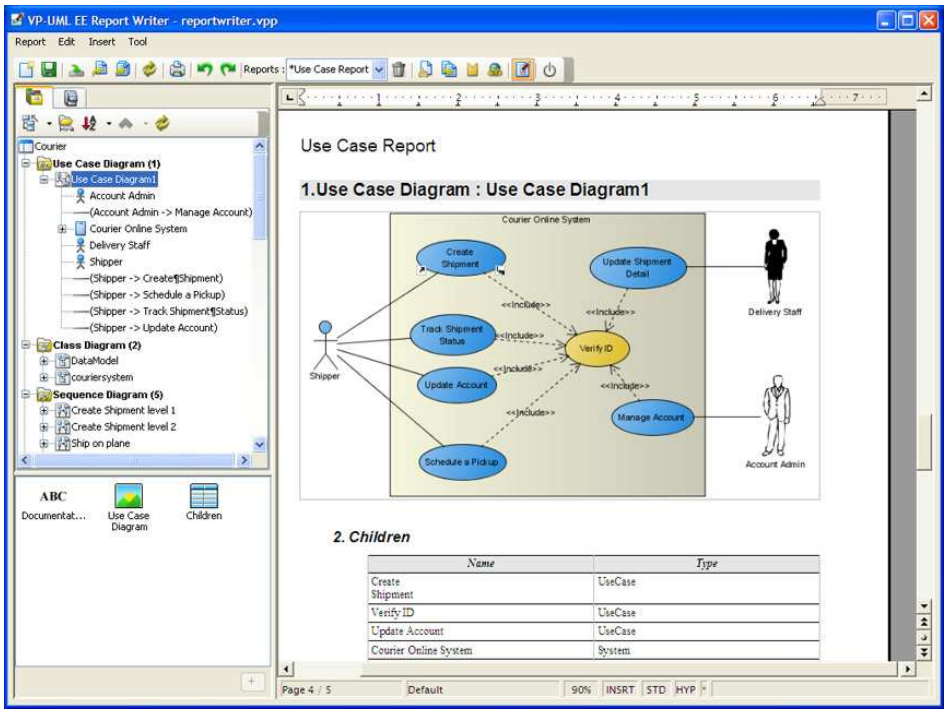

Figura 3 – Ferramenta de geração de documentação do Visual Paradigm (extraída de [12])

É uma funcionalidade que dá ao utilizador a possibilidade de escolher um dos *templates* já definidos na ferramenta para servir de *placeholder* para o conteúdo relevante a incluir na documentação. Usando esta abordagem, o utilizador apenas tem de preencher alguns campos comuns (como o título do documento ou nome da organização, por exemplo) e escolher, por meio de *checkboxes*, os diagramas ou entidades a incluir.

É possível também que seja o utilizador a indicar a configuração de cabeçalhos, rodapés, capas e mesmo organização dos conteúdos no documento. Esta abordagem dá uma maior flexibilidade ao utilizador, à custa de maior complexidade na construção do modelo do relatório. Este pode ser actualizado de maneira a manter a consistência com os modelos construídos e exportado para o formato pretendido. De realçar que, após a exportação, o documento PDF, HTML ou MS Word produzido não pode ser actualizado.

Apenas o relatório que lhe serve de base tem essa possibilidade, sendo necessário re-exportá-lo para ter acesso às actualizações efectuadas.

Este último aspecto é um grande inconveniente no que à manutenção da consistência da informação diz respeito. Se se pretender que o relatório gerado sirva de base à documentação técnica, adicionando posteriormente mais descrições textuais ou conteúdos vindos de outras fontes, o processo de construção do documento final tem de ser sempre repetido e efectuado manualmente de cada vez que há uma alteração nos modelos.

Apesar disso, a possibilidade de ser o utilizador a definir a forma e organização do relatório a produzir torna-se bastante interessante, em especial devido a possíveis questões de certificações de qualidade, onde são tipicamente exigidos formatos bem definidos de documentação e que sigam alguns padrões já existentes.

3.2.3 Enterprise Architect

O Enterprise Architect [11], propriedade da SparxSystems, é uma ferramenta que apresenta funcionalidades ao nível da modelação de software, permitindo ainda geração de código, gestão de requisitos e suporte para gestão de projectos e de testes.

Uma das suas características é o facto de suportar a partilha dos modelos desenvolvidos pelos utilizadores. Esta partilha pode ser conseguida através do armazenamento do ficheiro fonte da modelação numa *drive* partilhada numa rede, através da replicação dos ficheiros fontes, tendo cada utilizador a sua cópia, e inclusivamente através da utilização de um repositório de bases de dados relacionais como local de armazenamento. Ao optar por esta última possibilidade, o utilizador pode usar diversos sistemas de gestão de bases de dados, tais como Oracle 9i e 10g, SQL Server, MySQL, PostgreSQL.

O Enterprise Architect tem também um utilitário para realizar comparações (operações de *diff*) entre modelos, comparando o modelo actual com outro ficheiro no formato XMI ou até com uma *baseline* do projecto actual previamente armazenada (funcionalidade esta também suportada nativamente pela aplicação).

Além de suportar a modelação da aplicação a desenvolver, também dá uma ajuda nas tarefas relacionadas com testes (testes unitários, de integração, de sistema, de aceitação), com a gestão de projecto (identificação de riscos, detalhes ao nível do esforço dispendido nos elementos dos modelos, definição e utilização de métricas) e com a parte de manutenção da aplicação desenvolvida (recolha de informação de falhas e funcionalidades para permitir a rastreabilidade dos problemas encontrados).

Tal como o Visual Paradigm, apresenta um módulo que permite a geração de documentação. Esta pode ser obtida no formato RTF ou HTML, sendo baseada em *templates*. Estes podem ser definidos pelos utilizadores, utilizando para o efeito um editor WYSIWYG ("*What You See Is What You Get*") bastante flexível. O editor presente na ferramenta permite que seja definido o aspecto da documentação, ao nível de capas, cabeçalhos, rodapés, tabelas de conteúdos, entre outros, e também o seu conteúdo. Utilizando um sistema de *tags*, permite um correcto posicionamento dos conteúdos pretendidos que podem inclusivamente ser filtrados através de critérios de selecção previamente definidos.

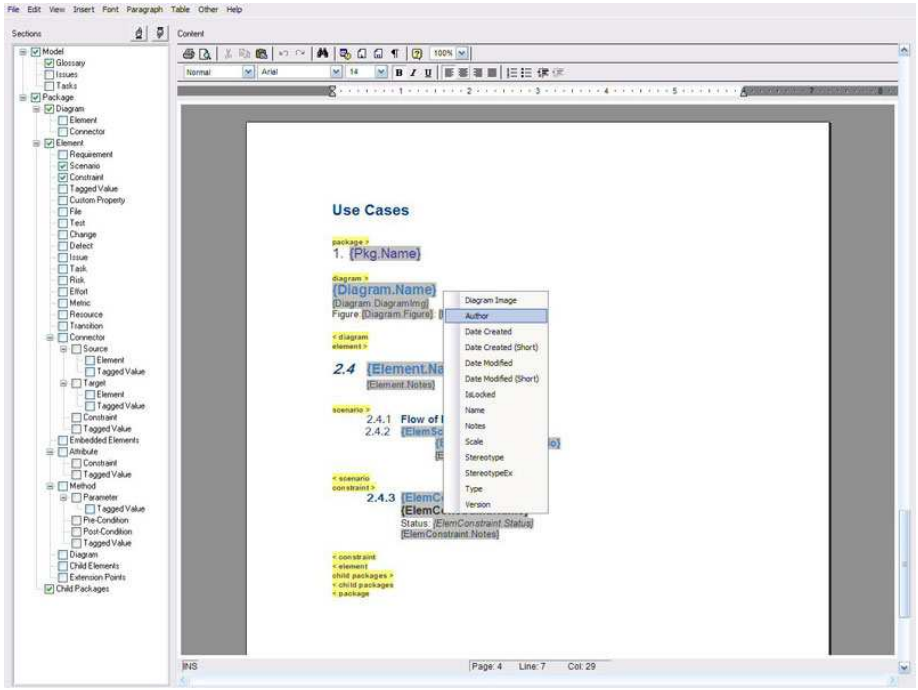

Figura 4 – Editor de *templates* do Enterprise Architect (extraída de [11])

Este editor é a grande força deste módulo. No entanto, e à semelhança da solução presente no Visual Paradigm, tem o inconveniente de os documentos saírem do controlo

da ferramenta a partir do momento em que são gerados. Assim, para futuras alterações dos modelos ter-se-á que re-gerar os documentos correspondentes, perdendo as alterações já efectuadas ao primeiro documento (estas podem ser mantidas através do método manual de cópia de conteúdos).

3.3 Geração de Relatórios

Algumas das ferramentas de modelação existentes, tal como o Enterprise Architect, por exemplo, apresentam a capacidade de armazenar os conteúdos ali produzidos em bases de dados relacionais.

Assim, faz sentido mencionar algumas ferramentas específicas de *reporting* que podem ser utilizadas com vista a produzir secções de documentos a partir dos dados presentes nas bases de dados das ferramentas de modelação.

Ferramentas como SQL Reporting Services [37], Crystal Reports [38] ou BIRT [39] oferecem funcionalidades de extracção, processamento e apresentação de dados. Estes dados podem ser provenientes de várias fontes (SQL Server, no caso de SQL Reporting Services, e diversas bases de dados ou ficheiros XML ou de texto no caso das outras duas ferramentas mencionadas) e podem ser agrupados de diversas formas para que façam sentido no contexto pretendido.

É possível criar listagens, tabelas, gráficos e especificar a forma de organização destes elementos. Os relatórios produzidos são, tipicamente, exportáveis para outros formatos, nomeadamente XML, HTML, CSV, PDF ou alguns formatos de imagem e podem ser construídos recorrendo às funcionalidades de integração destas soluções com ambientes de desenvolvimento integrados (Microsoft Visual Studio – no caso de SQL Reporting Services – ou Eclipse – no caso do BIRT).

Apesar de terem como grande inconveniente o facto de, devido a gerarem relatórios, não poderem actualizar automaticamente todo ou parte de um documento (já que estes são produzidos a partir da exportação dos relatórios), podem permitir facilitar a tarefa de geração de documentos heterogéneos através das suas capacidades avançadas de construção e formatação de documentos. Posteriormente será necessário ou copiar os conteúdos obtidos para a documentação ou então acrescentar as descrições textuais, ou

outros conteúdos que não sejam possíveis de gerar com estas ferramentas, ao documento produzido.

3.4 Síntese das Possibilidades Estudadas

A pesquisa levada a cabo permitiu perceber melhor as possibilidades existentes ao nível de integração de artefactos de software, nomeadamente integração de conteúdos de ferramentas de modelação em documentos.

Assim, é importante reter que existem várias abordagens possíveis para este problema.

As abordagens como Literate Programming ou Wiki-based Documentation permitem a construção dos documentos e integração de conteúdos através de notações que possibilitam a identificação de partes de documentos, tornando-se bastante flexíveis e extensíveis.

As abordagens como a geração de documentação a partir das próprias ferramentas de modelação ou a geração de relatórios a partir de conteúdos presentes em bases de dados são tipicamente mais fáceis de utilizar e com as quais se podem obter documentos bastante completos e apelativos.

Capítulo 4.
Integração Flexível de Modelos em Documentos

A pretensão de facilitar a construção e actualização de documentos heterogéneos constituídos por dados presentes em ferramentas de modelação tem sido extensamente procurada, como se verifica pelo descrito no capítulo anterior. No entanto, continuam a haver questões que se levantam neste processo.

A flexibilidade de escolha dos dados que serão visíveis nos documentos, as possibilidades de evolução dos documentos ao longo do seu ciclo de vida, mantendo a consistência de informação entre as ferramentas de modelação e os documentos e as funcionalidades ao nível de escrita colaborativa dos mesmos são algumas destas questões que não se encontram simultaneamente respondidas de uma forma geral.

As questões acima enumeradas são abordadas neste capítulo de uma forma mais extensa, procurando-se perceber qual o caminho a seguir para a obtenção de uma solução satisfatória.

4.1 Flexibilidade

Devido à sua natureza, os documentos heterogéneos podem incluir os mais variados conteúdos. Estes podem passar por diagramas, código-fonte ou descrições textuais de elementos tais como requisitos, testes e especificações de bases de dados.

Estes conteúdos podem-se tornar bastante extensos devido à quantidade de informação existente nas ferramentas de modelação, sendo bastante útil que o colaborador que constrói o documento possa ter a possibilidade de escolher apenas os dados que pretende que sejam incluídos nesse momento no documento.

O conceito de flexibilidade na integração de dados é, então, baseado na escolha dinâmica de conteúdos visíveis no documento, por parte do colaborador que o constrói e na altura da geração/actualização do artefacto.

Este conceito está presente nas abordagens baseadas em *literate programming* e *wiki-based documentation*. Efectivamente, ambas as abordagens permitem a definição dos conteúdos da documentação na hora da sua construção uma vez que é o próprio colaborador que escreve a documentação, definindo, por meio de marcas, o local onde quer inserir o conteúdo e também qual o conteúdo a inserir.

As abordagens baseadas em soluções de *reporting* e geração directa de documentação a partir de ferramentas de modelação não têm esta flexibilidade tão visível. Como são baseadas em *templates*, a flexibilidade existente é obtida a partir da definição de vários *templates*. Para cada combinação de conteúdos pretendida é, então, necessário definir um novo *template*. Este factor provoca um esforço extra na quantidade de trabalho necessária para obter esta funcionalidade, também porque a definição destes *templates* é, tipicamente, da responsabilidade de elementos ligados à área da Qualidade de Software, com vista a garantir a manutenção dos *standards* de Qualidade existentes na organização.

Além disso, por vezes é ainda necessário que esta definição de *templates* seja acompanhada da sua adição na ferramenta responsável por gerar os documentos pretendidos. Esta adição pode até ter de ser feita por cada utilizador da ferramenta. Exemplo desta situação é o uso do motor de *reporting* do Enterprise Architect, no qual cada *template* definido tem de ser explicitamente adicionado a cada projecto criado na ferramenta e em cada computador onde o ficheiro relativo ao projecto é usado.

4.2 Evolução Documental

À medida que um projecto de software vai evoluindo (por questões relacionadas com avanço nas fases de desenvolvimento, correcção de erros ou, simplesmente, manutenção evolutiva), também a sua documentação deve evoluir.

Este factor é de extrema importância para o sucesso global do projecto. Efectivamente, todas as alterações a qualquer parte do projecto devem ser reflectidas na documentação existente, de maneira a que possam ser consultadas quando necessário e por qualquer elemento ligado ao mesmo.

A questão da evolução de documentos heterogéneos é ainda mais complexa, dada a possibilidade de inclusão de dados provenientes de várias fontes.

Esta evolução pode ocorrer de forma incremental ou iterativa. De forma incremental, se forem sendo acrescentadas novas informações à documentação existente ou de forma iterativa se forem sendo revistas as informações presentes na documentação. Estas duas formas complementam-se bastante bem, dado que dentro de cada ciclo de incrementação podem existir um ou mais ciclos de iteração, potenciando assim a qualidade e completude da documentação produzida [40].

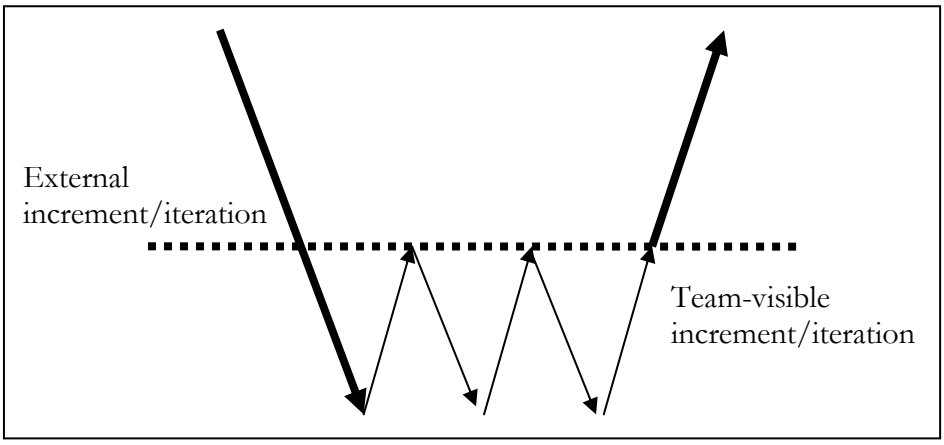

Figura 5 – Ciclos de incrementação e iteração (adaptada de [40])

A evolução da informação relativa a um projecto de software é feita nas suas várias vertentes (modelos, descrições textuais, código fonte) recorrendo às ferramentas específicas para cada tipo de conteúdo que são usadas no projecto. O grande desafio é conseguir fazer com que os documentos heterogéneos produzidos contenham sempre os dados mais

actuais, oriundos das mais diversas fontes, com o menor esforço por parte dos elementos da equipa.

A abordagem de documentação baseada em wikis permite que os documentos aí criados possam integrar conteúdos actuais de ferramentas de modelação, mediante o desenvolvimento de funcionalidades de integração que serão executadas na altura da gravação da página editada ou da exportação da documentação para formatos como PDF (*Portable Document Format*), DOC (formato nativo de documentos MS Word) ou outro. Estes documentos apresentam uma estrutura orgânica, significando isto que a sua estrutura e o seu conteúdo estão abertos a edição e evolução naturais e não pré-definidas [41]. Esta aparente "falta de controlo" pode ser colmatada com a adição de mecanismos de validação através de *XML-schemas*, que permitem garantir determinados *standards* ao nível da qualidade da documentação produzida.

A abordagem de documentação baseada em soluções de *reporting* existentes em ferramentas de modelação é, neste aspecto, bem menos capaz. Efectivamente, apesar de permitir uma passagem mais fácil da informação das ferramentas para formatos de documentação devido à existência de motores que realizam esta operação de forma fácil e rápida (como se verificou no capítulo anterior), esta abordagem descura normalmente a questão da actualização dos documentos produzidos.

Os documentos gerados por estas soluções são considerados finais e independentes. Finais porque não é suposto que sejam re-alimentados ao motor de onde saíram para serem actualizados e independentes porque são gerados tipicamente documentos inteiros, por vezes até com índices, sendo depois necessário executar operações de cópia para que esta informação seja integrada nos documentos heterogéneos. Isto faz com que, a cada alteração efectuada na ferramenta de modelação, haja a necessidade de re-gerar um documento e, manualmente, copiar o seu conteúdo para o documento heterogéneo, apagando a informação desactualizada.

Este trabalho procura apresentar uma solução que permita criar e manter actualizada documentação, mesmo quando esta necessita de manter certos padrões devido, por exemplo a questões relacionadas com certificações obtidas.

4.3 Escrita Colaborativa de Documentos

Em paralelo com a questão da evolução documental, verifica-se também a questão da escrita colaborativa de documentos.

Sendo a documentação de software uma actividade tipicamente levada a cabo por mais do que um elemento, verifica-se a necessidade de existência de um mecanismo eficaz e eficiente para garantir que todos os elementos trabalham sobre a mesma base e que não há perda de alterações efectuadas.

Em organizações onde se verifica a existência de uma abordagem de documentação mais tradicional, baseada na manutenção de documentos em formatos de *suites* de escritório, a existência de um repositório de controlo de versões é a solução mais adoptada.

Muitas destas organizações armazenam os seus documentos usando, por exemplo, um repositório CVS (*Concurrent Versioning System*) [42], que pode ser acedido mais facilmente através de um cliente como o WinCVS (Figura 6).

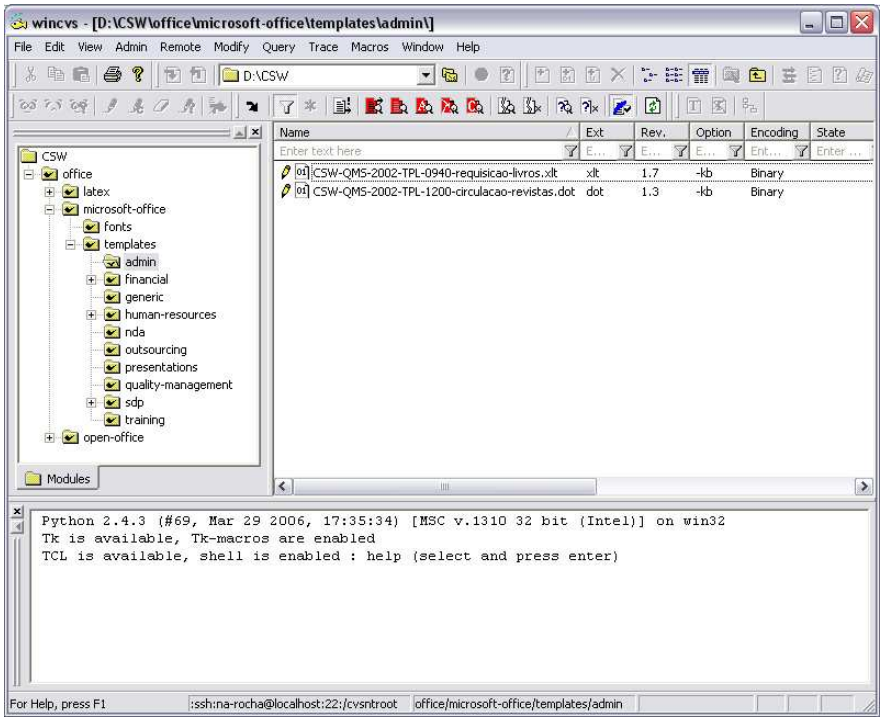

Figura 6 – WinCVS

Este sistema apresenta funcionalidades de controlo de versões muito básicas, ainda mais limitadas devido ao facto de os documentos estarem em formatos binários, correspondentes às *suites* de escritório utilizadas. Assim, é impossível, por exemplo, proceder à visualização das alterações efectuadas por cada colaborador, servindo o

repositório simplesmente para armazenar as diferentes versões. Apesar disso é possível implementar um mecanismo de acesso mutuamente exclusivo a cada documento, fazendo com que apenas um elemento esteja a editar o documento, o que diminui a probabilidade de perda de informação.

Por outro lado, verifica-se a existência de organizações que necessitam de funcionalidades mais avançadas em termos de *Business Inteligence*, gestão de conteúdos e colaboração, por exemplo. Estas podem optar, por exemplo, por uma solução integrada baseada na plataforma SharePoint Server da Microsoft [43]. Esta plataforma apresenta funcionalidades de gestão documental e escrita colaborativa de documentos mais completas que o recurso a apenas um repositório CVS. À semelhança de um CVS, o SharePoint permite a exclusividade de acesso de um utilizador a um documento e o versionamento de documentos, acrescentando-lhe a possibilidade de associação de meta-dados a documentos para facilitar pesquisas futuras (Figura 7).

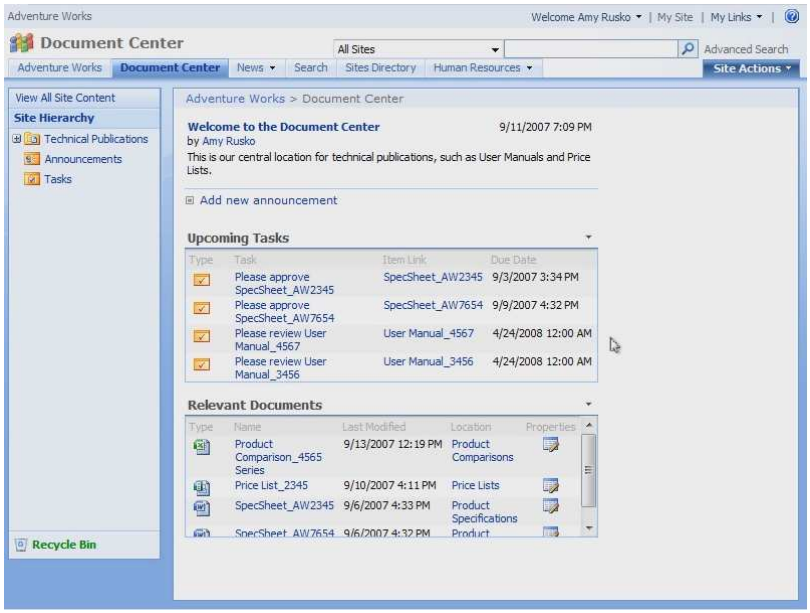

Figura 7 – Gestão de documentos em Sharepoint (extraída de [40])

Tem também um sistema de alertas que podem ser subscritos pelos utilizadores que queiram ser notificados quando alguma alteração é efectuada a um documento e um sistema de definição de *workflow* de documentos. Simultaneamente existe um mecanismo de integração com a *suite* de escritório da Microsoft, o Microsoft Office, para que seja possível reservar os documentos para edição ou ver o seu histórico directamente da aplicação cliente. Estas funcionalidades, em conjunto com o suporte integrado para wikis, que

permite a troca de ideias e informação entre os vários elementos da equipa, fazem com que esta seja uma solução válida para colaboração e gestão documental, em especial quando existe também necessidade de funcionalidades mais avançadas e integradas noutras áreas, como pesquisas e *Business Inteligence*.

As duas soluções baseadas em controlo de versões têm como grande inconveniente o facto de cada elemento necessitar de "reservar para si" (*check-out*) uma cópia do documento quando o pretende alterar. Este facto permite que vários elementos estejam a editar um mesmo documento em simultâneo, mas obriga a que no final tenham de conseguir juntar as alterações efectuadas por cada um numa só cópia (fazer o *merge* das alterações). Se em ficheiros de texto, como os ficheiros de código, por exemplo, este processo é ajudado pela existência de ferramentas específicas para o efeito, no caso de documentos em formato MS Word e OpenOffice já não é tão imediato. Isto deve-se ao facto de estes serem armazenados como documentos binários.

A abordagem baseada em wikis, por seu lado, permite que um documento seja trabalhado simultaneamente por mais do que um elemento. Ao dividir o documento por secções e criando uma *wiki page* para cada secção, pode ter-se um elemento a trabalhar em cada secção. Devido ao mecanismo de controlo de acesso existente nas wikis, apenas um elemento pode estar a editar uma destas partes do documento, solução que é bem menos restritiva do que a apresentada pelas abordagens baseadas em controlo de versões.

4.4 Consistência

Os documentos heterogéneos apresentam aos elementos dos projectos de software um problema que é de extrema importância que seja resolvido de forma eficaz.

Com efeito, garantir a consistência entre a informação presente nos documentos e a informação presente nas suas fontes (as ferramentas de modelação, no âmbito do presente trabalho) não é uma tarefa fácil nem imediata.

Frequentemente as actualizações são feitas apenas na ferramenta específica para o conteúdo a alterar, sendo desleixada a sua propagação para a documentação. O inverso também pode acontecer (actualizações feitas apenas na documentação) mas é, de todo, menos comum. Quer uma quer outra situação podem ter (e certamente virão a ter, a longo ou médio prazo, pelo menos) graves consequências para o projecto dado que vários

elementos podem consultar fontes diferentes e ficar com ideias e informações completamente erradas sobre o que se passa e como está a ser conduzido o projecto.

O projecto XSDoc, relacionado com documentação baseada em wikis, propõe uma abordagem dinâmica e eficaz para a manutenção da consistência da informação, já que desenvolveu um método para integração de código fonte na documentação [44].

No entanto, quando é necessário manter documentos em formatos de *suite* de escritório, este é ainda um problema com aspectos a analisar. Efectivamente, são usadas as funcionalidades que a maior parte das ferramentas de modelação apresentam para criar relatórios da informação nelas contidas, mas a passagem desta informação para a documentação existente e passível de ser editada em ferramentas de processamento de texto é completamente manual. Cabe ao elemento responsável copiar os conteúdos pretendidos para a documentação, apagando (caso seja necessário) o que já lá existam e estejam desactualizados. Como um processo manual que é, está sujeito a inconsistências e esquecimentos, tornando-se relativamente pouco fiável.

Este trabalho procura atenuar estas limitações, tentando aproximar a realidade da documentação existente em formatos binários da realidade da documentação construída com base em wikis, que é comprovadamente mais flexível.

4.5 Custo

A problemática do custo é algo a que é impossível fugir em qualquer área e o desenvolvimento de software não é excepção.

As organizações têm como objectivo último ser lucrativas, necessitando para isso de equilibrar as suas despesas com os seus ganhos, procurando ter de despender o mínimo possível para garantir a qualidade dos seus produtos e serviços.

O investimento em ferramentas de modelação capazes é algo tipicamente efectuado pelas organizações na área do desenvolvimento de software. Sendo o âmbito deste trabalho a integração destas ferramentas com a documentação (e ferramentas de suporte a esta), procura-se uma solução que não imponha mais investimentos significativos, minimizando o custo da plataforma de suporte.

4.6 Principais Problemas em Aberto

A análise feita nas secções anteriores ilustra o problema que se pretende resolver com este trabalho.

Verifica-se que a abordagem de documentação baseada em wikis apresenta boas características para responder às questões enunciadas.

Por outro lado, uma abordagem de documentação mais tradicional, com base em documentos armazenados em formatos de suites de escritório, é de momento mais limitada em todos os aspectos (flexibilidade, evolução, colaboração).

Apesar disto, e porque se procura que este trabalho tenha aplicação em organizações reais com grandes preocupações ao nível de certificações e onde possa existir já uma base de documentação suportada por ferramentas de modelação e de processamento de texto, procurar-se-á algo intermédio que inclua algumas características de wiki mantendo a base das ferramentas referidas. Esta decisão foi tomada também para minimizar o impacto que a transformação total dos processos e ferramentas de apoio à documentação poderia vir a ter no caso da passagem imediata para documentação totalmente *wiki-based*. Assim, o esforço foi no sentido de obter uma solução intermédia, que comece a preparar as organizações que se encontram neste patamar no que à documentação diz respeito para algo com maiores capacidades ao nível de flexibilidade, evolução e colaboração, sempre mantendo os *standards* já existentes (ou mesmo aproveitando as potencialidades das abordagens envolvidas para evoluir nesta problemática).

Capítulo 5.
Abordagem Proposta

A solução encontrada para o problema enunciado no capítulo anterior passou pela definição de um mecanismo genérico de identificação de determinados locais em documentos e integração de diversos componentes, tais como um configurador dinâmico de conteúdos e um gerador de documentos, de maneira a conseguir a flexibilidade e a possibilidade de evolução de documentos já referidas.

Esta abordagem foi validada através da sua implementação em um caso de estudo concreto e permitiu obter resultados bastante interessantes e aplicáveis no domínio em causa.

Neste capítulo é feita uma referência aos requisitos funcionais e não funcionais da plataforma de suporte à abordagem desenvolvida. Seguidamente é apresentada uma visão geral da mesma e clarificados os papéis que os utilizadores podem assumir. Finalmente é descrito o processo de documentação após a adopção desta abordagem.

5.1 Requisitos

Nesta secção são identificados os requisitos da solução desenvolvida a nível funcional e não funcional.

Para servir de base à flexibilidade pretendida é necessária a presença de documentos-tipo, vulgarmente designados por *templates*, quer para os conteúdos de cada documento, quer para os conteúdos de cada secção do documento que possa conter dados de ferramentas de modelação. Estes *templates* têm a função de indicar que informação pode estar presente, cabendo depois ao utilizador escolher aquela que realmente lhe interessa, e são constituídos por marcas indicando qual o conteúdo e onde se localiza no documento.

5.1.1 Requisitos Funcionais

Foi realizada uma análise das questões envolvidas nesta tese, obtendo-se assim uma lista de requisitos que foram considerados importantes para procurar a abordagem mais correcta para o problema.

Estes requisitos são apresentados de seguida em forma tabular, contendo uma breve descrição, a sua prioridade e dificuldade estimadas de implementação.

Tabela 1 – Requisito 'Pré-processamento de *templates*'

Pré-processamento de *templates*			
Prioridade	Alta	**Dificuldade**	Alta
Os *templates*, definidos numa organização para servir de base aos documentos finais, devem ser analisados pela plataforma para que sejam compreendidos.			

Tabela 2 – Requisito 'Apresentação do conteúdo dos *templates*'

Apresentação do conteúdo dos *templates*			
Prioridade	Alta	**Dificuldade**	Média
O conteúdo dos *templates* deve ser apresentado aos utilizadores de uma forma lógica e adequada. Devem ser claramente visíveis as diferentes secções que podem fazer parte de um documento e os diferentes conteúdos que cada secção pode conter.			

Tabela 3 – Requisito 'Configuração dos conteúdos dos documentos'

Configuração dos conteúdos dos documentos			
Prioridade	Alta	**Dificuldade**	Média
A plataforma deve disponibilizar um componente para que os utilizadores possam escolher os conteúdos que pretendem visualizar no documento final. Este configurador deve ser alimentado pelos dados resultantes do pré-processamento dos *templates*.			

Tabela 4 – Requisito 'Geração de documentos heterogéneos'

Geração de documentos heterogéneos			
Prioridade	Alta	**Dificuldade**	Alta
A plataforma deve disponibilizar um componente para permitir a geração de documentos heterogéneos, compostos por texto e dados provenientes de ferramentas de modelação. Este gerador deve ser alimentado pelas opções escolhidas no configurador, pelos *templates* inseridos no sistema e pelos dados das ferramentas de modelação.			

Tabela 5 – Requisito 'Actualização de documentos heterogéneos'

Actualização de documentos heterogéneos			
Prioridade	Alta	**Dificuldade**	Alta
Os documentos heterogéneos gerados pela plataforma devem poder ser actualizados ao nível dos dados, provenientes de ferramentas de modelação, que os contêm. O utilizador deve poder rever as opções que escolheu na altura da geração original (ou da actualização mais recente), cabendo a este a opção de manter ou alterar as escolhas efectuadas.			

Em termos de prioridade de implementação de requisitos, as funcionalidades consideradas essenciais para o correcto funcionamento da plataforma foram classificadas como tendo prioridade "Alta", as funcionalidades importantes mas não críticas foram classificadas como sendo de prioridade "Média" e as funcionalidades exclusivamente acessórias como tendo prioridade "Baixa".

Em termos de dificuldade estimada de implementação, a classificação adoptada foi igualmente distribuída pelos valores "Alta", "Média" e "Baixa". Os requisitos com grande complexidade e que envolvam interacção entre vários componentes foram classificados com dificuldade "Alta", os com alguma complexidade e sem grande interacção entre componentes foram classificados como tendo dificuldade "Média" e os requisitos com implementação mais básica como sendo de dificuldade "Baixa".

5.1.2 Requisitos Não Funcionais

Foram também identificados requisitos que, não sendo funcionalidades a implementar, têm interferência na forma como estas são concretizadas, já que são referentes a certos atributos que o sistema deve apresentar.

Usabilidade

A plataforma de suporte a uma solução de construção de documentação deve ser sempre pensada tendo em conta o utilizador final e as suas expectativas. Deve-se procurar algo intuitivo, claro e simples de usar para que não se contribua para um possível "desleixo" em relação ao processo de documentação de software.

Desempenho

Todas as operações envolvidas na abordagem proposta devem procurar ser realizadas no menor tempo possível.

Disponibilidade

Sendo uma das características pretendidas de uma abordagem deste tipo a colaboração entre os diversos elementos de uma equipa, a plataforma de suporte deve estar sempre disponível através da Internet.

Segurança

Apenas utilizadores autorizados devem poder aceder à plataforma, daí a necessidade da presença de um mecanismo de controlo de acessos. Este facto é justificado pelo facto de se estar a lidar com documentação que, na maior parte das vezes, contém dados confidenciais dos clientes.

Consistência

A consistência de informação é um aspecto crucial nesta abordagem. Procura-se garantir consistência entre os dados existentes nas ferramentas de modelação e os documentos produzidos, de forma a minimizar o esforço na manutenção da documentação.

5.2 Visão Geral

De forma a conseguir satisfazer todos os requisitos até aqui apresentados, foi idealizada a abordagem ilustrada pela Figura 8.

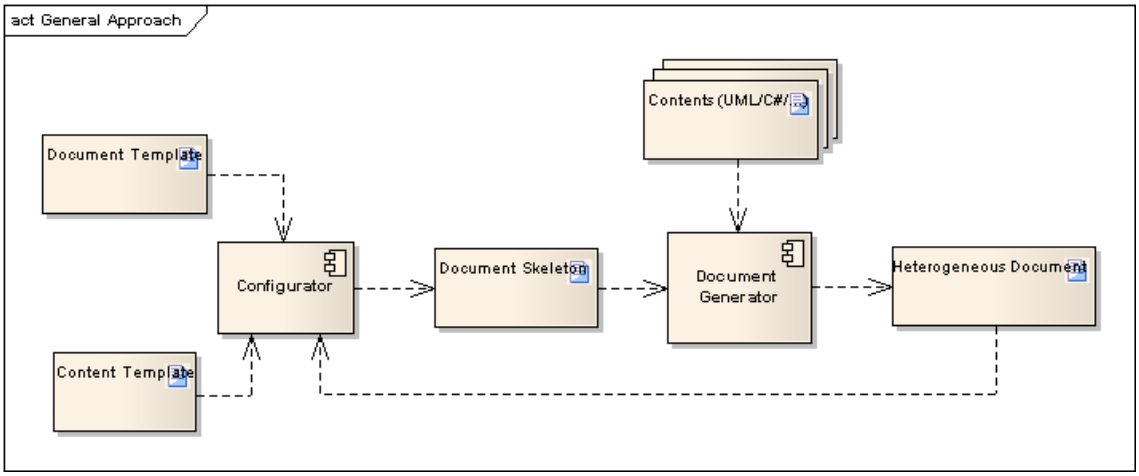

Figura 8 – Visão geral da abordagem idealizada

Nesta solução destaca-se ainda a presença de dois componentes essenciais: o configurador e o gerador de documentos heterogéneos.

O configurador de documentos heterogéneos tem como função permitir ao utilizador escolher os conteúdos que pretende que estejam presentes no documento final.

Actua tanto ao nível das secções visíveis no documento como ao nível dos conteúdos que serão visualizados em cada uma das secções que foram escolhidas para comporem o documento e que possam ter algum tipo de conteúdo vindo de ferramentas de modelação de software. Tem por base os *templates* definidos anteriormente.

O gerador de documentos heterogéneos tem como função juntar a informação proveniente do configurador e das ferramentas de modelação com os *templates* existentes, processando-os e construindo um documento com as características esperadas. Este documento gerado pode ser re-inserido neste sistema devido ao facto de conter, também ele, marcas indicando os locais onde existem dados provenientes de ferramentas de modelação. Assim, o sistema sabe onde actualizar os dados pretendidos. O facto de, na actualização de documentos, o utilizador voltar a passar pelo configurador adiciona ainda mais flexibilidade ao sistema. A presença de um mecanismo para guardar opções de geração originais do documento permite ao utilizador um maior controlo sobre a acção que pretende realizar na actualização: se pretende apenas actualizar o documento com a mesma informação que escolheu visualizar durante o processo de geração ou se pretende actualizar o documento com mais ou menos informação do que anteriormente.

5.3 Definição de Papéis dos Utilizadores

Esta abordagem pressupõe a existência de dois perfis de utilizadores do sistema:

- Administrador – tem a função de criar e manter os *templates* que servem de base aos conteúdos pretendidos, com as marcas necessárias;
- Utilizador comum – pode realizar as operações de geração e actualização de documentos.

De realçar que os utilizadores podem acumular o perfil 'Administrador' com o perfil 'Utilizador comum'.

5.4 Processo de Documentação

Seguindo esta abordagem proposta, o processo de documentação de software pode ser apresentado, de forma genérica, nos passos ilustrados na Figura 9.

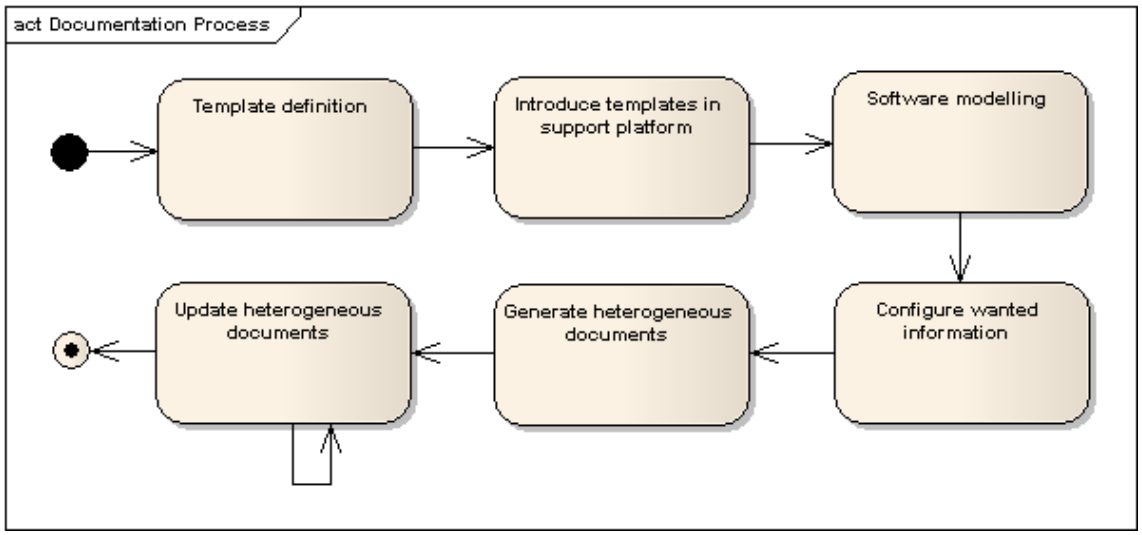

Figura 9 – Actividades envolvidas no processo de documentação

Definição dos *templates*

De maneira a garantir uma uniformidade da documentação produzida é fundamental que se definam as estruturas que servirão de base à construção dos documentos. Este trabalho pode ser efectuado por qualquer um dos elementos da equipa de projecto ou por um utilizador mais ligado à área da Qualidade de Software.

Introdução dos *templates* na plataforma de suporte

Essencial para que o sistema se torne conhecedor das características apresentadas pelos *templates* definidos.

Modelação de software

Modelação típica efectuada no âmbito de um processo de desenvolvimento de software. Pode estar relacionada com diagramas UML, definição de requisitos ou testes, entre outros. Realizada em ferramentas com suporte específico para as funcionalidades pretendidas.

Configuração da informação visível no documento final

Efectuada a partir de uma interface onde estão presentes as secções possíveis para o documento e o conteúdo possível para cada uma das secções escolhidas.

Geração de documentos heterogéneos

Despoletada pela necessidade de existência de um documento físico, esta acção pode estar envolvida por grande complexidade, dependendo das ferramentas envolvidas. Consiste em criar, num formato escolhido pelo utilizador, um documento contendo as informações escolhidas no configurador, podendo estas ser de várias naturezas tais como descrições textuais, diagramas ou código fonte.

Actualização de documentos heterogéneos

Re-alimentação da plataforma de suporte com documentos anteriormente gerados, permitindo ao utilizador redefinir os conteúdos que pretende ver/actualizar. A presença de marcas identificando os locais onde foi inserida informação proveniente das ferramentas de modelação permite uma actualização eficaz dos dados pretendidos, mantendo inalterada outra informação que tenha sido adicionada por outros meios.

Capítulo 6.
Aplicação da Abordagem Desenvolvida

A solução apresentada em linhas gerais no capítulo anterior teve de ser aplicada a um domínio concreto por forma a provar a sua aplicabilidade e utilidade.

Assim, foi desenvolvida uma aplicação que permite aplicar os conceitos e processos definidos à situação em que se tem como ferramenta de modelação o Enterprise Architect e como ferramentas de processamento de texto o Microsoft Word e o OpenOffice Writer.

Este capítulo procura descrever as especificidades do desenvolvimento efectuado, bem como a sua integração num ambiente real com necessidades efectivas ao nível da produção e manutenção de documentos heterogéneos.

6.1 Integração de Modelos Enterprise Architect com Documentos MS Word e OpenOffice

Os conceitos desenvolvidos no capítulo anterior foram aplicados a um domínio concreto, no âmbito de uma organização real.

A Critical Software, SA acolheu a abordagem da melhor forma. Esta organização, reconhecida pelo rigor que empresta aos seus processos e soluções demonstrado pelas certificações de qualidade obtidas, necessita de construir e manter documentação relativa a projectos que podem ser bastante complexos. Projectos complexos dão origem a documentação complexa e bastante heterogénea, no caso de se pretender efectuar um trabalho de elevado nível, como é o caso.

A extrema necessidade de documentar as soluções produzidas com um nível de detalhe elevado leva a que seja usada uma ferramenta de modelação para produzir os diagramas pretendidos e para armazenar informação de requisitos e testes, por exemplo. A ferramenta em questão é o Enterprise Architect, que já foi analisado na secção 3.2.3.

A documentação em si é produzida recorrendo ao Microsoft Word e ao OpenOffice Writer, sendo utilizados como base *templates* construídos a partir dos *standards* de referência na área (como, por exemplo, o IEEE [4]).

Estes factores condicionaram a escolha das ferramentas de modelação e de processamento de texto envolvidas no desenvolvimento do caso de estudo apresentado, de maneira a que este procurasse resolver um problema concreto, apresentado pela organização referida.

Apesar destas condicionantes, verificou-se estarem reunidas as condições para que a abordagem proposta pudesse ser validada mediante a construção de uma solução que permita integrar, de forma mais fácil, simples e rápida os conteúdos presentes no Enterprise Architect com a documentação produzida.

6.1.1 Documentation Framework

O trabalho produzido no âmbito desta tese assenta numa aplicação previamente desenvolvida pelo autor [6]. Surge assim como um complemento de funcionalidades a esta aplicação *web-based*.

Objectivos

O projecto Documentation Framework surgiu da necessidade de automatizar algumas tarefas de gestão e configuração dos vários *templates* existentes na organização. Estes são documentos-tipo que fornecem uma base comum para a realização das mais variadas acções que são possíveis de realizar. Documentos que apresentam um esqueleto para a apresentação de novas propostas de negócio, apresentações públicas, relatórios de apresentação de despesas, de requerimento de autorizações ou meios necessários à actividade diária dos colaboradores, documentação técnica de projectos (requisitos, arquitectura, detalhes de implementação, testes, manuais), entre outros.

Estes *templates* apresentam, como é natural, uma série de informação que é comum a muitos (ou por vezes todos) eles. Informação relativa aos dados institucionais da organização como o logótipo, o *website*, as moradas dos escritórios ou o *slogan* é apresentada em todos os documentos. Além da informação, também a própria estrutura é semelhante. Por exemplo, os *templates* que se referem a relatórios apresentam todos uma primeira página que funciona como capa (com a identificação da instituição e do documento em si), uma página seguinte onde é colocado o histórico de revisões e aprovações e uma tabela que apresenta os conteúdos do documento.

Por um lado, isto torna a tarefa de quem necessita de construir um documento bastante mais simples, uma vez que já tem uma estrutura base montada (sendo preciso no entanto moldá-la às suas necessidades) e garante que tudo o que é produzido na organização apresenta um aspecto semelhante e claro.

Por outro lado, a existência de um número tão grande de *templates* tem implicações bem visíveis no processo de actualização e manutenção dos mesmos. Se houver uma mudança de instalações de um escritório ou mesmo a abertura de um novo é necessário

percorrer todos os *templates* existentes e realizar actualizações de informação em cada um deles.

Todos estes *templates* estão armazenados num sistema de controlo de versões (no caso o CVS), o que significa que, para além de percorrer todos os *templates*, também é necessário obter a versão actual e enviar a nova para o sistema de controlo de versões.

Considera-se que por serem operações exclusivamente manuais, estão sujeitas a erros e tornam-se muito lentas. Seria muito mais produtivo para a organização libertar os seus colaboradores destas tarefas rotineiras.

Sendo os *templates* existentes o mais genéricos possível para a função que procuram satisfazer, verifica-se a necessidade de os utilizadores finais os configurarem de acordo com aquilo que efectivamente irão utilizar, numa dada altura. Isto é mais visível na documentação técnica de projectos. Há projectos das mais variadas áreas e com os mais diversos âmbitos. No entanto, todos devem respeitar as normas definidas, usando para isso as estruturas existentes. Apesar disso, há sempre uma ou mais secções que não fazem sentido para todos os projectos e que necessitam de ser removidas ou propriedades que estão definidas com conteúdos exemplificativos (nome do projecto e do cliente a que se refere o documento, por exemplo).

Assim, os objectivos do trabalho desenvolvido seriam:

- criação de um mecanismo que tornasse a tarefa de manutenção de *templates* mais simples e rápida;
- integração deste mecanismo com o repositório onde actualmente estão armazenados todos os *templates* da organização, de uma forma suave e invisível para os utilizadores finais;
- criação de uma ferramenta permanentemente disponível para permitir a realização das tarefas pretendidas, de preferência baseada em tecnologias web;
- elaboração de uma solução que permita aos colaboradores da empresa adaptar mais facilmente os *templates* existentes às suas necessidades específicas;
- construção de um repositório para armazenar informação relativa à documentação e aos dados que a compõem;

- suporte de duas das mais utilizadas ferramentas de processamento de texto: o Microsoft Word e o OpenOffice Writer.

Abordagem

A abordagem seguida foi a de uma aplicação web com as funcionalidades pretendidas.

Para tentar resolver o problema da quantidade de trabalho necessária na actualização de *templates*, optou-se por dividir estes em dois tipos:

- *Templates* base – uma colecção de *templates* que contêm a informação que é comum a uma série de *templates* (*template* base para relatórios técnicos, por exemplo). Estes incluem, tipicamente, a capa, a página com o histórico de revisões e alterações e a tabela de conteúdos, em conjunto com alguma informação que potencialmente se repita.
- *Templates* específicos – correspondem aos conteúdos específicos de cada um dos tipos de documento (*template* específico para especificação de requisitos ou de arquitectura, por exemplo). Contêm secções que apenas aparecem nesse *template* em particular.

Apesar de não diminuir o número efectivo de *templates* existentes, esta abordagem torna a tarefa de manutenção e actualização de informação comum mais fácil uma vez que passam a existir muito menos *templates* para actualizar, já que os conteúdos comuns existirão apenas nos *templates* base, um por cada tipo de documento, contrastando com o cenário anterior em que cada alteração necessitava de ser propagada para dezenas de *templates*. A juntar a isto, passa a haver um local comum e centralizado para editar as propriedades que estão repetidas em vários *templates* base, onde o utilizador apenas terá de preencher os locais correspondentes com os valores pretendidos e tudo o resto (operações sobre o CVS e manipulação de ficheiros) é da responsabilidade do sistema.

Para ser possível esta edição de propriedades existentes em ficheiros, pensou-se em marcar todos os locais passíveis de edição. Assim, todos os lugares que correspondam a alguma propriedade desejável de editar devem ser marcadas por meio do mecanismo de *bookmarks*, existente quer em MS Word, quer em OpenOffice Writer.

À semelhança dos *templates*, também as propriedades foram conceptualmente divididas em duas classes:

- Propriedades gerais – propriedades que aparecem em mais do que um *template* (tipicamente em *templates* base) e que têm o mesmo valor em todos eles.
- Propriedades específicas – propriedades que podem aparecer em vários *templates*, mas que o seu valor é distinto para cada um deles.

Além de tudo isto, as funcionalidades de manipulação de documentos desenvolvidas permitem a geração de documentos adaptados às necessidades de cada utilizador. Uma vez que os *templates* existentes têm de ser o mais genéricos possível (caso contrário teria de existir um para cada tipo de projecto e isto seria claramente uma solução muito difícil de manter), um utilizador quando precisa de começar um documento selecciona aquilo que provavelmente irá necessitar e preenche alguma informação pertinente acerca do projecto e do documento em si. Esta funcionalidade procura eliminar algumas das dificuldades que podem surgir nesta tarefa: lidar manualmente com secções, quebras de secção, quebras de página, formatações de títulos e actualização de propriedades na maioria das soluções de software de escritório existentes no mercado (nomeadamente MS Office e OpenOffice).

Para ter acesso mais fácil e mais rápido aos dados necessários para todos estes processos foi criada uma base de dados com alguma informação relevante acerca dos *templates* e dos seus dados constituintes.

Para garantir a consistência de informação entre os ficheiros binários e a informação constante da base de dados, foi desenvolvido um mecanismo de integração com o repositório de CVS que permite propagar as alterações efectuadas em ambos os sentidos.

Podem ser distinguidos claramente dois tipos de actores deste sistema (Figura 10). São eles o "*Quality Administrator*" e o "*User*".

O actor "*Quality Administrator*" representa os membros da equipa de Qualidade, cujo papel é essencialmente administração a manutenção dos *templates* existentes, sendo também capazes de gerar novos documentos.

O actor "*User*" representa um membro de uma equipa de projecto, que é capaz de gerar documentos adaptados às necessidades específicas do seu projecto.

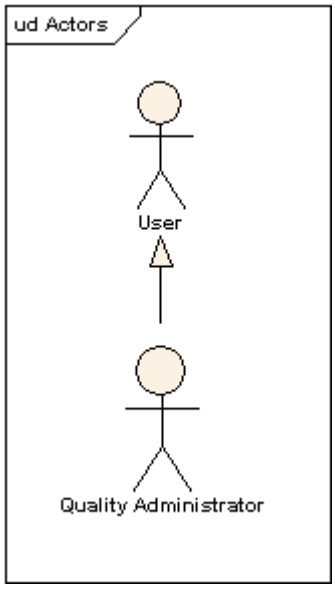

Figura 10 – Lista de actores

Arquitectura base

A Figura 11 demonstra uma visão global da aplicação desenvolvida e das partes com as quais interage.

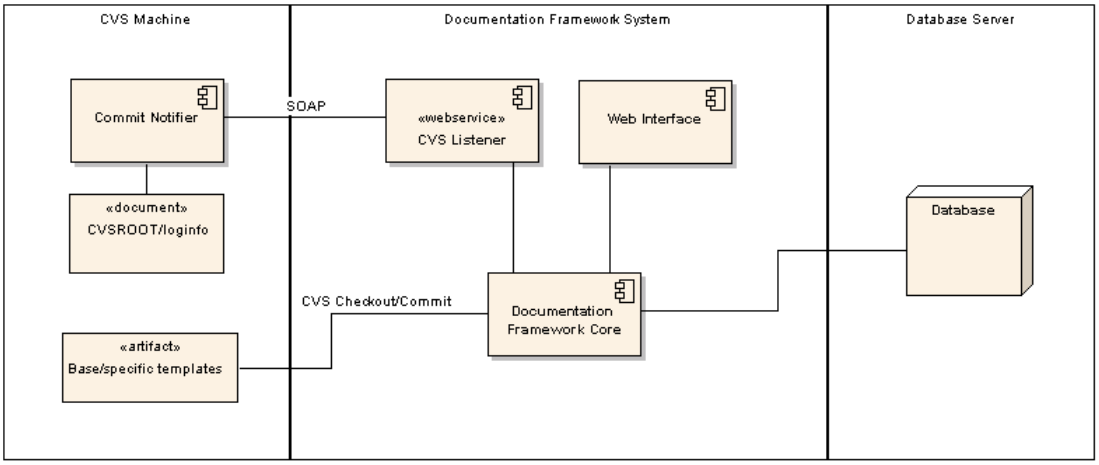

Figura 11 – Visão global da arquitectura do sistema

Este baseia-se numa aplicação web, que tem como fontes de informação os ficheiros binários dos *templates* da organização e uma base de dados que foi construída para esta aplicação em particular. Esta base de dados contém informação relacionada com os dados existentes nos *templates*, de maneira a permitir um comportamento mais estável e rápido do sistema, uma vez que se tornaria bastante pesado e ineficaz estar a manipular constantemente os ficheiros binários.

Assim sendo, a informação existente na base de dados deve estar sempre sincronizada com a informação presente nos *templates*, para garantir o bom funcionamento do sistema. Para isto está presente um mecanismo para detectar alterações no repositório CVS onde estão armazenados os ficheiros, para que algumas mudanças efectuadas pelos utilizadores sem ser através da aplicação web (adição de novas propriedades, por exemplo) sejam propagadas para a base de dados de suporte.

Este mecanismo consiste numa pequena "aplicação" cliente – componente *Commit Notifier* – para alertar o Documentation Framework que uma alteração foi efectuada no CVS, para que os ficheiros em causa na alteração sejam analisados e a base de dados actualizada com o seu novo conteúdo. O Documentation Framework usa um *web service* – componente *CVS Listener* – que está sempre à escuta de pedidos desta aplicação-cliente para despoletar o processo de análise. Apresenta ainda uma interface web para interacção com os utilizadores (administradores e de "uso comum"). O componente *Documentation Framework Core* representa toda a lógica do sistema e acesso a dados, ao nível de manipulação de ficheiros e acesso à base de dados utilizada.

A arquitectura lógica é uma arquitectura simples de três camadas, separando a interface, a lógica de negócio e o acesso e armazenamento de dados em camadas distintas, de maneira a obter independência no desenvolvimento e manutenção destes componentes críticos.

Tecnologias envolvidas

Devido ao facto de manipular documentos no formato MS Word, o projecto baseia-se em tecnologias Microsoft.

Assim, a interface foi desenvolvida com recurso à tecnologia ASP.NET 2.0 com algumas funcionalidades a utilizar ASP.NET Ajax, sendo a lógica de negócio em C#. O

sistema de gestão de base de dados escolhido foi o PostgreSQL (para minimizar custos envolvidos).

A manipulação de documentos MS Word utiliza a biblioteca PIA (*Primary Interop Assemblies*) for Office 2003 e a manipulação de documentos OpenOffice Writer é baseada em processamento simples de XML.

O componente que permite a integração com o repositório CVS foi desenvolvido recorrendo à linguagem de *scripting* Perl.

Funcionalidades disponibilizadas

As funcionalidades disponibilizadas por esta aplicação cobrem em pleno os objectivos traçados.

Assim, está presente a possibilidade de edição de propriedades gerais através de uma interface web, com todo o trabalho de processamento e actualização de ficheiros, base de dados e repositório CVS a ser da responsabilidade do sistema.

Existe também a funcionalidade de configuração de *templates* de acordo com as necessidades dos utilizadores, cabendo a estes escolher as secções a incluir no documento e definir certas propriedades do mesmo.

Além disto, esta aplicação disponibiliza aos utilizadores a possibilidade de modificar algumas propriedades das secções de documentos presentes na base de dados (tais como a sua obrigatoriedade num determinado documento), modificar o *template* base associado a um determinado *template* específico e as tradicionais capacidades de controlo de acessos e listagem de últimas alterações efectuadas pelo sistema.

É possível consultar uma descrição mais detalhada destas funcionalidades no Anexo A, na secção dedicada ao manual de utilização da Documentation Framework.

6.1.2 Abordagem Seguida

Com as funcionalidades de gestão e geração de *templates* já presentes na plataforma, considerou-se que o passo lógico seguinte para a evolução desta seria a introdução de

conteúdos nos *templates*, procurando aumentar a sua utilidade no processo de documentação de software. Assim, as funcionalidades implementadas no âmbito do presente trabalho permitiram que a influência da ferramenta para o processo de documentação não parasse no momento em que era gerado um novo *template* apenas com conteúdos exemplificativos mas continuasse durante o ciclo de vida do documento correspondente ao *template* obtido previamente. Estas novas funcionalidades estão relacionadas com a inclusão e actualização de conteúdos nos *templates* produzidos.

A integração deste trabalho com o projecto Documentation Framework foi, então, conseguida através da adição das funcionalidades pretendidas como novas páginas Web ou mesmo apenas alterando as páginas já desenvolvidas para que possam ter o desempenho pretendido.

Além da interface, também a base de dados necessitou de ser estendida de maneira a que conseguisse armazenar mais informação, relacionada com os dados vindos do Enterprise Architect possíveis de integrar nos documentos já produzidos pela aplicação.

A lógica de negócio que necessitou de ser acrescentada foi-o recorrendo a novas classes ou novos métodos em classes já existentes, daí que o impacto na arquitectura do sistema previamente existente tenha sido quase nulo.

Para que a abordagem defendida neste trabalho consiga ser aplicada ao caso em que o Enterprise Architect é usado como ferramenta de modelação é necessário que os dados sejam armazenados numa base de dados e não num simples ficheiro. Esta pode ser qualquer uma das referidas no capítulo 3.2.3. Esta potencialidade permite que se possa aceder aos dados presentes nos modelos desenvolvidos mediante o mecanismo de interrogações a uma base de dados, usando a linguagem SQL.

Aplicando os princípios gerais explicados no capítulo anterior ao caso específico formado pela combinação das ferramentas apresentadas – Enterprise Architect, MS Word/OpenOffice Writer e Documentation Framework – é possível documentar esta solução com o esquema apresentado na Figura 12.

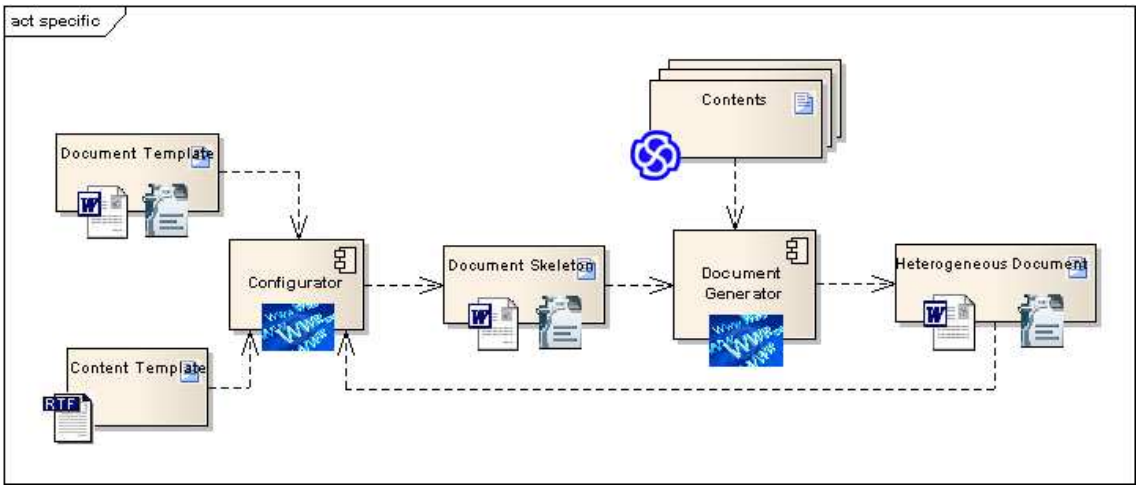

Figura 12 – Visão geral da abordagem idealizada aplicada ao caso de estudo

Como referido, esta solução baseia-se na existência de *templates* quer para os documentos a construir, quer para os conteúdos das ferramentas de modelação. Foi decidido que a construção destes últimos continuaria a ser feita recorrendo ao módulo de geração de documentação presente no Enterprise Architect.

Com efeito, o editor de *templates* presente nesta ferramenta é bastante poderoso e flexível no que diz respeito à indicação de conteúdos a incluir e à formatação dos mesmos. Permite indicar com exactidão que conteúdo inserir onde e, recorrendo a uma interface e funcionalidades de formatação bastante semelhantes às do MS Word, configurar estilos e formatações pontuais para aplicar ao *template* a construir. A juntar a isto o facto de os utilizadores já se encontrarem habituados a esta ferramenta e de os *templates* serem produzidos nos formatos RTF ou HTML (possíveis de serem processados por uma aplicação), fez com que se considerasse que a construção de raiz de um configurador de *templates* seria uma opção desnecessária, para este caso concreto.

Assim, o primeiro passo a executar passa pela construção dos *templates* de documentos (recorrendo ao MS Word/OpenOffice Writer) e dos *templates* relacionados com os conteúdos do Enterprise Architect.

Estes necessitam de ser introduzidos no repositório CVS da organização para que fiquem disponíveis para todos os utilizadores potenciais da ferramenta e vão servir de base para a informação a ser visualizada no configurador de documentos.

O componente de configuração de documentos está implementado num ambiente *web-based*, acessível por todos os utilizadores com permissões de acesso ao sistema, permitindo que seja escolhido o tipo de documento a gerar e as secções a incluir neste documento (funcionalidades já existentes na Documentation Framework). Quando alguma das secções escolhidas for passível de conter dados do Enterprise Architect, o utilizador pode optar quais (se alguns) os que pretende visualizar (Figura 20). Este procedimento é semelhante ao já existente para as secções de documentos.

Toda esta informação relativa às preferências do utilizador em relação aos conteúdos a visualizar no documento é compilada e processada pela aplicação, permitindo a obtenção de um documento apenas com as secções pretendidas. Os dados de Enterprise Architect são, então, adicionados a este documento configurado (caso seja esse o objectivo do utilizador), tarefa só possível devido ao mecanismo de identificação de locais em documentos que, neste caso, é conseguido através de *bookmarks*, colocadas junto das secções definidas como podendo ter conteúdos da ferramenta de modelação em causa. Estas identificam o local onde deve ser inserido o conteúdo e qual o conteúdo a inserir, mediante a referenciação do *template* – construído no editor do Enterprise Architect – que lhe servirá de base.

Cabe, então, ao gerador de documentos – mais uma vez, *web-based* e já presente na aplicação, tendo sido melhorado no âmbito deste trabalho – as tarefas de acesso aos dados da ferramenta de modelação, construção da apresentação destes mediante o *template* previamente definido e sua integração no documento que irá ser apresentado ao utilizador final.

O documento resultante encontra-se completamente configurado de acordo com as opções indicadas na interface web e contém em si a indicação das secções que apresentam conteúdos vindos do Enterprise Architect. Esta indicação é transparente para os utilizadores do documento (mais uma vez, recorrendo ao mecanismo de *bookmarks*) mas de extrema importância para a execução da funcionalidade de actualização de dados em documentos.

Assim, o documento obtido pode, a qualquer altura, voltar a ser inserido na ferramenta se se pretender que seja actualizado ao nível das secções com informação da ferramenta de modelação. Através da indicação referida no parágrafo anterior, o sistema

consegue perceber onde necessita de intervir, deixando o resto do documento intacto. De realçar que esta actualização é mais do que uma mera actualização dos dados já existentes, já que volta a ser apresentado o configurador (neste caso, apenas dos dados a incluir nas secções com informação do Enterprise Architect), podendo o utilizador efectuar alterações aos dados que pretende ter no documento ou mantendo as suas opções de geração iniciais.

6.1.3 Casos de Uso

Associados a esta abordagem, podem ser identificados alguns casos de uso, ilustrados na Figura 13.

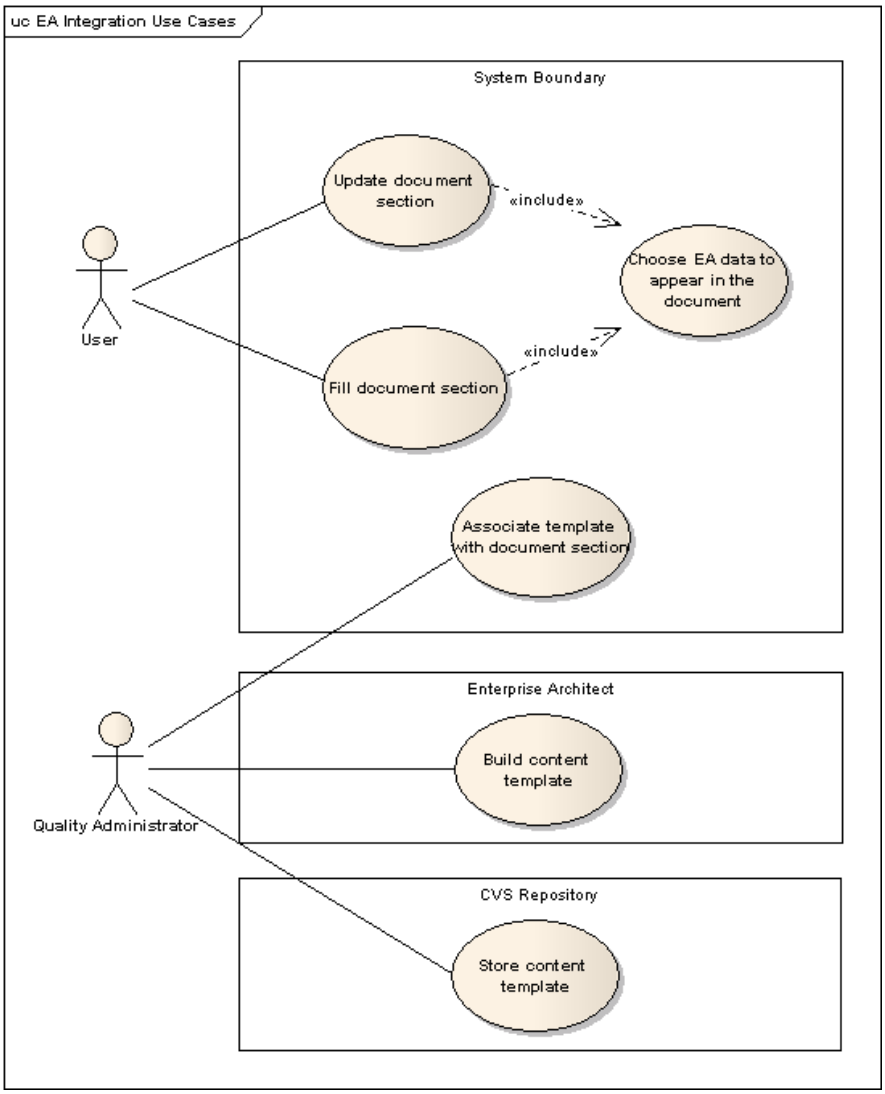

Figura 13 – Casos de uso da integração de conteúdos

A interacção de um utilizador com o sistema para a realização das funcionalidades de integração de informação é bastante simples.

Este tem a possibilidade de preencher uma secção do documento que pretende produzir com a informação vinda do Enterprise Architect ou então pode optar por actualizar secções de um documento previamente obtido através do processo anterior.

Em ambos os casos, o passo seguinte é a escolha dos conteúdos que pretende incluir, usando para isso as potencialidades da interface web desenvolvida.

De realçar que o preenchimento de uma secção com conteúdos do Enterprise Architect está completamente integrado na funcionalidade de construção de documentos configurados pelo utilizador, já existente na aplicação Documentation Framework. Por outro lado, e como descrito na secção anterior, a construção e configuração de *templates* para estes conteúdos é feito integralmente na ferramenta de modelação utilizada. No entanto, cabe ao *Quality Administrator* quer esta construção, quer a colocação dos *templates* obtidos no repositório CVS da organização e a associação destes *templates* a secções específicas de certos documentos, como ilustrado no diagrama da Figura 13.

6.1.4 Desenvolvimento da Solução

Na secção 6.1.2 foram já apresentadas de uma forma algo geral alguns dos mecanismos utilizados para conseguir implementar a abordagem desenvolvida ao domínio em questão.

Nesta secção são descritos estes e outros mecanismos e procedimentos relevantes para a obtenção de uma solução com as características pretendidas.

Armazenamento e processamento de *templates*

Todo este processo começa com a criação dos *templates* recorrendo ao Enterprise Architect. Depois da criação, estes devem ser armazenados num local central à organização para que sejam mais facilmente acedidos e difundidos pelos seus elementos constituintes. Até aqui isto não era um procedimento comum. Procedia-se à construção dos *templates* (na

maior parte das vezes até por elementos das próprias equipas de projecto), sendo estes utilizados e armazenados de forma independente por cada equipa. Esta abordagem fomentava a possibilidade de inconsistências na apresentação e conteúdos dos documentos produzidos na organização, já que muitas vezes não existia sequer intervenção do departamento de Qualidade neste processo.

A solução proposta pressupõe que sejam os elementos do departamento de Qualidade a construir e administrar os *templates*, uniformizando assim o processo e os conteúdos.

Os *templates* devem ser também colocados no repositório CVS para uma maior organização e manutenção de alterações efectuadas aos mesmos. Assim, foi estabelecido que deveria ser criada uma directoria para este tipo de documentos-tipo.

Garantida a existência de um local para armazenar os *templates* produzidos, tornou-se necessário desenvolver um mecanismo que permitisse a integração destes no sistema de uma forma simples e o mais transparente possível para os utilizadores. Este mecanismo foi adaptado do já existente anteriormente, consistindo num pequeno *script* Perl, corrido no servidor CVS, que tem como função indicar ao sistema quais os ficheiros adicionados, modificados ou removidos do repositório.

Assim é possível, ao sistema desenvolvido, analisar os ficheiros em questão de maneira a conseguir perceber os seus conteúdos e a forma como estão analisados.

Para que a ferramenta desenvolvida consiga dar a possibilidade de escolha de conteúdos aos utilizadores finais, é necessário que saiba previamente quais os conteúdos possíveis de incluir. Isto foi conseguido através do processamento dos *templates* produzidos no editor presente no Enterprise Architect.

Estes são armazenados no repositório CVS no formato RTF. Apesar disso, foi considerado mais fácil a utilização do formato WordML [45] para a análise do conteúdo existente e mesmo para a geração das secções dos documentos com os dados da ferramenta de modelação, por ser um formato aberto e baseado em XML (encontrando-se bem estruturado), ao contrário do RTF que, apesar de processável, se torna mais incómodo no processamento devido à fraca estruturação e às diferenças existentes no código

produzido pelas ferramentas que utilizam este formato, sendo que as marcas de formatação podem vir por qualquer ordem, desde que se tornem em algo válido.

O processo utilizado para proceder à análise dos *templates* produzidos encontra-se, então, ilustrado na Figura 14.

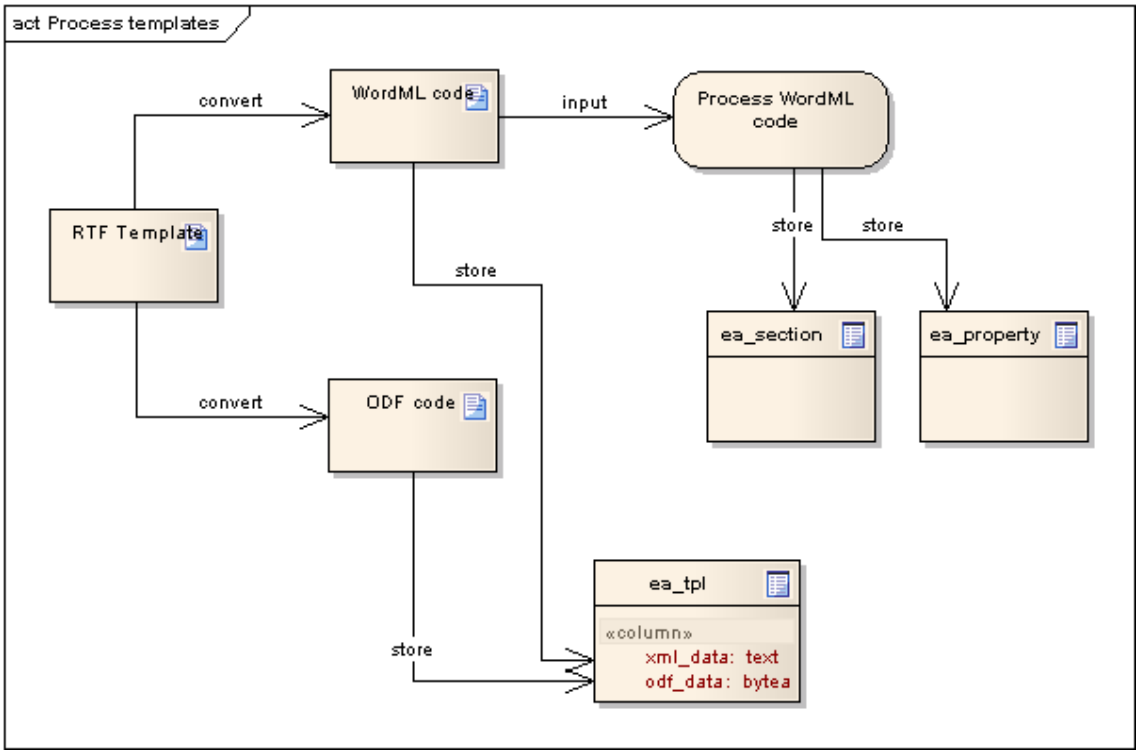

Figura 14 – Processamento de *templates*

A ferramenta desenvolvida começa, após efectuar o *checkout* do *template* do repositório CVS, por convertê-lo do formato RTF em que se encontra para o formato WordML. O código WordML obtido é armazenado na base de dados construída para servir de suporte à aplicação, para ser utilizado na geração dos conteúdos pretendidos. De maneira a conseguir criar de forma o mais correcta possível documentos também no formato ODF (OpenDocument Format), é também armazenado o código relativo a este formato, após a sua conversão a partir do RTF. A conversão RTF → ODF é mais fiável do que a conversão WordML → ODF, daí ter sido tomada esta escolha. O uso dado ao código ODF armazenado é descrito mais à frente neste documento.

Para facilitar o processo de construção da interface relativa ao configurador de documentos, optou-se por, além de guardar o WordML na base de dados, processá-lo para conseguir estruturar a informação nele presente de uma forma que seja mais simples e

rápida de pesquisar. Assim, o *template* foi analisado à procura de secções – referentes a um tipo de conteúdo, por exemplo, um *package*, um elemento ou um diagrama – e propriedades destas secções (por exemplo, o nome e o tipo do *package*).

Este processamento foi possível devido à elevada estruturação dos *templates* produzidos no editor do Enterprise Architect, que identifica as secções por meio de *bookmarks* (com indicação do seu início e do seu fim) e as propriedades por meio de campos automáticos. Isto é representado no formato WordML de uma forma *standard*, ilustrada na Figura 15 no caso das secções e na Figura 16 no caso das propriedades.

```
<aml:annotation aml:id="3" w:type="Word.Bookmark.Start" w:name="Pkg_Diagram_Begin"/>
<aml:annotation aml:id="3" w:type="Word.Bookmark.End"/>
- <w:p wsp:rsidR="00000000" wsp:rsidRDefault="00766B64">
  + <w:pPr></w:pPr>
  + <w:r></w:r>
    <aml:annotation aml:id="4" w:type="Word.Bookmark.Start" w:name="Pkg_Diagram_Begin_Inner"/>
    <aml:annotation aml:id="4" w:type="Word.Bookmark.End"/>
  - <w:r>
    - <w:rPr>
        <w:rStyle w:val="SSBookmark"/>
        <wx:font wx:val="Lucida Sans"/>
        <w:spacing w:val="-5"/>
        <w:highlight w:val="yellow"/>
      </w:rPr>
      <w:t></w:t>
    </w:r>
  </w:p>
+ <w:p wsp:rsidR="00000000" wsp:rsidRDefault="00766B64"></w:p>
+ <w:p wsp:rsidR="00000000" wsp:rsidRDefault="00766B64"></w:p>
  <aml:annotation aml:id="5" w:type="Word.Bookmark.Start" w:name="Pkg_Diagram_End_Inner"/>
  <aml:annotation aml:id="5" w:type="Word.Bookmark.End"/>
- <w:p wsp:rsidR="00000000" wsp:rsidRDefault="00766B64">
  + <w:pPr></w:pPr>
  + <w:r></w:r>
    <aml:annotation aml:id="6" w:type="Word.Bookmark.Start" w:name="Pkg_Diagram_End"/>
    <aml:annotation aml:id="6" w:type="Word.Bookmark.End"/>
  + <w:r></w:r>
```

Figura 15 – Representação de secções de *templates* EA em WordML

Aqui, verifica-se que os conteúdos que fazem parte de uma secção estão entre o nó `<aml:annotation>` com `w:name="<section-name>_Begin_Inner"` e o nó `<aml:annotation>` com `w:name="<section-name>_End_Inner"` (com `<section-name>` a representar o nome da secção a ser identificada). Este conteúdo é processado de uma forma recursiva para permitir que sejam identificadas correctamente

secções encadeadas e que esse comportamento seja retratado na estrutura hierárquica a construir.

```
− <w:r>
  + <w:rPr></w:rPr>
    <w:fldChar w:fldCharType="begin" w:fldLock="on"/>
  </w:r>
− <w:r>
  + <w:rPr></w:rPr>
    <w:instrText>MERGEFIELD </w:instrText>
  </w:r>
− <w:r>
  + <w:rPr></w:rPr>
    <w:instrText>Pkg.Notes</w:instrText>
  </w:r>
− <w:r>
  + <w:rPr></w:rPr>
    <w:fldChar w:fldCharType="separate"/>
  </w:r>
− <w:r>
  + <w:rPr></w:rPr>
    <w:t>{Pkg.Notes} </w:t>
  </w:r>
− <w:r>
  + <w:rPr></w:rPr>
    <w:fldChar w:fldCharType="end"/>
  </w:r>
```

Figura 16 – Representação de propriedades de *templates* EA em WordML

No caso das propriedades, verifica-se que estas são representadas por uma estrutura constante. O seu início é identificado pela existência dos nós `<w:fldChar w:fldCharType="begin"/>` e `<w:instrText>MERGEFIELD </w:instrText>`. O nó `w:instrText` seguinte contém o nome da propriedade – Pkg.Notes, no caso – e dá-se a separação entre os nós identificativos da propriedade e os nós com o texto contido na mesma através de um nó `<w:fldChar w:fldCharType="separate"/>`. Todos os nós que apareçam a seguir a este e até ser encontrado um nó `<w:fldChar w:fldCharType="end"/>` (identificador do final da propriedade), são considerados como o texto da propriedade que é mostrado ao utilizador.

O WordML obtido depois da conversão efectuada é, então, percorrido à procura de estruturas coincidentes com as acima ilustradas, sendo construída uma estrutura hierárquica de objectos de lógica de negócio para armazenar a informação correspondente.

No final, a estrutura produzida é gravada para as tabelas correspondentes da base de dados.

Modelo de dados

Para conseguir suportar as funcionalidades de integração de documentos tornou-se necessário criar algumas tabelas para armazenar a informação relacionada com alguns elementos do domínio. A Figura 17 representa as tabelas criadas e alteradas no âmbito destas funcionalidades.

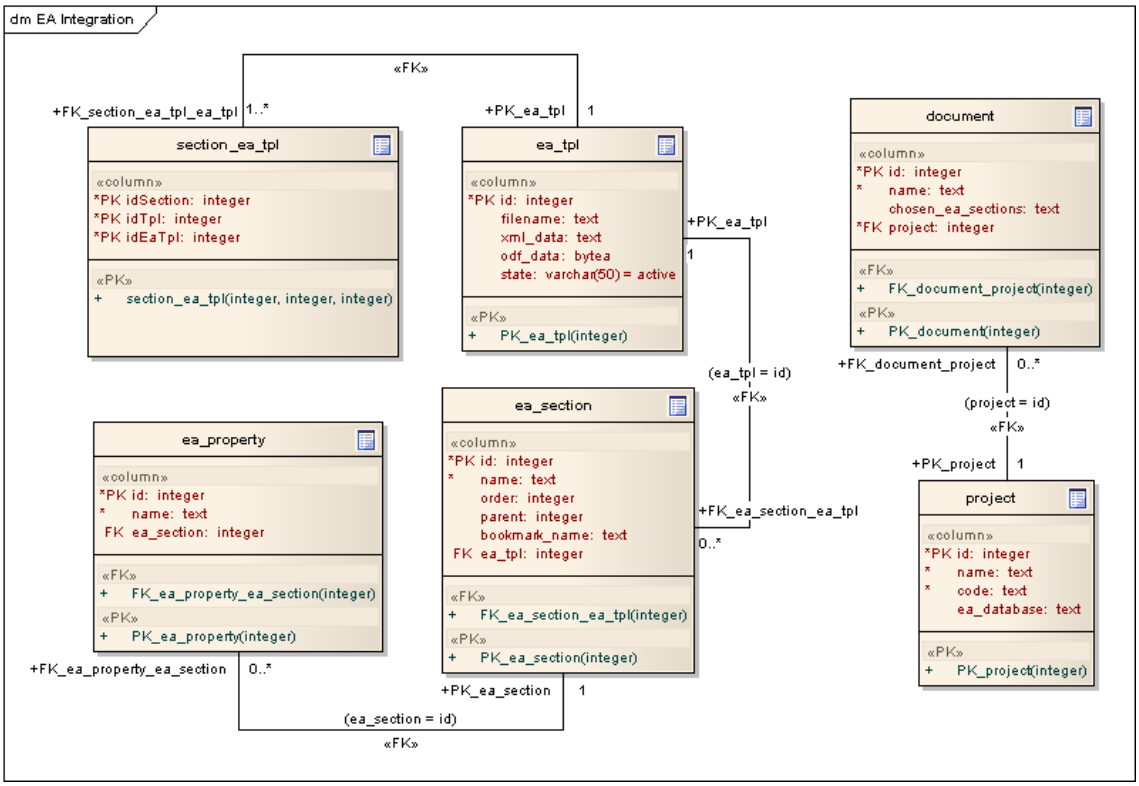

Figura 17 – Modelo de Dados

As tabelas apresentadas são apenas referentes às necessidades levantadas pela integração de documentos, servindo a tabela `section_ea_tpl` de ligação com as já existentes. Estas podem ser consultadas no Anexo B, onde está presente o modelo de dados global ao sistema.

Foram, então, criadas as seguintes tabelas:

- `ea_tpl` – armazena informação relativa a cada um dos *templates* sob controlo do sistema. Guarda o nome do ficheiro e também o código WordML e o código ODF do *template*;
- `ea_section` – armazena cada uma das secções de um *template* Enterprise Architect, guardando também informação da ordem por que esta aparece no documento ou a secção que é a sua secção-pai (as secções podem apresentar uma estrutura hierárquica);
- `ea_property` – armazena cada uma das propriedades de cada secção de um *template* Enterprise Architect;
- `section_ea_tpl` – serve de ligação entre uma secção de um *template* de um documento e o *template* dos conteúdos de Enterprise Architect possíveis de colocar na secção (caso existam);
- `document` – armazena os documentos criados a partir da ferramenta desenvolvida, guardando também as opções escolhidas relacionadas com os dados de Enterprise Architect a incluir. Todos os documentos gerados têm de se encontrar associados a um projecto em específico. Tem como objectivo facilitar a actualização de documentos, que será detalhada mais à frente neste documento.

A tabela project transita do projecto Documentation Framework, tendo sido estendida com a coluna `ea_database` que permite armazenar, para cada projecto, a base de dados onde está a informação dos modelos criados com o Enterprise Architect.

Associação de *templates* a secções de documentos

Sendo os *templates* dos documentos e os *templates* dos conteúdos de Enterprise Architect produzidos independentemente, foi necessário arranjar uma forma de conseguir associar estes dois tipos.

Aproveitando a interface de gestão de secções existente, foi adicionada a esta uma nova potencialidade, a de escolha do *template* EA pretendido para cada secção.

A Figura 18 demonstra um exemplo desta situação.

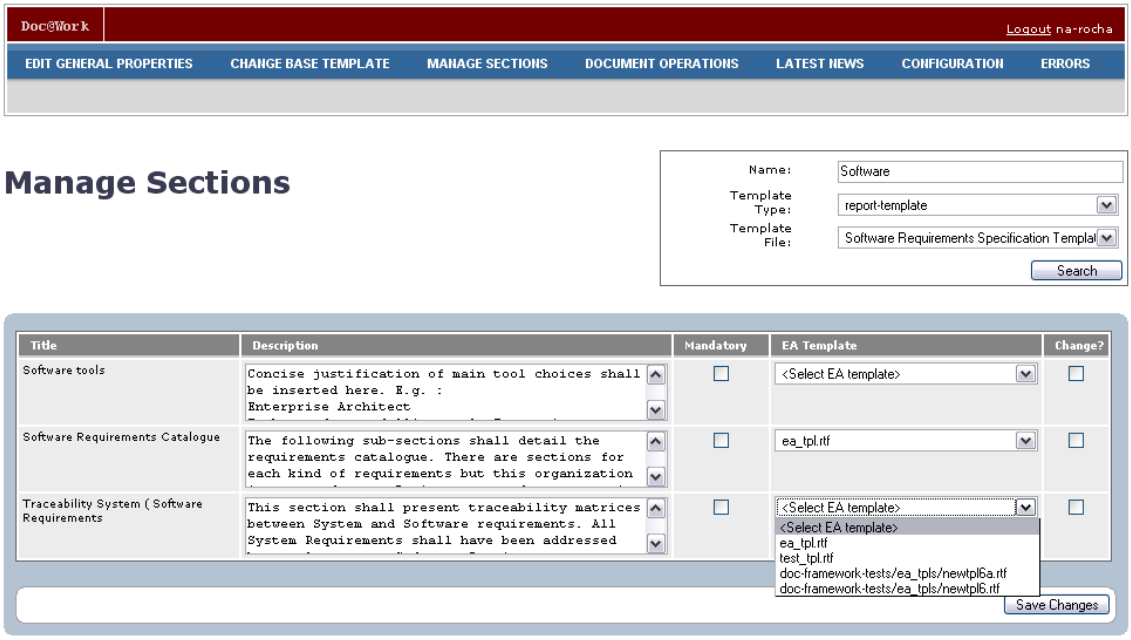

Figura 18 – Associação de *templates* a secções

Cada secção de cada documento é possível de ser associada a um determinado *template* de Enterprise Architect, para que o sistema saiba quais os conteúdos e a disposição dos mesmos dentro da secção. Isto foi conseguido com a introdução de uma lista com todos os *templates* do conhecimento do sistema, cabendo ao utilizador a escolha daquela que pretender.

Ao serem guardadas as alterações efectuadas é preenchida a tabela `section_ea_tpl` com as referências dos elementos intervenientes na associação (*template* de documento, *template* de secção EA e secção de documento).

Esta informação é também gravada no documento físico presente no repositório de CVS. Assim, o *template* do documento em questão é recuperado do repositório e é nele colocado um *bookmark*, na secção pretendida, indicando que esta tem a potencialidade de conter dados da ferramenta de modelação. No caso de já existir um *bookmark* na localização pretendida – significando que a secção em questão já tinha um *template* associado, estando-se a modificar ou remover esta associação – este é alterado ou simplesmente removido, consoante a acção realizada.

Este *bookmark* tem a forma `EA_<id_ea>`, em que `<id_ea>` indica o código de identificação do *template* pretendido na base de dados.

No fim do processo, o *template* actualizado do documento é novamente colocado no repositório CVS, contando como uma nova revisão.

Carregamento dinâmico do configurador de documentos

O componente de configuração de documentos é carregado dinamicamente primeiro com os conteúdos do *template* escolhido como base para o documento a gerar. A esta funcionalidade já existente, adicionou-se o carregamento, também dinâmico, dos conteúdos dos *templates* de Enterprise Architect associados a cada secção escolhida para fazer parte do documento final.

Após a escolha do documento-tipo pretendido e do carregamento da interface com as secções e propriedades deste, o utilizador pode então proceder à escolha das secções a incluir. Estas, caso tenham sido previamente definidas como podendo ter conteúdos da ferramenta de modelação (como descrito na secção anterior), despoletam o aparecimento de novas *tabs* onde estão presentes os seus conteúdos possíveis. Assim, por cada secção com esta possibilidade, é apresentada uma *tab*. A Figura 19 apresenta esta situação.

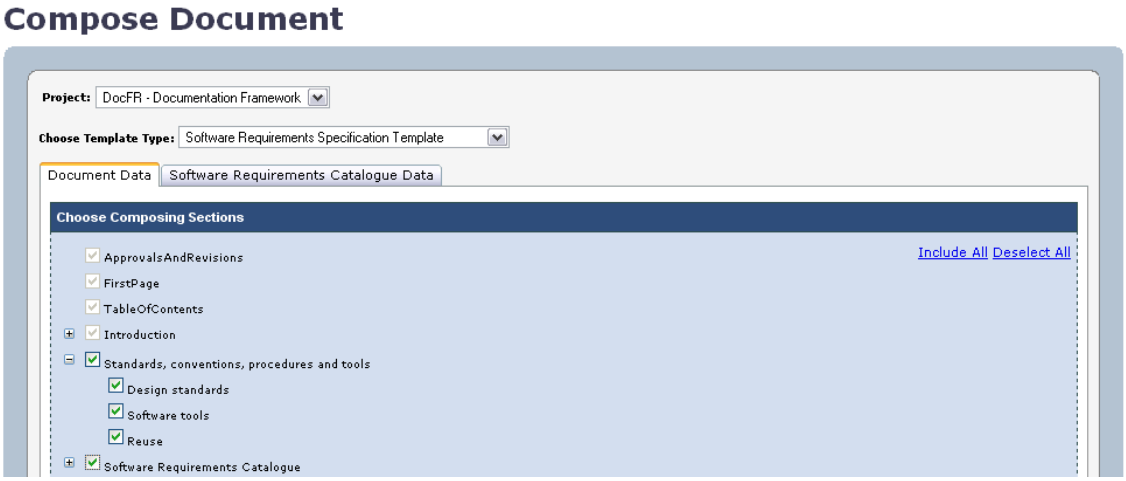

Figura 19 – Apresentação de *tabs* mediante as secções escolhidas

A secção *Software Requirements Catalogue* é a única configurada como podendo ter dados do Enterprise Architect e, estando seleccionada para fazer parte do documento, é mostrada uma *tab* com o mesmo nome que irá conter os dados possíveis de incluir nesta secção.

O conteúdo destas *tabs* é carregado previamente e é construído a partir da estrutura lida do WordML do *template* e gravada nas tabelas descritas anteriormente. Segue uma estrutura hierárquica, com secções e subsecções e com as propriedades destas funcionando como "folhas" na estrutura em árvore construída. A Figura 20 retrata um exemplo desta situação.

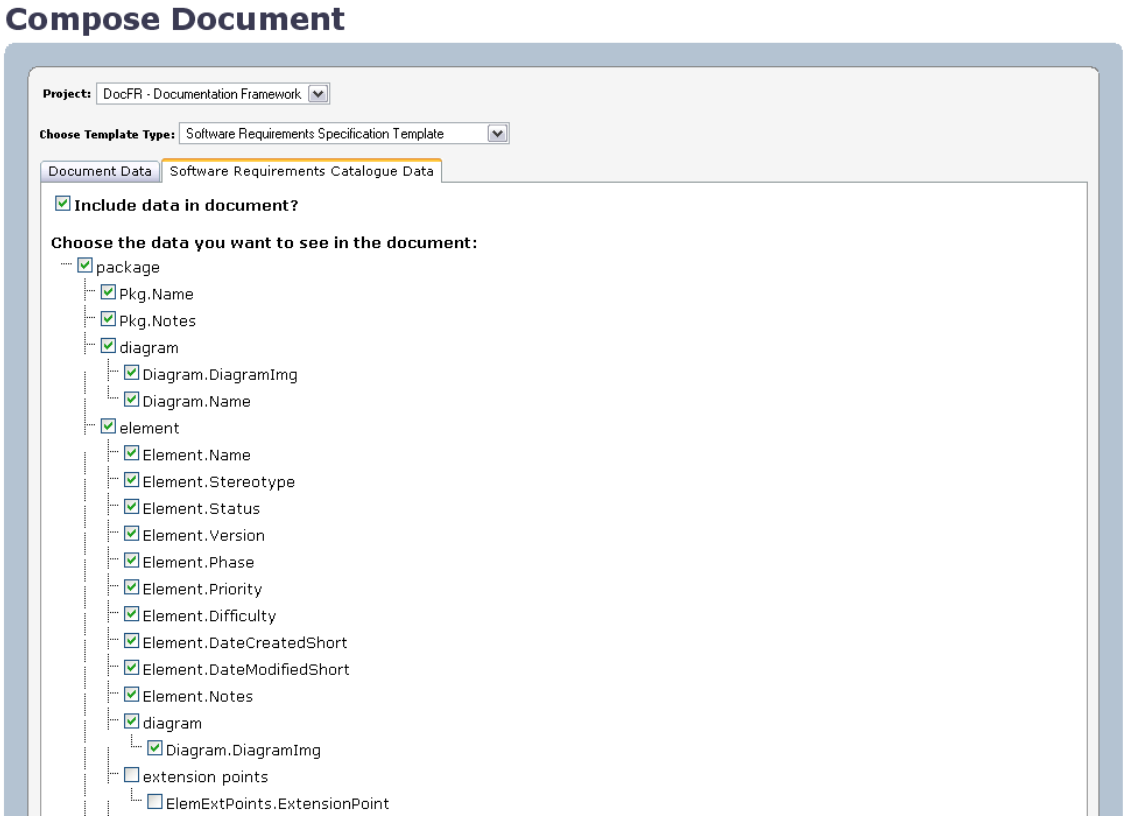

Figura 20 – Conteúdo de uma *tab* referente a conteúdos de Enterprise Architect

Verifica-se nesta *tab*, além da estrutura hierárquica representativa dos conteúdos possíveis, a existência de uma *checkbox* – *Include data in document?* – que o utilizador deve seleccionar para que a secção seja efectivamente preenchida. Esta *checkbox* foi adicionada já que o utilizador pode querer incluir apenas o esqueleto da secção sem os conteúdos da ferramenta de modelação, mesmo que isto seja possível.

Integração de conteúdos

A integração dos conteúdos presentes no Enterprise Architect com os documentos configurados através do mecanismo desenvolvido na Documentation Framework é

conseguida através das marcas – *bookmarks*, neste caso específico – incluídas nos documentos-tipo durante a associação de *templates* EA a secções.

Assim, após a geração do documento com as secções pretendidas pelo utilizador, o sistema passa para o processo de integração de conteúdos.

Mediante as escolhas efectuadas em cada *tab* correspondente a cada secção a preencher, o sistema interroga a base de dados do Enterprise Architect, no projecto correspondente, pelos dados pretendidos. Para isto foi necessário efectuar um mapeamento entre as propriedades possíveis de aparecer nos *templates* e as tabelas e colunas correspondentes, de maneira a que se saiba onde ir buscar os dados. Este mapeamento foi efectuado, na sua generalidade, com sucesso, estando ainda em aberto algumas propriedades que, à altura da entrega deste trabalho, ainda não tinham sido descobertas na base de dados. O mapeamento conseguido está em parte presente no Anexo C.

A aplicação percorre a lista das propriedades e secções a incluir, interrogando a base de dados do projecto e gerando a informação a incluir nos documentos a partir do WordML gravado aquando da análise ao *template*. Com os dados do projecto, são feitas pesquisas no WordML do *template*, gerados os pedaços de documento correspondentes à quantidade de dados encontrada e incluídos cada um destes no documento final. Este é um processo recursivo, que se repete para todas as secções e propriedades escolhidas na interface.

Por acordo com os *key users* deste domínio de aplicação, foi decidido que as propriedades não escolhidas para fazer parte do documento final seriam na mesma incluídas, mas com o valor "*Not Applicable*". Isto fica-se a dever a questões relacionadas com certificações de qualidade e garantia que a não inclusão destas propriedades não se deveu a simples esquecimento mas foi sim propositada.

No final de todo este processo, o documento completamente configurado e preenchido com conteúdos é disponibilizado ao utilizador.

Apesar de praticamente todos os dados pretendidos estarem presentes na base de dados que serve de suporte à modelação de conteúdos, as imagens dos diagramas estão ausentes. Isto deve-se ao facto de estas imagens serem geradas em tempo real pelo Enterprise Architect, estando apenas presente na base de dados informações relacionadas com as coordenadas dos elementos constituintes dos diagramas (linhas, setas, caixas, etc.).

Devido a isto, verificou-se a necessidade de usar automação do Enterprise Architect para quando as imagens são escolhidas para fazer parte dos documentos construídos. De realçar que, por razões de *performance* da aplicação, o Enterprise Architect não é lançado e fechado por cada diagrama que é necessário produzir. A abordagem seguida é antes a de lançar a ferramenta e fechá-la ao fim de 350 diagramas gerados ou no final da operação relacionada com o documento a ser processado (o que se verificar primeiro). O número 350 conseguiu-se através de testes de carga efectuados ao Enterprise Architect na parte de geração de documentação com diagramas.

A Figura 21 demonstra uma visão geral deste processo.

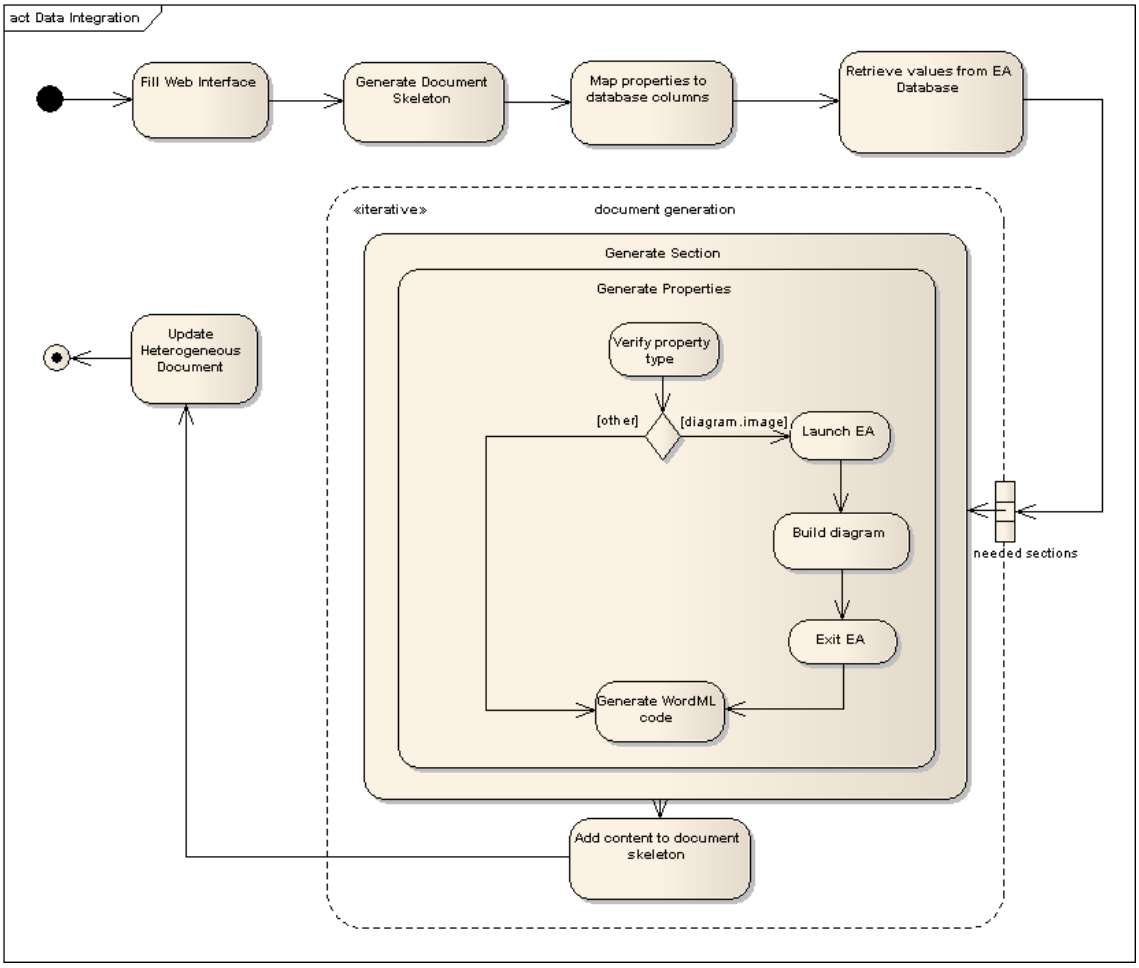

Figura 21 – Integração de dados de Enterprise Architect em documentos

A geração de documentos no formato ODF é em tudo idêntica ao processo descrito acima, mas utiliza como base para construção de documentos não o formato WordML mas sim o código ODF que é gravado durante o procedimento de análise aos *templates*. Esta opção foi tomada devido às incompatibilidades existentes, ao nível de estilos, à data da elaboração deste trabalho entre os formatos Microsoft e o formato ODF. A ideia original era a geração em WordML dos conteúdos pretendidos e a sua posterior conversão para ODF antes de os integrar no documento final. Devido às incompatibilidades referidas, tornou-se uma tarefa impossível de realizar com a qualidade desejada, tendo-se optado por usar o mesmo formato para geração dos conteúdos e para apresentação final. Sendo um formato baseado também ele numa norma aberta e baseada em XML, esta tarefa não se tornou de uma complexidade exagerada, sendo realizável no tempo útil disponível para o trabalho.

Relevância da escolha do projecto ao qual o documento fica associado

A escolha do projecto ao qual o documento a gerar/actualizar vai ficar/está associado está presente em qualquer um destes dois procedimentos (detalhados com maior pormenor quer nas secções correspondentes deste capítulo, quer no Anexo A, na secção dedicada ao manual de utilização da ferramenta).

Figura 22 – Escolha do projecto ao qual associar o documento

Esta escolha, ilustrada na Figura 22 assume-se da maior importância se for tido em conta que, tipicamente, as questões relacionadas com projectos diferentes são modeladas recorrendo a ficheiros de Enterprise Architect diferenciados. No presente caso, e como foi referido, esta modelação é efectuada utilizando uma base de dados relacional para armazenar os conteúdos (ao contrário de ficheiros .eap), mas o princípio de separação dos mesmos deve também ser adoptado. Quer isto dizer que deve ser usada uma instância de uma base de dados para cada projecto em causa.

Para que o sistema saiba onde ir buscar os dados relevantes é, então, necessário que haja algum tipo de informação que indique qual a base de dados a utilizar.

A Critical Software, S.A. tem já algumas ferramentas internas onde são guardadas informações sobre os projectos e as suas características, tendo sido proposta, no âmbito deste trabalho, a criação de um campo onde pudesse estar presente a indicação da base de dados Enterprise Architect utilizada.

Assim, o sistema dá a possibilidade ao utilizador de escolha do projecto associado para que depois possa aceder à informação de qual a base de dados por ele utilizada. Foi também definido que as credenciais de acesso a esta base de dados seriam idênticas às usadas por cada utilizador para aceder ao sistema desenvolvido, facilitando assim o processo de ligação à base de dados pretendida e contribuindo também para uma interligação mais suave entre ferramentas.

Memorização das opções de geração

Com o objectivo de melhorar a funcionalidade de actualização de dados em documentos já construídos procurou-se, de alguma forma, memorizar as opções tomadas no momento da geração do documento (ao nível dos conteúdos incluídos provenientes da ferramenta de modelação). Desta forma, o utilizador consegue não só actualizar os conteúdos existentes para que estes sejam substituídos pelos mais recentes, mas também colocar mais dados que lhe interessem ou retirar dados que deixaram de ter relevância para o documento e secção em questão.

A abordagem seguida para resolver esta questão baseou-se no armazenamento das opções tomadas na base de dados do projecto, na tabela `document` criada para o efeito. Esta armazena, além do projecto ao qual foi associado o documento gerado (escolha feita durante o processo de geração original), uma estrutura XML, hierárquica, de secções e propriedades dos *templates* de conteúdos Enterprise Architect.

A estrutura armazenada contém, para cada secção e propriedade do *template* escolhido, a indicação se esta foi escolhida para fazer parte do documento gerado e também alguns dados necessários para o correcto processamento desta informação, tais como seus os identificadores e nomes. Assim, a aplicação apenas necessita de percorrer esta estrutura para criar a interface relativa ao documento que se pretende actualizar.

Na Figura 23 está presente um exemplo de uma estrutura deste tipo guardada na base de dados.

```
- <eatpls>
  - <eatpl id="1" name="Software Requirements Catalogue">
    - <section id="1" name="package" bookmarkname="Pkg" order="1" checked="True">
        <property id="1" name="Pkg.Name" checked="True"/>
        <property id="2" name="Pkg.Notes" checked="True"/>
      - <section id="2" name="diagram" bookmarkname="Pkg_Diagram" order="2" checked="False">
          <property id="3" name="Diagram.DiagramImg" checked="False"/>
          <property id="4" name="Diagram.Name" checked="False"/>
        </section>
      - <section id="3" name="element" bookmarkname="Pkg_Element" order="3" checked="True">
          <property id="5" name="Element.Name" checked="True"/>
          <property id="6" name="Element.Stereotype" checked="False"/>
          <property id="7" name="Element.Status" checked="True"/>
          <property id="8" name="Element.Version" checked="True"/>
          <property id="9" name="Element.Phase" checked="False"/>
          <property id="10" name="Element.Priority" checked="False"/>
          <property id="11" name="Element.Difficulty" checked="False"/>
          <property id="12" name="Element.DateCreatedShort" checked="False"/>
          <property id="13" name="Element.DateModifiedShort" checked="False"/>
          <property id="14" name="Element.Notes" checked="False"/>
        - <section id="9" name="diagram" bookmarkname="Pkg_Element_Diagram" order="4" checked="False">
            <property id="23" name="Diagram.DiagramImg" checked="False"/>
          </section>
        - <section id="7" name="extension points" bookmarkname="Pkg_Element_ElemExtPoints" order="5" checked="False">
            <property id="16" name="ElemExtPoints.ExtensionPoint" checked="False"/>
          </section>
        - <section id="8" name="transition" bookmarkname="Pkg_Element_ElementTransition" order="6" checked="False">
            <property id="17" name="ElementTransition.DurationConstraint" checked="False"/>
            <property id="18" name="ElementTransition.Event" checked="False"/>
            <property id="19" name="ElementTransition.Note" checked="False"/>
            <property id="20" name="ElementTransition.TimeConstraint" checked="False"/>
            <property id="21" name="ElementTransition.Variable" checked="False"/>
            <property id="22" name="ElementTransition.Value" checked="False"/>
          </section>
          <section id="4" name="scenario" bookmarkname="Pkg_Element_ElemScenario" order="7" checked="False"/>
        </section>
        <section id="5" name="child packages" bookmarkname="Pkg_Pkg" order="8" checked="True"/>
      </section>
    </eatpl>
  </eatpls>
```

Figura 23 – Estrutura-exemplo de gravação de opções escolhidas

Esta estrutura é constituída por um conjunto de *templates* de Enterprise Architect, representados, cada um, por uma *tag* `<eatpl>` (é necessário armazenar um conjunto de *templates* uma vez que um documento pode ter vários *templates* associados, dependendo das secções escolhidas para ele). Esta *tag* tem como atributos o identificador do *template* e o nome da secção ao qual se encontra associado.

Cada *template* pode ter várias secções. Estas estão representadas pelas *tags* `<section>` e apresentam como atributos o seu identificador, o seu nome, a ordem pela

qual aparecem no *template*, o nome do *bookmark* a ela associado no *template* Enterprise Architect e uma indicação se foi ou não escolhida para fazer parte do documento (`checked`).

Cada secção pode ter várias propriedades representadas pela *tag* `<property>`, cada uma destas tendo como atributos o seu identificador, o seu nome e um atributo `checked` que é em tudo idêntico ao existente para as secções.

As opções são regravadas na base de dados de cada vez que o utilizador se dirige à ferramenta para actualizar o documento em questão, de maneira a que esteja sempre visível o conteúdo correcto.

Actualização de documentos

O procedimento adoptado para proceder à actualização dos conteúdos do Enterprise Architect em documentos já existentes é bastante semelhante ao existente na geração dos mesmos.

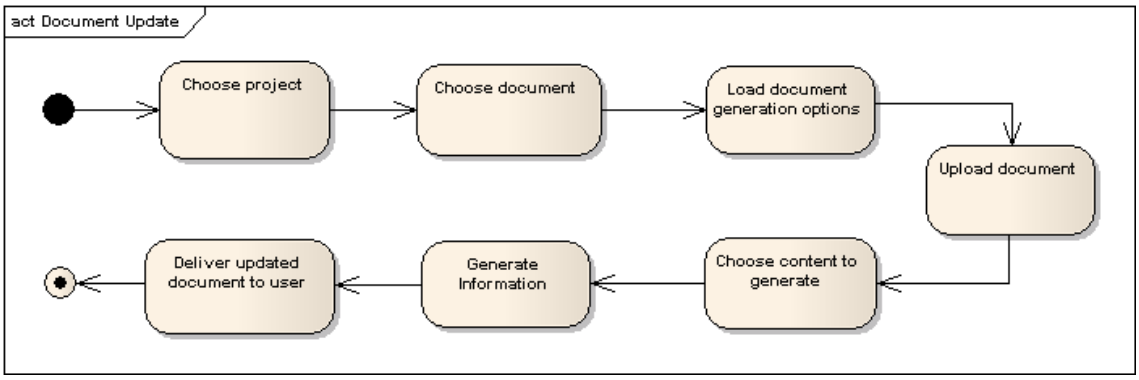

Figura 24 – Actualização de documentos

Com efeito, e seguindo o esquema ilustrado na Figura 24, o utilizador necessita de dar algumas informações sobre o documento que pretende actualizar (a que projecto foi associado, qual o seu nome) e fazer *upload* deste para a ferramenta. Aproveitando a funcionalidade de manutenção das opções de geração descrita na secção anterior, torna-se relativamente simples saber que secções do documento podem ser actualizadas e qual o seu conteúdo (quer o efectivamente gerado, quer todo o possível), sendo este apresentado

novamente sob a forma de *tabs*, cada uma relativa a uma secção e com a indicação dos dados em si presentes.

Após este processo de escolha, a ferramenta trata de aceder à base de dados do Enterprise Architect para ir colher os dados pertinentes, utilizando mais uma vez o código WordML armazenado – ou o código ODF, consoante o formato pretendido para o ficheiro de saída – para construir os pedaços de documento relativos a estes conteúdos. Estes pedaços são posteriormente inseridos no local exacto pretendido, deixando o resto do documento completamente intacto.

Isto é conseguido, mais uma vez, através da identificação do local específico pelos *bookmarks* colocados no documento. Este mecanismo, que é uma concretização do mecanismo de *tags* para identificar locais em documentos referido no capítulo 5.2, permite a realização das tarefas pretendidas de forma eficaz e praticamente transparente para o utilizador final.

6.2 Conclusões Obtidas com o Caso de Estudo

Apesar de se ter focado em específico na combinação de ferramentas Enterprise Architect e MS Word/OpenOffice Writer, esta implementação permitiu retirar algumas conclusões ao nível de aplicabilidade e flexibilidade da abordagem.

Assim, uma das conclusões mais importantes que se pode retirar é que a identificação de partes de documentos para inclusão de conteúdos é adequada ao contexto e permite um bom controlo da estrutura da documentação a produzir.

No entanto, a instanciação da abordagem proposta no caso de estarem envolvidas outras ferramentas de modelação está dependente das funcionalidades por estas apresentadas em termos de acesso aos seus conteúdos. Este facto pode aumentar o esforço associado à implementação de uma aplicação informática de suporte à abordagem ou mesmo torná-la impossível de automatizar.

Por outro lado, a necessidade de existência de *templates* para os conteúdos a incluir nas secções dos documentos, não se torna num factor de exagerada complexidade, dado que várias ferramentas já dispõem de componentes para os criar. Será uma questão de

procurar processar estes *templates* para incluir os seus dados no sistema desenvolvido e, numa fase posterior, conseguir efectuar a geração das partes relevantes dos documentos.

Pode-se concluir que a abordagem desenvolvida é possível de ser implementada com sucesso. No entanto, e dado que há imensas ferramentas de modelação com as suas características específicas, verifica-se que a aplicabilidade depende das possibilidades fornecidas pelas ferramentas, em especial em termos de acesso a dados nelas contidos.

Capítulo 7.
Validação

Uma das fases mais importantes de um projecto de investigação é a validação dos conceitos teóricos desenvolvidos numa situação real e concreta.

Assim, é importante estabelecer um paralelismo numa determinada situação entre a abordagem proposta e a pré-existente.

Este capítulo procura explicar o contexto e procedimentos efectuados para levar a cabo a validação do trabalho desenvolvido, ao mesmo tempo que identifica os principais resultados obtidos. Todos estes factores referem-se à implementação da solução concreta descrita no capítulo anterior.

7.1 Experiência

A condução de uma ou várias experiências, num ambiente de uma situação concreta que esteja relacionada com os conceitos desenvolvidos, permite avaliar a qualidade e aplicabilidade do trabalho desenvolvido. Assim é possível identificar o seu real valor e os benefícios por si trazidos.

O cenário para experimentação dos princípios defendidos foi a Critical Software, S.A.. Assim, o trabalho foi efectuado mediante uma relação estreita com o departamento de Qualidade da organização, já que é este que tem à sua responsabilidade a gestão das questões relacionadas com a documentação.

As funcionalidades foram sendo testadas à medida que iam sendo desenvolvidas, obtendo-se assim um sempre importante *feedback* para possibilitar o melhoramento da abordagem proposta.

7.2 Metodologia

Foi seguida uma metodologia de experimentação da nova ferramenta de suporte aos conceitos desenvolvidos, sendo posteriormente efectuada uma comparação com a abordagem previamente seguida. Esta última consiste em gerar/actualizar os conteúdos mediante a utilização do motor de *reporting* do Enterprise Architect. A inclusão dos conteúdos na documentação existente é conseguida através de cópia manual efectuada a partir dos relatórios produzidos pela ferramenta.

De maneira a quantificar os resultados obtidos, foram escolhidas como métricas principais de análise o tempo total necessário para realizar as tarefas pretendidas e a consistência de informação entreos conteúdos da ferramenta de modelação e a documentação. Outras métricas que também contribuíram para comparação efectuada foram a qualidade da documentação produzida e o grau de definição dos processos seguidos (nomeadamente no que diz respeito à organização de todos os ficheiros envolvidos no processo de integração de dados de várias fontes).

Os documentos utilizados para validar esta abordagem fazem parte da documentação técnica dos projectos, nomeadamente documentos de especificação de

requisitos, documentos de especificação de arquitectura e documentos de especificação de testes.

Das tarefas realizadas pelos utilizadores destacam-se a definição, configuração e armazenamento de *templates* de conteúdos Enterprise Architect, a geração de documentos heterogéneos e a actualização dos mesmos.

7.3 Execução e Resultados

Como foi referido, as experiências foram sendo conduzidas ao longo do desenvolvimento do trabalho, com especial incidência na sua parte final, onde a plataforma já se encontrava mais estável.

Os *templates* de Enterprise Architect utilizados foram alguns dos já existentes na organização, bem como os *templates* da documentação técnica. Estes conteúdos foram criados pelos elementos do departamento de Qualidade, sendo que esta última classe de *templates* – relativos à documentação técnica de projectos – segue a abordagem de *templates* base e *templates* específicos proposta no âmbito do projecto Documentation Framework e encontram-se armazenados no repositório CVS da empresa.

O projecto de Enterprise Architect utilizado como fonte de dados para incluir nos documentos produzidos foi um projecto exemplo fornecido com a ferramenta, entretanto estendido com mais dados – relacionados com requisitos, testes e alguns modelos de arquitectura – introduzidos durante os testes efectuados.

Em termos de resultados ao nível de documentação produzida, estes podem ser ilustrados pela Figura 25 e pela Figura 26.

A Figura 25 demonstra uma possível escolha de conteúdos Enterprise Architect efectuada na configuração do documento pretendido.

Verifica-se a escolha das propriedades relativas ao nome e notas de cada *package*, nome e tipo de cada elemento e imagem relativa ao diagrama de cada elemento.

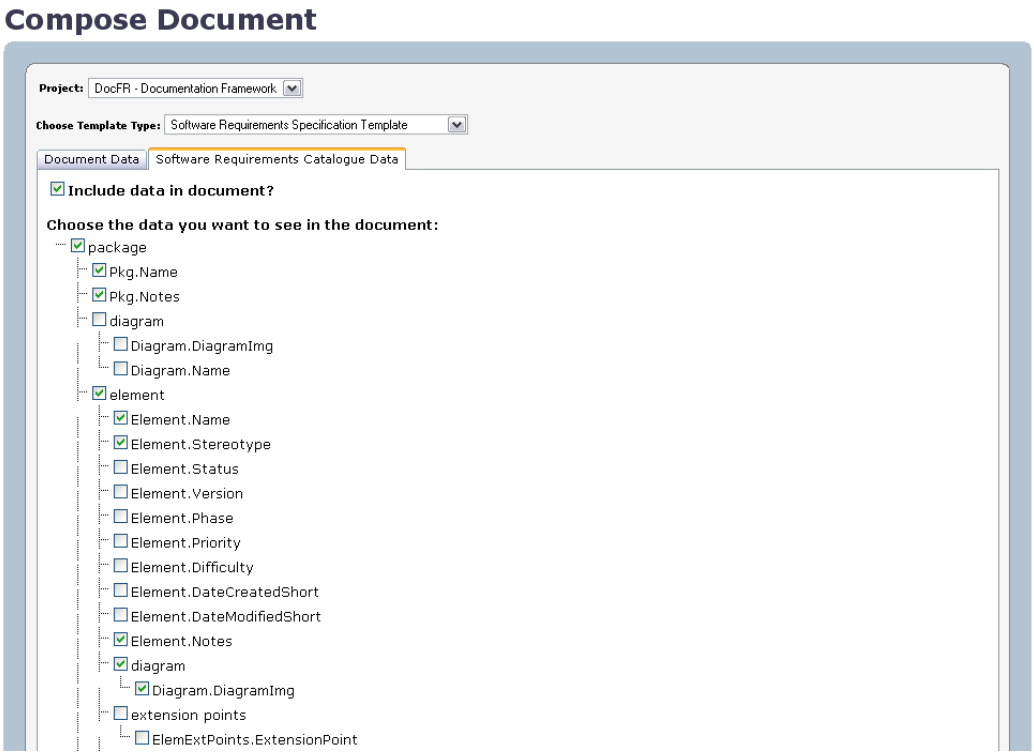

Figura 25 – Exemplo de configuração escolhida

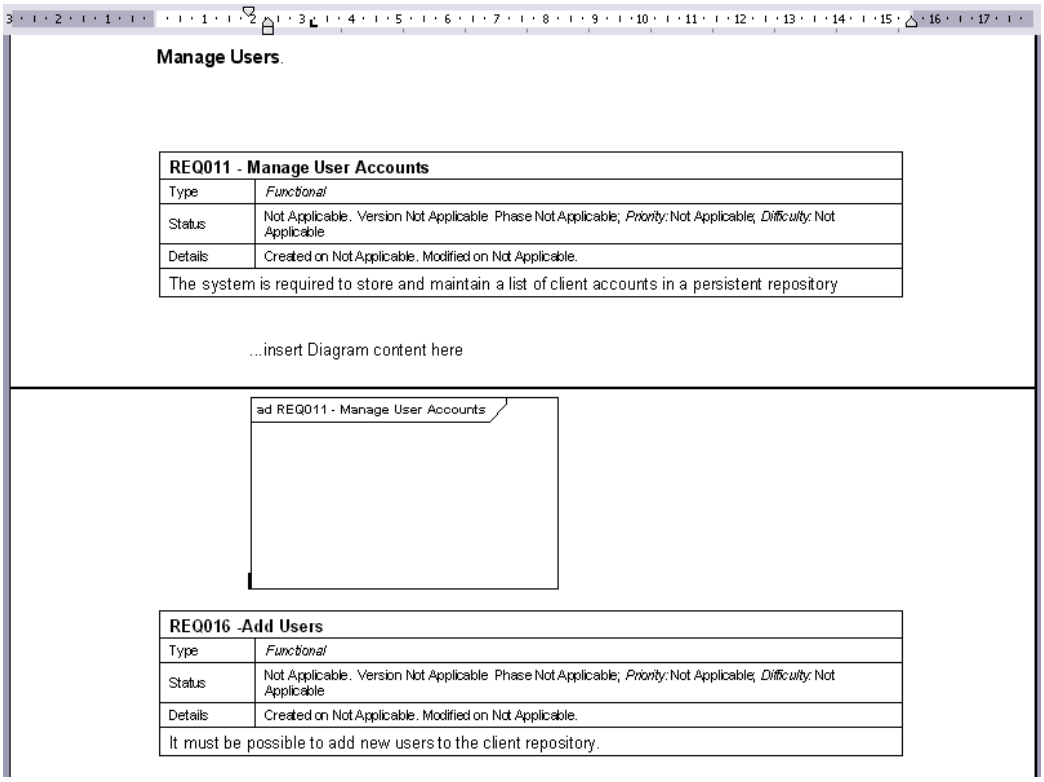

Figura 26 – Exemplo de documento produzido

Já na Figura 26 encontra-se parte do documento correspondente, verificando-se a existência de elementos (no caso, requisitos), estando algumas das suas propriedades preenchidas, tal como o indicado na interface Web. As propriedades não escolhidas são, conforme definido, preenchidas com o valor *"Not Applicable"*.

As duas figuras anteriores podem servir também para comprovar a elevada flexibilidade da solução obtida, no que diz respeito aos conteúdos a introduzir na documentação. Com efeito, a escolha de conteúdos através do sistema de *checkboxes* permite um maior grau de adaptação dos documentos às necessidades efectivas.

Ao nível da quantidade de dados testada – com implicações directas na performance da aplicação e, consequentemente, no tempo total de execução das tarefas pretendidas – uma parte importante dos testes incidiu sobre uma elevada quantidade de elementos. Isto aconteceu para se verificar se os ganhos em casos algo extremos seriam ou não efectivamente visíveis.

Segue-se um resumo dos principais resultados obtidos, divididos por métrica utilizada e respectivo processo testado.

Tempo total necessário para realização das tarefas

O processo de geração/actualização de documentos heterogéneos apresenta uma performance bastante melhor com a abordagem proposta. Este ganho é mais visível com o aumento de dados envolvidos no processo e está também relacionado com a necessidade existente na abordagem tradicional de proceder à cópia manual dos conteúdos pretendidos do documento gerado pelo motor de reporting do Enterprise Architect para a documentação existente.

A Tabela 6 contém o resumo de alguns testes efectuados. Os valores de requisitos testados (10, 50 e 100) foram escolhidos tendo em conta que são alguns dos valores mais frequentes para um projecto. Já o valor de 2300 requisitos refere-se a um dos projectos com maior quantidade de dados existente na organização. De realçar que os valores de tempo apresentados incluem as tarefas de *checkout* do repositório CVS do *template* original do documento a construir e a sua configuração (escolha das secções pretendidas),

funcionalidades já implementadas na Documentation Framework. Manualmente, estas tarefas têm tipicamente o "custo" de, aproximadamente, 2 minutos.

Já o processo de configuração e armazenamento de *templates* de Enterprise Architect apresenta valores mais altos de tempo envolvido na abordagem proposta. No entanto, este facto é perfeitamente justificável já que está relacionado com a necessidade de associar as secções dos documentos a *templates* produzidos e também com o facto de ter sido definido um procedimento para manuseamento destes *templates*. Com efeito, estes passam a estar armazenados no repositório CVS, à semelhança de outros conteúdos da organização, ao contrário do que acontecia até aqui (com estes a serem armazenados nos computadores locais de alguns elementos do projecto ou dos elementos do departamento de Qualidade, não sendo gerais à organização, mas sim específicos de cada projecto).

Tabela 6 – Resultados obtidos

	Abordagem manual	**Abordagem proposta**
10 requisitos / 1 diagrama	Aprox. 2:30 minutos	Aprox. 30 segundos
10 requisitos / 10 diagramas (1 diagrama por requisito)	Aprox. 2:25 minutos	Aprox. 35 segundos
50 requisitos / 1 diagrama	Aprox. 2:32 minutos	Aprox. 45 segundos
50 requisitos / 50 diagramas (1 diagrama por requisito)	Aprox. 2:40 minutos	Aprox. 55 segundos
100 requisitos / 1 diagrama	Aprox. 2:40 minutos	Aprox. 1 minuto
100 requisitos / 100 diagramas (1 diagrama por requisito)	Aprox. 2:55 minutos	Aprox. 1:20 minutos
2300 requisitos / 0 diagramas	Aprox. 7 minutos	Aprox. 1:50 minutos
2300 requisitos / 20 diagramas (1 diagrama por *package*)	Aprox. 8 minutos	Aprox. 3:30 minutos
2300 requisitos / 2300 diagramas (1 diagrama por requisito)	Aprox. 3 horas	Aprox. 47 minutos

Consistência de informação

A consistência entre a informação presente na ferramenta de modelação e a documentação é mais facilmente conseguida com a abordagem proposta.

O facto de se tratar de uma abordagem que apresenta as tarefas de geração e actualização de informação bastante automatizadas, com menor grau de intervenção humana, contribui para que a actualização de documentos seja efectuada mais frequentemente. Isto diminui a possibilidade de ocorrência de inconsistências, que aumenta consoante o tempo de desfasamento entre a actualização dos modelos e a actualização da documentação.

Em relação aos conteúdos propriamente ditos, como as suas fontes de informação são os modelos produzidos com a ferramenta adequada, verifica-se que o que aparece na documentação corresponde por inteiro ao que foi previamente criado.

Qualidade da documentação produzida

Neste item verifica-se que as duas abordagens são bastante semelhantes e podem ser avaliadas em duas vertentes (que se combinam para obter o parâmetro pretendido): a manutenção de um aspecto geral da documentação e a sua completude.

Os *templates* de conteúdos Enterprise Architect são produzidos tendo em conta o estilo utilizado nos documentos da Critical Software. Uma vez que as duas abordagens utilizam estes artefactos como base para a geração pretendida, ambas mantêm um nível bastante elevado neste parâmetro de avaliação.

Em termos de completude da documentação ambas as abordagens permitem a geração de documentos com bastante informação associada, sendo esta quantidade definida pelos *templates* definidos.

Grau de definição dos processos seguidos

A abordagem proposta apresenta um maior grau de definição em todo o processo de integração das ferramentas de modelação e de processamento de texto.

Com efeito, propõe a criação de um local específico e central à organização para armazenamento dos *templates* utilizados para o conteúdo das ferramentas de modelação.

Permite também efectuar o processo de geração/actualização de documentos com uma só aplicação e sem a necessidade de intervenção manual.

Capítulo 8.
Conclusões

A realização deste trabalho contribuiu para dar mais um passo no que diz respeito à produção eficaz de documentação com qualidade no âmbito de um projecto de software.

Com efeito, é proposta uma abordagem com algumas características importantes relacionadas com a flexibilização do processo de configuração de documentação e integração dos dados provenientes de ferramentas de modelação com os documentos produzidos.

Num domínio mais concreto, a aplicação dos conceitos desenvolvidos a um caso concreto, num ambiente real, permitiu ter uma verdadeira noção da aplicabilidade dos mesmos, ao mesmo tempo que contribuiu para a integração das ferramentas utilizadas no caso de estudo (Enterprise Architect e MS Word/OpenOffice Writer).

Este capítulo desenvolve mais em detalhe estas conclusões, apresentando também um sumário das principais dificuldades encontradas, bem como das contribuições para a comunidade e pontos a ter em conta para futuros melhoramentos.

8.1 Principais Resultados

Os resultados principais deste trabalho podem ser vistos do ponto de vista mais teórico e do ponto de vista mais prático.

Assim, do ponto de vista mais teórico, salienta-se o desenvolvimento de uma abordagem ao processo de produção/actualização de documentação com vista a uma mais fácil integração de conteúdos provenientes de diferentes fontes de dados para documentos processados com vulgares processadores de texto de *suites* de escritório. Esta contempla uma elevada flexibilidade, quer ao nível de configuração de *layout* e escolha de conteúdos, quer ao nível aplicabilidade a vários cenários de utilização. Além disso, procurou-se aumentar o nível de definição e organização do processo de documentação, apresentando uma abordagem com linhas orientadoras concretas.

Em termos de ferramentas informáticas para servir de suporte aos conceitos apresentados, pode ser referido que o desenvolvimento de um sistema com um *front-end web-based* para englobar dados provenientes de Enterprise Architect em documentação nos formatos MS Word e OpenOffice Writer se traduziu num factor essencial para a comprovação da aplicabilidade dos princípios defendidos. Para além disso, esta ferramenta permitiu a obtenção de valores muito interessantes em termos de tempo poupado e facilidade no processo de geração e manutenção de documentação com características visíveis de heterogeneidade.

Foram, assim, alcançados bons resultados no que diz respeito à usabilidade, disponibilidade, desempenho e segurança da aplicação de suporte e também à consistência da informação existente nos modelos e nos documentos.

8.2 Maiores Desafios

Ao longo de todo o processo que culminou com a apresentação do trabalho documentado, foram sendo encontradas algumas dificuldades, ultrapassadas com esforço e persistência na persecução dos objectivos propostos.

Estas dificuldades centraram-se principalmente na implementação do sistema que serviu de *proof-of-concept* da abordagem desenvolvida, estando relacionadas com algumas limitações tecnológicas das ferramentas escolhidas para o caso de estudo.

Assim, podem ser referidos os problemas encontrados, em especial na questão de conversão/uniformização de formatos (RTF, DOC e ODF, essencialmente), que se fizeram sentir na dificuldade em manter o estilo definido no *template* original nos documentos finais apresentados nos formatos pretendidos.

Outro dos aspectos mais problemáticos centrou-se no desenvolvimento do componente de configuração de documentos, dado o elevado nível de dinamismo pretendido na interface a desenvolver.

Apesar disto, os objectivos propostos foram atingidos, mesmo tendo sido por vezes necessário um reajuste dos princípios, métodos e fluxos de informação envolvidos no sistema a implementar.

8.3 Resumo de Contribuições

Uma das maiores contribuições deste trabalho encontra-se ao nível da flexibilidade permitida no processo de escolha de conteúdos visíveis na documentação, permitindo ao utilizador uma escolha exacta do que pretende visualizar.

A aplicação desenvolvida para dar suporte à abordagem conceptualizada permitiu dar um passo em frente no que diz respeito à integração de dados de Enterprise Architect com documentação em formatos MS Word e OpenOffice Writer. Assim, com este sistema, torna-se mais simples e rápido a integração pretendida, mantendo a qualidade da documentação produzida.

8.4 Trabalho Futuro

Todos os projectos e abordagens desenvolvidas têm espaço para melhoramentos. Como tal, também este não foge à regra.

Assim, um dos aspectos considerados como passíveis de mais investigação é a abordagem de actualização da informação inserida nos documentos, de maneira a dar maior possibilidade aos elementos da equipa de projecto de incluírem os seus comentários em paralelo com esta informação, sem o risco de estes serem perdidos no processo de actualização das secções correspondentes.

Ao nível da aplicação desenvolvida, foram identificados alguns pontos onde algum trabalho futuro poderá contribuir para o aumento do valor para os seus utilizadores:

- fazer com que o sistema tenha total controlo sobre os documentos que produz, evitando assim a necessidade do upload dos mesmos para o processo de actualização;
- procurar melhorar ainda mais os ganhos de performance obtidos;
- adicionar suporte para outras ferramentas de modelação e outros formatos de *output* de documentação;
- possibilidade de integração com algumas ferramentas de gestão interna que possam existir nas organizações onde a aplicação está em funcionamento, para facilitar o acesso a certo tipo de informação, como por exemplo, informação relacionada com os projectos e as suas características.

8.5 Considerações Finais

Como conclusão pode ser referido que este trabalho, mediante uma série de factores, consegue trazer valor para as organizações onde for aplicado. De entre estes podem-se destacar a sua abrangência mediante a conceptualização de uma solução genérica, a sua aplicação prática mediante a criação de uma ferramenta de suporte aplicável a um domínio específico e o facto de ter sido desenvolvido e testado com sucesso num ambiente empresarial bastante exigente nesta área.

Bibliografia

[1] Aguiar, A. (2003). *A minimalist approach to framework documentation*, PhD thesis, Faculdade de Engenharia da Universidade do Porto.

[2] *Software Documentation*, http://en.wikipedia.org/wiki/Software_documentation.

[3] *Documentation definition by the Linux Information Project* (2006), http://www.linfo.org/documentation.html.

[4] IEEE (2001). *Standard for Software User Documentation - IEEE Std 1063-2001*. IEEE.

[5] Microsoft Corporation (2008). *What is document management?*, http://technet.microsoft.com/en-us/library/cc261933.aspx.

[6] Rocha, N (2007). *Documentation Framework*, Internship Report, Faculdade de Engenharia da Universidade do Porto.

[7] Silva, João Pedro (2004). *Gestão Documental: uma fonte de vantagem competitiva*. Disponível em http://www.novabase.pt/showNews.asp?idProd=resgestaodocvantcompet.

[8] Soares, António Lucas (2007). Acetatos de apoio à disciplina de Planeamento Estratégico de Sistemas de Informação.

[9] *Knowledge Management*, http://en.wikipedia.org/wiki/Knowledge_management.

[10] *UML Modeling Tools*, http://en.wikipedia.org/wiki/UML_tool.

[11] Sparx Systems (2008). *Enterprise Architect – UML Design Tools and UML CASE for software development*, http://www.sparxsystems.com.au/ea.htm.

[12] *Visual Paradigm for UML*, http://www.visual-paradigm.com/product/vpuml/.

[13] *IBM Software – Rational Rose*, http://www-306.ibm.com/software/awdtools/developer/rose/index.html.

[14] *Visio Home Page*, http://office.microsoft.com/en-us/visio/default.aspx.

[15] *What is word processing? – A Word Definition from the Webopedia Computer Dictionary*, http://www.webopedia.com/TERM/W/word_processing.html.

[16] *LyX | The Document Processor*, http://www.lyx.org/Home.

[17] Knuth, D. (1984). *The TeXbook (Computers and Typesetting),* Addison-Wesley. ISBN 0-201-13448-9.

[18] Aguiar A., David G. (2005), *WikiWiki weaving heterogeneous software artifacts*, Proceedings of the 2005 international symposium on Wikis, p.67-74, October 16-18, 2005, San Diego, California.

[19] Kunishima, T., Yokota, K., Liu, B. and Miyake, T. (1999). *Towards Integrated Management of Heterogeneous Documents*, Cooperative Da*tab*ases and Applications '99, pp.39-51, Springer, Sep., 1999. Disponível em http://alpha.c.oka-pu.ac.jp/yokota/paper/codas99.ps.

[20] *Literate Programming*, http://www.literateprogramming.com.

[21] Smith, M. (2001). *Towards modern literate programming*, Project Report HONS 10/01, Department of Computer Science, University of Canterbury, Christchurch, New Zealand.

[22] Sun Microsystems (2003). *Javadoc Tool Home Page*. Disponível em http://java.sun.com/j2se/javadoc/.

[23] van Heesch, D. (2002). *Doxygen — a documentation system for C++, Java and other languages*. Disponível em http://www.doxygen.org.

[24] Sametinger, J. and Pomberger, G. (1992). *A hypertext system for literate C++ programming*. Journal of Object Oriented Programming, 4(8):24–29.

[25] Walsh, N. (2002). *Literate Programming in XML,* in Proceedings of XML 2002.

[26] *DocBook.org*, http://www.docbook.org/.

[27] Badros, G. J. (2000). *JavaML: a markup language for Java source code*. Computer Networks (Amsterdam, Netherlands: 1999), 33(1–6):159–177.

[28] Mamas, E. and Kontogiannis, K. (2000). *Towards Portable Source Code Representations Using XML*. In Proceedings of WCRE'00, Brisbane Australia, pages 172–182.

[29] O. M. Group (2005). *XML Metadata Interchange (XMI)*, http://www.omg.org/technology/documents/formal/xmi.htm.

[30] *Wiki*, http://pt.wikipedia.org/wiki/Wiki.

[31] *CamelCase*, http://en.wikipedia.org/wiki/CamelCase.

[32] SnipSnap, wiki e weblog, http://snipsnap.org/space/start.

[33] Aguiar A., David G. and Padilha, M. (2003). *XSDoc: an Extensible Wiki-based Infrastructure for Framework Documentation*. In E. Pimentel, N. R. Brisaboa, and J. Gómez, editors, JISBD, pages 11–24.

[34] *IBM – Rational Rose Enterprise*, http://www-306.ibm.com/software/awdtools/developer/rose/enterprise/.

[35] *IBM – Rational SoDA – Features and Benefits*, http://www-306.ibm.com/software/awdtools/soda/features/index.html.

[36] Freire, Vanessa F. D. and Garcia, Francilene (2002). *Rational SoDA*. Disponível em http://www.dsc.ufcg.edu.br/~patricia/pct/aulasfinais/c17.PDF.

[37] *Microsoft SQL Server: Reporting Services*, http://www.microsoft.com/sql/technologies/reporting/default.mspx.

[38] *Crystal Reports*, http://en.wikipedia.org/wiki/Crystal_Reports.

[39] *Eclipse BIRT Home*, http://www.eclipse.org/birt/phoenix/.

[40] Cockburn, A. (1997). *Using VW staging to clarify spiral development,* in OOPSLA'97 Practitioner's Report, Humans and Technology technical report. Atlanta (GA).

[41] Muller, Claudia and Birn, Lukas (2006). *Wikis for collaborative software documentation,* International Conference on Knowledge Management, Special track on Business Process oriented Knowledge Infrastructures.

[42] *Concurrent Versions System*, http://en.wikipedia.org/wiki/Concurrent_Versions_System.

[43] Microsoft Corporation (2008). *Microsoft Office SharePoint Server*, http://www.microsoft.com/sharepoint/default.mspx.

[44] Aguiar A. (2005). *DocIt! – Agile Documentation of Object-oriented Frameworks*, disponível a partir de http://doc-it.fe.up.pt/aaguiar/space/Teaching/Training+courses/2006/Agile+Project+Management-22nd+May/Area+Reservada/doc-it-talk-handouts.pdf.

[45] Microsoft Corporation (2003). *Microsoft Office 2003 XML Reference Schemas*, http://rep.oio.dk/Microsoft.com/officeschemas/wordprocessingml_article.htm.

[46] Critical Software, S.A. (2007). *Documentation Framework Software Architecture Specification*, CSW-2007-SAS-3436, Internal Report.

Anexo A
Manual de Utilização

A.1. Introdução

Para que as funcionalidades do sistema desenvolvido se tornem mais perceptíveis e simples para o utilizador final torna-se necessário criar um manual de utilização onde estas possam ser ilustradas e descritas.

A apresentação das funcionalidades foi dividida nas referentes ao projecto Documentation Framework e nas que foram adicionadas ou alteradas no âmbito do trabalho descrito neste documento. Estas últimas são descritas em maior pormenor, dada a sua maior relevância nesta fase.

A.2. Descrição das funcionalidades da Documentation Framework

Editar propriedades gerais

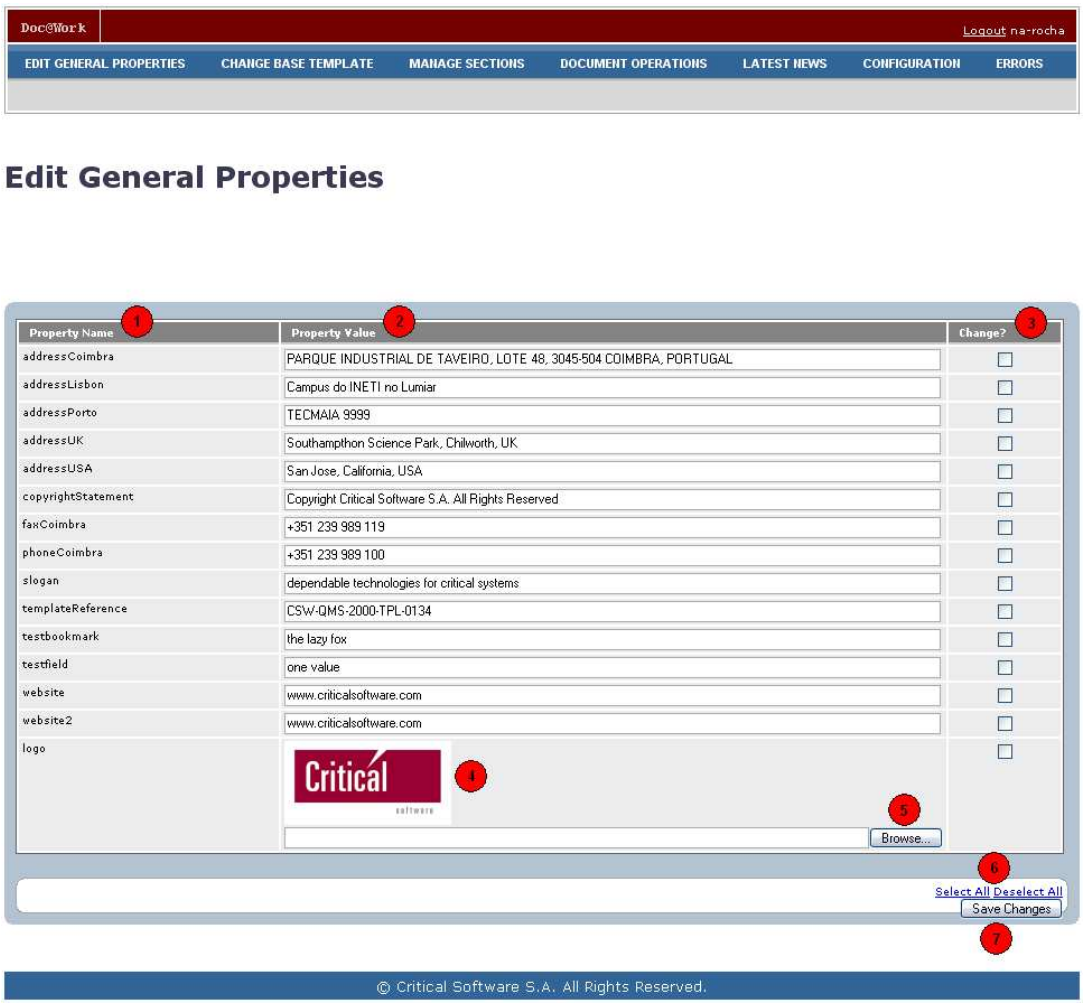

Figura 27 – Funcionalidade de edição de propriedades gerais

1. *Property Name*: indicação do nome das propriedades.
2. *Property Value*: indicação do valor actual das propriedades. Para as propriedades de texto é uma caixa de texto editável, permitindo ao utilizador inserir novos valores.
3. *Change*: *checkbox* que, quando seleccionada, significa que a propriedade respectiva é para ser efectivamente alterada.
4. *Brow*se: permite ao utilizador escolher uma nova imagem para associar à propriedade a partir dos ficheiros existentes no seu computador. Apenas existente em propriedades de imagem.

5. *Select All* e *Deselect All*: forma mais rápida de seleccionar/desseleccionar todas as *checkboxes* da página.
6. *Save Changes*: botão que despoleta a acção de actualização pretendida.

Alterar o *template* base de um *template* específico

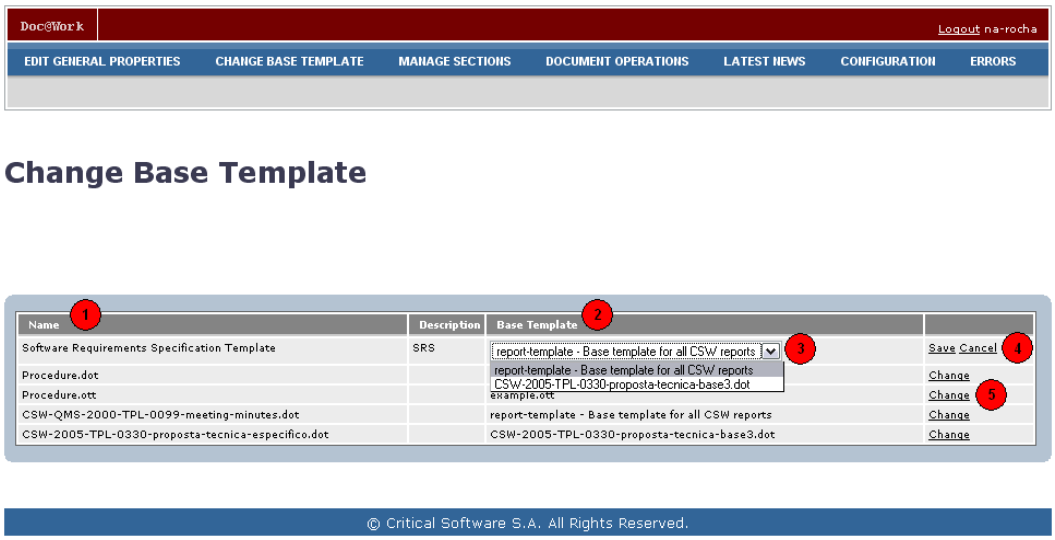

Figura 28 – Funcionalidade de alteração de *templates* base

1. *Name*: nome do ficheiro relativo a cada *template* específico existente no sistema.
2. *Base Template*: indicação do *template* base que se encontra associado a cada *template* específico.
3. *DropDownList* de *templates* base: lista contendo todos os *templates* base existentes no sistema.
4. *Save/Cancel*: botões para guardar na base de dados a alteração do *template* base ou cancelar a operação sem guardar alterações.
5. *Change*: botão para alterar o *template* base associado. Faz com que o texto indicado pelo número 2 se transforme na *dropdown list* indicada pelo número 3. O administrador pode então escolher da lista aquele que pretende que passe a ser o novo *template* base.

Listar erros ocorridos na aplicação

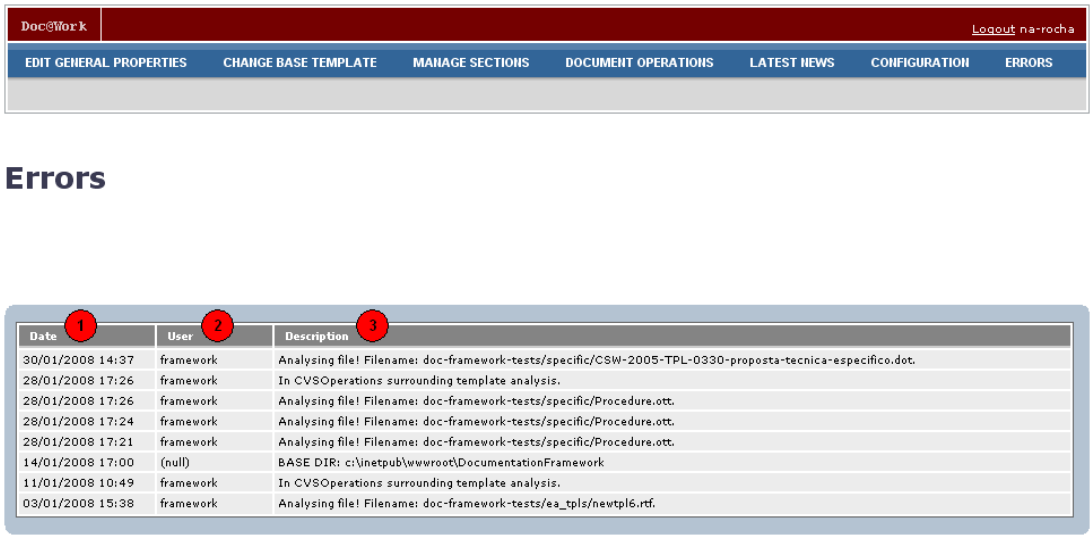

Figura 29 – Funcionalidade de listagem de erros ocorridos na aplicação

1. *Date*: data e hora em que o erro ocorreu.
2. *User*: utilizador autenticado na altura do erro.
3. *Description*: descrição sumária do erro ocorrido.

Listar as últimas alterações efectuadas aos *templates* através da aplicação

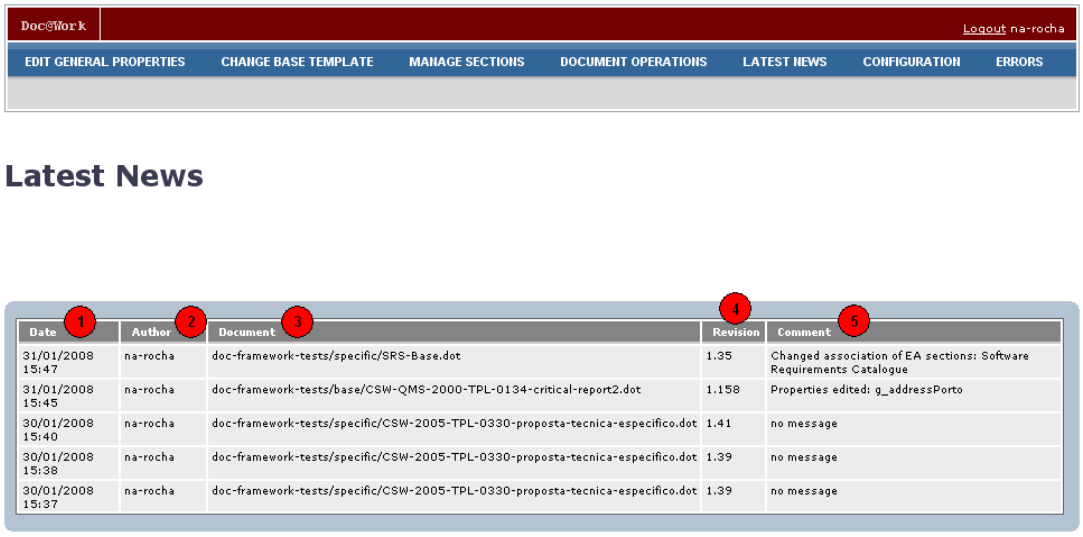

Figura 30 – Funcionalidade de listagem das últimas alterações efectuadas

1. *Date*: data e hora em que a alteração foi efectuada.

2. *Author*: autor responsável pela alteração.
3. *Document*: documento no qual a alteração foi efectuada.
4. *Revision*: revisão do documento após a alteração.
5. *Comment*: descrição sumária da alteração efectuada.

Configurar o número de dias de alterações visíveis

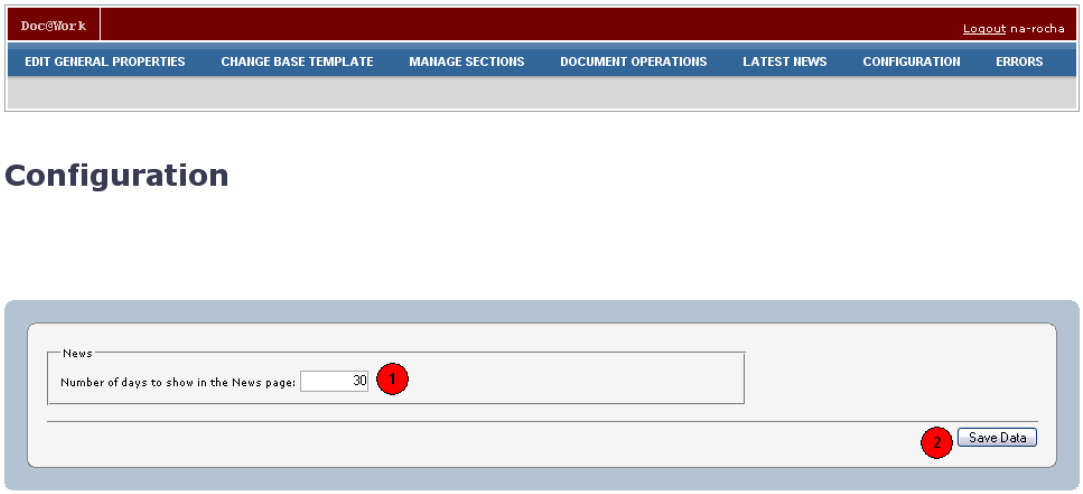

Figura 31 – Funcionalidade de configuração de dias de alterações visíveis

1. *Number of days to show in the News page*: número de dias visualizados na página relativa a alterações efectuadas.
2. *Save Data*: botão para guardar o valor inserido.

A.3. Descrição das funcionalidades relacionadas com integração de ferramentas

Associar *templates* de Enterprise Architect a secções de documentos

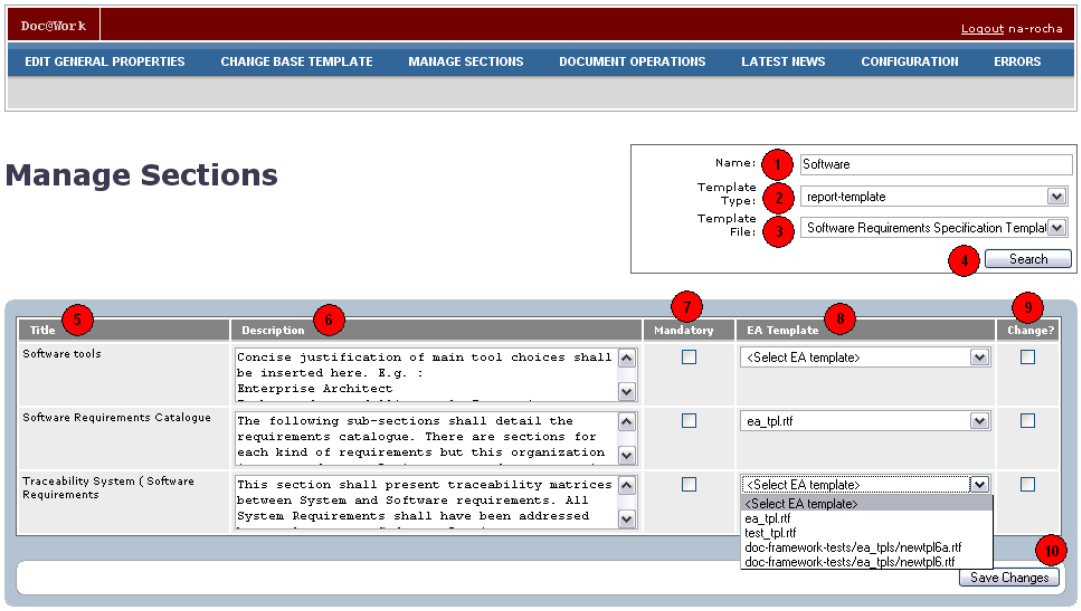

Figura 32 – Funcionalidade de gestão de secções de documentos

1. *Name*: caixa de texto para inserir caracteres que devem estar presentes no nome das secções a listar. Caso esteja vazia considera-se que este parâmetro não deve ser incluído na pesquisa.
2. *Template Type*: apresenta uma lista dos *templates* base existentes e permite apresentar as secções que são apenas de um *template* base.
3. *Template File*: lista para escolher *templates* específicos. Se tiver sido escolhido um *template* base na lista anterior, apenas aparecerão nesta lista os *templates* específicos que o têm como base. Caso nenhum *template* base tenha sido seleccionado, serão listados aqui todos os *templates* específicos existentes.
4. *Search*: inicia a pesquisa e preenche a listagem de secções com aquelas que satisfazem os critérios de pesquisa.
5. *Title*: título de cada secção.
6. *Description*: descrição de cada secção existente na base de dados. Esta é lida inicialmente dos ficheiros físicos que são analisados quando são inseridos ou modificados através do repositório CVS.

7. *Mandatory*: *checkbox* que indica se a secção está definida como sendo obrigatória ou não.
8. *EA Template*: lista com os *templates* de conteúdos Enterprise Architect existentes no sistema. Caso a secção já tenha algum *template* associado, este é mostrado como estando seleccionado.
9. *Change*: *checkbox* que, quando seleccionada, significa que a secção respectiva é para ser efectivamente alterada.
10. *Save Changes*: guarda as alterações efectuadas.

Compor documentos com dados de Enterprise Architect

A composição de documentos presente no projecto Documentation Framework foi actualizada tendo em conta as novas necessidades verificadas.

As figuras 33, 34, 35 e 36 ilustram todo o processo envolvido.

O processo começa com a escolha do projecto ao qual o documento irá estar associado, importante para a manutenção das opções de geração. Posteriormente, o utilizador tem de escolher o tipo de documento pretendido para que sejam carregadas na interface as secções e propriedades respectivas.

Ao escolher as secções pretendidas, o sistema detecta se alguma delas pode conter dados provenientes de Enterprise Architect. Em caso afirmativo, é mostrada uma *tab*, com o nome da secção em questão, que irá conter uma representação do *template* de dados associado. Este contém uma estrutura hierárquica onde o utilizador pode escolher o que pretende para incluir na secção actual, mediante um mecanismo de *checkboxes* semelhante ao existente para as secções do documento (Figura 35).

Como pode ser consultado na Figura 34, também é possível preencher algumas propriedades previamente definidas presentes no documento, aumentando o grau de personalização do mesmo.

Após todo o processo de configuração, o sistema gera o documento pretendido, disponibilizando-o ao utilizador final mediante a interface ilustrada na Figura 36.

As figuras incluídas a seguir contêm, quando necessário, uma legenda onde são identificados os locais de principal relevo nas interfaces desenvolvidas.

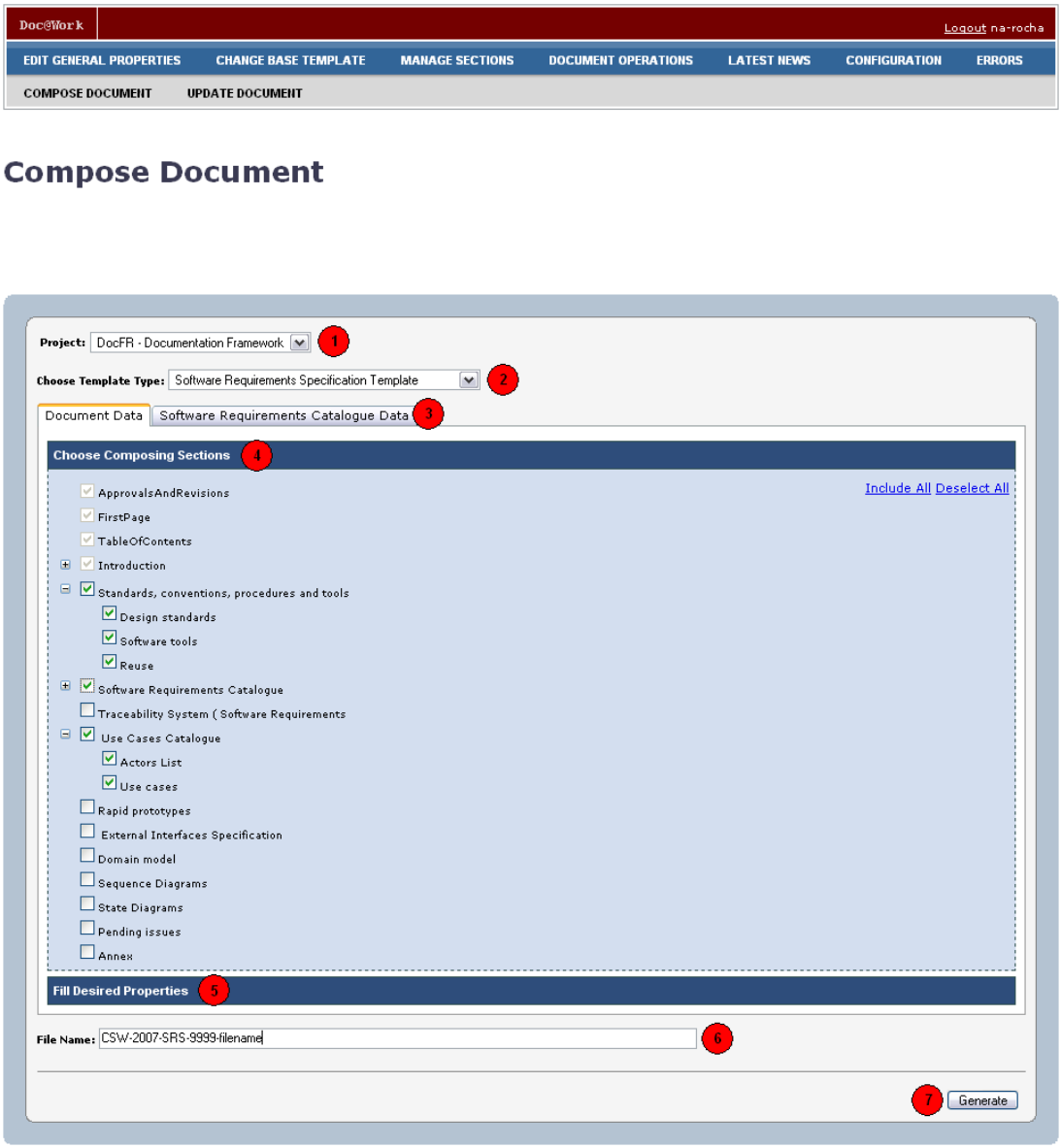

Figura 33 – Composição de documentos (passo 1 – escolha de secções do documento)

1. *Project*: lista com todos os projectos existentes no sistema.
2. *Choose Template Type*: lista com todos os *templates* que podem ser utilizados para composição de documentos.
3. *Tab Container* onde vão aparecendo as *tabs* relativas a cada secção passível de conter dados de Enterprise Architect. A *tab Document Data* está sempre presente e inclui os dados para configuração geral do documento a produzir.
4. *Choose Composing Sections*: local onde são visualizadas as secções do *template*. Estas são visualizadas hierarquicamente.
5. *Fill Desired Properties*: local para preenchimento das propriedades do *template*.

6. *File Name*: caixa de texto para inserção do nome que o novo documento terá. Tem obrigatoriamente de ser preenchida.
7. *Generate*: inicia o processo de geração do novo documento com os dados existentes na interface.

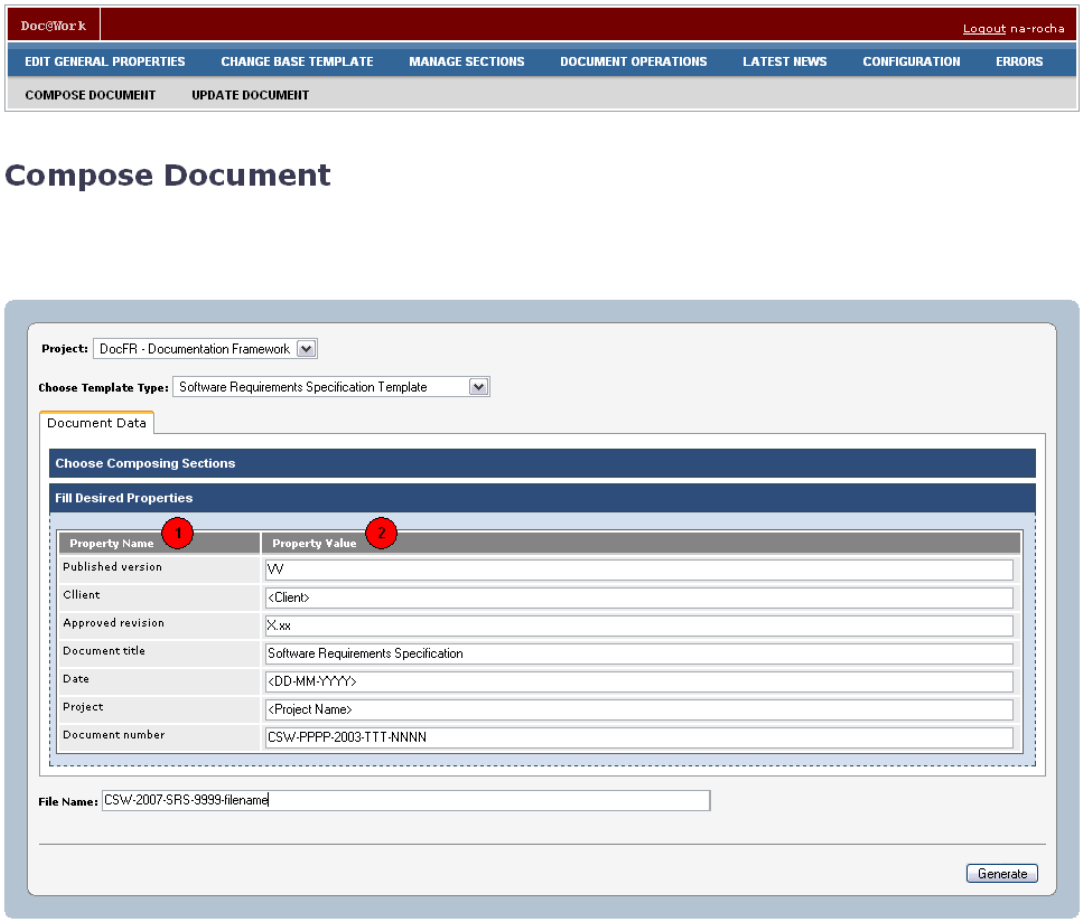

Figura 34 – Composição de documentos (passo 2 – preenchimento de propriedades)

1. *Property Name*: nome das propriedades existentes no *template*.
2. *Property Value*: caixas de texto para inserção dos valores pretendidos para cada propriedade. Inicialmente encontram-se preenchidas com os valores por defeito existentes no *template*.

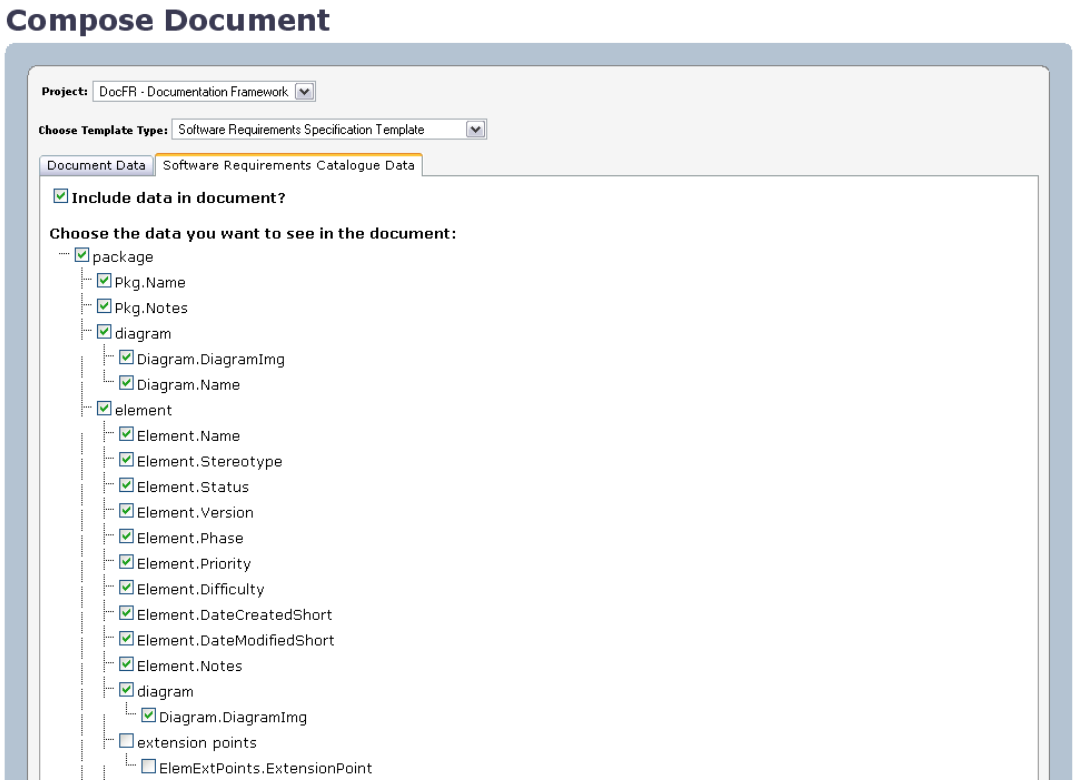

Figura 35 – Composição de documentos (passo 3 – escolha de conteúdos de EA)

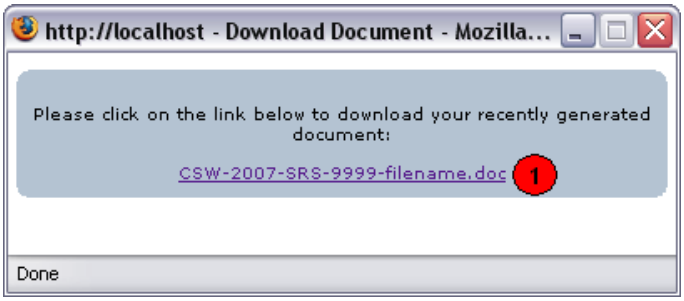

Figura 36 – Composição de documentos (passo 4 – gravação do documento final)

Actualizar documentos com dados de Enterprise Architect

O processo de actualização de documentos tem algumas semelhanças com o processo de geração dos mesmos, em especial na sua parte final.

Inicialmente, o utilizador deve escolher o projecto ao qual o documento a actualizar se refere e também a referência deste, através do nome preenchido aquando da sua geração. Este processo tem como objectivo que seja possível visualizar últimas escolhas efectuadas ao nível de conteúdos de secções de Enterprise Architect, uma vez que estas foram previamente guardadas na base de dados do projecto.

Após estes passos, a interface mostra uma lista das secções possíveis de actualizar. À medida que o utilizador as selecciona, um esquema em tudo igual ao presente na composição de documentos permite a visualização de *tabs* com os seus conteúdos. Cada uma destas apresenta a estrutura hierárquica correspondente, com as opções gravadas já seleccionadas. Desta forma, o utilizador consegue saber que conteúdos estão presentes no documento, sendo também possível a escolha de uma nova configuração ou a manutenção da já existente (Figura 38).

O passo seguinte é o *upload* do documento físico para o servidor (Figura 39). Este é um passo necessário nesta altura, em que o sistema não tem o controlo total sobre os documentos gerados (este controlo é partilhado com os utilizadores que geram os documentos e os editam localmente no seu computador). O nome do ficheiro físico deve ser igual ao escolhido na lista disponibilizada ao utilizador.

Após o *upload* ter sido efectuado com sucesso, surge um novo botão na interface – identificado com o número 1 na Figura 40 – que permite dar início à actualização dos dados pretendidos. No final, o documento actualizado é retornado ao utilizador mediante a interface já referida na composição de documentos e ilustrada na Figura 36.

As figuras incluídas a seguir contêm, quando necessário, uma legenda onde são identificados os locais de principal relevo nas interfaces desenvolvidas.

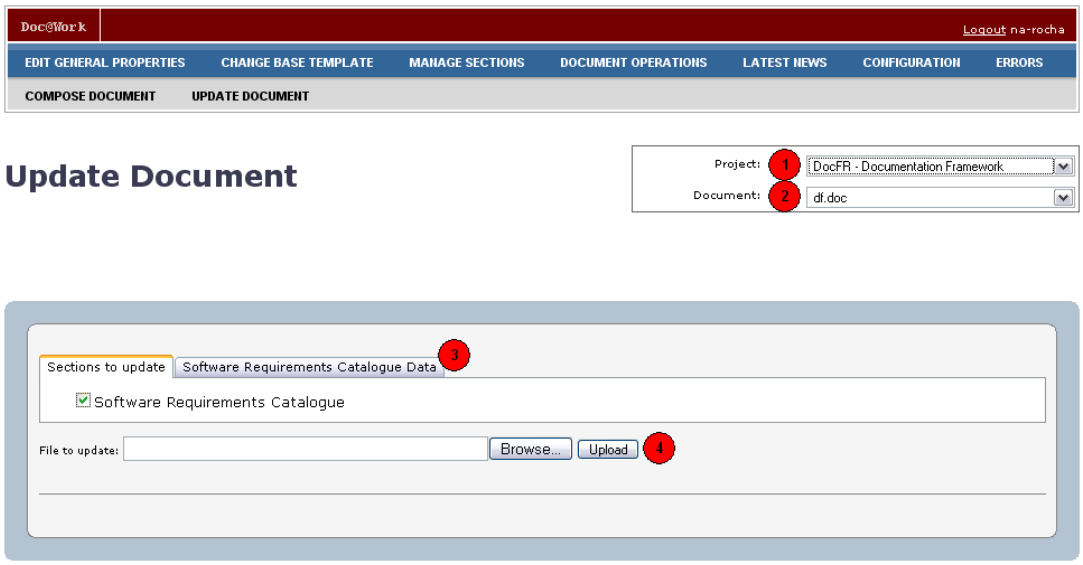

Figura 37 – Actualizaçã de documentos (passo 1 – escolha do documento a actualizar)

1. *Project*: lista com os projectos associados ao utilizador autenticado.
2. *Document*: lista com os documentos associados ao projecto escolhido pelo utilizador.
3. *Tab Container* onde vão aparecendo as *tabs* relativas a cada secção passível de conter dados de Enterprise Architect. A *tab 'Sections to update'* está sempre presente e inclui as secções referidas, necessitando o utilizador de seleccionar as que pretende efectivamente actualizar.
4. *Upload*: botão que permite realizar o *upload* do documento a actualizar para o servidor.

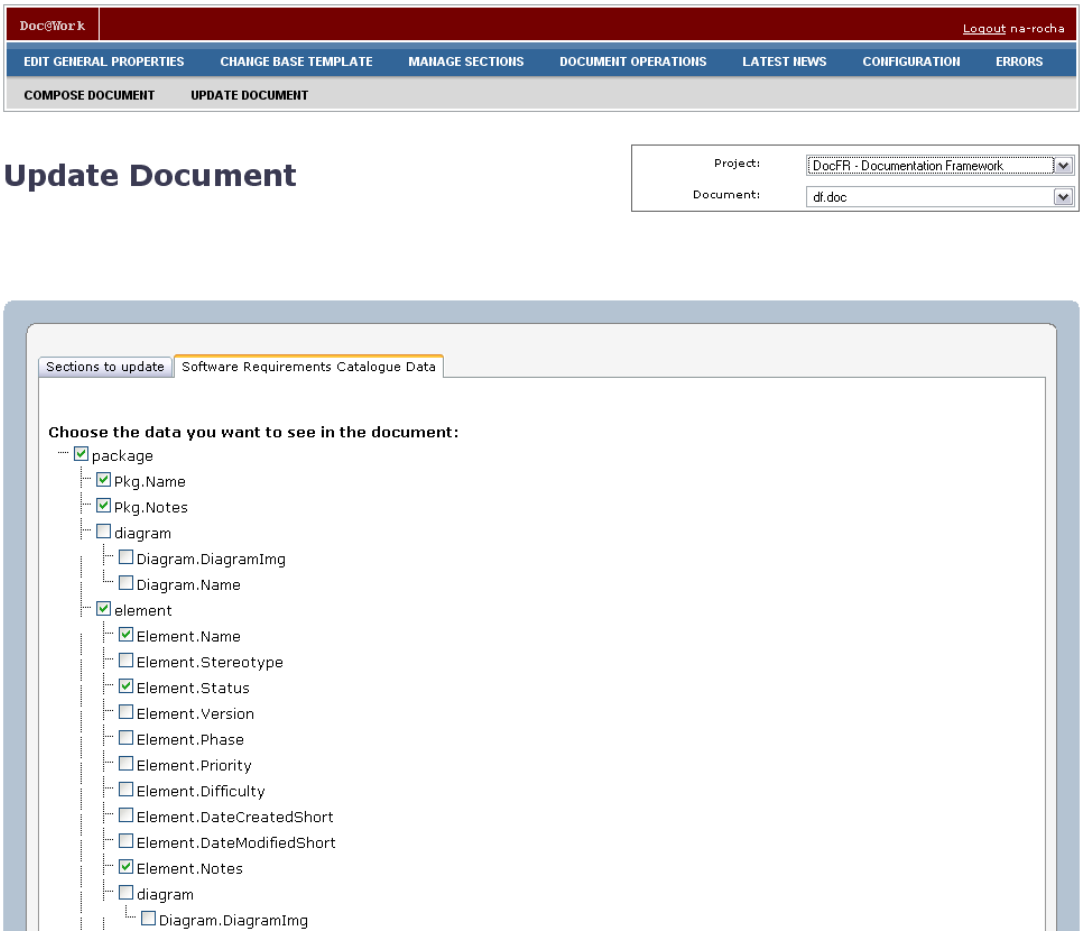

Figura 38 – Actualização de documentos (passo 2 – escolha dos conteúdos a actualizar)

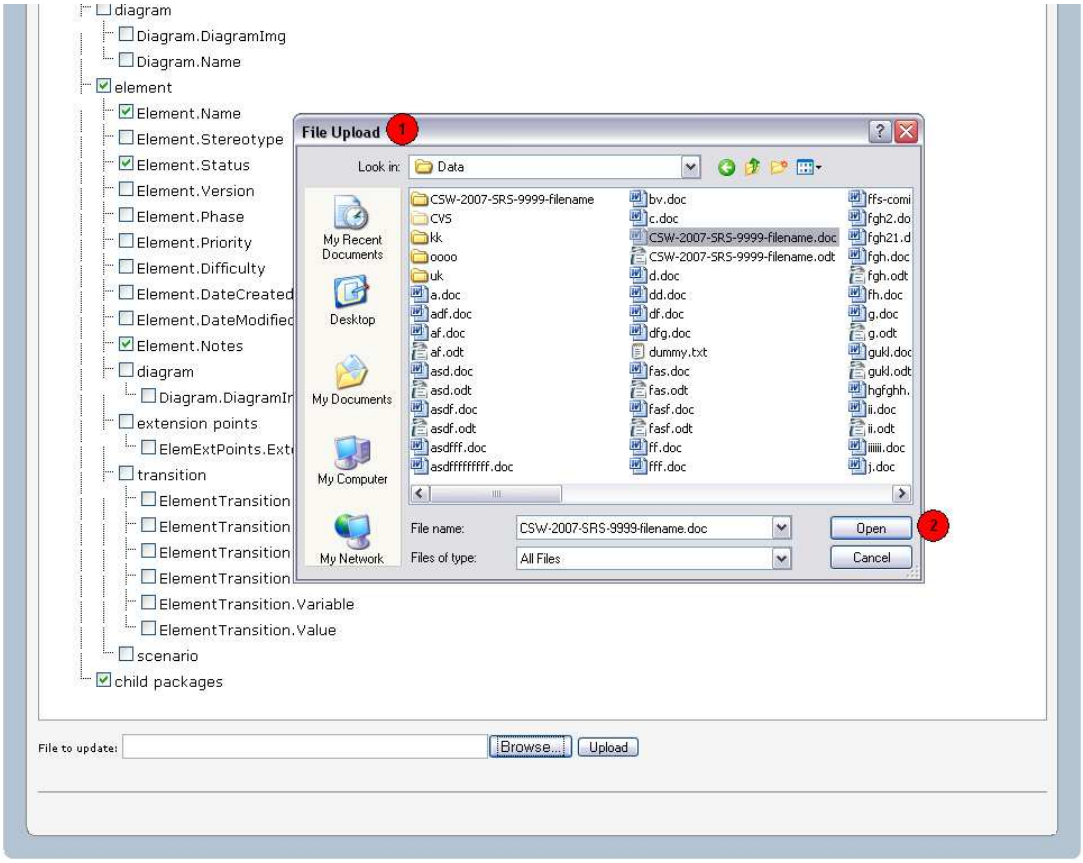

Figura 39 – Actualização de documentos (passo 3 – upload do documento)

1. *File Upload*: ecrã que possibilita a escolha do documento pretendido para actualização.
2. *Open*: botão para realizar a escolha efectiva do documento a actualizar.

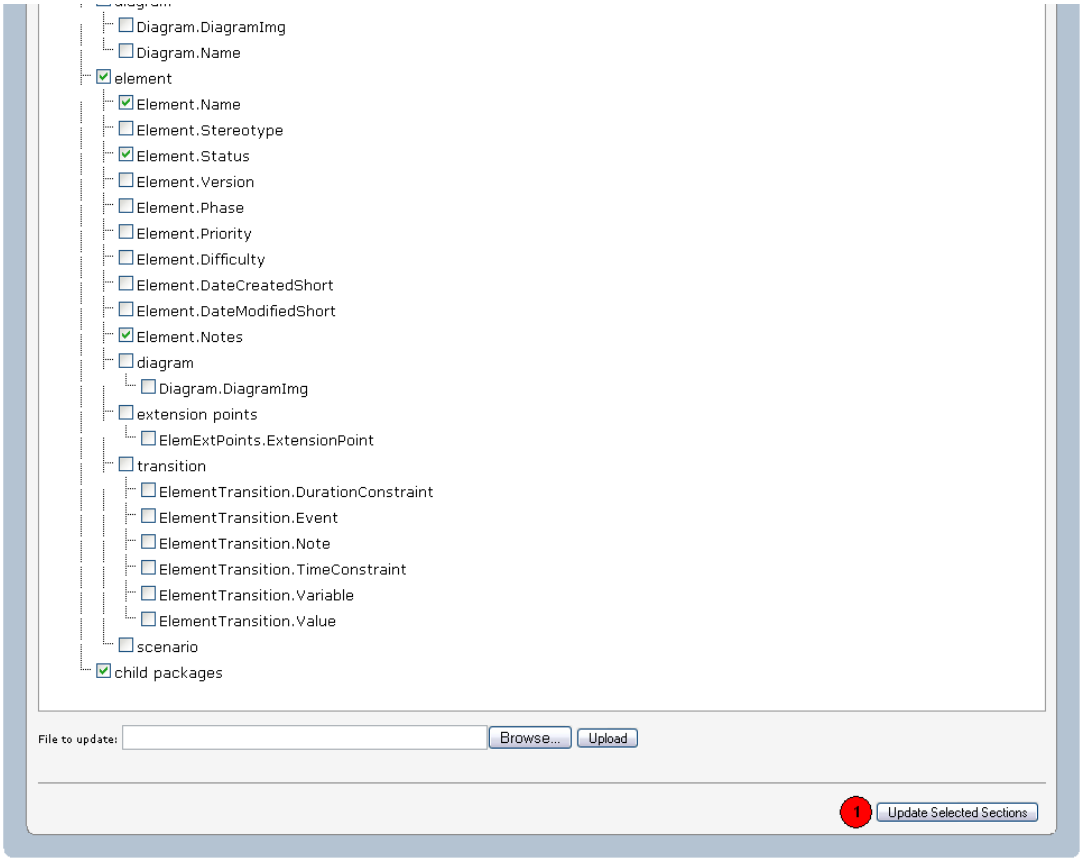

Figura 40 – Actualização de documentos (passo 4 – começo do processo de actualização)

Anexo B
Modelo de dados

Neste anexo está incluído o modelo de dados global do sistema desenvolvido, com as tabelas criadas no âmbito do projecto Documentation Framework e as criadas no âmbito do projecto descrito no presente documento.

Uma descrição mais pormenorizada das tabelas e colunas envolvidas pode ser consultada em [46].

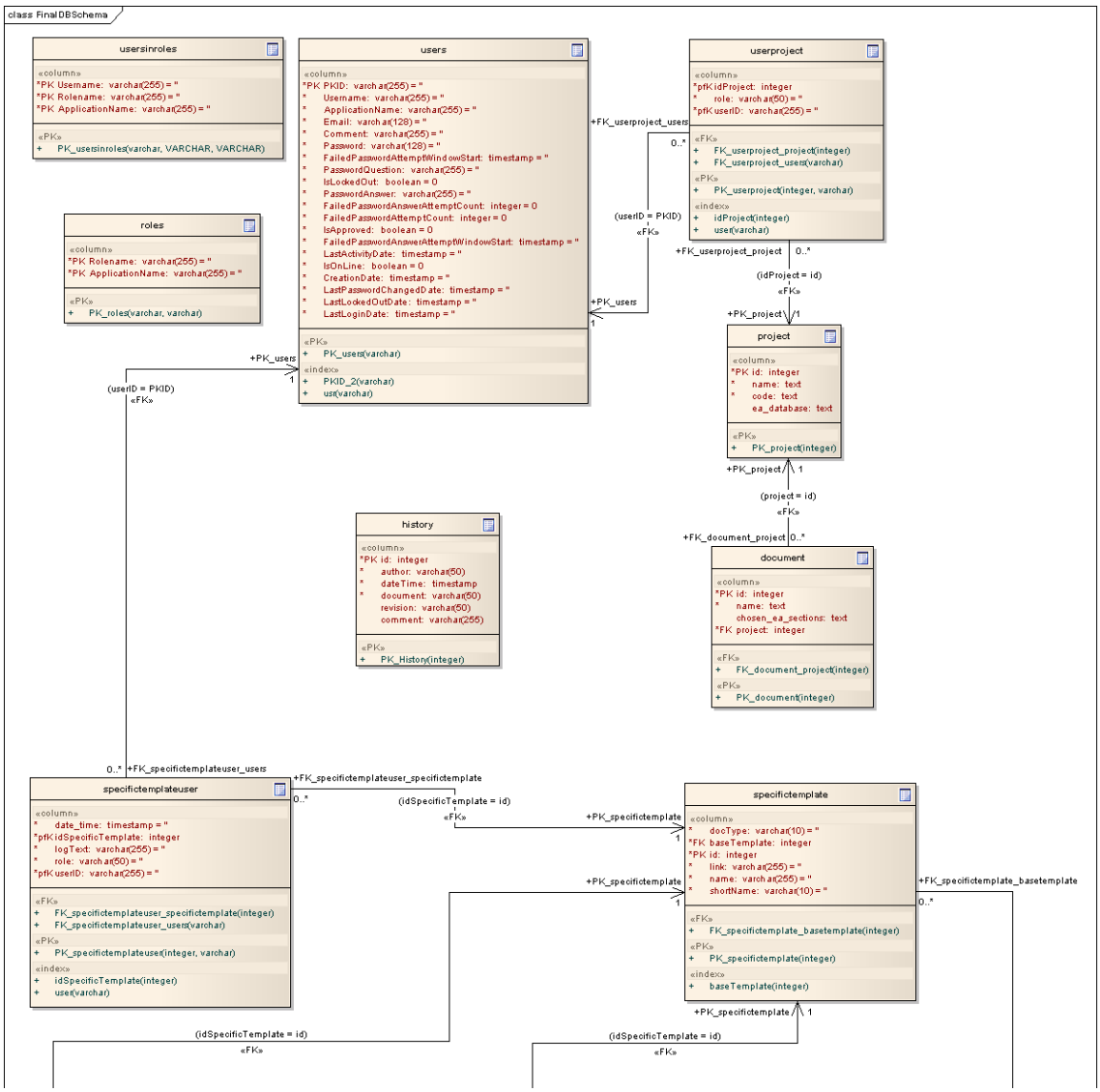

Figura 41 – Modelo de dados global (parte 1)

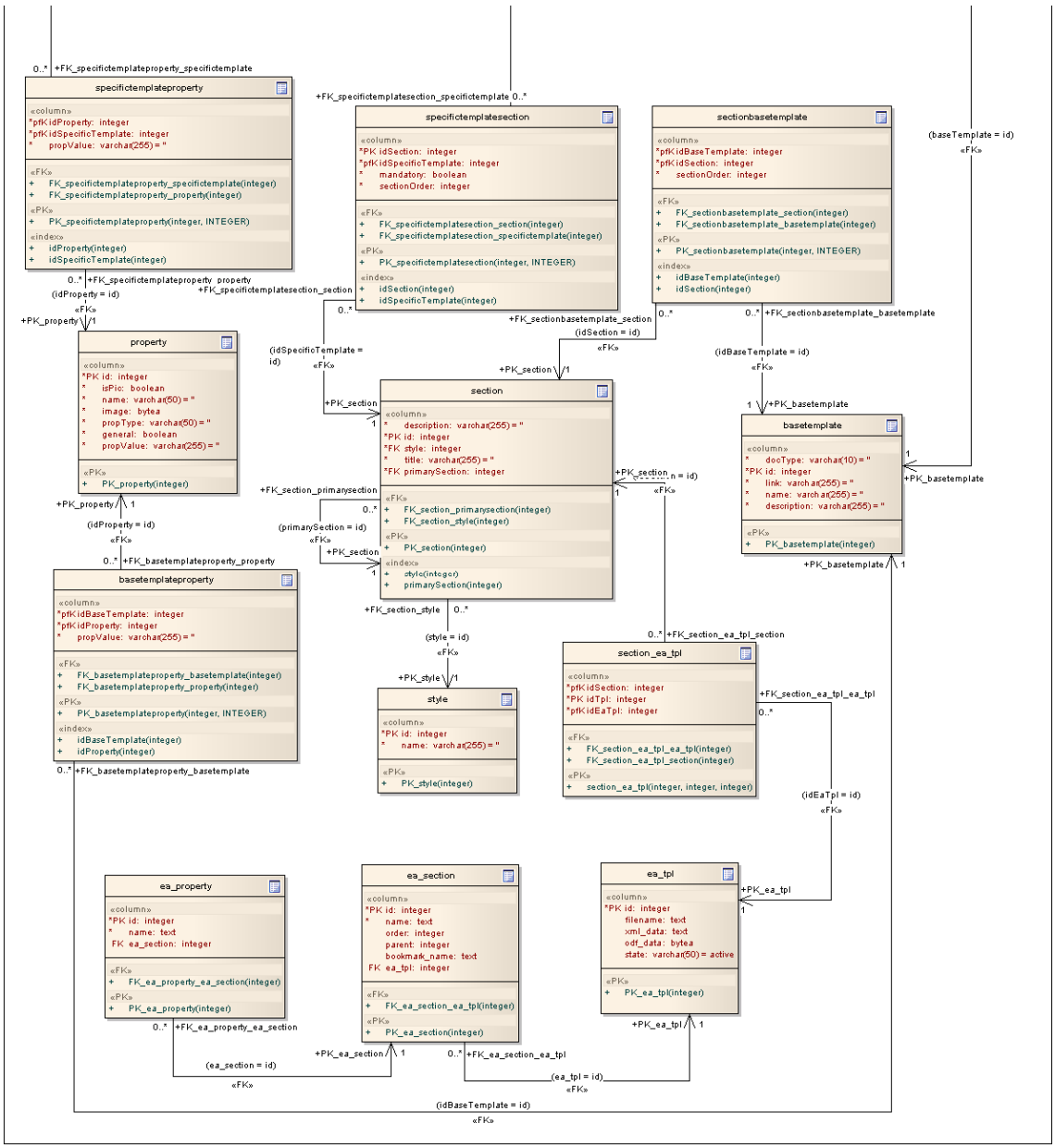

Figura 42 – Modelo de dados global (parte 2)

Anexo C

Mapeamento entre propriedades de *templates* e base de dados Enterprise Architect

Neste anexo encontra-se apresentado o mapeamento construído para fazer corresponder às propriedades possíveis de encontrar nos *templates* de Enterprise Architect as respectivas tabelas e colunas da base de dados.

Devido à extensão deste mapeamento, apenas são aqui incluídos os dados considerados como tendo uma utilização mais frequente na documentação. O mapeamento completo pode ser consultado em [46].

Tabela 7 – Mapeamento entre propriedades de *templates* EA e base de dados

Secção	**Propriedade**	Tabela**/Coluna**
Model		Table t_package where t_package.ParentID = 0
Model/Glossary		Table t_glossary
	Meaning	t_glossary.Meaning
	Team	t_glossary.Team
	Type	t_glossary.Type
Model/Issues		Table t_issues
	Date	t_issues.IssueDate
	Description	t_issues.Notes
	Issue	t_issues.Issues
	Owner	t_issues.Owner
	Priority	t_issues.Priority
	ResolutionComments	t_issues.Resolution
	ResolvedBy	t_issues.Resolver
	ResolvedDate	t_issues.DateResolved
	Status	t_issues.Status
Model/Tasks		Table t_tasks
	ActualTime	t_tasks.ActualTime
	AssignedTo	t_tasks.AssignedTo
	EndDate	t_tasks.EndDate

	Description	t_tasks.NOTES
	History	t_tasks.History
	Owner	t_tasks.Owner
	Percent	t_tasks.Percent
	Phase	t_tasks.Phase
	Priority	t_tasks.Priority
	StartDate	t_tasks.StartDate
	Status	t_tasks.Status
	Task	t_tasks.Name
	TotalTime	t_tasks.TotalTime
	Type	t_tasks.TaskType
Package		Table t_package where t_package.ParentID != 0
	Abstract	t_object.Abstract from t_object, t_package where t_package.Package_ID = t_object.PDATA1 and t_object.Object_Type = 'Package'
	Alias	t_object.Alias from t_object, t_package where t_package.Package_ID = t_object.PDATA1 and t_object.Object_Type = 'Package'
	DateCreated	t_package.CreatedDate
	DateModified	t_package.ModifiedDate
	Flags	t_package.Flags
	Name	t_package.Name
	Notes	t_package.Notes
	ParentPackage	t_package.Parent_ID
	Phase	t_object.Phase from t_object, t_package where t_package.Package_ID = t_object.PDATA1 and t_object.Object_Type = 'Package'
	Scope	t_object.Scope from t_object, t_package where t_package.Package_ID = t_object.PDATA1 and t_object.Object_Type = 'Package'
	Status	t_object.Status from t_object, t_package where t_package.Package_ID = t_object.PDATA1 and t_object.Object_Type = 'Package'
	Stereotype	t_object.Stereotype from t_object, t_package where t_package.Package_ID = t_object.PDATA1 and t_object.Object_Type = 'Package' (first element of the comma separated list)
	Version	t_object.Version from t_object, t_package where

		t_package.Package_ID = t_object.PDATA1 and t_object.Object_Type = 'Package'
Package/Diagram		Table t_diagram where t_diagram.Package_ID = currPack.ID
	DiagramImg	INEXISTENT
	Author	t_diagram.Author
	Name	t_diagram.Name
	Notes	t_diagram.Notes
	Type	t_diagram.Type
Package/Element		Table t_object where t_object.Package_ID = currPkg.ID
	Abstract	t_object.Abstract
	Alias	t_object.Alias
	Author	t_object.Author
	Complexity	t_object.Complexity
	DateCreated	t_object.CreatedDate
	DateModified	t_object.ModifiedDate
	Difficulty	Only Applies to Requirement, Change and Issue types, otherwise ignored if (requirement) t_objectrequires.Difficulty else if(change OR issue) t_objectproblems.Severity
	Multiplicity	t_object.Multiplicity
	Name	t_object.Name
	Notes	t_object.Note
	Phase	t_object.Phase
	Priority	Only Applies to Requirement, Change and Issue types, otherwise ignored if (requirement) t_objectrequires.Priority else if(change OR issue) t_objectproblems.Priority
	Scope	t_object.Scope
	Status	t_object.Status
	Stereotype	t_object.Stereotype (first element of the comma separated list)
	Type	t_object.Type

	Version	t_object.Version
Package/Element/ Requirement		Table t_objectrequires where t_objectrequires.ParentID = currElement.ID
	Difficulty	t_objectrequires.Difficulty
	LastUpdate	t_objectrequires.LastUpdate
	Name	t_objectrequires.Requirement
	Notes	t_objectrequires.Notes
	Priority	t_objectrequires.Priority
	Stability	t_objectrequires.Stability
	Status	t_objectrequires.Status
	Type	t_objectrequires.ReqType
Package/Element/External Requirements		Table t_object where t_object.Object_Type = 'Requirement' and go through the connector (Source → Element; Target → Ex Req)
	Difficulty	t_object.PDATA3
	LastUpdate	t_object.ModifiedDate
	Name	t_object.Name
	Notes	t_object.Notes
	Priority	t_object.PDATA2
	Status	t_object.Status
Package/Element/Test		Table t_objecttests where t_objecttests.Object_ID = currElement.ID
	AcceptanceCriteria	t_objecttests.AcceptanceCriteria
	Class	t_objecttests.TestClass
	CheckedBy	t_objecttests.CheckedBy
	Input	t_objecttests.InputData
	Name	t_objecttests.Test
	Notes	t_objecttests.Notes
	RunBy	t_objecttests.RunBy
	RunDate	t_objecttests.DateRun
	Status	t_objecttests.Status
	TestResults	t_objecttests.Results
	Type	t_objecttests.TestType
Package/Element/Attribute		Table t_attribute where t_attribute.Object_ID = currElement.ID
	Alias	t_attributes.Style
	Container	t_attributes.Container

	Default	t_attributes.Default
	Lenght	t_attributes.Lenght
	LowerBound	t_attributes.LowerBound
	Multiplicity	t_attributes.[t_attributes.LowerBound .. t_attributes.UpperBound]
	Name	t_attributes.Name
	Notes	t_attributes.Notes
	Stereotype	t_attributes.Stereotype (first element of the comma separated list)
	Type	t_attributes.Type
	UpperBound	t_attributes.UpperBound
Package/Element/Scenario		Table t_objectscenarios where t_objectscenarios.Object_ID = currElement.ID
	Notes	t_objectscenarios.Notes
	Scenario	t_objectscenarios.Scenario
	Type	t_objectscenarios.ScenarioType
	Weight	t_objectscenarios.EValue
Package/Element/Diagram		Table t_diagram where t_object.Diagram_ID = t_diagram.Diagram_ID and t_object.Object_ID = currElement.ID
	Rest of Fields	Repetition of Package/Diagram mapping
Package/Element/Method		Table t_operation where t_operation.Object_ID = currElement.ID
	Abstract	t_operation.Abstract
	Behaviour	t_operation.Behaviour
	Code	t_operation.Code
	Name	t_operation.Name
	Notes	t_operation.Notes
	Scope	t_operation.Scope
	Stereotype	t_operation.Stereotype (first element of the comma separated list)
Package/Element/Method/ Parameter		Table t_operationparams where t_operationparams.OperationID = currMethod.ID
	Const	t_operationparams.Const
	Default	t_operationparams.Default
	Kind	t_operationparams.Kind
	Name	t_operationparams.Name
	Notes	t_operationparams.Notes

	Position	t_operationparams.Pos
	Type	t_operationparams.Type
Package/Element/ ChildElements		Table t_object where t_object.Parent_ID = currElement.ID
	Rest of fields	Repetition of Element fields mapping
Package/ChildPackages		Table t_package where t_package.Parent_ID = currPkg.ID
	Rest of fields	Repetition of Package fields mapping