Universidade Nova de Lisboa

Faculdade de Ciências e Tecnologia

Departamento de Informática

Cenários Visuais: Rastreio de Requisitos, Documentação e Animação para Sistemas Legados

Vítor Emanuel Gomes Gouveia

(Licenciado)

Dissertação para a obtenção do Grau de Mestre em

Engenharia Informática

Julho de 2008

[Esta página foi intencionalmente deixada em branco]

Dissertação realizada sob a orientação do

Professor Doutor Fernando Brito e Abreu

Professor Auxiliar do Departamento de Informática

Faculdade de Ciências e Tecnologia

Universidade Nova de Lisboa

[Esta página foi intencionalmente deixada em branco]

Resumo

A rastreabilidade dos requisitos de software nos sistemas legados é um problema recorrente na maioria das empresas. Existem várias causas para este problema, mas as razões principais estão na falta de documentação e unem-se à inexistência de um suporte adequado que permita manter a rastreabilidade entre os requisitos, o desenho e o código fonte de um sistema de software. Nos casos em que existe alguma documentação do sistema esta apresenta-se normalmente na forma de documentos de texto não estruturado, escritos em língua natural. A falta de um standard para representar a informação, torna mais difícil perceber a estrutura e a complexidade de um sistema legado.

Esta dissertação vem tentar mitigar o problema apresentado, da falta de rastreabilidade entre os requisitos e a implementação, propondo uma abordagem inovadora designada por Cenários Visuais. A técnica proposta tenta mitigar o problema que existe da rastreabilidade do software e vai facilitar a compreensão do sistema por parte de todos os envolvidos, desde a sua criação até às fases posteriores do seu ciclo de vida. Os cenários visuais são filmagens da execução de cenários, onde é possível encontrar informação adicional sobre as iterações entres os componentes de um sistema. Os cenários visuais são uma representação dinâmica e sincronizada entre duas perspectivas de um sistema de software. A primeira corresponde à perspectiva do utilizador do sistema (caixa preta) e é constituída por um filme descrevendo cada cenário de um caso de utilização. A segunda corresponde à perspectiva do construtor de sistemas de software (caixa branca) e é concretizada por um diagrama de sequência (UML). O processo de criação de um cenário visual servirá de suporte para a criação automática de documentação de sistemas legados e facilitará a validação dos requisitos de um sistema de software.

Em suma, espera-se que com esta dissertação a compreensão de um sistema de software legado possa vir a ser melhorada.

Palavras-Chave:

Rastreabilidade dos requisitos; Sistemas legados; Ligações de rastreabilidade; Cenário; Automático

[Esta página foi intencionalmente deixada em branco]

Abstract

Requirements traceability in legacy systems is a recurrent problem faced by many companies. There are several sources for this problem, but the main reasons are the lack of documentation and the limited support for maintaining the traceability links among requirements, design and source code. Even when some documents exist, they are usually written in natural language and presented in an unstructured way. The lack of a standard for representing the information makes it more difficult to understand the structure and complexity of a legacy system.

This dissertation tries to mitigate the presented problem, the missing of traceability links between requirements and the implementation of a system, through the introduction of an innovative approach called Visual Scenarios. The proposed technique tries to mitigate the problem of software traceability and helps the comprehension of a system by all practitioners involved, since its creation until the end of the life cycle. Visual scenarios are footages of scenarios where we can find additional information about all interactions between system components. Visual scenarios are a dynamic and synchronized representation between two perspectives from a software system. The first one corresponds to the system user perspective or black-box perspective. In a nutshell, is a movie showing user interactions during an execution of a scenario from a use case. The second one corresponds to the programmer perspective or white-box perspective and is realized by an UML animated sequence diagram. The process of creating visual scenarios will support the creation of automatic documentation for legacy systems and is expected to help the requirements validation.

In conclusion, with this dissertation we expect to improve the comprehension of a legacy system.

Keywords:

Requirements Traceability; Legacy Systems; Traceability Links; Scenario; Automated

[Esta página foi intencionalmente deixada em branco]

Índice

Índice de figuras

Índice de tabelas

Capítulo 1

Introdução

Conteúdo

1. Introdução

1.1. Introdução Geral

Hoje em dia é normal ouvirmos falar da importância que as tecnologias da informação têm para a sociedade actual. Os computadores e o seu software aparecem nas mais diversas áreas de negócio, desempenhando papéis de grande relevo. Assim sendo, a criação de software tornou-se um negócio rentável para muitas empresas actuais. Claro está, que todos os projectos de software são iniciados pela identificação de uma nova necessidade no mundo dos negócios [1]. As constantes alterações que se fazem sentir no nosso contexto actual levam muitas vezes a termos a necessidade de corrigir um defeito numa aplicação já existente, expandir o sistema actual, criar um novo produto, serviço ou sistema.

Desenvolver software não é uma tarefa fácil e desenvolvê-lo com qualidade é ainda mais difícil [2]. Quando se começa alguma coisa de novo é importante que seja algo cuidadosamente pensado. Assim sendo, torna-se conveniente recorrer a modelos que representem aquilo que irá ser desenvolvido. Um modelo consiste na interpretação de um dado domínio de um problema, segundo uma determinada estrutura de conceitos, criando uma abstracção da realidade. A técnica da modelação é bastante utilizada na generalidade das áreas de engenharia, permitindo a partilha de conhecimentos entre os diferentes grupos que desenvolvem um projecto (técnicos e não técnicos). À medida que o software e o hardware foram evoluindo, também as linguagens e paradigmas de programação o foram. Um paradigma é um modelo interpretativo duma realidade. Do mesmo modo, um paradigma de programação é uma conceptualização da computação. Actualmente o paradigma dominante no desenvolvimento de software é o orientado aos objectos.

Para ajudar o processo de desenvolvimento de software orientado aos objectos foi proposta a linguagem UML (*Unified Modeling Language*) [3]. Esta nasceu depois de um esforço de unificação de um grande número de outras linguagens de modelação, preexistentes para o mesmo paradigma. A UML é uma linguagem que utiliza uma notação padrão para especificar, construir, visualizar e documentar sistemas de informação orientados por objectos. Pela abrangência e simplicidade dos conceitos utilizados, ela facilita o desenvolvimento de sistemas de informação, permitindo integrar os aspectos de natureza organizacional que constituem o negócio e os elementos de natureza tecnológica dos mesmos. Pelo facto de utilizar um conjunto de símbolos padrão, a UML funciona como um meio de comunicação entre os diversos elementos envolvidos no processo, como os peritos do domínio, gestores e membros da equipa de desenvolvimento. A linguagem UML é constituída por um conjunto de modelos, construídos com base num conjunto de diagramas.

Na indústria, infelizmente, pouco se aproveitou do UML e das suas potencialidades. De todos os diagramas que a linguagem UML disponibiliza (13 na versão 2 do UML) apenas dois são recorrentemente utilizados: os diagramas de casos de utilização [4] e os diagramas de classes [3]. Isto pode ser visto até numa simples pesquisa feita no motor de busca *Google* [5], onde uma procura por “casos de utilização” devolve quase sete vezes mais ocorrências, que uma pesquisa relativa a outro diagrama da mesma linguagem.

O diagrama de casos de utilização permite representar as interacções dos actores (quem faz despoletar uma acção) com o sistema de software. Estes diagramas são normalmente utilizados para representar o levantamento de requisitos de um sistema. Os requisitos de um sistema são funcionalidades que este deve possuir. Ou seja, características (serviços) que devem ser disponibilizadas a um utilizador [6]. Os requisitos são necessários, não só para construir sistemas de raiz, mas também sempre que é necessário reestruturar ou simplesmente fazer evoluir sistemas já existentes, muitas vezes chamados de **sistemas legados**. Na maior parte das vezes esses sistemas são pouco extensíveis e com pouca ou nenhuma documentação disponível [7]. Os requisitos dividem-se em funcionais e não funcionais. Para os identificar adequadamente é habitual aplicar um conjunto de técnicas, de modo a obter a percepção detalhada das funcionalidades que o sistema deverá fornecer. Dentro destas técnicas podemos encontrar reuniões, entrevistas, questionários ou a elaboração de pequenos

protótipos [2]. Para uma melhor compreensão de um caso de utilização é normal que este seja detalhado em vários cenários (principal e secundários). Um cenário especifica uma possível execução de um caso de utilização. A sua descrição pode assumir uma de texto livre, mas é uma boa prática que essas descrições sejam estruturadas, detalhadas, passo a passo.

Os diagramas de classes são uma das técnicas mais utilizadas no desenvolvimento orientado por objectos. Este diagrama permite modelar a estrutura de objectos num sistema. Para cada classe descreve os seus atributos, as suas operações e o seu relacionamento com outras classes.

O sucesso de um sistema de software depende de como este serve a necessidade dos utilizadores e do ambiente envolvente [8]. Perceber as necessidades dos *stakeholders* (utilizadores, clientes, etc…) é um processo delicado de onde resultam normalmente os requisitos do sistema. Quando os *stakeholders* e os analistas de requisitos criam os documentos de requisitos, estes podem assumir vários formatos. Podemos encontrar desde simples notas retiradas de reuniões, até entrevistas gravadas com os clientes. Definir e descrever requisitos é um processo delicado, onde as interpretações que cada indivíduo tem relativo a um determinado sistema são quase sempre subjectivas. Os casos de utilização, pela sua simplicidade, são recorrentemente utilizados pelos analistas como meio de comunicação e entendimento com os *stakeholders*.

1.2. Motivação e descrição do problema

Como foi referido anteriormente, os diagramas mais utilizados, na indústria, são os diagramas de casos de utilização e os diagramas de classes. Os diagramas de casos de utilização descrevem o comportamento do sistema enquanto que os diagramas de classes representam a estrutura interna do mesmo. Ambos os diagramas representam uma abstracção do sistema, mas a relação que existe entre ambos é indirecta e esta, na generalidade dos casos práticos, não é consubstanciada. Por vezes, nem sequer os diagramas de casos de utilização são usados para especificar os requisitos, existindo apenas documentos expressos em língua natural. Apesar da desconexão entre os diagramas, a relação entre a especificação dos requisitos e o desenho do sistema é muito

importante para desenvolvimento do software. Sem essa relação torna-se muito mais difícil compreender o sistema com o qual estamos a lidar, para além de dificultar as tarefas de manter, actualizar ou reutilizar um sistema em funcionamento. Assim sendo, é necessário manter actualizados os modelos e os documentos da análise dos requisitos ao longo do ciclo de vida do software, dado que os mesmos evoluem constantemente. Se não se conseguir gerir as modificações que ocorrem nos requisitos, dificilmente se conseguirá conduzir com sucesso o desenvolvimento e manutenção do software. Como tal, é importante perceber como os requisitos estão implementados e relacionados com o resto do sistema, e quais serão os efeitos resultantes de alterações. A isto chama-se a rastreabilidade dos requisitos.

A rastreabilidade dos requisitos já foi definida de várias formas, como por exemplo:

- *”the ability to describe and follow the life of a requirement, in both a forwards and backwards direction”* [9];
- *"a software requirements specification is traceable if (i) the origin of each of its requirements is clear and if (ii) it facilitates the referencing of each requirement in future development or enhancement documentation”* [10];
- *“the ability to relate requirements specifications with other artefacts created in the development life-cycle of a software system”* [11].

Todas estas abordagens têm uma coisa em comum: conseguir acompanhar os requisitos no ciclo de vida de desenvolvimento de software. O acompanhamento dos requisitos pode ser feito de uma maneira sincrónica e/ou diacrónica. Rastreabilidade sincrónica dos requisitos é a possibilidade de se criar ligações de rastreabilidade entre os vários documentos do ciclo de vida numa determinada altura. Por exemplo, estabelecer uma ligação entre um caso de utilização e o código fonte por ele executado. Rastreabilidade diacrónica é a capacidade de perceber a evolução de um documento ao longo do tempo. Por exemplo, perceber como evolui um diagrama de classes ao longo do tempo. Através das ligações de rastreabilidade entre requisitos e código fonte, é fácil perceber que este alterar-se-á sempre que um determinado requisito se altere.

Apesar da importância destas ligações, muitas empresas têm falhado na implementação de boas práticas de rastreabilidade dos requisitos [12]. As principais

dificuldades encontradas, relativas às ligações de rastreabilidade, consistem na sua criação, manutenção e utilização. Estas ligações tornam-se especialmente importantes para os programadores que estão envolvidos na manutenção do software [13]. Várias abordagens comerciais têm tentado resolver estes problemas [14]. No entanto, o que normalmente acontece é que essas ferramentas focam principalmente a gestão das ligações de rastreabilidade, que sendo sem dúvida uma grande ajuda, não é suficiente. Fica a faltar a definição das ligações (que é manual) e o suporte para a sua utilização em tarefas da engenharia de software [14].

Não obstante, a comunidade científica já foi mais além, tornando possível automatizar a criação de ligações de rastreabilidade, embora depois a validação das mesmas acabe por ser, muitas vezes, manual. Deste modo, é fácil perceber que a geração de ligações e a sua posterior validação é complexa, muito demorada e com elevados custos.

Nos sistemas legados, a importância da rastreabilidade dos requisitos torna-se ainda mais importante. Normalmente, estes sistemas são muito críticos para o ambiente envolvente e é muito importante perceber que repercussões terá no sistema uma "*pequena*" alteração. Uma dificuldade acrescida nos sistemas legados é a falta de documentação que estes apresentam, tornando muito mais problemático a possibilidade de haver rastreabilidade de requisitos.

Em conclusão, um dos grandes problemas que actualmente o desenvolvimento de software sofre é não existirem ferramentas que auxiliem a compreensão das alterações que um requisito pode causar num sistema e a possibilidade de conseguir a documentação automática dos mesmos.

1.3. Solução apresentada

Para resolver os problemas descritos anteriormente, pretende-se apresentar um processo inovador de rastreabilidade dos requisitos em sistemas legados. Este processo também servirá de suporte à documentação desses sistemas. O processo irá ser suportado e baseado no conceito de cenários visuais.

Os cenários visuais visam diminuir a desconexão que existe entre os modelos de requisitos e a arquitectura de um sistema. Estes são uma representação dinâmica e

sincronizada entre duas perspectivas do sistema de software: caixa preta e caixa branca. A primeira corresponde à perspectiva do utilizador do sistema e é consubstanciada por um filme descrevendo cada cenário de um caso de utilização. Cada cena desse filme corresponde a um passo do cenário. A segunda (caixa-branca), corresponde à perspectiva do construtor do sistema de software e é concretizada por um diagrama de sequência (UML), representando as interacções relevantes (seleccionadas através de um mecanismo de filtragem) que ocorrem durante o cenário. Cada passo do cenário é colorido com uma cor diferente para facilitar a compreensão.

1.4. Principais contribuições previstas

Nesta dissertação pretende-se contribuir para ajudar a resolver o problema da rastreabilidade dos requisitos e da falta de documentação, que muitas vezes os sistemas enfrentam.

1. **Mitigar o problema dos sistemas legados.**

Com esta dissertação pretende-se contribuir com uma nova abordagem, que ajude a revolver um problema recorrente na área da engenharia informática: os sistemas legados.

2. **Introduzir uma nova técnica de rastreabilidade de requisitos**

Espera-se contribuir para a solução do problema da rastreabilidade de requisitos com uma técnica inovadora. Esta técnica terá em vista a automatização da criação das ligações de rastreabilidade, a partir do que se designa de cenários visuais.

3. Facilitar a validação dos requisitos de um sistema de software

Como muitas vezes a validação dos requisitos num sistema de software é demorada e complicada, pretende-se introduzir uma nova abordagem que facilite essa validação. Uma vez que os cenários visuais contêm o que acontece exteriormente e

interiormente num sistema, espera-se que através da sua visualização seja mais fácil a validação de requisitos de software.

4. Gerar automaticamente manuais de utilizador de sistemas em execução

A geração automática de documentação para sistemas legados é algo necessário, mas nem sempre disponível. Para mitigar esta falta, pretende-se introduzir uma técnica que visa auxiliar essa geração. Com a geração dos cenários visuais pretende-se que seja criado simultaneamente um manual de utilizador do sistema.

5. Enriquecer as especificações de requisitos

As especificações de requisitos podem ter vários formatos. O que se propõe nesta dissertação é enriquecê-las através dos cenários visuais para uma melhor compreensão das mesmas.

6. Melhorar o desenvolvimento do software através dos modelos

A adopção de representações mais abstractas (modelos UML) em todas as fases do ciclo de vida do desenvolvimento é algo ainda muito pouco explorada. Pretende-se com esta dissertação melhorar o estado da arte actual com uma abordagem que utilize os modelos UML em mais fases do ciclo de vida do software.

1.5. Estrutura da dissertação

Para além deste capítulo, esta dissertação é composta por mais seis capítulos.

- O segundo capítulo apresenta uma série de propostas que existem para suportar a rastreabilidade de requisitos. O capítulo começa por descrever uma taxionomia que servirá para classificar as propostas apresentadas. Seguidamente, essas mesmas propostas são descritas e detalhadas. No

final do capítulo é apresentada uma conclusão relativa ao estado da arte actual na rastreabilidade de requisitos.

- O terceiro capítulo descreve a proposta desta dissertação para o problema da rastreabilidade de requisitos nos sistemas legados. A proposta consiste na apresentação de um processo suportado por um protótipo que visa mitigar o referido problema. Neste capítulo, são definidos os requisitos do protótipo e apresentados os vários passos que constituem o processo.

- No quarto capítulo é descrita a arquitectura do *ReModeler*. Inicialmente é apresentado o protótipo como se fosse uma caixa preta, ou seja, são descritos os seus *inputs* e *outputs*. Seguidamente, o protótipo é apresentado como uma caixa branca, ou seja, passamos a conhecer os seus componentes e os pacotes necessários à sua implementação.

- O quinto capítulo descreve os aspectos relacionados com a implementação dos pacotes descritos no capítulo anterior. Neste capítulo são apresentados os diagramas de classes dos vários pacotes, bem como os diagramas de sequência de algumas das funcionalidades do protótipo. São também apresentadas as tecnologias que foram utilizadas para a criação do *ReModeler*.

- O sexto capítulo apresenta um caso de estudo da utilização do *ReModeler Process*. No caso de estudo são descritos os vários passos do processo e apresentados os resultados obtidos em cada passo. No fim do capítulo são apresentadas as ameaças à validação

- O sétimo capítulo encerra a dissertação. São apresentadas as conclusões obtidas e de que forma foram atingidas as contribuições da dissertação. A encerrar é traçado o caminho para o trabalho futuro que fica aberto por esta dissertação.

[Esta página foi intencionalmente deixada em branco]

Capítulo 2

Estado da Arte

Conteúdo

Este capítulo apresenta uma série de propostas que existem para suportar a rastreabilidade de requisitos. O capítulo começa por descrever uma taxionomia que servirá para classificar as propostas apresentadas. Seguidamente, essas mesmas propostas são descritas e detalhadas. No final do capítulo é apresentada uma conclusão relativa ao estado da arte actual na rastreabilidade de requisitos

2. Estado da Arte

2.1. Introdução

Neste capítulo propõe-se uma taxionomia que ajudará a classificar os artigos relacionados com a rastreabilidade no desenvolvimento de software. A utilização de uma taxionomia, para este tipo de análise é vantajosa, porque permite seguir uma abordagem metódica para avaliar as técnicas propostas. Esta abordagem é melhor face a uma não estruturada, porque permite ganhar objectividade e comparabilidade na análise. Esta taxionomia inclui um conjunto de características qualitativas e uma avaliação quantitativa, baseada numa escala ordinal. As características qualitativas e quantitativas irão ajudar a fazer uma crítica construtiva ao trabalho realizado.

As características qualitativas devem dar uma ideia geral de como a técnica funciona e onde esta se enquadra. Os escolhidos, para apresentar os artigos, são os seguintes; âmbito, técnica, crítica e classificação segundo a taxionomia proposta.

- **Âmbito –** Uma breve descrição da técnica e dos motivos da sua criação, enquadrando a técnica no estado da arte actual.
- **Técnica –** Uma descrição geral, sem ser muito exaustiva, de como funciona a técnica de rastreabilidade presente no artigo.
- **Crítica –** Uma crítica construtiva face à técnica apresentada, enumerando as suas vantagens e desvantagens.
- **Classificação –** Uma avaliação, apresentada sob a forma tabular, segundo os critérios definidos na taxionomia.

Como já foi referido anteriormente, a rastreabilidade de requisitos tenta criar ligações entre os requisitos, desenho, código e casos de testes de um sistema de software. A rastreabilidade ajuda os analistas a perceber que implicação tem alterar os requisitos de um sistema. Infelizmente, as técnicas de rastreabilidade não têm tido muito sucesso nas empresas que as tentam implementar, isto porque a criação e manutenção das ligações é normalmente uma tarefa muito demorada e com grandes custos [15].

Na prática, muitas empresas focam o seu esforço de rastreabilidade nos requisitos funcionais, deixando os não funcionais para segundo plano, por se tratarem de requisitos muito mais complicados de tratar. Isto leva a que a maioria das técnicas de rastreabilidade actuais apenas se preocupem em conseguir ligações para os requisitos funcionais, sendo uma minoria as que já tentam abranger os dois tipos.

O modo como se criam as ligações de rastreabilidade é também um factor muito importante. Tradicionalmente a maioria das técnicas realizava essa ligação de forma manual, ou seja, de forma dispendiosa e com muitos custos para as empresas [14]. Ultimamente tem havido uma evolução no sentido de tornar essa criação mais automática. Na verdade, quanto mais automática for a geração, melhor, pela contenção de custos associados.

A possibilidade de criar ligações automáticas originou um outro problema que é a validade das ligações criadas. Na maior parte dos casos, é necessário proceder a uma posterior validação, realizada por parte dos intervenientes no sistema, que podem ser o cliente, o gestor de projecto, o analista ou até utilizador final. No entanto, a variedade dos papéis desenrolados pelos intervenientes, origina diferentes visões do sistema, alterando e condicionando a validade da verificação efectuada [16].

2.2. Descrição da Taxionomia proposta

De acordo com a abordagem taxionómica de análise que escolhida, passa-se a descrever as categorias de cada critério que vão guiar a classificação dos vários artigos que consultados. A tabela 1 apresenta uma visão condensada dos critérios escolhidos.

Acrónimo	Critério
GL	**Geração das ligações**
CR	**Cobertura de requisitos**
VA	**Validação automática**
LPC	**Ligações pré construídas**
ASL	**Aplicação a sistemas legados**

Tabela 1 – Critérios da taxionomia

2.2.1. Geração das ligações

A **geração das ligações (GL)** relaciona-se com à possibilidade de automatizar a geração de ligações de rastreabilidade, sendo que, quanto mais automática for essa geração, menor será a intervenção humana. A maioria das ferramentas comerciais tenta, de alguma forma, conseguir o maior nível de automatização possível, embora nem todas o façam de um modo eficiente. Em baixo, descrevo os vários níveis de geração, segundo uma escala ordinal.

- **Manual –** No primeiro nível encontrado, não é possível gerar ligações de rastreabilidade de forma alguma, ou seja, alguém tem de criar uma ligação entre dois elementos manualmente.

- **Semi-automático –** Aqui já é possível ter algum automatismo na geração das ligações. Embora, seja necessário a execução de algumas acções, para a geração automática das ligações.

- **Automático –** Neste nível, a geração das ligações é toda automatizada sem necessidade de nenhuma intervenção humana, nem de nenhum *input*.

2.2.2. Cobertura de requisitos

A **cobertura de requisitos (CR)** refere o tipo de requisitos que se podem encontrar na engenharia de requisitos (ER) [17]. Os tipos de requisitos definidos são os podem ser rastreáveis pelas ligações de rastreabilidade.

- **Funcionais –** No primeiro nível apenas os requisitos funcionais são rastreáveis. Os requisitos funcionais são aqueles que descrevem o comportamento do sistema face a um determinado *input* ou acção do utilizador. Às vezes, para este tipo de requisitos é esperado um determinado output do sistema.

- **Funcionais e não Funcionais –** No nível mais elevado, é possível fazer a rastreabilidade dos requisitos funcionais e não funcionais. Os requisitos não funcionais demonstram capacidades do sistema, por exemplo: segurança, estabilidade, robustez, etc. Não faz sentido falar da possibilidade de apenas se conseguir fazer a rastreabilidade de requisitos não funcionais, pois todas as técnicas que conseguem fazer a rastreabilidade de requisitos não funcionais também conseguem cobrir os funcionais.

2.2.3. Validação automática

A **validação automática (VA)** é um ponto muito importante na validação de requisitos, porque mesmo que se consigam gerar ligações automaticamente é importante saber se as ligações geradas são válidas. Uma ligação de rastreabilidade válida é uma ligação que de facto existe.

- **Sem validação automática –** Neste nível, a validação das ligações de rastreabilidade tem de ser feita de uma forma não automática. Ou seja, por alguém que perceba o sistema.

- **Com validação automática –** No segundo nível, as ligações geradas são de facto válidas, pois é feita uma validação automática das mesmas.

2.2.4. Ligações pré construídas

As várias técnicas que vão ser apresentadas têm diferentes requisitos de *input*, para começar o seu processo de rastreabilidade dos requisitos. As mais automáticas apenas requerem os requisitos e outros documentos do ciclo de vida do software. Outras precisam de uma ajuda externa em que o analista tem de fornecer ligações já **pré construídas (LPC)** por ele.

- **Com necessidade –** Todas as técnicas que caem neste nível precisam de algum suporte externo feito por parte de um analista ou utilizador do sistema em questão.
- **Sem necessidade –** Este nível é superior, pois deixa de ter a necessidade do requisito anterior. É possível automatizar o processo de rastreabilidade.

2.2.5. Aplicação a Sistemas legados (ASL)

Não só interessa ter rastreabilidade de requisitos de sistemas em desenvolvimento mas também é importante que se consiga ter em sistemas legados. Se tivermos rastreabilidade em sistemas legados é muito mais fácil compreender e alterar o sistema.

- **Não –** Não é possível aplicar esta técnica a sistemas legados.
- **Sim –** É possível aplicar esta técnica a sistemas legados.

2.2.6. Método de geração das ligações de rastreabilidade (MGLR)

O último ponto da minha taxionomia serve para enquadramento das várias técnicas que existem. Muitas das técnicas de rastreabilidade têm em comum o método que usam para criar as ligações. Alguns dos apresentados neste documento, estão identificados em baixo.

- **Matrizes** – Uma matriz de rastreabilidade é uma tabela que relaciona dois documentos entre si. Na tabela encontram-se na coluna à esquerda os atributos de um dos documentos e na linha superior os atributos do outro documento. Quando um atributo de um documento se relaciona com algum outro documento é colocada

uma marca na célula de intersecção [18]. As matrizes de rastreabilidade são muito usadas pela indústria para definir relações entre os requisitos e outros artefactos de software [19].

- **Frases Chave** (***Keyphrased)*** – Nesta aproximação, quer os requisitos, quer o código fonte são marcados com frases de maneira a se puder estabelecer uma ligação entre ambos. A ligação é expressa através das próprias frases [20].

- **Recuperação de Informação** (***Information Retrieval***)- Aqui, a criação das ligações é feita com base na semelhança que existe entre o código fonte e os documentos de requisitos [21].

- **Baseado em Eventos** (***Event-Based***) – Este método utiliza um tipo de arquitectura subscritor/consumidor ("*publish/subscribe*"). Nesta arquitectura, um processo (*subscriber* /consumidor) subscreve o interesse num conjunto de eventos junto de outro processo (*publisher*/ produtor). Neste caso, um processo que subscreve um evento estabelece uma ligação de rastreabilidade para com o produtor. Quando houver alterações no produtor este tem de avisar todos os subscritores. Este tipo de arquitectura é usado para manter ligações de rastreabilidade actualizadas [22].

- **Baseado em cenários (Scenario-Based) –** Os cenários são muitas vezes utilizados para descrever requisitos. Neste método, os cenários são utilizados ao longo do ciclo de vida, originando a criação de todas as relações de rastreabilidade [23].

- **Baseado em aspectos (Aspect Weaving)** – o surgimento da programação por aspectos, levou ao aparecimento de novas técnicas de rastreabilidade, baseadas na facilidade com que os aspectos conseguem fazer introspecção no código fonte [19].

2.3.1. Recovering Tracebility Links between Code and Documentation [21]

Âmbito

Apesar de existirem várias técnicas de representação dos requisitos de uma aplicação, uma regra de facto é que a maioria dos requisitos acabam por ficar expressos em documentos de texto em língua natural. A ideia dos autores do artigo é que estes documentos acabam, muitas vezes, por ser preferíveis aos diagramas, isto porque a informação que um documento de texto contém é sempre superior à abstracção de um diagrama. Se houver maneira de estabelecer ligações directas entre os documentos de texto e o código fonte, a compreensão, a manutenção, a reutilização de um sistema de software pode ser facilitada.

Com base nas vantagens anteriormente referidas é apresentada no artigo uma técnica de rastreabilidade entre documentos de texto e código fonte, baseada em técnicas de IR (*Information Retrieval*) já aplicadas a outros domínios [24]. O objectivo principal de uma técnica de IR é extrair a informações contidas em documentos para posterior utilização. O processo de IR começa quando é introduzido no sistema uma consulta, na forma de uma expressão formal (*query*), sobre uma necessidade de informação. Através da *query* é possível identificar um conjunto de documentos, com diferentes graus de relevância.

Técnica

A criação das ligações de rastreabilidade vai ser conseguida através de duas actividades distintas. A primeira prepara os documentos de requisitos para ser possível extrair informações e a outra extrai *queries* do código fonte.

A actividade que prepara o documento de requisitos culmina com uma indexação do mesmo, após uma fase de normalização do texto. A indexação é feita com base no vocabulário extraído do documento. A normalização é feita em três passos. No primeiro, todas as letras maiúsculas são transformadas em letras minúsculas. A seguir

todas *stopwords* [25] (tais como artigos, pontuação, números, etc.) são removidas. Por último, são convertidos os plurais para singulares e passados todos os verbos para o infinitivo, através de uma análise morfológica. A actividade que prepara o código fonte constrói e indexa uma *query*, por cada componente do mesmo. A construção de uma *query* consiste em: extrair uma lista de identificadores, separar os identificadores compostos por mais de duas palavras (i.e., a palavra ListaLigada ou Lista_Ligada é separada em Lista e Ligada) e normalizar, conforme descrito acima, o texto do código fonte.

Depois desta preparação, é calculada a similaridade entre as *queries* e os documentos de requisitos. O resultado obtido tem os documentos de requisitos ordenadamente afectos ao código fonte, sendo que os documentos de requisitos mais similares com o código estão colocados na posição mais elevada na ordenação.

Esta classificação e ordenação dependem do modelo IR usado. No artigo é discutida a aplicação de dois modelos de IR distintos. O primeiro é o modelo probabilístico em que a ordenação dos documentos de texto é feita com base na probabilidade destes serem relevantes para a *query* calculada. O segundo modelo é o vector espacial onde os documentos e as *queries* são tratados como vectores num espaço de dimensão n, onde o n é o número de palavras no vocabulário. A ordenação dos documentos é feita com base na distância entre ambos os vectores. A figura 1 ilustra as duas actividades que compõem esta técnica.

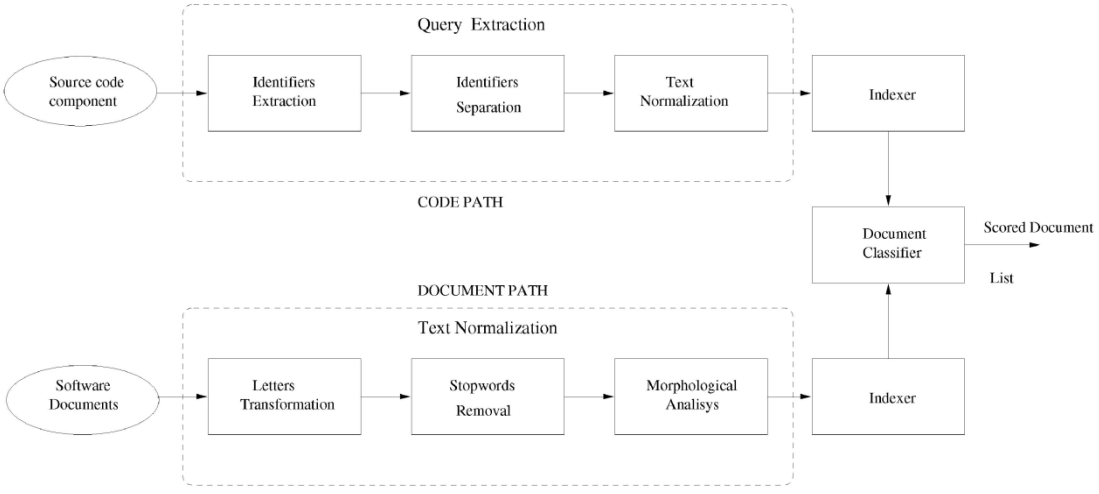

Figura 1 - Método de geração das ligações de rastreabilidade (em [21])

Crítica

O bom funcionamento desta técnica depende da similaridade entre as palavras dos documentos de texto e identificadores do código fonte, ou seja, a similaridade tem de ser bastante alta. Quando isto não acontece é mais difícil conseguir as ligações de rastreabilidade. Sendo esta uma limitação reconhecida pelos autores, eles discutem a possibilidade desta técnica poder ser estendida com uma outra para suplantar essa deficiência. A ideia é conseguirem modelar o comportamento do programador a escrever código de modo a conseguir estabelecer as ligações de rastreabilidade [21].

Como os autores alegam que os requisitos são melhor expressos em texto do que em modelos, apenas se preocuparam em estabelecer ligações de rastreabilidade entre o código fonte e os documentos de texto. As ligações de rastreabilidade são de facto calculadas de forma automática, mas a sua verificação e validação fica sempre dependente de terceiros.

Existe ainda uma outra limitação, que no âmbito desta dissertação é bastante importante. No caso de sistemas legados, a documentação dos requisitos pode não existir ou estar tão obsoleta que se torna inviável a sua utilização. Sem esta documentação a técnica proposta deixa de ser possível de usar.

Classificação

	MGLR	GL	CR	VA	LPC	ASL
[21]	Recuperação de Informação	Automático	Funcionais	Sem validação automática	Sem necessidade	Não

2.3.2. A Keyphrase Based Traceability Scheme [20]

Âmbito

Este é um artigo que demonstra como é feita uma técnica manual de rastreabilidade. Esta técnica foi usada para melhorar o desenvolvimento de um sistema

criado pela *Racal Radar Defense Systems Ltd*, actualmente comprada pela Thales Group [26].

Técnica

No processo de rastreabilidade proposto, são introduzidas manualmente frases chave entre os vários documentos do ciclo de vida do software, que são depois lidos por uma ferramenta, para detectar a presença das frases introduzidas. As frases chave introduzidas servem para exprimir relações entre os documentos. A relação é expressa entre o documento onde a frase é introduzida e outros a que esta se refira. Posteriormente, é produzido um documento com uma lista das frases chave, o nome e a secção do documento onde estas foram encontradas.

As ligações de rastreabilidade são criadas através de sucessivas comparações dos vários documentos anteriormente referidos. É possível comparar várias versões do mesmo documento ou documentos relativos a diferentes fases do ciclo de vida. Por fim, as ligações de rastreabilidade são confirmadas nas revisões dos documentos.

Crítica

Este artigo é um bom exemplo de uma técnica de rastreabilidade totalmente manual. Tudo é feito de uma forma manual, desde a criação das ligações até a sua posterior validação. É evidente que isto apresenta problemas de escalabilidade e de manutenção a longo prazo, que levaram ao aparecimento das técnicas automáticas. Cria-se uma grande dependência do programador, que inclusivamente se pode esquecer de introduzir uma frase chave, deixando de criar uma determinada ligação.

Não é feita nenhuma referência ao tipo de requisitos que esta técnica consegue abranger, mas acredito que ambos os tipos possam ser abrangidos. Sendo uma técnica manual é possível indicar no código fonte onde um determinado requisito não funcional está a ser tratado.

Mais uma vez esta técnica só pode ser usada durante a construção de novos sistemas, porque para sistemas já construídos e principalmente complexos, torna-se inviável a marcação do código

Classificação

	MGLR	GL	CR	VA	LPC	ASL
[20]	Frases chave	Manual	Funcionais e não Funcionais	Sem validação automática	Sem necessidade	Não

2.3.3. Using Scenarios to Support Traceability [23]

Âmbito

Os autores deste artigo acreditam que a utilização de cenários ao longo de todo o ciclo de vida do software melhora a rastreabilidade dos requisitos. As representações e a semântica que os cenários podem ter no ciclo de vida do software podem ser muito diferentes de fase para fase. Os cenários podem descrever o sistema em diferentes níveis de abstracção. Os autores investigaram sobre quais as possíveis utilizações dos cenários, estes podem estar ligados directamente à arquitectura do software, serem usados para testes e podem, inclusivamente, ser expressos nas mais variadas formas [23]. Neste artigo podemos ver a exploração da relação entre cenários

Foram definidos cinco tipos de cenários para acompanhar o ciclo de vida do software. Os primeiros, e mais próximos dos stakeholders, são os **cenários de requisitos** (*requirements scenarios*), que descrevem os requisitos após a sua fase de criação. Os seguintes são os **cenários de análise** (*analyses scenarios*), que adicionam detalhe aos cenários de requisitos e estabelecem uma perspectiva mais interna sobre o sistema. Um exemplo deste tipo é um diagrama de sequência UML, no qual as mensagens que são trocadas estão ao nível de abstracção do diagrama de classes do sistema. Nos **cenários de arquitectura (***arquitecture scenarios***)** é adicionando ainda mais detalhe, sendo que os eventos descritos por eles estão ao nível da arquitectura do sistema. Os **cenários de desenho** (*design scenarios*) aumentam o detalhe dos cenários de arquitectura. Os conceitos usados nestes cenários são os objectos que entram na arquitectura do sistema e as mensagens trocadas estão ao nível desses objectos. Os **cenários de código (***code scenarios*) descrevem os eventos ao nível do código do sistema, ou seja, representam as invocações dos métodos. Todos os cenários anteriores,

quando executados, resultam neste tipo de cenário. Por último, foram definidos os **cenários de teste (***test scenarios*), que representam o conjunto dos cenários definidos anteriormente, utilizados em fases de testes sobre o sistema desenvolvido.

Técnica

Para se relacionarem os vários cenários descritos anteriormente, os autores do artigo resolveram descrevê-los através de uma linguagem própria de descrição de cenários**,** scenarioML [27]. Os cenários descritos pela scenarioML são ficheiros XML criados manualmente, onde as ligações entre os cenários se encontram dispersas. A figura 2 ilustra um ficheiro escrito em scenarioML.

Figura 2 - (em [23])

Crítica

Neste artigo é proposta uma técnica onde os cenários são artefactos que são usados ao longo de todo o ciclo de vida. Usar a mesma linguagem para descrever os cenários de diferentes níveis de abstracção vai facilitar a compreensão, criação e manutenção das ligações entre os mesmos. Infelizmente, esta é uma técnica manual e tudo o que se ganharia na utilização de uma linguagem única, perde-se na manutenção e criação das ligações. O processo inerente à técnica proposta será sempre demorado e susceptível de erros, quando feito de uma maneira manual. Para além disso, também o escalonamento se torna um problema a ter em conta. Os autores reconhecem as limitações existentes e acreditam que se se conseguir automatizar o processo de criação e manutenção das ligações, poderão ter bons resultados com ela. Como este é um processo manual e com possíveis problemas de escalonamento, torna-se inexequível de ser aplicado sistemas de legados de grandes dimensões.

Os requisitos funcionais são os únicos abrangidos por esta técnica. No entanto, se os requisitos não funcionais fossem expressos em cenários, também poderiam ser abrangidos, embora os autores não clarificaram essa situação.

Classificação

	MGLR	GL	CR	VA	LPC	ASL
[23]	Baseado em cenários	Manual	Funcionais	Sem validação automática	Sem necessidade	Não

2.3.4. A Scenario-Driven Approach to Trace Dependency Analysis [15]

Âmbito

Como a geração e validação das ligações de rastreabilidade é um processo complexo e com muitos custos, este artigo propõe uma técnica automática de geração e validação das mesmas**.** Ela requer apenas que sejam fornecidos pelo programador um conjunto de hipóteses de ligações. Essas devem ligar um conjunto de cenários de teste aos artefactos que compõem o sistema. Os artefactos do sistema são entendidos, por exemplo, como documentos de requisitos escritos em língua natural, ou diagramas UML. A ideia é posteriormente executar os cenários descritos e perceber o comportamento do sistema de forma a conseguir criar novas ligações e validar as fornecidas.

Técnica

Como introduzido no âmbito, esta técnica necessita de um conjunto de requisitos iniciais para poder ser utilizada. À partida precisa de um sistema de software executável, uma lista dos artefactos desenvolvidos (i.e., elementos dos modelos), um conjunto de cenários descrevendo casos de utilização e casos de teste e de um conjunto de hipóteses iniciais de ligações entre os artefactos e os cenários. Depois de obtidos os requisitos essenciais, é necessário executar os cenários. Essas execuções permitem saber quais as linhas de código que cada cenário usa. Ao conjunto das linhas resultantes da

execução foi atribuído o nome de "*footprint*". Por sua vez, após os "*footprints*" terem sido criados, são globalmente combinados num "*footprint graph*", onde se identificam as linhas de código que os vários cenários possam ter em comum, calculando-se as dependências existentes. Se um cenário usar um subconjunto de linhas de código de um outro, isto pode indicar uma dependência entre ambos.

Havendo como hipótese inicial duas ligações entre um determinado cenário e um artefacto, é possível com esta técnica saber se existe uma relação extra de dependência entre os dois artefactos. Tal acontece, sempre que, entre os cenários, existir também uma relação de dependência (A->B e C->D, B->D sempre que A->C). A Figura 3 ilustra os passos que são efectuados para a obtenção de ligações de rastreabilidade.

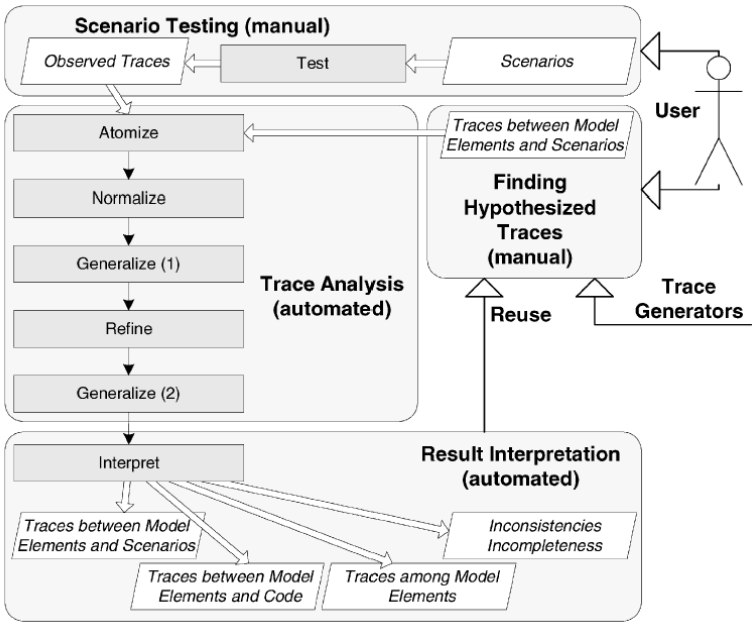

Figura 3 - Visão global das actividades para criação das ligações de rastreabilidade (em [15])

Crítica

A descrição que fiz da técnica foi apenas uma visão geral. Na verdade, o processo de análise e criação das ligações de rastreabilidade é bastante mais complexo, com inúmeras regras, embora não deixe de ser um processo automático que consegue validar as ligações criadas. As ligações de rastreabilidade estão directamente ligadas ao código fonte da aplicação. Como tal, apenas é possível detectar dependências entre modelos relacionados com o código.

Nesta técnica existe outra pequena desvantagem: a necessidade de terem de ser criadas inicialmente algumas ligações. Para além disso, a qualidade do resultado está fortemente dependente do *input* fornecido. É referido também no artigo que é possível gerar ambiguidades nas ligações de rastreabilidade, mas que após sucessivas utilizações do método, essa ambiguidade desaparece. Embora esta técnica abranja claramente os requisitos funcionais, a abrangência dos não funcionais não fica muito clara. Sendo esta uma técnica baseada em cenários, o que teria de ser feito para resolver o problema, seria expressar os requisitos não funcionais também em cenários, tal como na técnica [23] referida anteriormente.

Como precisamos de ter um sistema a funcionar e o resto do processo é relativamente automático, esta é uma técnica possível de ser aplicada a sistemas legados.

Classificação

	MGLR	GL	CR	VA	LPC	ASL
[15]	Baseado em cenários	Semi-Automático	Funcionais	Com validação automática	Com necessidade	Sim

2.3.5. Recovering and Using Use-Case-Diagram-To-Source-Code Traceability Links [13]

Âmbito

Quer os requisitos, quer as funcionalidades de um sistema de software, são normalmente descritos através de casos de utilização. Como já foi referido anteriormente, os casos de utilização raramente são ligados ao código fonte. A técnica proposta neste artigo tentar criar ligações directas entre os casos de utilização e o código fonte. A ideia principal é tentar encontrar nomes comuns entre os casos de utilização e o código fonte, estabelecendo assim uma ligação de rastreabilidade entre ambos.

Técnica

A solução apresentada é chamada de "*LEarning and ANAlyzing Requirements Traceability*" (*LeanArt*). A figura 4 ilustra a arquitectura da solução. Para iniciar o processo é necessário, o código fonte do programa (2), os casos de utilização (1) e algumas ligações (3) criadas pelo programador do sistema. Após fornecido o input descrito, o *LeanArt* instrumenta o programa (4) de modo conseguir monitorizar as variáveis do mesmo enquanto este estiver a ser executado. Aquando da execução do programa, vão sendo recolhidos dados sobre as variáveis instrumentadas. Os valores recolhidos, as ligações inicialmente fornecidas, os casos de utilização e os nomes das entidades do programa vão servir de dados de entrada ao *Learner* (9) para este estabelecer relações, ou seja, identificar entidades com valor e nome semelhantes. No final, o Learner depois vai relacionar as restantes entidades, que ainda não estavam relacionadas, com os casos de utilização.

O resultado fornecido pelo *Learner* é depois validado pelo *Validator* (11) e fornecido ao utilizador (programador) que, após atenta revisão, rejeita ou aceita os dados recebidos. As ligações classificadas como verdadeiras são posteriormente guardadas numa base de dados (12), para mais tarde serem usadas pelo *Learner* para melhorar a sua estratégia.

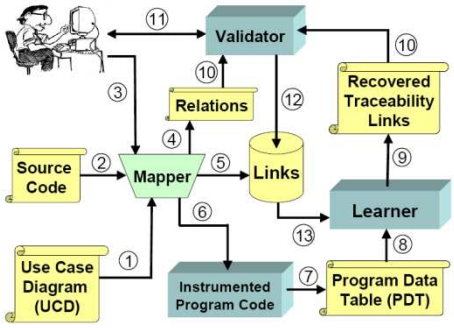

Figura 4 - Arquitectura do LeanArt (em [13])

Crítica

A ideia principal da proposta apresentada é que é possível imitar o procedimento que um ser humano tem, quando procura por semelhanças entre nomes no código e nos casos de utilização. Foram utilizadas técnicas de "*machiner learning*" (ML) para conseguir relacionar os nomes dos casos de utilização com o código fonte. O problema das técnicas ML é não terem uma precisão de 100%, tendo o programador que acabar

por validar manualmente as ligações criadas. Os autores do processo reconheceram essa falha e melhoraram o processo de detecção das ligações. Mesmo assim, em última instância, o programador acaba sempre por ter de validar algumas ligações.

É ainda apresentado no artigo um *plugin* de *Eclipse*, que facilita a navegação entre os casos de utilização e o código fonte através das ligações criadas. Nesta técnica apenas existe a preocupação em construir ligações entre casos de utilização e o código fonte, esquecendo o resto dos possíveis documentos que existem no ciclo de vida.

Os requisitos abrangidos por este processo são naturalmente os requisitos funcionais, que estão declarados nos casos de utilização.

Classificação

	MGLR	GL	CR	VA	LPC	ASL
[13]	Machine Learning	Automático	Funcionais	Sem validação automática	Com necessidade	Sim

2.3.6. Experiences Using Scenarios to Enhance Traceability [28]

Âmbito

O artigo proposto não apresenta nenhuma técnica específica de rastreabilidade. Trata-se de um relato de experiências, onde os cenários são usados para melhorar a rastreabilidade no desenvolvimento de software. São apresentadas três situações, onde os cenários são usados no desenvolvimento de software.

Técnica

Não é apresentada nenhuma técnica.

Crítica

Na primeira situação apresentada, os cenários são usados ao longo de todo o ciclo de vida do software e são criadas ligações directas entre estes e os artefactos do ciclo de vida. Pelo que consegui perceber, as ligações são criadas de uma forma manual

e a relação fica guardada numa ferramenta que faz a gestão dessas mesmas ligações. Esta visão não acrescenta nada de novo ao estado de arte actual, apenas é apresentado um framework de gestão de cenários e ligações.

Na segunda situação, é apresentada a ideia de melhorar a rastreabilidade no desenvolvimento ágil de software, mais especificamente na abordagem de *Extreme programming* (XP) [29]**.** Quando o software é desenvolvido segundo uma abordagem XP, a maior parte dos requisitos é expressa numa estrutura chamada *user stories*. Depois de se iniciar a programação, esses requisitos são descartados. A ideia apresentada é que as *user stories* não devem ser descartas, mas sim ficar guardadas num sistema, na forma de cenários. Mais uma vez, a criação e gestão das ligações entre os cenários e o resto do sistema é feita de uma forma manual. Esta situação traz um conceito bastante interessante, da rastreabilidade de requisitos no desenvolvimento ágil de software, onde normalmente se descarta ou não existem os requisitos do sistema em desenvolvimento.

A última situação apresentada é a utilização de cenários no desenvolvimento *open source*. O desenvolvimento de código *open source* é um exemplo onde a rastreabilidade de requisitos se baseia unicamente no código. Muitas vezes, para se perceber as intenções motivadoras da criação de um determinado componente, os únicos elementos disponíveis que existem são os comentários no código. Isto torna difícil a percepção da estrutura do sistema e consequentemente a avaliação das consequências que uma alteração a uma determinada parte do código pode trazer ao resto do sistema.

A ideia proposta é utilizar a descrição dos cenários como substitutos dos comentários do código. Como os cenários são mais ricos que simples comentários, será mais fácil traçar a rastreabilidade do sistema. Os cenários ficam intimamente ligados ao código.

Os autores pensam ainda desenvolver ferramentas que suportem a criação e manutenção desta junção do código com os cenários. De todas as situações onde se usaram cenários para melhorar a rastreabilidade, esta última é sem dúvida bastante interessante. No entanto o modo de criação e gestão dos cenários é, mais uma vez, um processo manual, que traz as desvantagens já previamente mencionadas.

2.3.7. Automating Software traceability in Very Small Companies: A Case Study and Lessons Learned [14]

Âmbito

A rastreabilidade dos requisitos ainda é muito pouco utilizada na indústria, sendo várias as razões apontadas como causadoras desta situação. O autor identificou como as principais razões a complexidade e a dimensão da rastreabilidade dos requisitos em grandes projectos. No meio empresarial, estabelecer e manter ligações de rastreabilidade é uma tarefa demorada e com muitos custos associados. Isto porque as técnicas usadas, continuam a ser maioritariamente manuais e as muitas das ferramentas de rastreabilidade que existem apenas se concentram em fazer uma gestão das ligações criadas [30].

Num panorama onde as ferramentas existentes no mercado têm como foco principal a manutenção de ligações de rastreabilidade, uma empresa austríaca sentiu a necessidade de desenvolver um ambiente capaz de suportar a geração e manutenção dessas ligações de rastreabilidade. Era importante que nesse ambiente fosse possível a utilização das ligações em actividades da engenharia de software, tais como análise de impacto. O artigo apresenta o ambiente desenvolvido e descreve as conclusões inerentes à sua utilização.

Técnica

A empresa já possuía várias ferramentas de desenvolvimento, sendo a maioria delas desenvolvidas pela própria. Entre elas encontram-se um sistema de rastreio, neste são guardados os dados relevantes aos produtos desenvolvidos, tais como os seus atributos. Contam ainda com um sistema para controlo de versões baseado no CVS [31] e um repositório central de documentação baseado num wiki [32]. Para melhorar a rastreabilidade de requisitos foi decidido desenvolver um ambiente capaz de ligar os artefactos das várias ferramentas que já disponham.

A primeira dificuldade encontrada foi como fazer a gestão das diferentes tecnologias de desenvolvimento. Optaram por uma estratégia baseada num repositório central que fosse automaticamente preenchido. A maioria dos artefactos relativos ao ciclo de vida já estavam guardados numa base de dados Oracle, como tal decidiram que

o novo sistema iria ser desenvolvido em Oracle. A primeira coisa que desenvolveram foi o software necessário para importar toda a informação externa. Foi criado um modelo de dados específico para receber a informação importada. Seguidamente, desenvolveram uma maneira de gerar as ligações automaticamente. Esta apenas se baseou nos ids que os artefactos já possuíam no sistema de rastreio.

As ligações que puderam produzir foram apenas entre módulos de um programa e ficheiros de nomes, entre código fonte e a sua informação respectiva. Não é actualmente possível estabelecer outro tipo de ligações por exemplo entre tabelas e código fonte.

Por último, construíram um motor de pesquisa para poder utilizar a informação relativa as ligações estabelecidas.

Crítica

Achei bastante importante incluir este artigo no estado da arte actual porque é bastante recente e demonstra muito bem o que se passa no mundo empresarial. Os autores após investigarem o mercado concluíram que as ferramentas que existem actualmente disponíveis são bastante limitadas, que estas apenas se focam na gestão das ligações de rastreabilidade. Assim sendo, decidiram criar um ambiente de rastreabilidade. O ambiente criado além de tentar gerar as ligações de rastreabilidade automaticamente, tenta também que possam ser utilizadas para melhorar o desenvolvimento do software.

A grande desvantagem que eu veja no ambiente apresentado é que a geração das ligações assenta na presença de ids previamente introduzidos. Se as convenções de nomes que especificaram não forem utilizadas é impossível estabelecer qualquer ligação de rastreabilidade. O ambiente descrito no artigo não faz a rastreabilidade de um sistema, mas sim do processo de desenvolvimento e pelo que dá para perceber o ambiente desenvolvido apenas faz uma gestão dos requisitos funcionais. A sua aplicabilidade a sistemas legados é inviável, visto que estes já foram desenvolvidos e possivelmente não têm documentação associada.

A conclusão que os autores chegaram sobre a utilização da técnica desenvolvida foi que apesar da geração das ligações ser muito simples, conseguiram retirar benefícios da sua criação e automatizar o desenvolvimento de software. Esta é apenas uma crítica à

aplicação do ambiente desenvolvido. Conseguiram-se mais resultados mas que caem fora deste âmbito.

Classificação

	MGLR	GL	CR	VA	LPC	ASL
[14]	Palavras-chave (ids)	Automático	Funcionais	Sem validação automática	Sem necessidade	Não

2.3.8. Event-Based Traceability for managing Evolutionary Change [22]

Âmbito

Os autores deste artigo identificaram que um dos principais problemas na rastreabilidade de requisitos é a manutenção das ligações entre vários artefactos durante o ciclo de vida do software. À medida que um sistema é desenvolvido, os seus documentos também deveriam evoluir. O que na realidade se passa é exactamente o contrário. Após a criação dos documentos de requisitos, a manutenção da sua rastreabilidade é completamente descurada.

Tendo por base a ideia de que as ligações de rastreabilidade devem acompanhar a evolução de um sistema de software, é proposta uma técnica de rastreabilidade baseada em eventos. Nesta técnica, requisitos e outros artefactos da engenharia de software são ligados através de uma relação produtor-consumidor. Os requisitos e artefactos que possam causar alterações ao sistema tomam o papel de produtores, enquanto que os artefactos dependentes adoptam o papel de consumidores. Quando os requisitos mudam é feita uma publicação desse evento. Posteriormente todos os consumidores (assinantes) são notificados dessa alteração. Os assinantes depois devem tomar as acções apropriadas para se adaptarem à nova situação dos requisitos.

Técnica

Para se poder implementar convenientemente a técnica de rastreabilidade deste artigo os autores implementaram a arquitectura representada pela figura 5. Os componentes principais da arquitectura são: o *requirements manager*, o *event server* e o *subscriber manager*.

Requirements Manager

O *requirements manager* é responsável pela gestão dos requisitos e por criar os eventos relativos às alterações nos requisitos. A definição dos eventos teve por base a observação da evolução dos requisitos em vários projectos. As acções definidas que se podem ter sobre os requisitos são de: criação, modificação, decomposição, desactivação, junção, refinamento, substituição. É importante salientar que é preciso detectar quando estes eventos ocorrem. Para tal foi concebido um algoritmo [33], que é usado pelos autores, para suportar o reconhecimento dos eventos durante a especificação dos requisitos.

Event Server

O *event server* é responsável pela gestão das subscrições e pela gestão de todas as mensagens possam ocorrer no sistema. O *event server* guarda ainda as acções apropriadas devem ser tomadas por tipos de assinantes específicos em resposta a eventos padrão.

Subscriber Manager

O *subscriber manager* é responsável por receber as notificações de eventos e de entregá-las aos artefactos dependentes. A resolução dos eventos recebidos pelos artefactos é feita pelo programador. Este actualiza o artefacto e as suas ligações de maneira a responder ao evento recebido.

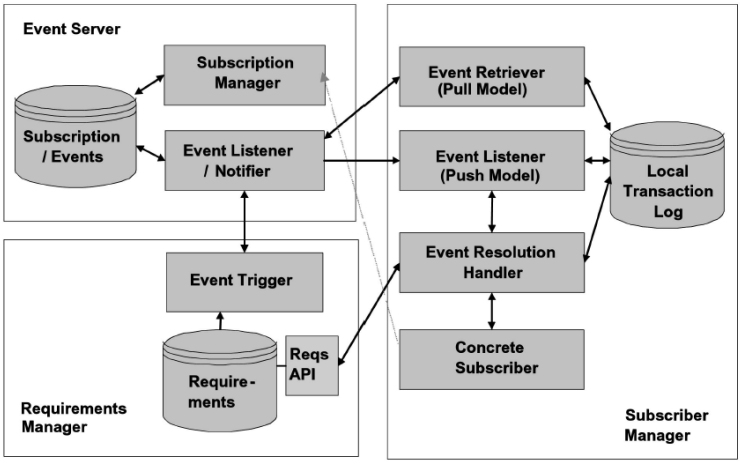

Figura 5 - (em [22])

Manutenção das alterações nos artefactos

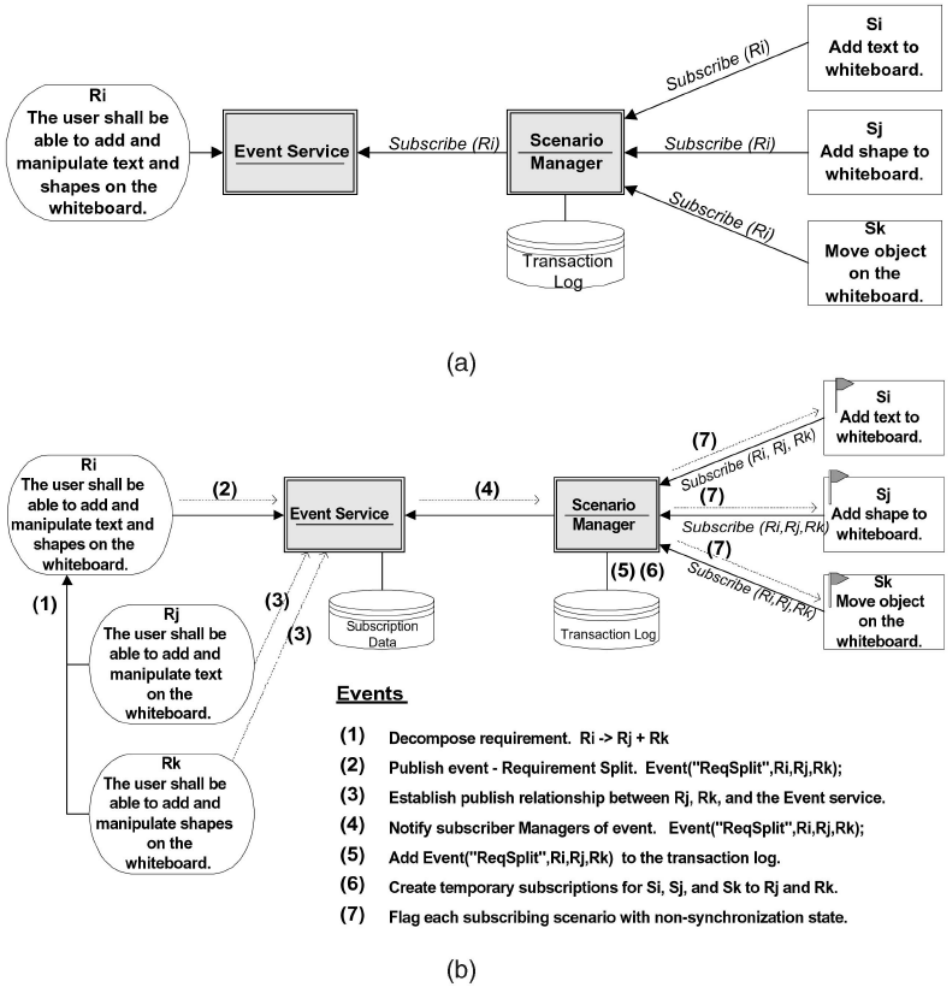

Figura 6 - (em [22])

A figura 6 ilustra a situação da evolução de um requisito no sistema. Os cenários Si, Sj e Sk estão ligados ao requisito Ri por subscrições feitas pelo "*scenario subscriber manager*". Quando o requisito Ri é decomposto em duas partes, Rj e Rk, um evento de alteração é automaticamente publicado no *event server*. O *event server* verifica na sua base de dados interna que são os assinantes daquele requisito e notifica-os através do *subscriver manager*. O *subscriber manager* é responsável pela gestão dos diferentes tipos de eventos de acordo com as necessidades dos seus assinantes. Neste caso, ocorreu uma decomposição e como tal é necessária intervenção manual para serem estabelecidas novas ligações de dependência. O *scenario subscriber manager* actualiza o seu log de transacções com os eventos publicados e avisa os assinantes, dependentes do evento. Os assinantes (Si,Sj e Sk) terão de sincronizar o seu estado, de acordo com o log, antes de puderem ser usados. Por fim, o *scenario manager* actualiza as assinaturas dos subscritor de modo a estes serem notificados de eventos relacionados com os requisitos Ri,Rj e Rk.

Crítica

A rastreabilidade baseada em eventos é uma técnica que vem tentar dar resposta ao problema que existe de manutenção e actualização das ligações de rastreabilidade. Ao contrário das abordagens anteriores, onde a preocupação era a aquisição das ligações de rastreabilidade, esta apenas se preocupa na sua manutenção. Nesta técnica não existe nenhuma maneira automática para criação de ligações de rastreabilidade. Tem de ser o próprio utilizador da arquitectura a introduzir as ligações. A sua aplicabilidade a sistemas ligados não é muito fácil visto que a criação das ligações é manual.

Classificação

	MGLR	**GL**	**CR**	**VA**	**LPC**	**ASL**
[22]	Baseado em Eventos	Manual	Funcionais e não Funcionais	Sem validação automática	Sem necessidade	Não

2.4. Conclusão

Todas as técnicas apresentadas na tabela 2, tentam mitigar o problema da rastreabilidade de requisitos. A rastreabilidade ajuda os analistas a perceber que implicação tem alterar os requisitos de um sistema. Infelizmente, as técnicas de rastreabilidade não têm tido muito sucesso nas empresas que as tentam implementar, isto porque a criação e manutenção das ligações é normalmente uma tarefa muito demorada e com grandes custos [15]. Em [14] foi investigado o mercado das ferramentas de rastreabilidade e concluiu-se que as ferramentas que existem actualmente apresentam bastantes limitações. Estas apenas se focam na gestão de ligações de rastreabilidade deixado para segundo plano a sua aquisição. Em [22] temos um exemplo que vai de encontro à preocupação da manutenção das ligações de rastreabilidade. O esforço de rastreabilidade dos requisitos por parte das empresas, actualmente é apenas focado nos requisitos funcionais, os não funcionais são deixados para segundo plano, por ser mais difíceis de tratar. Como os requisitos não funcionais são atributos de um sistema de software torna-se muito difícil criar um traço de rastreabilidade entre os vários componentes de um sistema. Isto leva a que a maioria das técnicas de rastreabilidade actuais apenas se preocupem em conseguir ligações para os requisitos funcionais, sendo uma minoria as que abrangem os dois tipos, por exemplo [20] e [22]. A viabilidade de criação de ligações de rastreabilidade de requisitos não funcionais é encontrada em técnicas manuais de rastreabilidade. As primeiras técnicas de rastreabilidade que apareceram as ligações eram calculadas e mantidas de uma forma manual [20], [23]. Em virtude disso, começaram a surgir problemas de escalabilidade e manutenção a longo prazo e por conseguinte elevados custos [22]. Logo à partida, num processo manual, é preciso um esforço extra para criação e manutenção das ligações. Mais recentemente, começaram a aparecer técnicas onde se encontram vários automatismos relativamente à rastreabilidade dos requisitos. [13], [14], [21]. Nestas técnicas as possíveis ligações de rastreabilidade são calculadas de forma automática, mas a sua validação e verificação fica sempre dependente de terceiros. De forma a facilitar a criação de ligações automáticas muitas vezes são fornecidas outras como input. O input fornecido se não for de qualidade desejada, ou seja, se as ligações forem insuficientes ou não fizerem sentido, o resultado final será afectado gravemente afectado. Em [28] é apresentada uma técnica que tenta melhorar a rastreabilidade de requisitos no desenvolvimento ágil de software, mais especificamente na abordagem de *Extreme Programming* (XP) [29]. Onde é apresentado a rastreabilidade de requisitos no

desenvolvimento ágil de software, onde normalmente se descartam ou não existem os requisitos do sistema em desenvolvimento. Contudo está técnica não deixa de ser manual tendo as desvantagens previamente expostas. Apesar de tudo quando é aplicada alguma técnica de rastreabilidade é possível encontrar vários benefícios no desenvolvimento de software. Em [14] conseguiram tirar benefícios da rastreabilidade de requisitos numa empresa de desenvolvimento de software. Apesar de existirem várias técnicas de rastreabilidade, estas dificilmente se conseguem aplicar a grandes sistemas em execução (sistemas legados). Estes, muitas vezes apresentam documentação obsoleta ou inexistente. O esforço para conseguir rastreabilidade de requisitos neste tipo de sistemas é tão elevado que afasta muitas das empresas da prática da rastreabilidade de requisitos.

	MGLR	GL	CR	VA	LPC	ASL
[21]	Recuperação de Informação	Automático	Funcionais	Sem validação automática	Sem necessidade	Não
[20]	Frases Chave	Manual	Funcionais e não Funcionais	Sem validação automática	Sem necessidade	Não
[23]	Baseado em Cenários	Manual	Funcionais	Sem validação automática	Sem necessidade	Não
[15]	Baseado em cenários	Semi-Automático	Funcionais	Com validação automática	Com necessidade	Sim
[13]	Machine Learning	Automático	Funcionais	Sem validação automática	Com necessidade	Sim
[14]	Palavras-Chave (ids)	Automático	Funcionais	Sem validação automática	Sem necessidade	Não
[22]	Baseado em Eventos	Manual	Funcionais e não Funcionais	Sem validação automática	Sem necessidade	Não

Tabela 2 – Estado da arte

Capítulo 3

ReModeler Process, *ReModeler* e a Modelação Aumentada

Conteúdo

Este capítulo descreve a proposta desta dissertação para o problema da rastreabilidade de requisitos nos sistemas legados. A proposta consiste na apresentação de um processo suportado por um protótipo que visa mitigar o referido problema. Neste capítulo, são definidos os requisitos do protótipo e apresentados os vários passos que constituem o processo.

3. *ReModeler Process, ReModeler* e a Modelação Aumentada

A manutenção dos modelos e dos documentos da análise de requisitos ao longo do ciclo de vida do software é geralmente uma tarefa demorada e por isso com custos associados elevados. Nesta dissertação é proposto um processo suportado por uma ferramenta, que juntamente com um conjunto de artefactos, visam aumentar a nossa compreensão sobre um sistema. Entre outras coisas, vai automatizar a utilização de modelação estática e dinâmica e simultaneamente garantir a rastreabilidade dos requisitos funcionais. O processo foi denominado de *ReModeler Process* e a ferramenta que o suporta foi nomeada de *ReModeler*.

3.1. Modelação Aumentada

A modelação aumentada tenta melhorar a rastreabilidade ao longo de todo o ciclo de vida do software, através de uma série de abstracções. Nestas abstracções podemos encontrar os cenários visuais, os manuais de utilizador automáticos e outros artefactos [34]. As abstracções da modelação aumentada vão tentar melhorar a nossa compreensão sobre os sistemas legados, através da sua documentação, reduzindo o fosso que existe entre a modelação estática e a dinâmica. A modelação aumentada advoga que a utilização de modelos deve ser estendida a todo o ciclo de vida do software, desde a definição dos requisitos até à entrega do produto final. Particularmente, tenta-se expandir as fronteiras do UML, adicionando novas perspectivas e mecanismos de percepção, que possibilitarão um aumento de produtividade dos seus utilizadores.

3.1.1. Cenários Visuais

As interacções de um utilizador com um sistema de software são normalmente descriminadas com recurso a casos de utilização, que por sua vez são detalhados em vários cenários (principal e secundários). A representação de um cenário não está uniformizada, podendo ser representada como um texto livre, embora seja preferível recorrer a uma descrição estruturada, apresentando um conjunto de passos. Um cenário visual é uma extensão à descrição estruturada. Através dos cenários visuais vamos criar cenários UML "aumentados", que contêm uma representação dinâmica e sincronizada entre duas perspectivas do sistema de software: caixa preta (cenário externo) e caixa branca (cenário interno). O cenário externo representa a perspectiva do utilizador do sistema e é consubstanciado por um filme. Por sua vez, um cenário interno representa a perspectiva do programador do sistema e é concretizado por um diagrama de sequência. Os cenários visuais além de se apresentarem como um artefacto para a documentação de sistemas, têm também como objectivo suportar a construção de cenários de teste, uma vez que eles contêm as acções dos utilizadores, aquando a utilização de um sistema. A figura 7 representa os conceitos do cenário visual presentes na modelação aumentada. Um cenário visual é constituído por várias cenas (Passo_Visual), que por sua vez são compostas por um conjunto de imagens. As imagens podem estar, ou não, associadas a uma mensagem de um diagrama de sequência, através das marcações temporais.

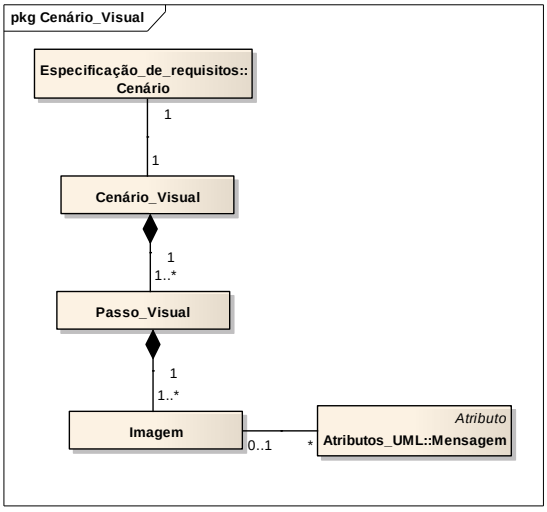

Figura 7 - Fragmento da ontologia do domínio da modelação aumentada (cenário visual)

3.1.1.1. Representação de cenários externos

O teste de caixa preta (*black-box testing*) é uma das técnicas de ensaio de software mais usada [1]. Neste tipo de testes apenas se conhece o comportamento externo de um sistema de software, que varia segundo um conjunto de dados de entrada fornecidos. De modo semelhante, num cenário externo apenas se conhece a parte exterior do sistema e o seu comportamento face às interacções do utilizador. Num cenário visual, a perspectiva da caixa preta é representada por um filme que contém todas as interacções de um utilizador com o sistema de software. Deve existir igual número de filmes e de cenários, para um sistema composto por vários casos de utilização, sendo que cada cena do filme representa um passo da descrição do cenário. De modo a aumentar a nossa compreensão sobre o filme, cada cena contêm uma legenda que corresponde à descrição do passo do cenário respectivo.

3.1.1.2. Representação de cenários internos

Uma outra técnica de teste de software muito conhecida é o teste de caixa branca (*white-box testing*) [1]. Neste tipo de testes o foco de análise é o interior do sistema, ou seja, o seu código fonte. Num cenário visual, o cenário interno representa uma perspectiva análoga aos testes de caixa branca. Um cenário interno representa as interacções que existem entre os vários componentes de um sistema, quando um utilizador está a interagir com o mesmo. Os cenários internos complementam os externos, na medida em que mostram um filme animado das chamadas que ocorrem entre as instâncias das classes de um sistema de software, onde é possível identificar os vários passos de um cenário externo. Este filme é conseguido fazendo animação de diagramas de sequência.

3.1.2. Manual automático de utilizador

Um problema recorrente na engenharia de software é a comunicação entre os *stakeholders* de um sistema e a equipa que o está a desenvolver. As falhas mais recorrentes ocorrem na definição de requisitos, que levam a que o produto final não disponha das funcionalidades desejadas. Outra vezes o manual de utilizador do produto

final não descreve devidamente as funcionalidades do sistema, levando a uma posterior rejeição do mesmo [35]. Uma das soluções apontadas para este problema foi o desenvolvimento do manual de utilizador no início do ciclo de vida do software [36]. Assim, os manuais de utilizador dão uma boa perspectiva do que um sistema tem para oferecer a quem o está a adquirir e conseguem simultaneamente fornecer descrições das funcionalidades a quem o está a desenvolver, beneficiando ambas as partes envolvidas. Basicamente, os manuais de utilizador são uma forma de documentação importante nos sistemas de software. Tipicamente descrevem as suas funcionalidades e como executá-las. Deste modo, um manual de utilizador contribui para o enriquecimento da especificação dos requisitos, auxilia fases posteriores do ciclo de vida do software e apresenta-se como uma importante ferramenta de validação de um sistema [35].

Nesta dissertação é apresentada uma proposta para a sua geração automática, combinando os cenários visuais de um sistema de software com os seus casos de utilização. O manual de utilizador gerado está organizado da seguinte forma: começa por estar dividido por capítulos, onde cada capítulo descreve um diagrama de casos de utilização, depois por secções, subsecções e páginas.

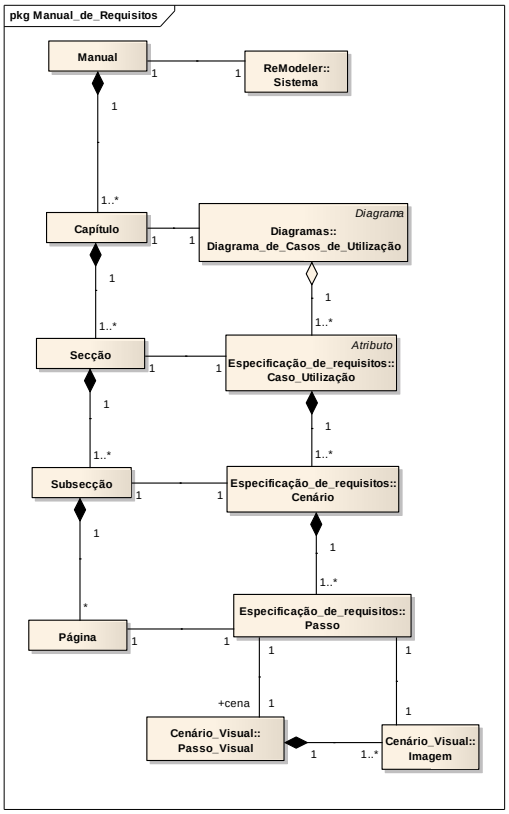

Figura 8 – Fragmento da ontologia da modelação aumentada (manual de utilizador)

As secções representam os casos de utilização que compõem um capítulo (um diagrama). Por sua vez, a cada subsecção correspondem as descrições de cenários de cada caso de utilização. Cada página de uma subsecção/descrição de cenário corresponde então a um passo. Ela deve mostrar não só a descrição do passo, mas também uma imagem representativa do mesmo, que é obtida a partir dos cenários visuais. A ideia subadjacente é que o manual seja gerado após a criação de todos os cenários visuais que compõem um sistema. A figura 8 representa a associação criada entre o manual de utilizador e a especificação de requisitos de um sistema.

3.1.3. Baterias de Cenário Teste

Testar é:

"…uma actividade realizada para avaliar a qualidade de um produto, com o intuito de o melhorar, através da identificação dos defeitos e problemas" [37].

"…uma verificação dinâmica do comportamento de um sistema num conjunto finito de casos de teste, escolhidos adequadamente a partir de um conjunto infinito de execuções do domínio, contra o comportamento esperado" [37].

Derivando do que foi exposto, é possível concluir que testar é uma actividade vital para a qualidade de um produto. Os produtos de software não fogem a essa regra. Para testar a qualidade de um produto de software é normal utilizarem casos de teste. Um caso de teste na engenharia de software é um conjunto de condições ou variáveis sobre as quais é determinado se um requisito, ou um caso de utilização, de uma aplicação é completamente ou parcialmente satisfeito [38]. É normal fazerem-se muitos casos de teste para estabelecer que um requisito foi completamente satisfeito. Os casos de testes são normalmente agrupados por similaridade e esses agrupamentos são chamados de baterias de casos de teste. Por exemplo, um sistema pode ter uma bateria de casos de teste para agrupar testes relativos a uma base de dados e pode ter outra para agrupar testes relativos às interfaces gráficas. A validade de um sistema depende de como este se comporta face aos requisitos dos seus *stakeholders*. As várias perspectivas dos *stakeholders* estão representadas nos casos de utilização e nas descrições de cenários. Isto leva a pensar que os testes deveriam ser feitos de acordo com as perspectivas dos utilizadores de um sistema, ou seja, devia-se falar de cenários de teste,

em vez de casos de teste, e de baterias de cenários de teste, em vez baterias de teste. Os cenários de teste vão permitir ter várias vistas de acordo com as perspectivas dos vários *stakeholders*. As baterias de cenário de teste são um conjunto finito de cenários de teste. Um cenário de teste pode estar presente em várias baterias de cenários de teste. É importante referir que as baterias de teste podem ser produzidas a partir dos requisitos (casos de utilização) desenvolvidos pelo engenheiro de requisitos em conjunto com o perito do domínio, logo no início do ciclo de vida. Através das baterias de cenários de teste tem-se uma aliança entre os requisitos, modelação e testes que é difícil encontrar nas práticas correntes.

Assim, a definição de testar um sistema pode ser um pouco diferente da anterior. Testar é:

"…uma actividade realizada para avaliar a qualidade de um produto, de forma a melhorá-lo, através da identificação de defeitos e problemas rastreáveis ao **nível dos modelos UML**."

"…uma verificação dinâmica do comportamento de um sistema através de um conjunto finito de **cenários de testes**, escolhidos adequadamente a partir de um conjunto infinito de execuções do domínio, contra o comportamento esperado"

3.1.3.1. Cobertura de Cenários de teste

As baterias de cenários de teste podem ter várias utilizações, por exemplo, para a realização de testes de cobertura. É possível fazer uma analogia com os testes de cobertura do código fonte. Nesses testes é calculado quantas linhas, funções, condições e caminhos possíveis foram executados. São chamados testes de caixa branca, onde é inspeccionado o código fonte directamente. No caso da cobertura de cenários de teste os testes são de caixa preta, não se inspecciona o código fonte, mas sim quais os requisitos e cenários que foram executados.

3.2. O *ReModeler Process* (RMP)

Para utilizar os vários artefactos anteriormente descritos foi definido o *ReModeler Process*. O principal intuito deste processo é a documentação automática de um sistema, obtida através da automatização do processo de criação de modelos estáticos e dinâmicos e da rastreabilidade dos requisitos funcionais, expressos em cenários de casos de utilização. Este processo é suportado por uma ferramenta, designada de *ReModeler*, onde é possível criar as abstracções da modelação aumentada anteriormente descritas A figura 9 representa alguns conceitos presentes na linguagem UML que constituem a base do *ReModeler Process*. Nele encontram-se facilidades de criação e edição de casos de utilização, que por sua vez constituem os diagramas de casos de utilização. O cenário interno de um cenário visual deve ser apresentar um diagrama de sequência. Estes são constituídos por vários objectos, que trocam mensagens entre eles. Todos os objectos são instâncias de uma determinada classe, que está contida num diagrama de classes. Para facilitar a organização dos diagramas anteriormente referidos, estes apresentam-se agrupados por pacotes, que podem ser representados em diagrama de pacotes.

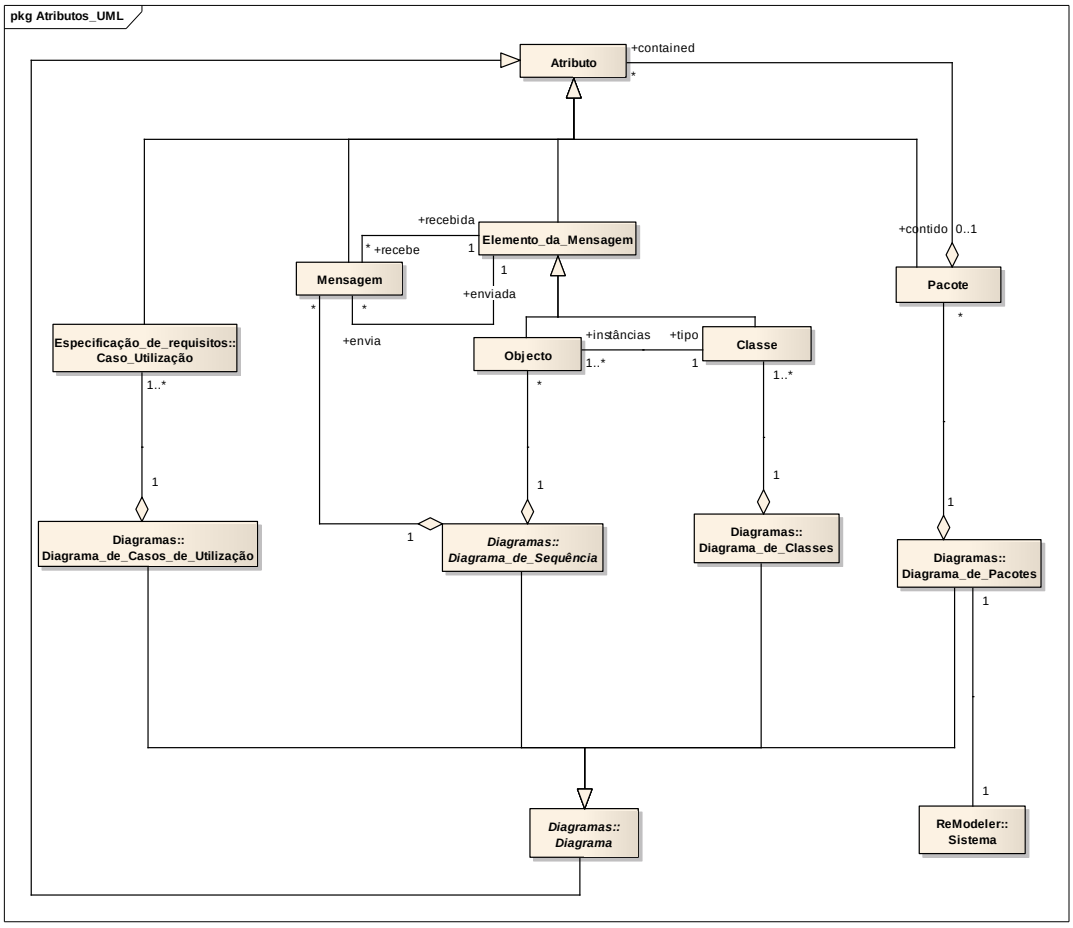

Figura 9 – Fragmento da ontologia do domínio da modelação aumentada (conceitos UML)

3.2.1. Inicialização do processo

O primeiro passo deste processo consiste em ligar a ferramenta *ReModeler* ao sistema a documentar, neste passo inicial os dois sistemas são compilados, criando-se um novo sistema. A figura 10 representa o diagrama de actividades deste passo.

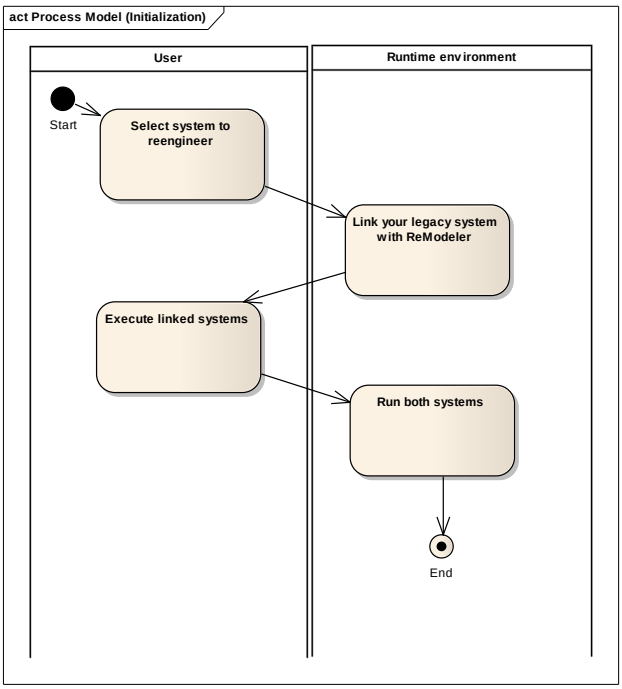

Figura 10 – Passo inicial do RMP

3.2.2. Criação dos Cenários dos Casos de Utilização

Depois de ligados ambos os sistemas e posto a correr o sistema resultante, o primeiro passo para a construção da documentação de um sistema de software, através do RMP, é dado por um perito no domínio. Este vai descrever detalhadamente o sistema, através de diagramas de casos de utilização. O RMP tanto permite a criação de casos de utilização na própria ferramenta, como permite a importação de diagramas de casos de utilização em formato XMI (*XML Metamodel Interchange*) [39]. O XMI é um standard para a exportação e importação de diagramas UML. Após a criação dos casos de utilização é necessário proceder a descrição estruturada de cenários. Esta é feita na própria ferramenta através dos passos e sub-passos que o constituem. Esta facilidade de detalhar os cenários não é encontrada nas ferramentas actuais de UML que, na sua maioria, apenas suportam diagramas de casos de utilização onde, no máximo, é possível

adicionar notas UML. A figura 11 representa os conceitos presentes no domínio da modelação aumentada, quando se especificam os requisitos de um sistema. É permitido a criação de casos de utilização com os respectivos cenários associados, que por sua vez são decompostos em vários passos. A um cenário está associado um cenário visual e um diagrama de sequência. Um passo está associado a uma cena (Passo_Visual) que faz parte de um cenário visual. A figura 12 representa o diagrama de actividades deste passo do processo.

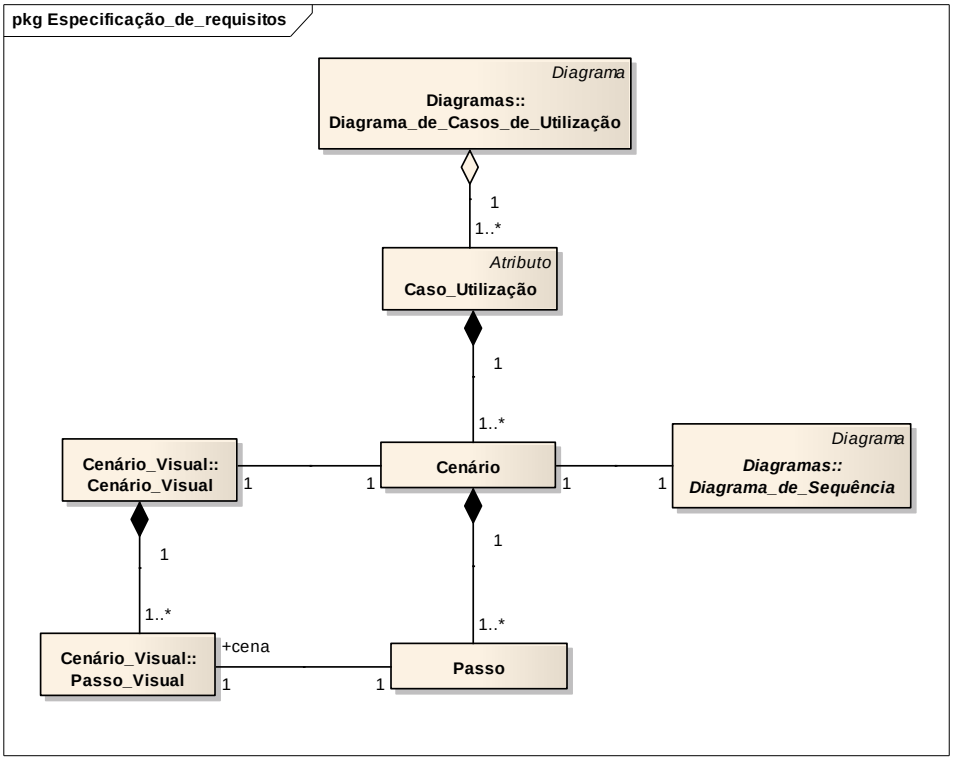

Figura 11 – Fragmento da ontologia do domínio da modelação aumentada (criação de casos de utilização)

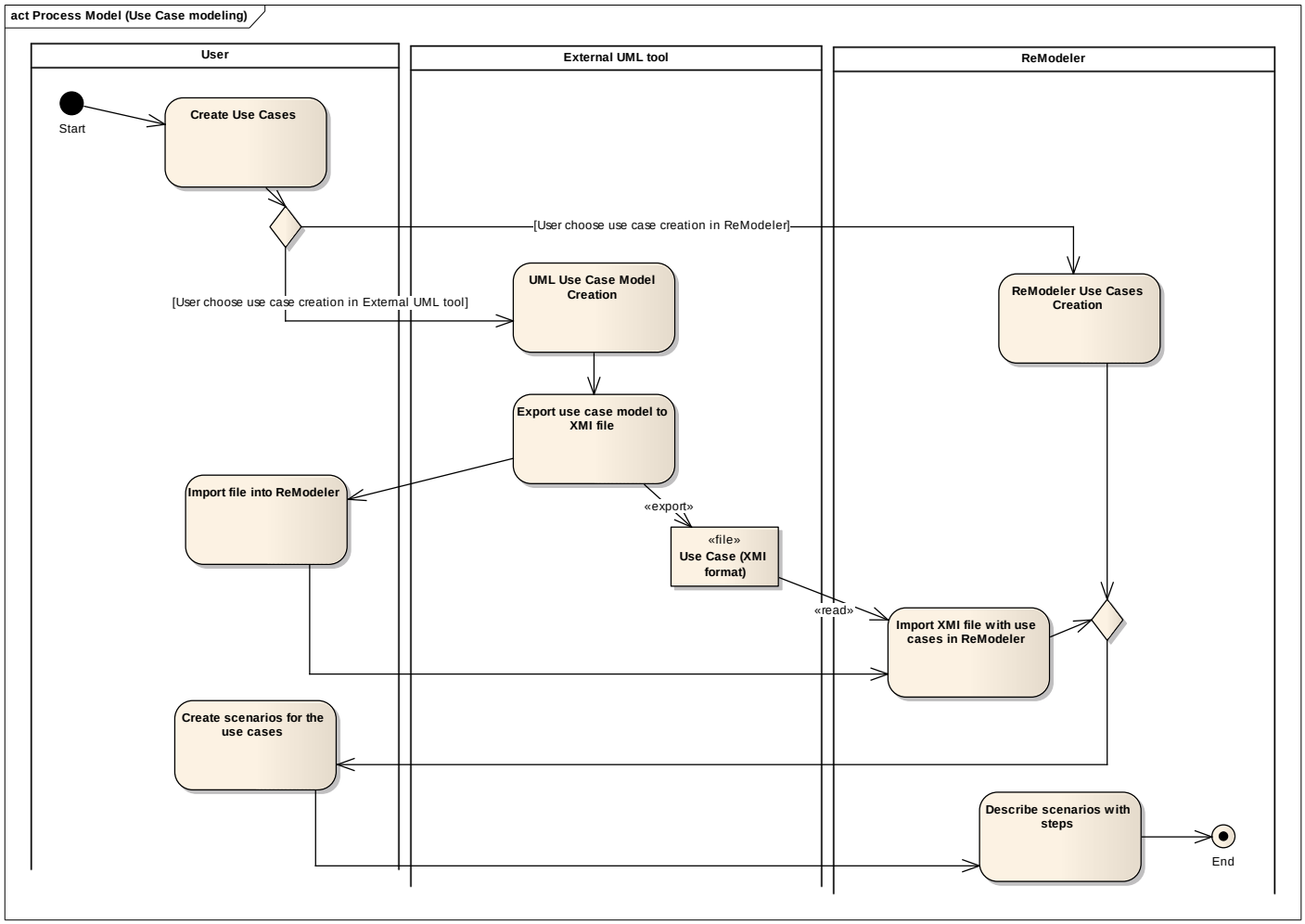

Figura 12 – Criação de casos de utilização

3.2.3. Captura de cenários

O passo seguinte do RMP é a captura dos cenários. Esta pode ocorrer de dois modos distintos. Uma é se estivermos a desenvolver um novo sistema, onde os casos de utilização são desenvolvidos incrementalmente, outra é se já tivermos perante um sistema completamente desenvolvido (sistema legado). No primeiro caso, é possível capturar os cenários de um caso de utilização, mal este esteja desenvolvido e considerado estável. No segundo caso, a captura dos cenários pode ser feita de uma única vez, uma vez que o sistema está completamente desenvolvido.

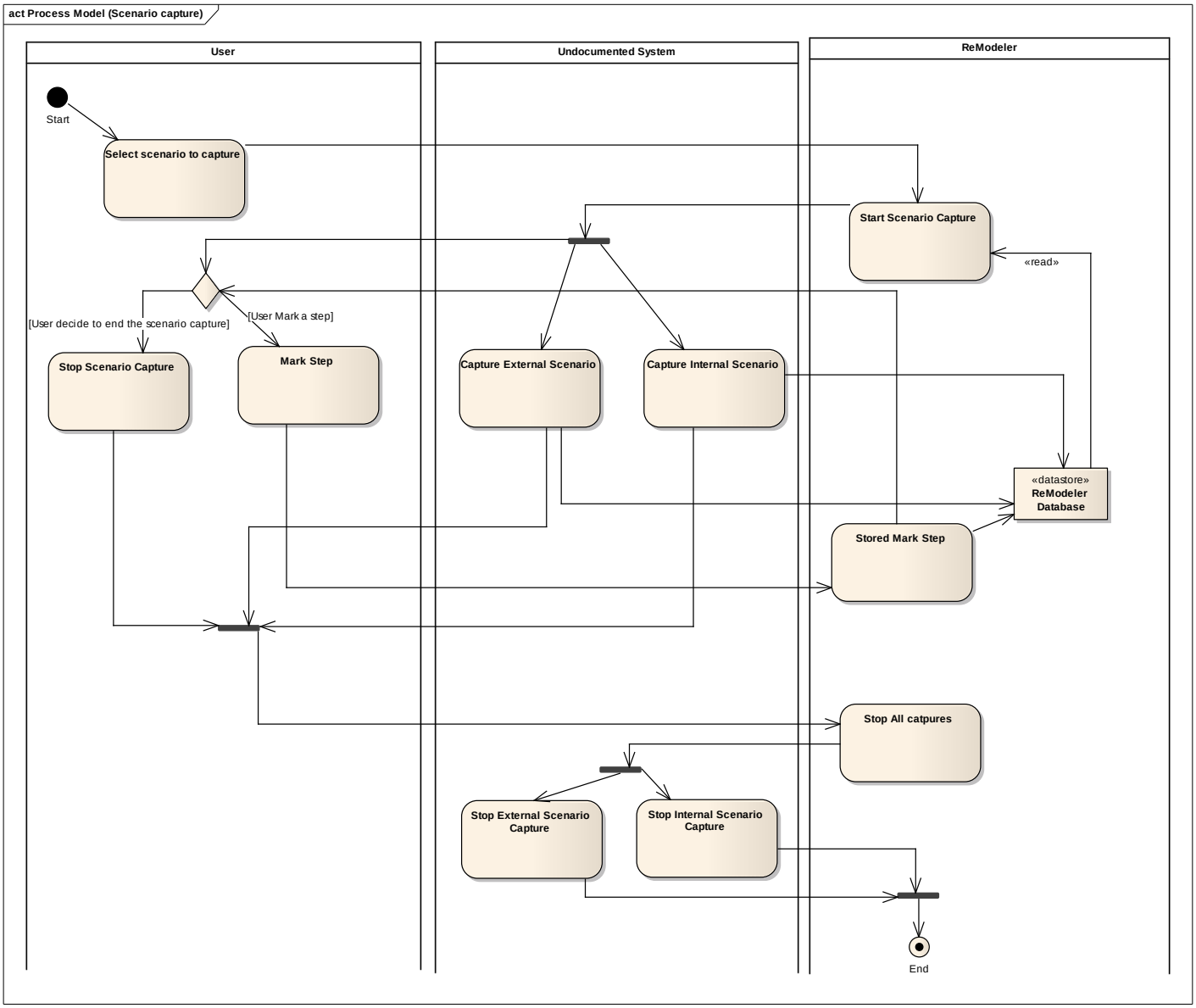

Figura 13 – Processo de captura de cenários

Nesta fase, o perito do domínio vai executar, separadamente, os cenários que descreveu no passo anterior. Durante a sua execução é necessário marcar, passo a passo, os vários passos que compõem o cenário. Esta marcação é feita numa janela disponibilizada pelo *ReModeler*. Enquanto é executado um cenário vai haver duas capturas simultâneas. Uma é composta pela captura das acções externas do utilizador, na forma de imagens. Estas devem conter as janelas que compõem o sistema em execução. Na outra é necessário capturar o funcionamento interno do sistema. Existem várias técnicas para esse efeito [40]. Em [40] descrevem-se outros componentes do sistema *ReModeler*, entre os quais um que através da facilidade de *weaving* dos

aspectos, consegue capturar as invocações que ocorrem entre várias instâncias de classes de um sistema, durante uma execução do mesmo, identificando a altura temporal exacta em que elas ocorreram. Com esta captura simultânea é criada uma ligação directa de rastreabilidade entre os requisitos funcionais do sistema e o seu comportamento real, uma vez que os cenários visuais estão intimamente associados aos casos de utilização. A figura 13 representa o diagrama de actividades deste passo do processo.

3.2.4. Geração dos artefactos da modelação aumentada

Nesta fase do processo, é possível criar os vários artefactos da modelação aumentada com base nos cenários capturados no passo anterior.

3.2.4.1. Criação e visualização das duas perspectivas capturadas (Cenário Visual)

Os cenários visuais são o primeiro artefacto, que neste ponto do processo podem ser gerados. Para se puder criar um cenário visual é necessário apresentar as duas perspectivas anteriormente descritas (externa e interna). Para tal, é preciso existir um *player* que as suporte. Esse *player* para além das funcionalidades básicas que encontramos num *player* de vídeos: reproduzir, parar, pausar, saltar entre cenas, diminuir ou aumentar a velocidade de reprodução e posicionar a reprodução numa altura do tempo, deve conter duas janelas distintas. Na primeira janela deve visualizar-se o filme externo, legendado, e na outra o filme interno. A sincronização entre as duas perspectivas deve ser feita aquando da captura, com o auxílio da marcação temporal que existe nas mensagens internas capturadas. O cenário interno deve estar colorido de modo a se conseguir perceber onde começa e acaba uma cena do filme, ou seja, onde está presente um passo da descrição de um cenário. Por outras palavras, deve usar-se a cor para identificar a extensão de cada passo de um cenário, permitindo a percepção das transacções entre os passos. O processo de visualização deve ser poder ser reproduzido mais tarde, quantas vezes forem necessárias. Este processo é totalmente automático com base nas capturas anteriormente feitas. Ao utilizador, basta pedir ao sistema para reproduzir um determinado cenário visual. A figura 14 representa o diagrama de actividades para a reprodução de um cenário visual.

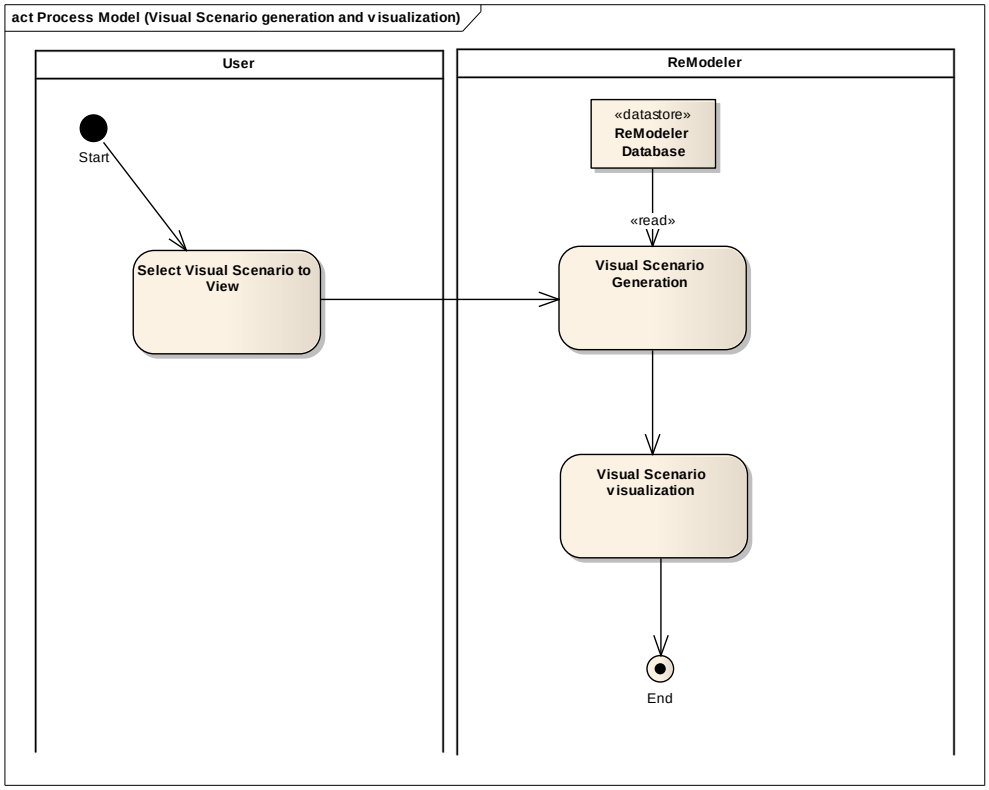

Figura 14 – Diagrama de actividades da criação e visualização de um cenário visual

3.2.4.2. Criação e visualização do manual automático de utilizador

Outro artefacto da modelação aumentada que é possível criar nesta fase do processo é o manual automático de utilizador. Temos duas opções para a sua visualização. A primeira é através de um *player* específico, que permite a navegação nas páginas do manual através de botões, tendo em conta os requisitos anteriormente referidos (capítulos, secções, subsecções e páginas). A segunda é através de um *browser*. O *ReModeler* pode gerar o manual de utilizador em formato HTML [54], com o mesmo conteúdo que aparece no *player*. Esta facilidade garante que a documentação gerada é independente *ReModeler*. Para se obter o manual em qualquer um dos formatos simplesmente tem de se pedir ao *ReModeler* que os gere. A geração é feita com base nos cenários de casos de utilização descritos e nas capturas feitas no *ReModeler*. A figura 15 representa o diagrama de actividades para a geração e visualização do manual automático de utilizador.

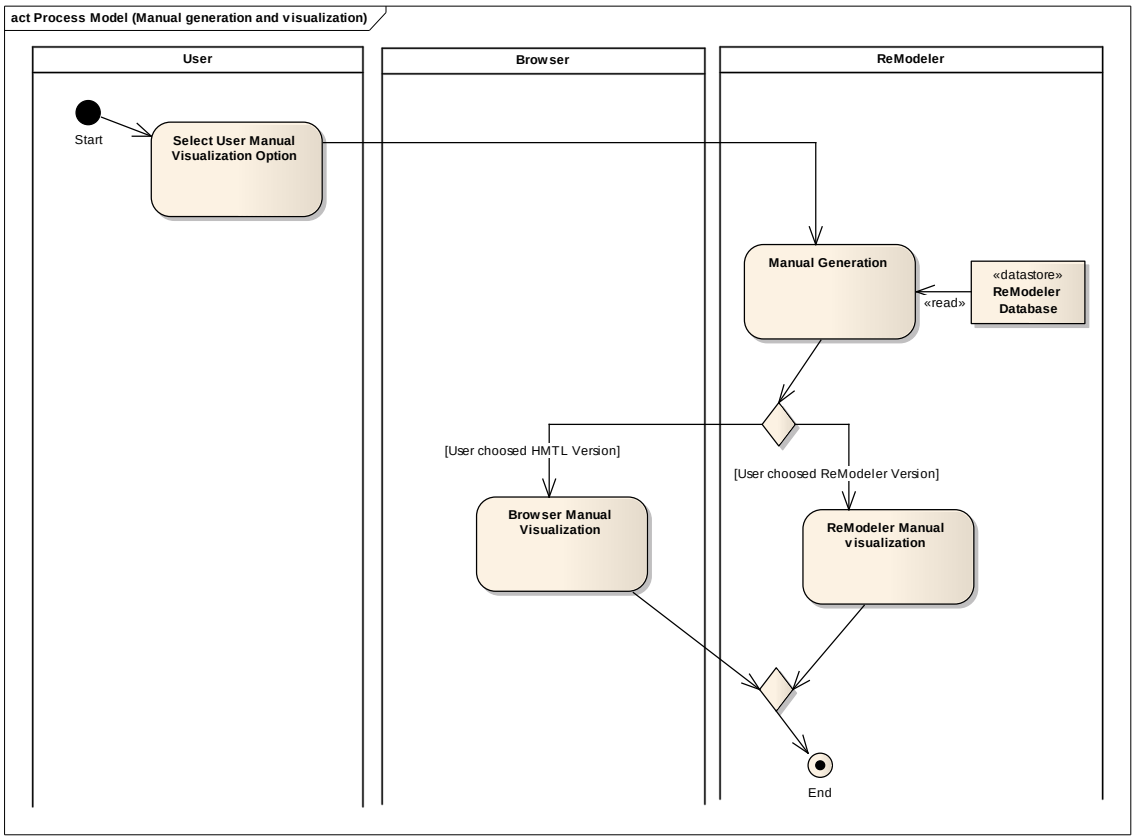

Figura 15 – Diagrama de actividades para a geração e visualização do manual.

3.2.4.3. Criação de baterias de Cenários de teste

O último passo do RMP consiste na criação de baterias de cenários de teste. O *ReModeler* tem a facilidade de adicionar ou remover os cenários dos casos de utilização às baterias de cenários de teste. É de realçar, que é possível ter um cenário em baterias diferentes. Estas baterias vão permitir que posteriormente sejam executados testes de cobertura. A figura 16 representa as actividades que um utilizador pode ter perante a bateria de cenários de teste.

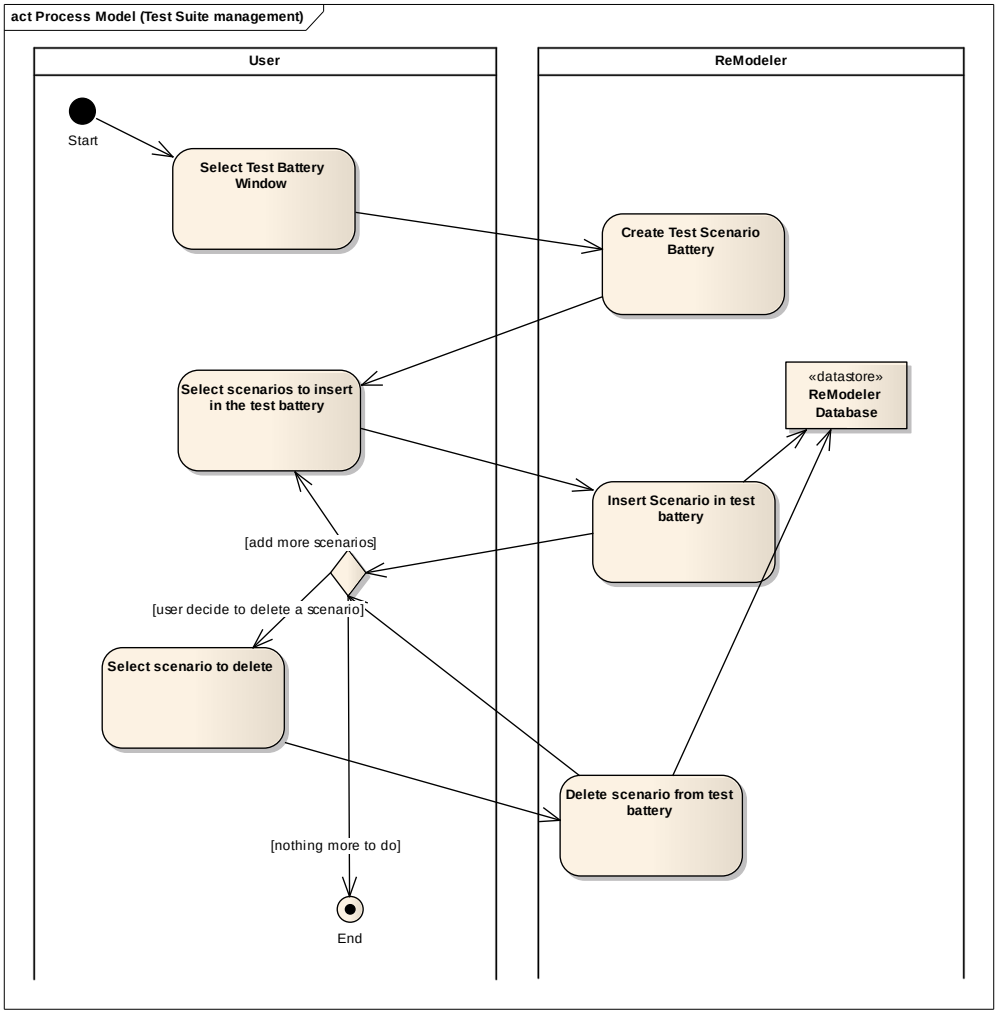

Figura 16 – Diagrama de actividades para a gestão de baterias de cenários de teste

3.3. *ReModeler*

De forma a suportar o RMP foi desenvolvido o *ReModeler*. Este foi realizado em conjunto com uma colega, que desenvolveu outros conceitos, não presentes nesta dissertação, relativos à modelação aumentada [34]. Seguidamente passo a apresentar algumas das funcionalidades que estão presentes no *ReModeler*, que são importantes para o RMP.

3.3.1. Edição de casos de Utilização

A figura 17 representa algumas das opções que estão disponíveis para a edição dos casos de utilização de um sistema.

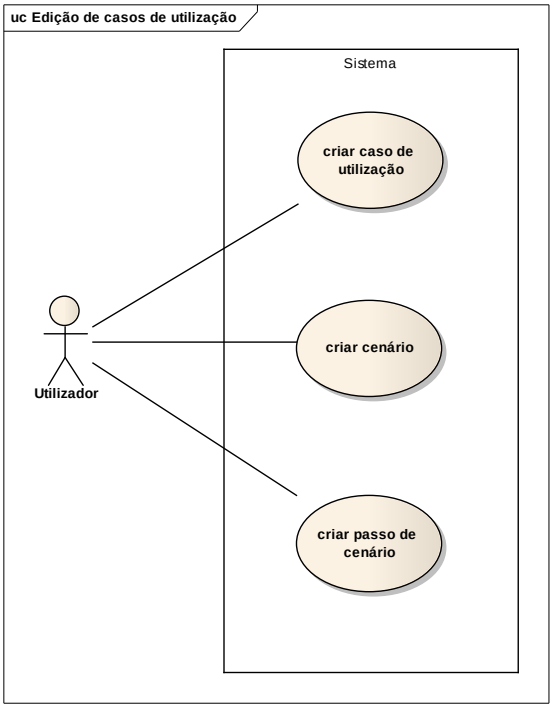

Figura 17 – Casos de utilização para a edição de casos de utilização.

Criar caso de utilização

Cenário principal

1. O caso de utilização começa quando um utilizador selecciona a opção de criar um caso de utilização na árvore de casos de utilização.
2. É um mostrado um formulário para a criação do caso de utilização.
3. O utilizador insere os dados do caso de utilização.
4. O utilizador confirma a criação do caso de utilização através da opção de "ok".
5. O sistema apresenta o caso de utilização criado na árvore de casos de utilização.

Cenário alternativo

1. A qualquer momento o utilizador cancela a criação do caso de utilização, voltando-se ao passo 1.

Criar cenário

Cenário principal

1. O caso de utilização começa quando o utilizador selecciona um caso de utilização e escolhe a opção de criar um cenário.
2. É mostrado na árvore do sistema um novo cenário com a possibilidade de edição do nome.
3. O utilizador edita o nome do novo cenário.
4. O utilizador carrega na tecla "*enter*" confirmando o nome do novo cenário.
5. O sistema mostra o novo cenário criado na árvore dos casos de utilização.

Cenário alternativo

1. Após o passo 4 o sistema mostra uma mensagem de erro "cenário já existente" e regressa ao passo 3.

Criar passo de cenário

Cenário principal

1. O caso de utilização começa quando o utilizador selecciona a opção de editar um cenário.
2. É mostrado um formulário para a criação dos passos de um cenário.
3. O utilizador selecciona a opção de criar um novo passo.
4. O sistema mostra o novo passo.
5. O utilizador edita o novo passo.
6. O utilizador confirma a edição do cenário através da opção "ok".

Cenário alternativo

1. A qualquer altura o utilizador cancela a edição do cenário, voltando-se ao passo 1.

3.3.2. Captura de Cenários (externo e interno)

A figura 18 representa as acções que estão disponíveis aquando da captura dos cenários (interno e externo) de um sistema.

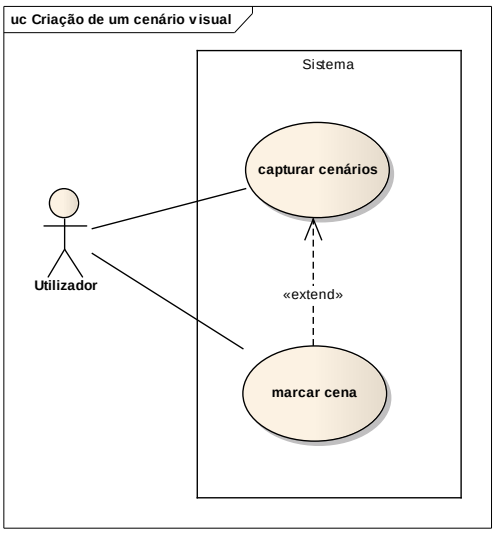

Figura 18 – Casos de utilização para a captura de cenários.

Capturar cenários

Cenário principal

1. A criação de um cenário visual começa quando o utilizador selecciona a opção de “criação de cenário visual”.
2. O sistema mostra um formulário onde se pode indicar o nome do cenário visual e a sua localização no disco rigído.
3. O utilizador selecciona a opção de captura.
4. O sistema começa com a captura externa e interna do sistema.
    a. ***Extend***: Marcar Cena.
5. O utilizador selecciona a opção para parar a captura dos cenários.

Cenário alternativo

1. O utilizador cancela a opção de captura no passo 2, voltando-se ao passo 1.

Marcar cena (Extensão)

Cenário principal

1. O utilizador marca uma cena durante a captura de um cenário visual.

3.3.3. Visualização de um cenário visual

A figura 19 representa as opções que estão disponíveis durante a visualização de um cenário visual

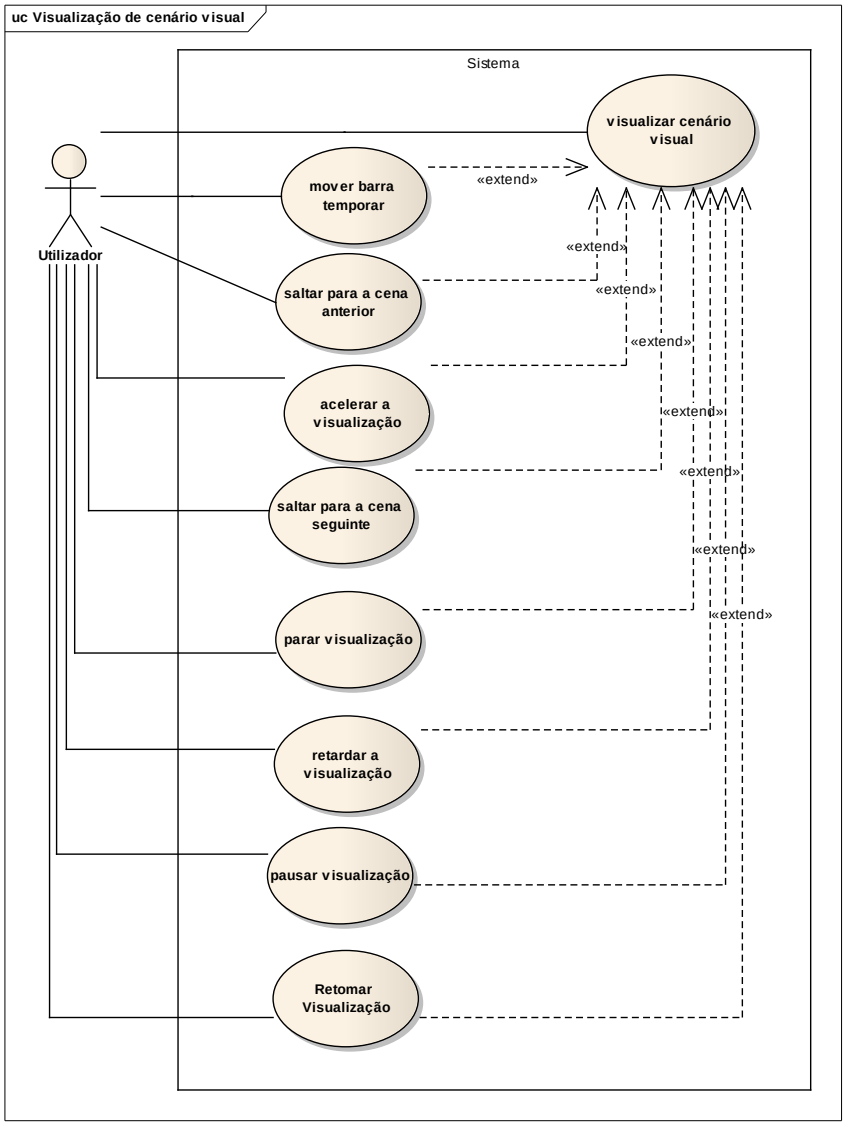

Figura 19 – Casos de utilização para a visualização de cenários visuais.

Visualizar cenário visual

Cenário principal

1. O utilizador selecciona a opção de visualizar um cenário visual na árvore do sistema.
2. O sistema mostra a janela do cenário visual e começa a reprodução do mesmo.
3. Se durante a reprodução de um cenário visual o utilizador move a barra temporal.
    a. ***Extend***: mover barra temporal.
4. Se durante a reprodução de um cenário visual o utilizador selecciona a opção de diminuir a velocidade de reprodução.
    a. ***Extend****:* retardar a visualização.
5. Se durante a reprodução de um cenário visual o utilizador selecciona a opção de aumentar a velocidade de reprodução.
    a. ***Extend***: acelerar a visualização.
6. Se o utilizador selecciona a opção de pausar a reprodução de um cenário visual.
    a. ***Extend***: pausar a visualização.
7. Se o utilizador selecciona a opção de retomar a reprodução de cenário visual durante uma pausa.
    a. ***Extend****:* retomar visualização.
8. Se durante a reprodução de um cenário visual o utilizador selecciona a opção de saltar para a cena anterior.
    a. ***Extend***: saltar para a cena anterior.
9. Se durante a reprodução de um cenário visual o utilizador selecciona a opção de saltar para a cena seguinte.
    a. ***Extend****:* salta para a cena seguinte.
10. Se o utilizador selecciona a opção de parar a reprodução de um cenário visual.
    a. ***Extend****:* parar visualização.

Cenário alternativo

1. No passo 2 o utilizador cancela a reprodução de um cenário visual.

Saltar para a cena anterior (Extensão)

Cenário principal

1. O sistema passa a reprodução para a cena anterior.

Saltar para a cena seguinte (Extensão)

Cenário principal

1. O sistema passa a reprodução para a cena seguinte.

Parar a visualização (Extensão)

1. O sistema pára a reprodução de um cenário visual.

Pausar a visualização (Extensão)

Cenário principal

1. O sistema pausa a reprodução de um cenário visual.

Acelerar a visualização (Extensão)

Cenário principal

1. O sistema duplica a velocidade de reprodução.

Retardar a visualização (Extensão)

Cenário principal

1. O sistema diminui para metade a velocidade de reprodução.

Mover barra temporal (Extensão)

Cenário principal

1. O sistema identifica o novo tempo da barra e salta para esse tempo na reprodução do cenário visual.

Retomar Visualização (Extensão)

Cenário Principal

1. O sistema retoma a reprodução de um cenário visual.

3.3.4. Visualização do manual automático de utilizador

A figura 20 representa as acções mínimas que estão disponíveis para o manual automático

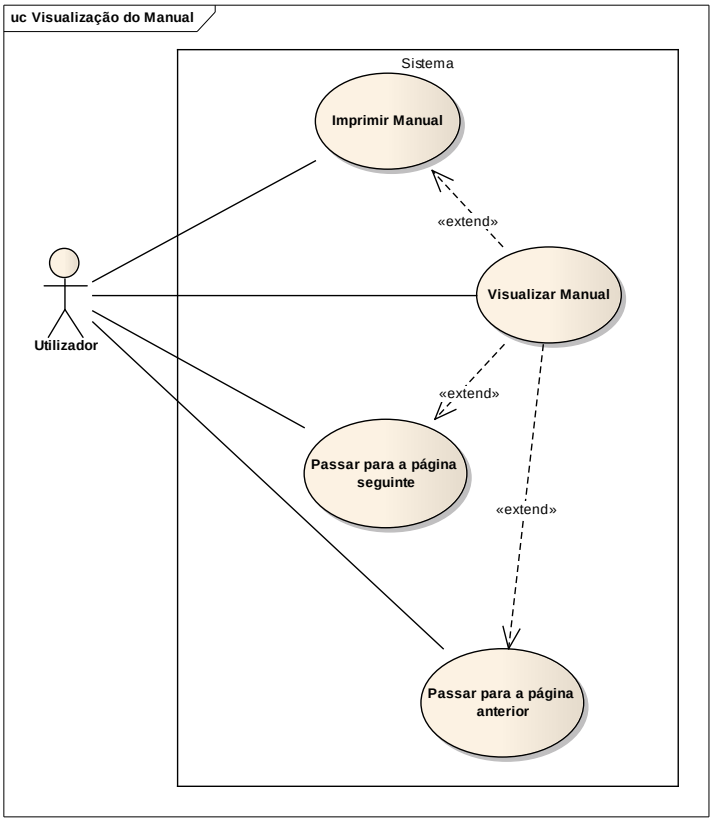

Figura 20 – Casos de utilização para o manual de utilizador.

Visualizar Manual

Cenário principal

1. O utilizador selecciona a opção de visualizar o manual automático do sistema.
2. O sistema mostra a janela do manual automático.
3. Se o utilizador selecciona a opção de visualizar a página seguinte do manual.

a. ***Extend*:** passar para a página seguinte.

4. Se o utilizador selecciona a opção de visualizar a página anterior do manual.
    a. ***Extend:*** passar para a página anterior.
1. Se o utilizador selecciona a opção de imprimir o manual
    a. ***Extend:*** imprimir manual.

Cenário alternativo

1. No passo 2 o sistema não consegue mostrar o manual porque não existem cenários visuais para o sistema.

Passar para a página seguinte (extensão)

Cenário principal

1. O sistema passa para a página seguinte do manual.

Cenário alternativo

1. Não existe página seguinte, o manual fica na última página.

Passar para a página anterior (extensão)

Cenário principal

1. O sistema passa para a página anterior do manual.

Cenário alternativo

2. Não existe página seguinte, o manual fica na primeira página.

Imprimir Manual (Extensão)

1. O sistema imprime o manual em formato HTML.

3.4. Utilização do RMP e dos artefactos da Modelação Aumentada

A rastreabilidade dos requisitos e a documentação de sistemas apresentam-se como um problema actual que está longe de ser resolvido. Os modelos, como já referido, são simplificações da realidade que visam auxiliar o desenvolvimento do

software. A linguagem de modelação mais usada actualmente é o UML e a sua introdução em todas as fases do ciclo de vida, poderá melhorar a nossa compreensão sobre os vários problemas que afectam o desenvolvimento de software.

Através do uso de cenários visuais é possível elevar o nível de abstracção da análise de um sistema, habitualmente feita apenas ao nível do código fonte (*debugging*, *profiling*), para o nível dos modelos. Assim espera-se melhorar a rastreabilidade entre uma aplicação executável e o seu modelo arquitectural. Por exemplo, seria relativamente fácil a adopção de cenários visuais num processo de desenvolvimento de sistemas como o *Rational Unified Process* (RUP) [41].

O RUP é um processo da engenharia de software que reúne as melhores práticas no desenvolvimento de software e que serve de guia às organizações que o usam. Assenta numa abordagem orientada aos objectos com uma estrutura regular e bem definida, mas permitindo às organizações que o implementam a possibilidade de modificá-lo e adaptá-lo às suas necessidades. O RUP estabelece duas dimensões: horizontal e vertical. A dimensão horizontal representa o tempo e os aspectos do ciclo de vida à medida que se vão desenvolvendo. A dimensão vertical agrupa as actividades de engenharia do software pela sua natureza e representa as várias disciplinas que constituem o processo. Em suma, a dimensão horizontal representa os aspectos diacrónicos do processo na forma de ciclos, fases, iterações, enquanto que a segunda dimensão (vertical) representa os aspectos sincrónicos do processo, isto é, mostra que actividades são desenvolvidas em simultâneo.

Um produto desenvolvido com o RUP passa por uma série de iterações, que permitem a sua validação, desde o início do ciclo de vida até à sua conclusão. É durante estas iterações que a utilização dos cenários visuais poderia dar um valioso contributo. Em qualquer altura se poderia gerar um cenário visual e, através da sincronização das suas vistas, perceber como um determinado cenário está implementado. Com essa expansão de conhecimento facilmente se poderia propor alterações, caso fosse necessário. Não só se poderiam adoptar os cenários visuais, como se poderia adoptar o manual automático de utilizador e as baterias de cenários de teste. Os casos de utilização no RUP estabelecem uma ligação entre os requisitos e o resto dos artefactos desenvolvidos, portanto as baterias de cenários de teste poderiam ser definidas logo numa fase inicial do ciclo de vida do processo, através dos casos de utilização. Mais

tarde, poderiam ser feitos testes ao sistema de acordo com as baterias de cenário de teste inicialmente definidas. Por último, o manual automático de utilizador podia ser produzido para documentar as diferentes iterações, que ocorreram durante o ciclo de vida. Na figura 21 está a sombreado a altura do ciclo do RUP onde seria possível fazer uma adopção dos conceitos anteriormente referidos.

O ciclo de vida do RUP é um exemplo de como poderiam ser utilizados os cenários visuais, manuais automáticos de utilizador e baterias de cenários de teste, mas a sua aplicação estende-se para lá do RUP. A abrangência dos conceitos anteriormente referidos é tão vasta que estes podem e devem ser utilizados sempre que se queira perceber um sistema. Por exemplo, os cenários visuais podem ser utilizados em situações em que se queira perceber como um requisito está implementado. O enriquecimento de especificações de requisitos pode ser feito, de uma maneira automática, através dos manuais de utilizador, e as baterias de teste promovem a criação de vários tipos de teste, de acordo com as perspectivas que existem de um sistema. A tabela 3 mostra, resumidamente, o interesse potencial que cada artefacto da modelação aumentada tem para os vários *stakeholders*, que normalmente estão envolvidos no desenvolvimento de sistemas de informação.

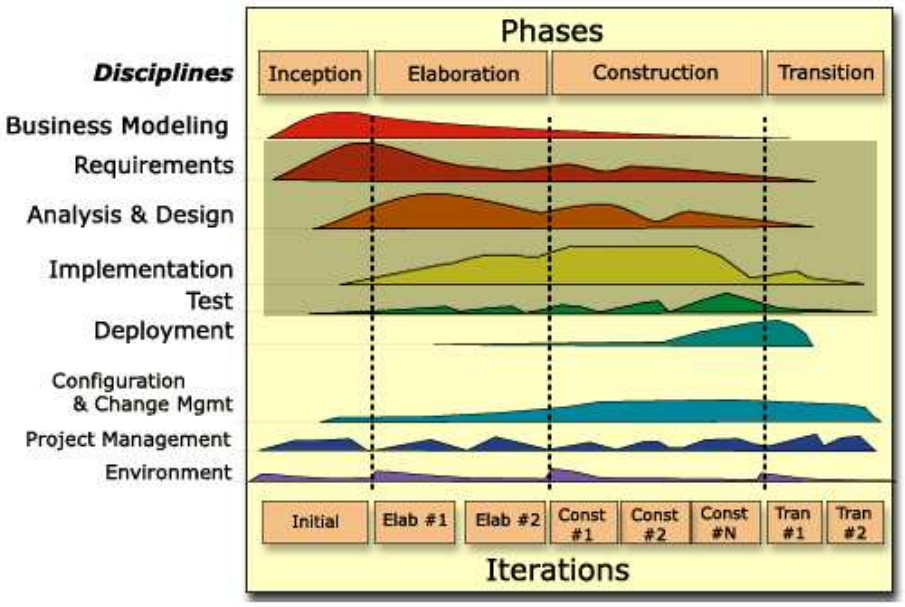

Figura 21 - A utilização da modelação aumentada no RUP

Um cliente pode usar a combinação do manual de utilizador com os cenários visuais para saber quais os passos que tem de executar num determinado cenário, de forma a concretizar o caso de utilização correspondente. A validade de um caso de

utilização pode ser posta à prova, perante um cliente, durante a produção do sistema. O uso dos cenários visuais vai demonstrar como se comporta o sistema num determinado cenário. Um cliente pode rejeitar ou aceitar esse comportamento, em virtude de este corresponder ou não ao desejado. Um engenheiro de requisitos, com o auxílio dos cenários visuais e/ou o manual, facilmente enriquece uma especificação de requisitos, que esteja a desenvolver e simultaneamente cria baterias de cenários de teste consoante a sua visão do sistema. Um gestor de projectos, através dos cenários visuais e do manual rapidamente percebe como está o andamento do projecto e simultaneamente tem uma ideia do impacto que determinadas alterações terão no sistema. Um programador, para alterar facilmente um requisito, precisará dos cenários visuais, para perceber como este está implementado. A sua visão (baixo nível) permite a definição de baterias de cenários (baixo nível) de forma a validar o sistema em implementação. As equipas de teste podem usar as baterias de cenários de teste para efectuar testes caixa preta sobre o sistema e com o auxílio dos cenários visuais, replicar exactamente as acções do utilizador, quando este está a executar uma funcionalidade do sistema. Assim se mostrou como a modelação aumentada tenta trazer a modelação para o todo ciclo de vida do desenvolvimento do software abrangendo os vários intervenientes do mesmo.

Utilizadores interessados	**Artefactos da Modelação Aumentada**		
	Cenários Visuais	Manual Automático Utilizador	Baterias de Cenário de Teste
Cliente	√	√	
Engenheiro de Requisitos/ Analista	√	√	√
Gestor de Projectos	√	√	√
Programador/Engenheiro de Software	√		√
Equipa de testes	√		√

Tabela 3 – Possíveis interessados na modelação aumentada.

3.4.1. Utilização do RMP em Sistemas Legados e no desenvolvimento de Sistemas de Informação

O RMP foi desenvolvido a pensar em sistemas legados e na recuperação da sua documentação, mas este pode ser aplicado a sistemas em desenvolvimento. Por exemplo, a criação de casos de utilização e respectivas descrições de cenários podem ser feitas aquando da criação das especificações de requisitos de um novo sistema. A criação de cenários visuais pode ser feita durante o desenvolvimento do sistema, após a programação de um requisito. Este pode ser imediatamente capturado e demonstrado ao utilizador final do sistema. O manual automático é facilmente gerado sempre que necessário, não sendo preciso ter o sistema completamente desenvolvido. A sua criação vai ficando mais completa à medida que um sistema vai ficando desenvolvido. Por último, as baterias de cenários de teste que entram na fase dos testes de um sistema, podem ser definidas durante a fase de criação dos requisitos do sistema. Desta forma, é possível usar o RMP e a modelação aumentada sobre um ciclo de vida de desenvolvimento incremental/iterativo. Num sistema legado a criação do manual e dos cenários visuais é mais imediata por este já estar completamente desenvolvido.

Capítulo 4

A arquitectura do *ReModeler*

Conteúdo

Neste capítulo é descrita a arquitectura do *ReModeler*. Inicialmente é apresentado o protótipo como se fosse uma caixa preta, ou seja, são descritos os seus *inputs* e *outputs*. Seguidamente, o protótipo é apresentado como uma caixa branca, ou seja, passamos a conhecer os seus componentes e os pacotes necessários à sua implementação

4. A arquitectura do *ReModeler*

No capítulo anterior foi apresentado um processo suportado por uma ferramenta, que utiliza os conceitos da modelação aumentada para a documentação automática de sistemas. Neste capítulo vão ser apresentados os componentes que fazem parte arquitectura da ferramenta e a respectiva implementação por pacotes. O *ReModeler* contém vários conceitos da modelação aumentada, que vão para além dos descritos nesta dissertação. Estão também presentes na sua implementação os conceitos descritos em [34]. Os diagramas apresentados neste capítulo são constituídos por elementos de duas cores: a azul são os módulos desenvolvidos por mim que implementam os conceitos presentes nesta dissertação e a cor-de-rosa são os módulos que foram desenvolvidos no âmbito de outra dissertação [40].

4.1. *ReModeler Inputs e Outputs*

A figura 22 mostra os artefactos, identificados pela cor azul e cor-de-rosa, da modelação aumentada que são produzidos pelo *ReModeler*, identificando a verde os dados recebidos e necessários para os produzir. O sistema legado faz parte dos dados recebidos pelo *ReModeler* para produzir a sua documentação. Ao contrário de muitas técnicas presentes no estado da arte actual, não é o código fonte que é um *input* de entrada, mas sim a execução do próprio sistema. Para tal, vai ser necessário ligar os dois sistemas de uma forma quase "simbiótica" através da compilação simultânea de ambos [40]. O *ReModeler* permite ainda a importação de dois diagramas UML (casos de utilização e classes) produzidos numa qualquer ferramenta UML, que exporte os modelos em formato XMI [39]. É deixando ao critério do utilizador criar os casos de utilização no *ReModeler* ou importar de outra ferramenta. A importação de diagramas

de classes é necessária para os outros artefactos da modelação aumentada [34]. Nos documentos exportados pelo *ReModeler* podemos encontrar o manual do utilizador, os filmes dos cenários visuais e ainda um conjunto de outros artefactos [34].

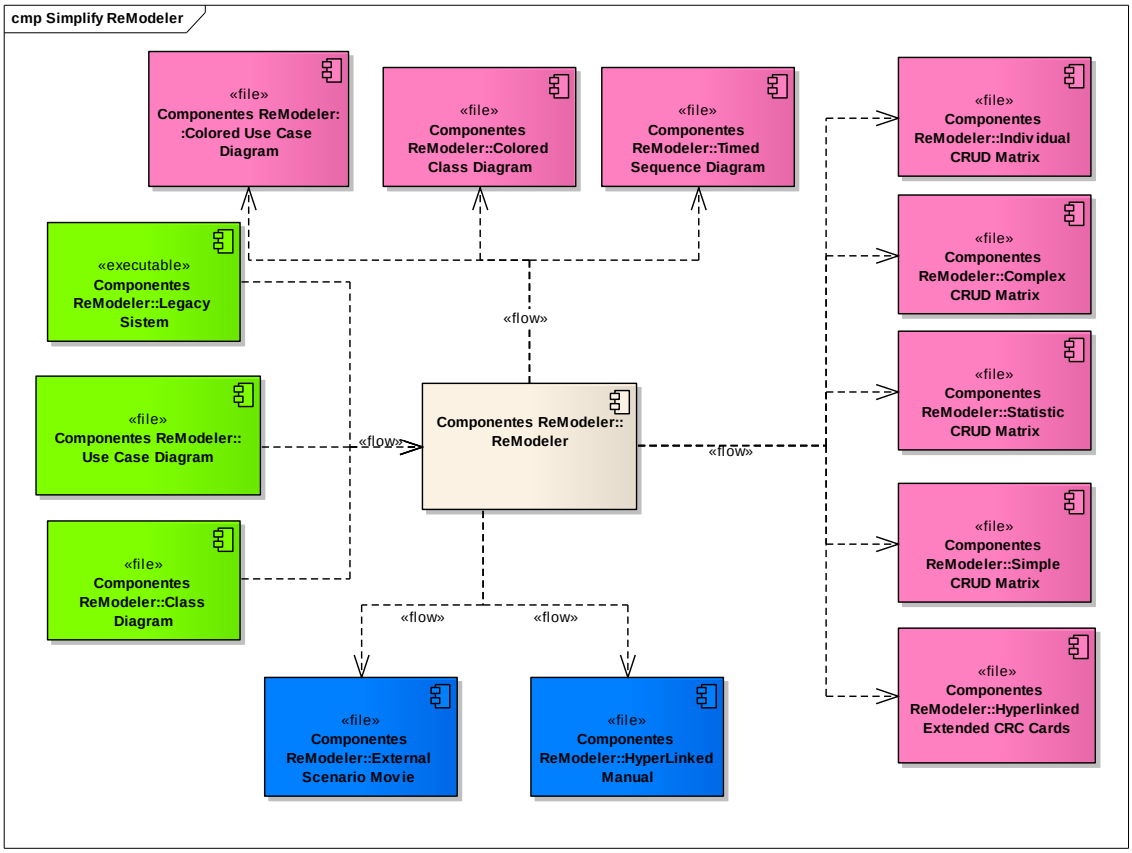

Figura 22 – Os *inputs* e *outputs* do *ReModeler*.

Na perspectiva da figura 22 anterior apenas se pode ver o que é recebido e produzido pelo *ReModeler*, ou seja, estamos perante uma perspectiva de caixa preta. Na figura 23 podemos ver a perspectiva de caixa branca, ou seja, podemos ver os componentes internos do *ReModeler*. É importante referir que o *ReModeler* funciona com uma base de dados [40], identificada pela cor amarela. Todos os dados inseridos, capturados e gerados, são guardados nessa base de dados.

Os componentes que a seguir se descrevem correspondem às várias funcionalidades que estão no *ReModeler*. A sua apresentação irá ser feita ao longo deste capítulo, juntamente com os seus pacotes. Os nomes dos componentes e dos pacotes aparecem escritos em Inglês porque o *ReModeler* foi desenvolvido nessa língua.

4.2. Diagrama de Componentes do *ReModeler*

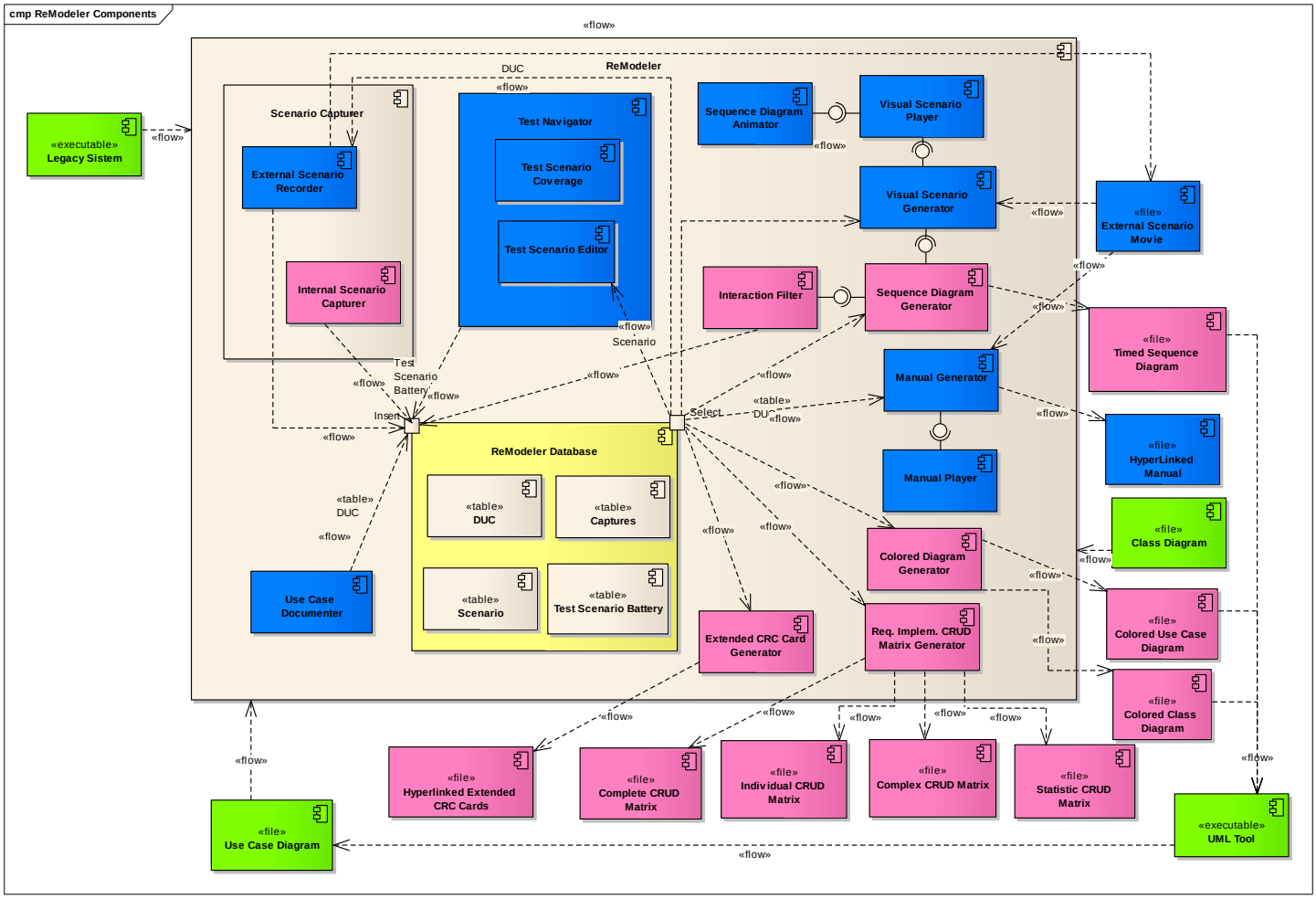

Figura 23 – Diagrama de componente do *ReModeler*

4.2.1. Componentes do Cenário Visual

Nesta secção vão descritos os componentes responsáveis pela geração e reprodução dos cenários visuais.

4.2.1.1. Desenho dos diagramas de sequência (*SequenceDiagramAnimator*)

A animação dos diagramas de sequência é uma parte importante dos cenários visuais. É através destes diagramas que se percebe como funciona a troca de mensagens entre os objectos do sistema. Foi, por isso, desenvolvido um componente (*SequenceDiagramAnimator)* para efectuar o desenho de diagramas de sequência. O pacote *SeqDiagPlayerLibrary* representa a biblioteca responsável pelo desenho de diagramas de sequência. Esta biblioteca interpreta uma linguagem própria, através da

qual gera um diagrama de sequência. Essa linguagem tem de ser gerada a partir das capturas presentes na base de dados. O componente responsável pela geração dessa linguagem é o *SequenceDiagramGenerator* [40].

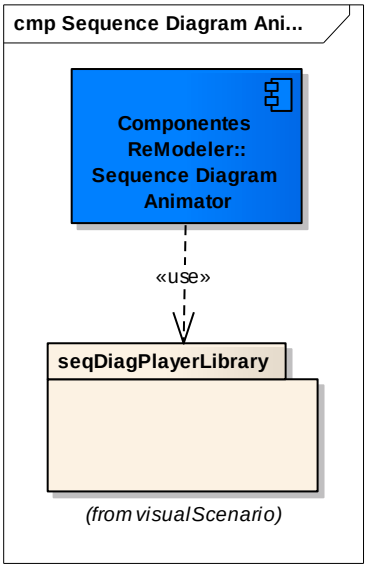

Figura 24 – Pacote do componente responsável pelo desenho dos diagramas de sequência.

4.2.1.2. Captura de Cenários Externos (*ExternalScenarioRecorder*)

Num cenário visual, um cenário externo é representado por um filme, que contém as acções de um utilizador durante a execução do mesmo. A criação do cenário externo é constituída por duas acções diferentes. Numa são capturadas as várias imagens do sistema durante a execução de um cenário. Na outra são marcados os passos que constituem o cenário. A produção do cenário externo está a cargo do componente *ExternalScenarioRecorder*.

A figura 25 contém os quatro pacotes que constituem este componente. Para iniciar a gravação de um cenário externo é necessário um formulário, que está presente no pacote *systemDialogForms*. O pacote *externalScenarioRecorder* é o responsável pela gravação do filme e de passar as marcações do utilizador ao pacote *databaseUpperController*, que é o responsável pela escrita na base de dados dos passos marcados durante a captura de um cenário externo. Temos também o pacote *utils* que irá aparecer junto de todos os componentes seguidamente apresentados, pois este pacote contém uma serie de utilitários necessários para a execução do *ReModeler*.

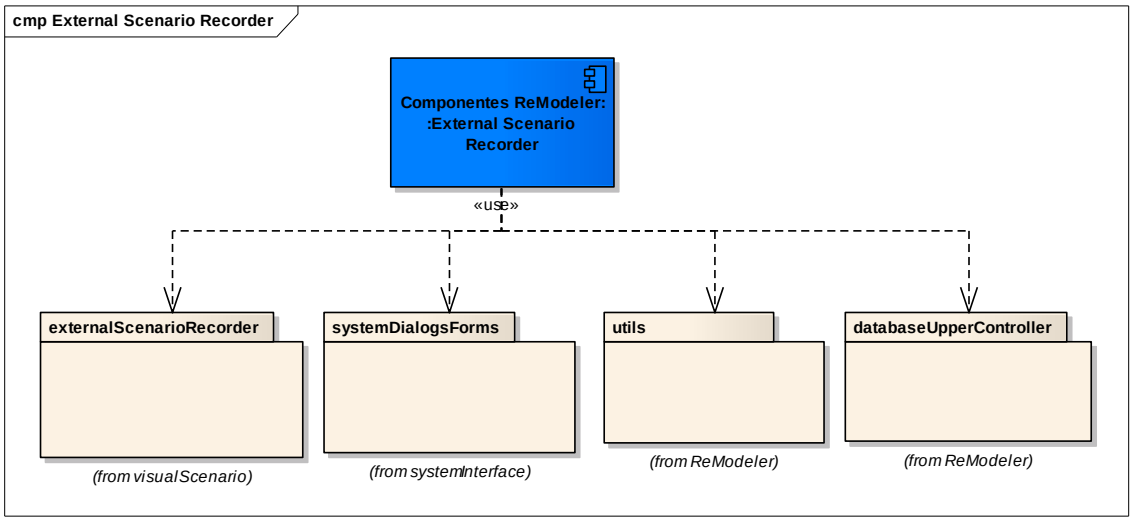

Figura 25 – Pacotes do componente de captura de cenários externos.

4.2.1.3. Geração de Cenários Visuais (*Visual Scenario Generator*)

O componente *VisualScenarioGenerator* é responsável pelo processo de geração de cenários visuais. Os cenários visuais são compostos por duas perspectivas do sistema: a perspectiva externa e interna. A perspectiva externa é concretizada por um filme, que contém as interacções do utilizador de um sistema A perspectiva interna é representada pelo diagrama de sequência, que contém as mensagens trocadas pelos objectos do sistema, aquando da captura de um cenário. Neste componente vão ser armazenados os comandos da linguagem dos cenários internos e as legendas do cenário externo. Os comandos do cenário interno são fornecidos pelo pacote XMI e são resultado da captura feita pelo componente *InternalScenarioCapturer* [40]. As legendas do cenário externo vêm directamente da base de dados do *ReModeler*, após a captura feita pelo componente *externlaScenarioRecorder*. Este componente tem ainda a responsabilidade de durante a reprodução de um cenário visual, fornecer os comandos e/ou as legendas para um determinado instante. O pacote principal deste componente é o *VisualScenarioGenerator*. Este contém a interface para se efectuarem invocações, acerca das informações que estão guardadas, sobre um cenário visual (legendas do cenário externo e comandos do cenário interno). A figura 26 contém os pacotes que fazem deste componente.

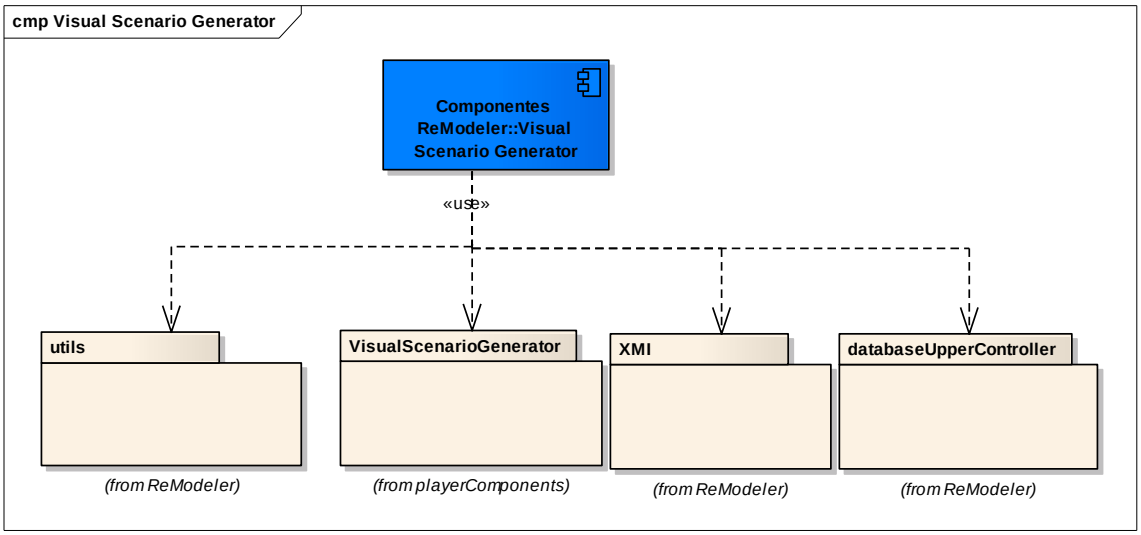

Figura 26 – Pacotes do componente responsável pela geração de cenários visuais

4.2.1.4. Player dos Cenários Visuais (*Visual Scenario Player*)

O *player* de cenários visuais é responsável por disponibilizar ao utilizador as funcionalidades básicas encontradas em qualquer *player* de vídeos: aumentar ou diminuir a velocidade de reprodução, saltar entre várias cenas, pausar e parar reprodução. O pacote responsável pelo *player* é o *playerComponents*. Neste pacote é que se inicia a reprodução de um cenário visual e são coordenados os seus cenários (externo e interno). Durante a reprodução de um cenário externo, capturado previamente pelo *ExternalScenarioRecorder*, vai ser requisitado ao *VisualScenarioGenerator* os comandos do cenário interno e as legendas do cenário externo. Os comandos do cenário interno são posteriormente fornecidos ao componente *SequenceDiagramaAnimator*, responsável pelo desenho dos diagramas de sequência. As legendas do cenário externo são mostradas juntamente com a apresentação do cenário visual. Este componente é uma possível interface gráfica de um *player* de cenários visuais. A figura 27 contém os pacotes deste componente.

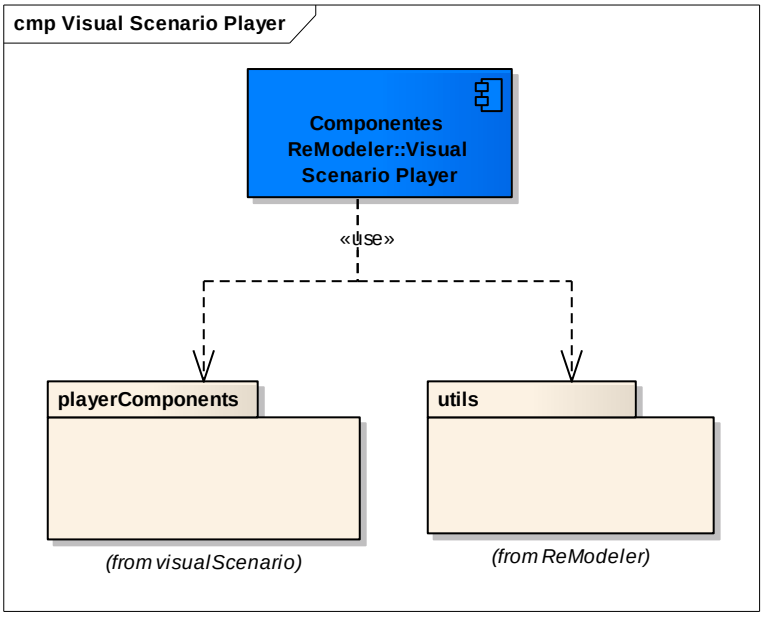

Figura 27 – Pacotes do componente responsável pela reprodução dos cenários visuais

4.2.2. Editor de Casos de Utilização (*Use Case Documenter*)

De modo a facilitar a documentação de sistemas, o *ReModeler* suporta funcionalidades de criação, edição e importação de casos de utilização. A criação e edição de casos de utilização pode ser feita no próprio *ReModeler*, enquanto que a importação de casos de utilização é possível através de diagramas UML em formato XMI, exportados por alguma ferramenta de UML externa ao sistema. Os casos de utilização irão aparecer graficamente numa estrutura em árvore, permitindo a criação dos respectivos dos cenários e suas descrições.

A figura 28 mostra os pacotes necessários à implementação do editor de casos de utilização. O pacote *systemInterface* contém os pacotes da interface gráfica do editor. É neste pacote que se encontra o sistema de janelas do *ReModeler*, os componentes da interface feitos à medida e as janelas de diálogo que interagem com o utilizador. O pacote *databaseUpperController* é o responsável por guardar as informações sobre os casos de utilização na base de dados.

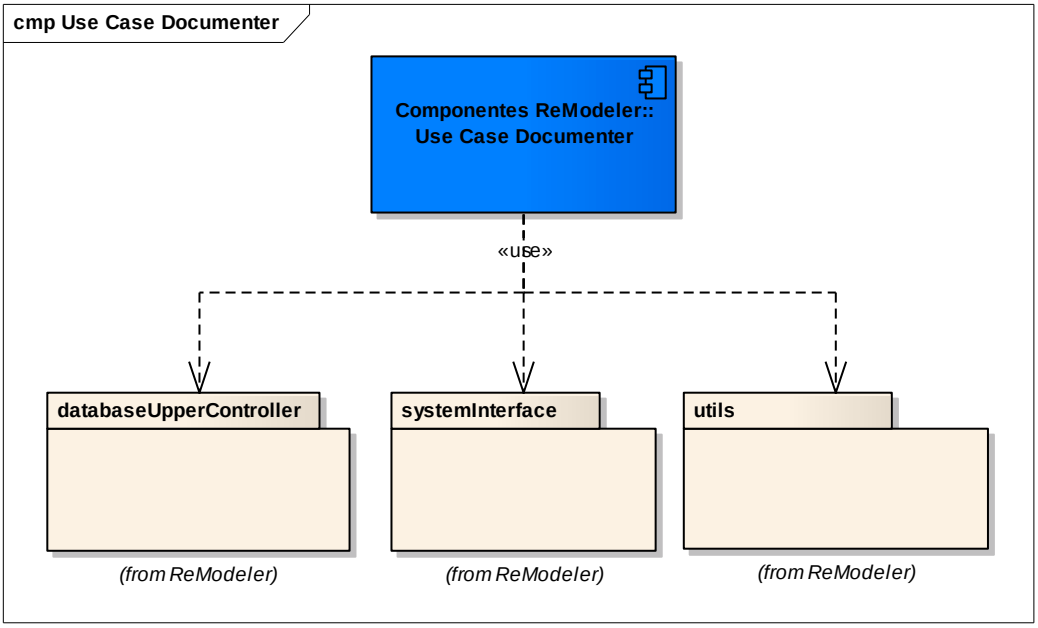

Figura 28 – Pacotes do *ReModeler Editor*.

4.2.3. Gerador do Manual automático de Utilizador (*Manual Generator*)

A geração do manual de utilizador vai depender de dados que foram gerados previamente por outros componentes do *ReModeler*. Portanto, foi necessário desenvolver um componente que fizesse a recolha desses dados para serem disponibilizados na forma do manual de utilizador. Foi criado o componente *ManualGenerator* que é responsável pela geração do manual de utilizador.

A figura 29 contém os pacotes necessários para a geração do manual de utilizador. O pacote *ManualGenerator*, juntamente com o pacote *databaseUpperController*, é responsável por ir recolher as informações ao sistema que devem estar no manual de utilizador: informações sobre os casos de utilização, sobre os cenários e as imagens dos cenários visuais. O pacote *ManualElements* é responsável por guardar em memória as informações relativas ao manual. Este cria virtualmente um manual de utilizador, que pode ser pesquisado pelo componente *ManualPlayer*.

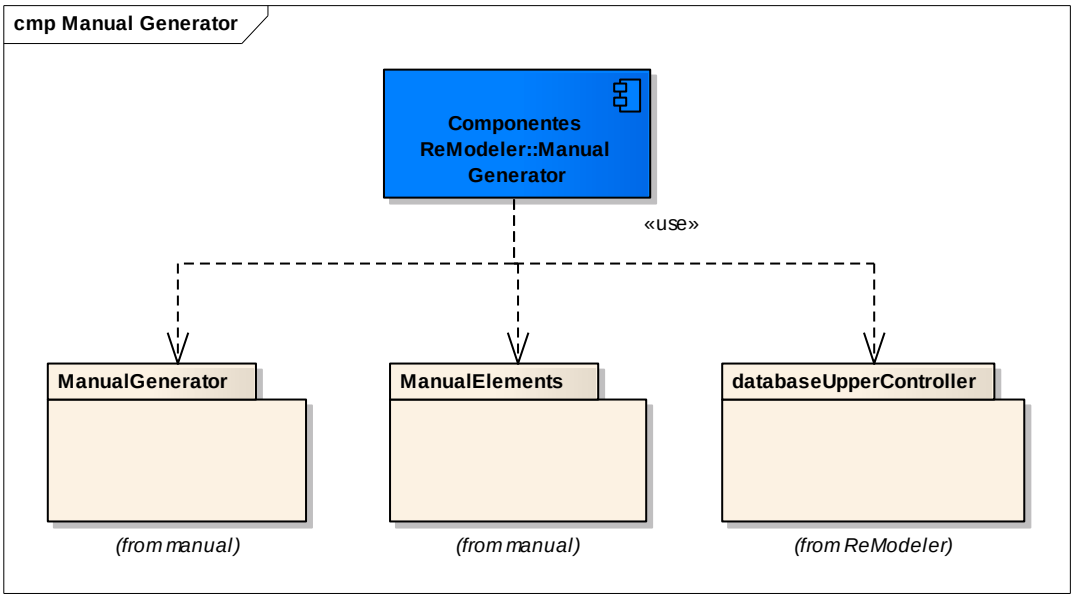

Figura 29 - Pacotes do componente responsável pela geração do manual de utilizador

4.2.4. Manual Player (*ManualPlayer*)

Para interagir sobre o manual gerado pelo *ManualGenerator* e mostrar as várias páginas que o constituem, é necessário um *player*. O componente *ManualPlayer* representa a interface gráfica responsável pelo sistema de janelas do manual de utilizador interactivo. Este componente é constituído pelo pacote *ManualPlayer* onde estão localizados os componentes da interface gráfica e o pacote *ManualBrowser* que serve para navegar entre as páginas do manual. Este componente é também responsável pela criação do formato HTML, com base no manual gerado. A figura 30 mostra os pacotes necessários à implementação deste componente.

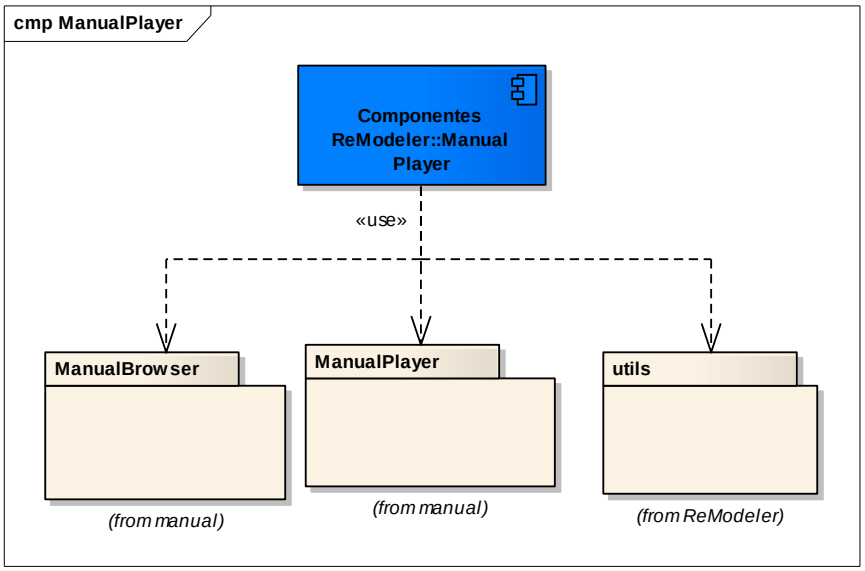

Figura 30 – Pacotes do componente responsável pela visualização do manual de utilizador

4.2.5. Editor de Cenários de teste (*TestScenarioEditor)*

A figura 31 mostra os pacotes que constituem o componente de edição de cenários de teste (*TestScenarioEditor*). As opções disponíveis no editor de cenários de teste são comuns à do editor de casos de utilização, com a facilidade de se puder agrupar cenários de teste em baterias de cenários de teste. A interface gráfica do editor é também comum à dos casos de utilização, isto é, uma estrutura em árvore, agrupada por baterias de cenários de teste.

À semelhança do editor de casos de utilização, o pacote responsável pelo editor de cenários de teste é o pacote que contém a interface gráfica do *ReModeler* (*systemInterface*). Mais uma vez temos presente o pacote *databaseUpperController* para se fazerem acessos à base de dados.

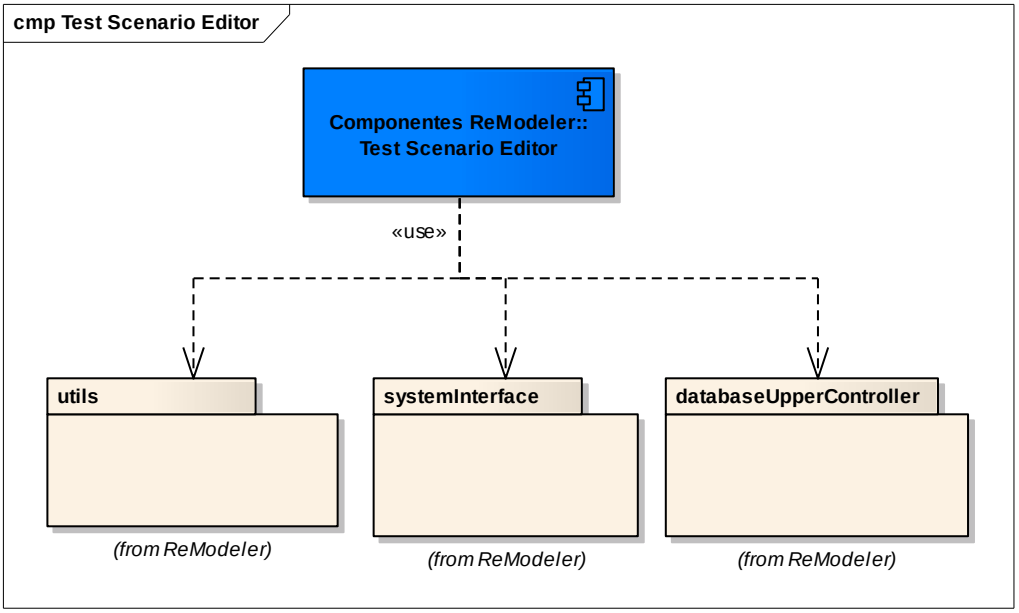

Figura 31 – Pacotes do componente de edição de cenário de teste

4.2.6. Cobertura dos cenários de teste (*TestScenarioCoverage*)

Quando um cenário é capturado no sistema, é necessário marcá-lo e guardar essa informação na base de dados. Posteriormente, podem (e devem) ser gerados indicadores de cobertura de execuções, tais como diagramas de casos de utilização [40]. A cobertura dos cenários de teste é baseada nas marcações feitas durante a captura. A sua representação fica a cargo do componente *TestScenarioCoverage*, figura 32. Este é constituído por componentes já descritos anteriormente, que seguem o princípio já descrito.

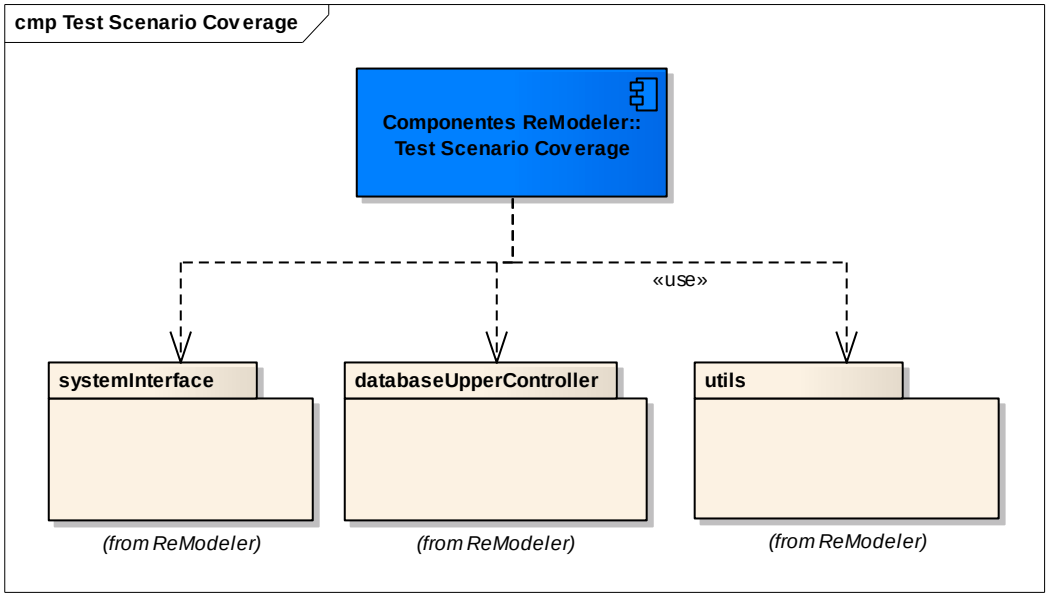

Figura 32 – Pacotes do componente de cobertura de cenários de teste

[Esta página foi intencionalmente deixada em branco]

Capítulo 5

Desenho do *ReModeler*

Conteúdo

Neste capítulo são descritos os aspectos relacionados com a implementação dos pacotes descritos no capítulo anterior. Neste capítulo são apresentados os diagramas de classes dos vários pacotes, bem como os diagramas de sequência de algumas das funcionalidades do protótipo. São também apresentadas as tecnologias que foram utilizadas para a criação do *ReModeler*.

5. Desenho do *ReModeler*

No capítulo anterior foram apresentados os componentes que fazem parte do *ReModeler*. Este implementa os conceitos da modelação aumentada e é a ferramenta que auxilia o *ReModeler Process*. Ao longo do presente capítulo, descrevem-se os principais aspectos da sua realização. O *ReModeler* foi desenvolvido na linguagem Java, mas foi necessário recorrer-se a algumas bibliotecas e programas de código aberto para auxiliar a sua implementação. Estes programas serão também devidamente apresentados, juntamente com os pacotes que os utilizam. Inicialmente vai ser apresentado o diagrama de pacotes global e em seguida vai ser detalhado cada um desses pacotes. Em alguns casos, para facilitar a compreensão do sistema, aumenta-se a granularidade da descrição, apresentando as classes que constituem os pacotes.

5.1. Diagrama de Pacotes Global

Os componentes do capítulo anterior estão implementados em cinco grandes pacotes, que por sua vez contêm vários pacotes internos. O diagrama de pacotes global, representado na figura 33, mostra os pacotes que implementam os conceitos propostos nesta dissertação, mas também outros que, embora não tenham sido desenvolvidos no âmbito da mesma, são relevantes para o seu funcionamento e compreensão (ver pacotes cor-de-rosa)

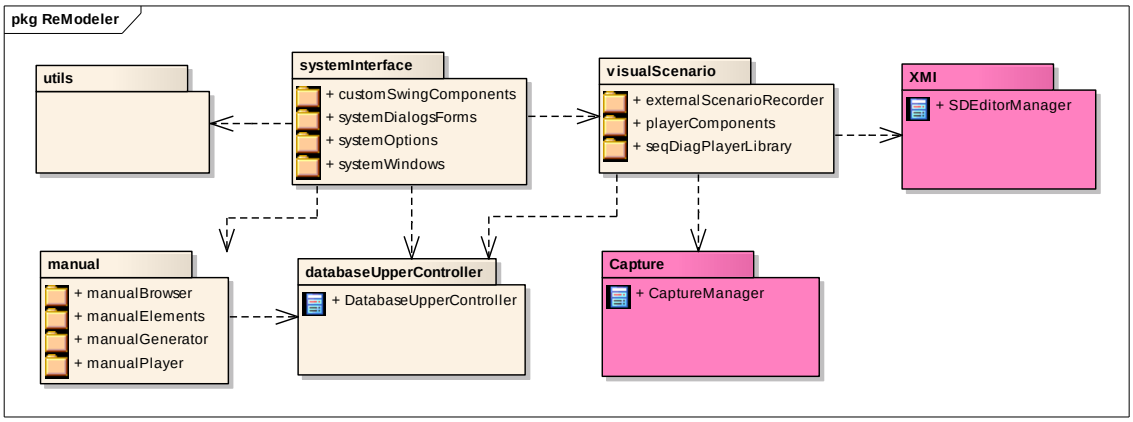

Figura 33 – Diagrama de pacotes global

- Pacote *databaseUpperController*: é responsável pela comunicação com outros pacotes desenvolvidos [40], que comunicam com a base de dados;
- Pacote *visualScenario*: agrupa todos os sub-pacotes e classes relativos à criação e apresentação dos cenários visuais;
- Pacote *manual*: é responsável pela criação e apresentação dos manuais de utilizador automáticos:
- Pacote *systemInteface*: agrupa todos os sub-pacotes e classes que implementam a interface gráfica do *ReModeler*;
- Pacote *utils*: agrupa um conjunto de utilitários necessários à execução do *ReModeler*.

De seguida é apresentado cada um dos pacotes detalhadamente, mostrando os diagramas de classes, sempre que necessário.

5.2. Pacote *databaseUpperController*

No capítulo anterior referiu-se que o funcionamento do *ReModeler* assentava sobre uma base de dados interna. A criação desta base de dados deveu-se à necessidade de manipular uma grande quantidade de dados, que não poderiam ser armazenados nem geridos em memória. Para tal, no âmbito de outra dissertação [40], foram desenvolvidos alguns pacotes e classes para tratar da comunicação com a base de dados, nomeadamente realizando acções de pesquisa, inserção, actualização e remoção de dados da mesma. O pacote *databaseUpperController* serve de elo de ligação entre essa

camada desenvolvida e os vários pacotes implementados no âmbito desta dissertação. A figura 34 mostra a interligação do funcionamento do pacote *databaseUpperController* com os pacotes que constituem as camadas inferiores do *ReModeler*. Sendo a base de dados um elemento comum à implementação dos vários componentes, este pacote será dos mais usados pelos restantes do sistema.

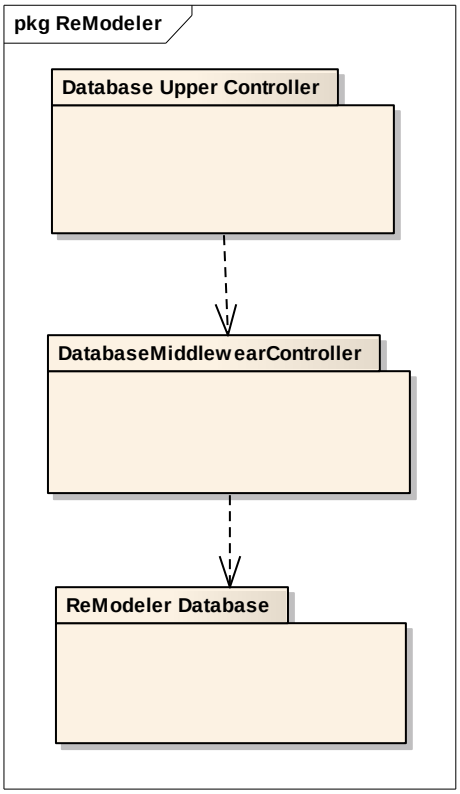

Figura 34 – Ligação do pacote *databaseUpperController* com os pacotes das camadas inferiores do *ReModeler*.

5.3. Pacote *visualScenario*

No capítulo anterior foram apresentados quatro componentes relativos aos cenários visuais: (i) um componente para fazer o desenho do cenário interno, (ii) um componente para fazer a captura de cenários externos, (iii) um componente para fazer a geração do cenário visual e (iv) um componente para fazer a sua visualização. O pacote *visualScenario* implementa todos os constituintes necessários à realização destes quatro componentes, ou seja, dos cenários visuais. Este pacote subdivide-se em três pacotes internos, que por sua vez podem ser agrupados em dois conjuntos: um relativo à captura dos cenários visuais e o outro relativo à sua visualização. A figura 35 mostra o diagrama de pacotes interno ao pacote *visualScenario.*

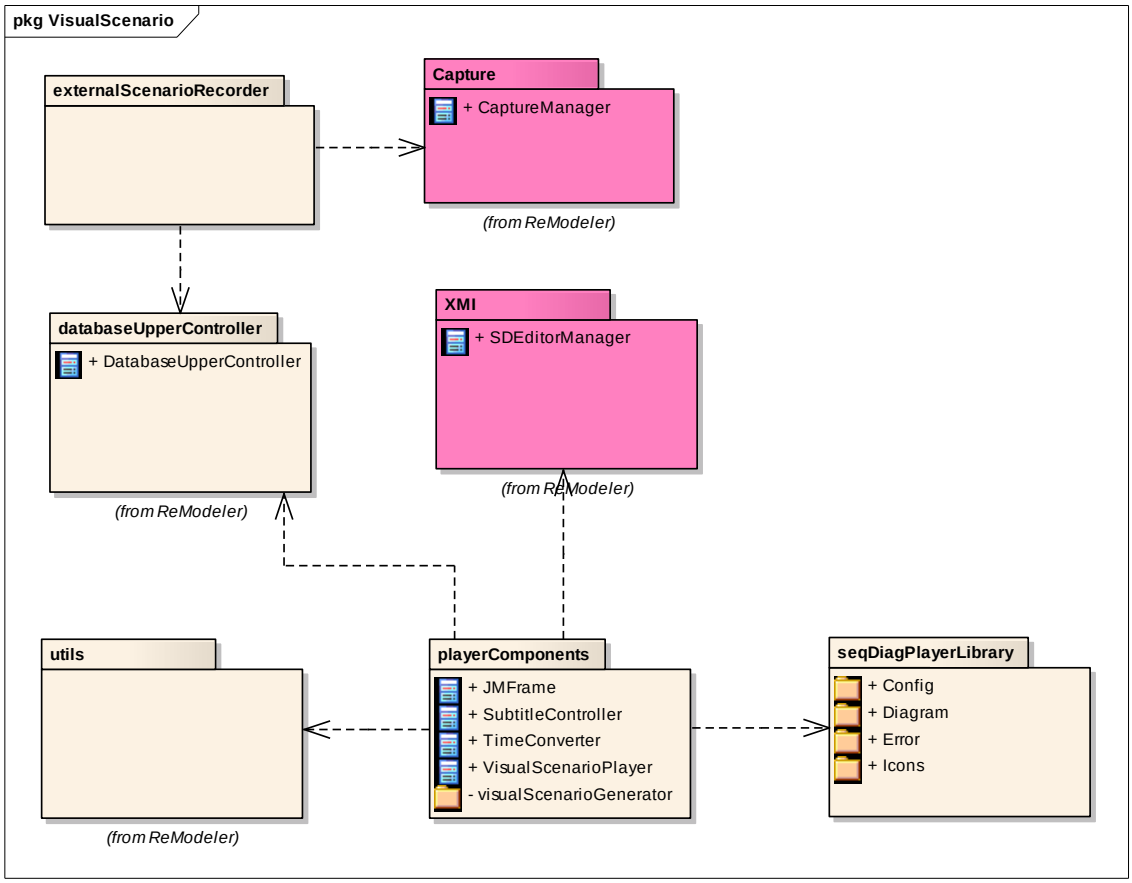

Figura 35 – Diagrama de Pacotes interno do pacote *visualScenario*.

5.3.1. Capturas de cenários visuais

Conforme descrito no segundo capítulo, um cenário visual é composto por duas visões diferentes de um cenário: a visão das interacções do utilizador com o sistema (externa) e a visão da sequência das mensagens trocadas interiormente, aquando da execução do mesmo (interna). A obtenção destas duas visões é um processo relativamente complexo, constituído por duas actividades distintas: a captura externa e a captura interna, respectivamente, que podem ser feitas de modo simultâneo ou separadamente. A figura 36 mostra a parte do diagrama de pacotes interno ao pacote *visualScenario* que respeita à implementação das capturas de um cenário.

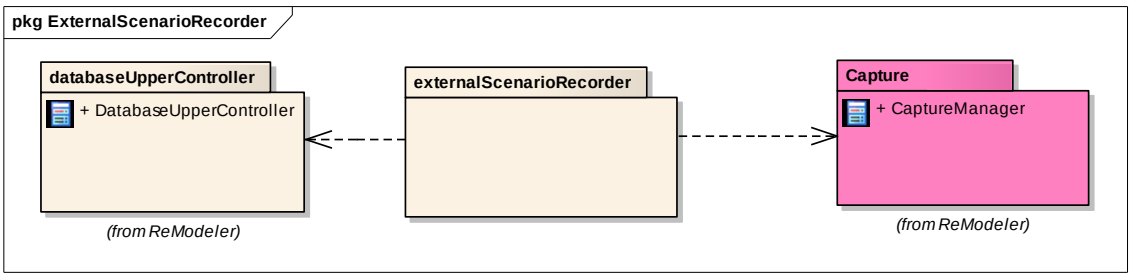

Figura 36 – Diagrama de pacotes relativo à captura de cenários.

A captura interna de um cenário foi realizada pelo pacote *Capture*, que não foi implementado no âmbito desta dissertação, mas sim em [40]. Dentro do pacote *visualScenario*, o sub-pacote responsável pela implementação da captura externa de um cenário é o *externalScenarioRecorder*. Este pacote contém classes que permitem capturar as imagens respeitantes às interacções do utilizador do sistema em análise, durante a execução de um cenário, de forma a produzir um filme das mesmas. Na figura é ainda possível ver o pacote *databaseUpperController*, que é responsável por fazer a ligação entre o *externalScenarioRecorder* e a camada que lida com a comunicação com a base de dados, como foi explicado anteriormente.

5.3.1.1. Classes do pacote *externalScenarioRecorder*

Para se efectuar a gravação de um cenário externo é necessário capturar várias imagens do ecrã do computador, relativas às acções do utilizador com o sistema, e depois transformá-las num filme. A criação do filme foi implementada com auxílio de uma API fornecida pela SUN, designada de *Java Media Framework* (JMF) [42]. A JMF fornece uma arquitectura unificada e um protocolo de mensagens para fazer gestão da aquisição, do processamento e da transmissão de dados multimédia, baseados no tempo. A JMF foi desenhada para suportar os formatos multimédia mais conhecido, tais como AIFF [43], AU [44], AVI [45], MIDI [46], MPEG [47], QuickTime [48], WAV [49] entre outros. Esta API segue o mesmo princípio da linguagem Java, "*write once, run anywhere*" (uma vez escrito, corre em qualquer lado), para o uso de dados multimédia em programas Java. A JMF permitiu não só a criação dos filmes do cenário externo, mas também a sua visualização.

O principal aspecto a considerar na criação do filme é a aquisição das imagens que o compõem. Foi desenvolvida para esse efeito uma classe, chamada de *MyScreenGrabber*, que consegue capturar imagens sequenciais do ecrã do computador,

como se de vários *PrintScreens* se tratassem, armazenando-as em disco em formato JPEG. Concluída a captura das imagens, é depois invocada a classe *JpegImagesToMovie* que é responsável pela leitura das várias imagens armazenadas e pela reconstituição do filme das mesmas. A figura 37 representa o diagrama de classes que compõe internamente o pacote *externalScenarioRecorder*.

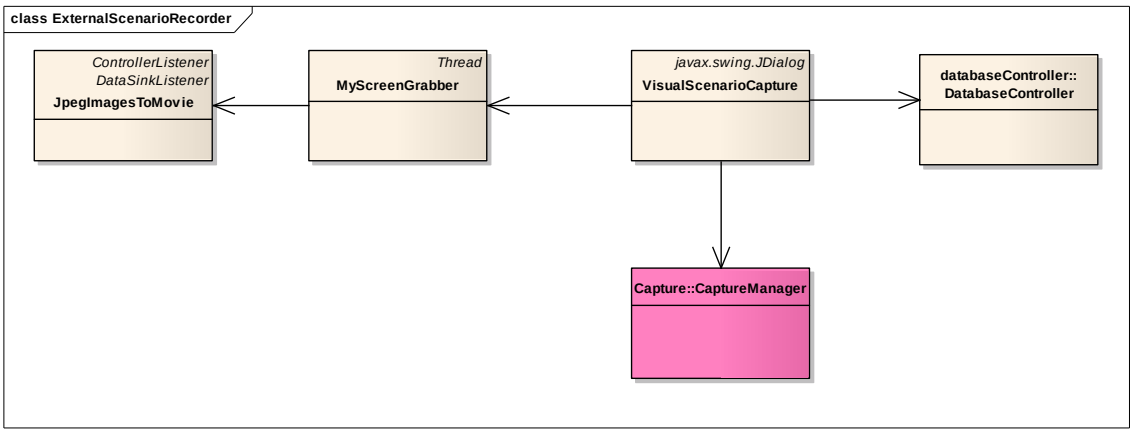

Figura 37 - Diagrama de classes do pacote *externalScenarioRecorder*

A classe *VisualScenarioCaptute* é a interface gráfica responsável por dar início a uma captura de um cenário visual. Esta classe tanto pode iniciar a captura de cenários externos ou internos separadamente, como de ambos em simultâneo. Para iniciar a captura interna, a classe *VisualScenarioCaptute* deve comunicar com a classe *CaptureManager*, do pacote *Capture*, enquanto que para dar início à captura externa é na classe *MyScreenGrabber* que os métodos devem ser invocados.

Durante a captura de um cenário externo é possível marcar cenas no filme, que vão corresponder aos vários passos que constituem o cenário. Isso é realizado na interface gráfica *VisualScenarioCapture*, em colaboração com a classe *DatabaseUpperController*. O instante de tempo em que a cena é marcada no filme vai ser transmitido à base de dados, originando a actualização do passo correspondente. A Figura 38 mostra o diagrama de sequência simplificado das acções efectuadas para concretizar o filme de um cenário externo.

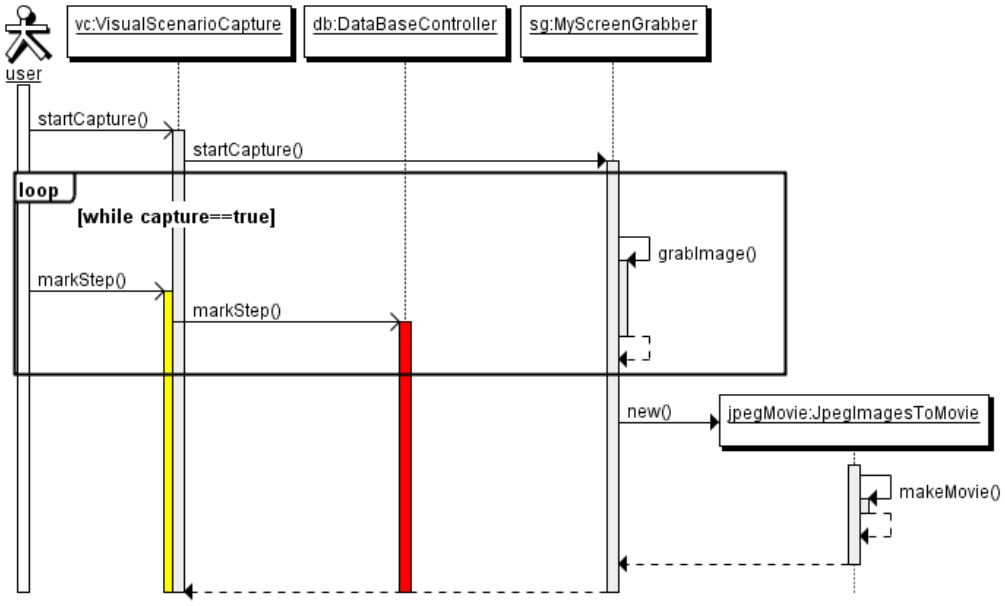

Figura 38 - Diagrama de sequência para captura de um cenário externo

5.3.2. Reprodução de Cenários Visuais

Após a captura dos cenários visuais é necessário existir um mecanismo que os permita reproduzir. Para a criação de cenários visuais não basta efectuar as capturas descritas anteriormente. Os resultados de ambas as capturas têm de ser tratados e sincronizados, de modo a criar um artefacto que possa ver visualizado e reproduzido. Deste modo, têm de ser executados três processos distintos, identificados no capítulo anterior pelos componentes *SequenceDiagramAnimator, VisualScenarioGenerator e VisualScenarioPlayer*. No primeiro processo devem ser desenhados os diagramas de sequência que representam as mensagens capturadas, através da interpretação de uma linguagem interna produzida para o efeito. O segundo processo prende-se com a sincronização dos vários resultados produzidos: filme, diagrama de sequência e descrição das várias cenas, de modo a gerar o artefacto composto designado de Cenário Visual. Por último, deve ser possível reproduzir o Cenário Visual gerado através de uma interface própria. A figura 39 representa a parte do diagrama de pacotes interno ao *visualScenario* que mostra apenas os pacotes que implementam a geração e reprodução de um cenário visual.

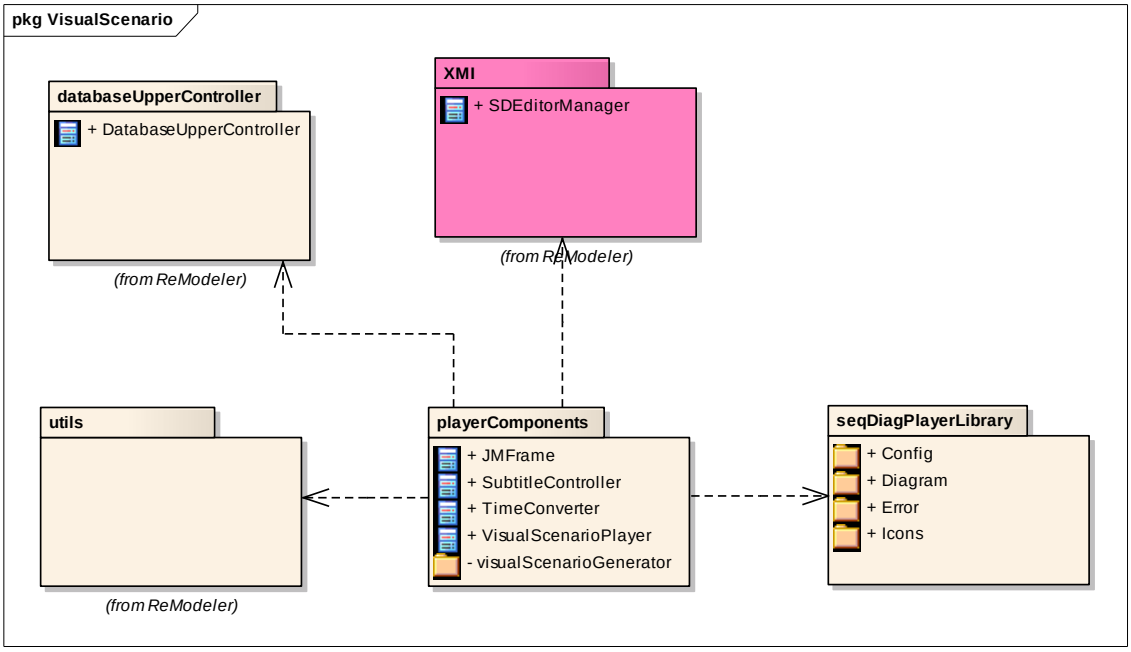

Figura 39 - Pacotes que implementam a geração e reprodução de um cenário visual

5.3.2.1. Desenho do cenário interno

De entre os sub-pacotes que constituem o pacote *visualScenario*, o que é responsável por realizar os processos descritos anteriormente, relativos ao desenho dos diagramas de sequência, chama-se *seqDiagPlayerLibrary*. Na verdade, este sub-pacote é uma biblioteca de desenho de diagramas de sequência que foi desenvolvida com base num programa de código aberto, disponível no *sourceforge* [50], chamado de *Quick Sequence Diagram Editor* (QSDE) [51]. O programa, desenvolvido por *Markus Strauch*, comporta-se como um editor de diagramas de sequência, que os cria a partir de descrições textuais, que seguem uma sintaxe textual, tal como aparece representado no lado esquerdo da Figura 40. Entre as funcionalidades disponibilizadas, encontramos opções para escalar os diagramas desenhados e para exportar os diagramas desenhados para alguns formatos externos, como por exemplo PDF [52]. A figura 40 mostra a janela principal da execução do programa.

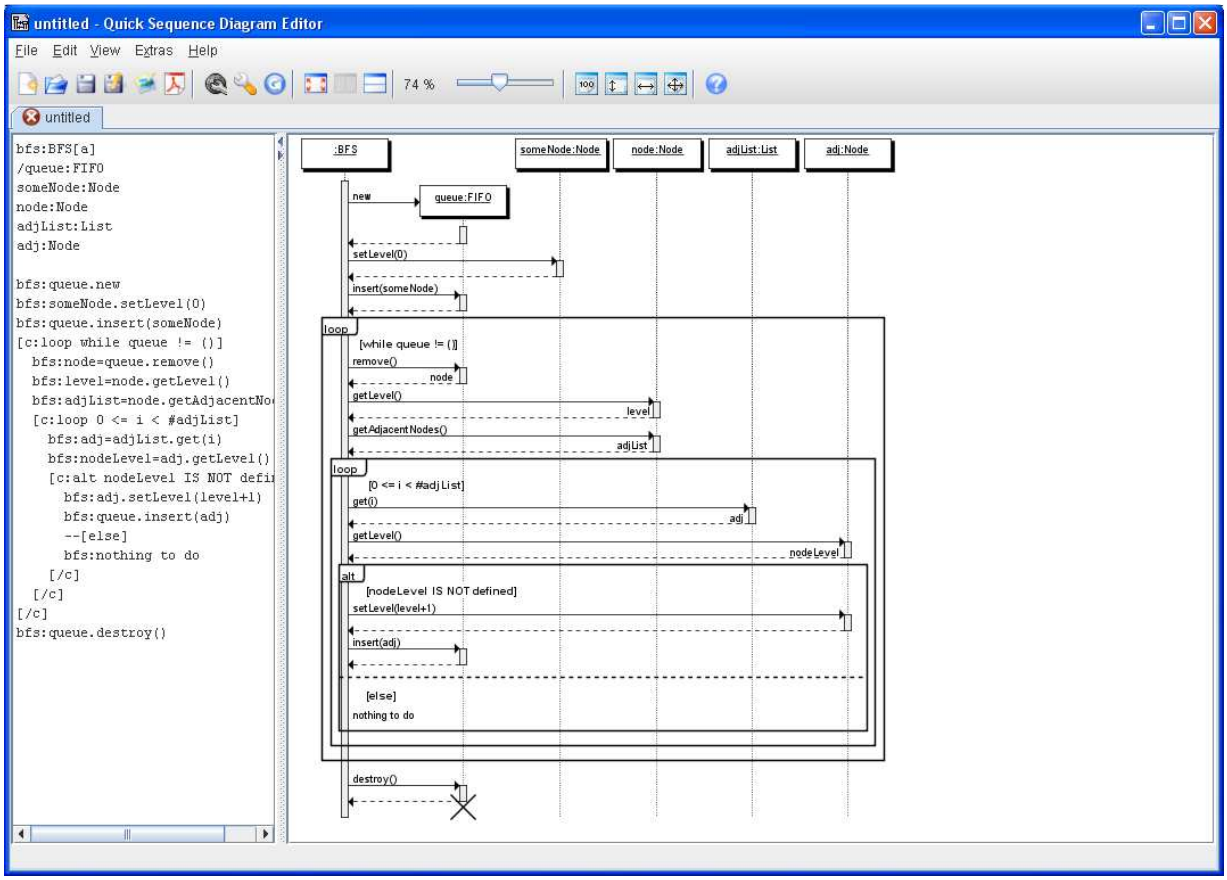

Figura 40 - Janela do *Quick Sequence Diagram Editor*

5.3.2.1.1. Funcionalidades da biblioteca de desenho de diagramas de sequência do *ReModeler*

Existem várias funcionalidades que devem estar presentes numa biblioteca de desenho de diagramas de sequência. No *ReModeler*, em particular, o principal requisito para uma biblioteca deste género seria que ela seja capaz de desenhar diagramas de sequência de uma forma automática. De forma a permitir implementar este requisito, a geração dos diagramas de sequência deveria ser feita a partir da interpretação de uma linguagem simples. Para além disso, o diagrama resultado da geração, deveria ser apresentado numa janela independente, possível de ser incorporada no *ReModeler*. Um diagrama de sequência representativo de um cenário interno tende a crescer e a ocupar uma área significativa do ecrã. Para que o desenho automático desses diagramas não ultrapasse as fronteiras da janela disponível, é necessário que a mesma implemente a funcionalidade de escala automática. Os métodos básicos que devem estar implementados na biblioteca devem permitir inserir os comandos da linguagem e visualizar o diagrama de sequência.

Alguns dos métodos da interface principal da biblioteca

public void appendText(String Text) -> inserção de um comando da linguagem.

public DiagramWindow getDiagramWindow() -> devolve a janela onde é desenhado o diagrama.

public void clean() -> limpa a janela onde é desenhado o diagrama.

public ScalePanel getScalePanel() -> devolve um painel com funções de escalamento para a janela que desenha o diagrama.

A introdução dos comandos da linguagem própria para desenho no diagrama é feita através do método *appendText*. Por sua vez, a visualização dos diagramas de sequência gerados é feita através da janela *DiagramWindow*.

5.3.2.1.2. Processo de construção da biblioteca

Inicialmente o QSDE continha tudo o que era necessário para fazer diagramas de sequência, mas após um estudo das funcionalidades que o programa disponibilizava, concluiu-se que para fazer a animação dos diagramas de sequência, era necessário fazer uma adaptação da linguagem e do desenho dos diagramas. Foi necessário fazer esta adaptação porque quer a linguagem, quer o desenho dos diagramas funcionam em conjunto e não é permitido marcar os passos de um cenário no diagrama de sequência. A última alteração que se fez ao QSDE foi ao nível da interface gráfica. Esta não era totalmente independente do resto do programa e apenas se necessitava da janela de desenho dos diagramas de sequência. Para o conseguir realizar todas as alterações com sucesso, teve de ser realizado um processo de reengenharia de software, que passou pelas fases de compreensão do sistema, recuperação da documentação para o sistema (legado), e de rescrita das partes identificadas como necessárias para implementar os novos requisitos.

O primeiro passo feito para se tentar compreender o programa foi a recuperação da sua estrutura interna, sob a forma de um diagrama de classes UML, conseguido através das facilidades de engenharia inversa de uma ferramenta UML. A ferramenta escolhida foi o *Enterprise Architect* [53], que dentro das restantes ferramentas disponibilizadas no mercado é das que apresenta o comportamento mais satisfatório, na

recuperação de diagramas de classes. Ainda assim, o diagrama de classes recuperado apresentou pequenas lacunas, que não dispensaram a observação atenta do código fonte para a sua completude.

Contudo, o diagrama de classes recuperado não foi suficiente para perceber o funcionamento e organização do QSDE. Embora fosse perceptível a relação entre as classes, o seu funcionamento interno não era de todo claro. Para se compreender o seu funcionamento interno foi necessário executá-lo, muitas e muitas vezes em processo de *debbugging*, num processo lento e demorado, em que o fluxo de execução foi seguido de uma forma manual. Este é exactamente o processo que espero que se possa melhorar com as contribuições desta dissertação.

Após o entendimento do sistema, foi possível definir que partes do mesmo teriam de ser alteradas ou mesmo removidas. A figura 41 representa o diagrama de pacotes recuperado, onde foram marcados com uma circunferência encarnada os pacotes que foram necessários alterar. Os restantes pacotes ou acabaram por ser completamente removidos, ou mantiveram-se inalterados, como o pacote *error* e *icons*. Um facto curioso foi ter sido encontrado nesta versão do QSDE um pacote aparentemente sem utilidade (*taglet*).

A alteração do sistema, segunda parte do processo de reengenharia, começou pelo desacoplamento da interface gráfica do resto do programa. Este foi um processo difícil e bastante demorado devido ao grande acoplamento existente neste componente. De seguida, foi alterada a linguagem própria de modo a permitir a identificação clara das mensagens que constituíam cada passo do cenário. A alteração da linguagem também originou a modificação da forma como alguns dos componentes gráficos eram desenhados. O último passo da rescrição do programa foi tornar alguns dos componentes gráficos do QSDE completamente independentes.

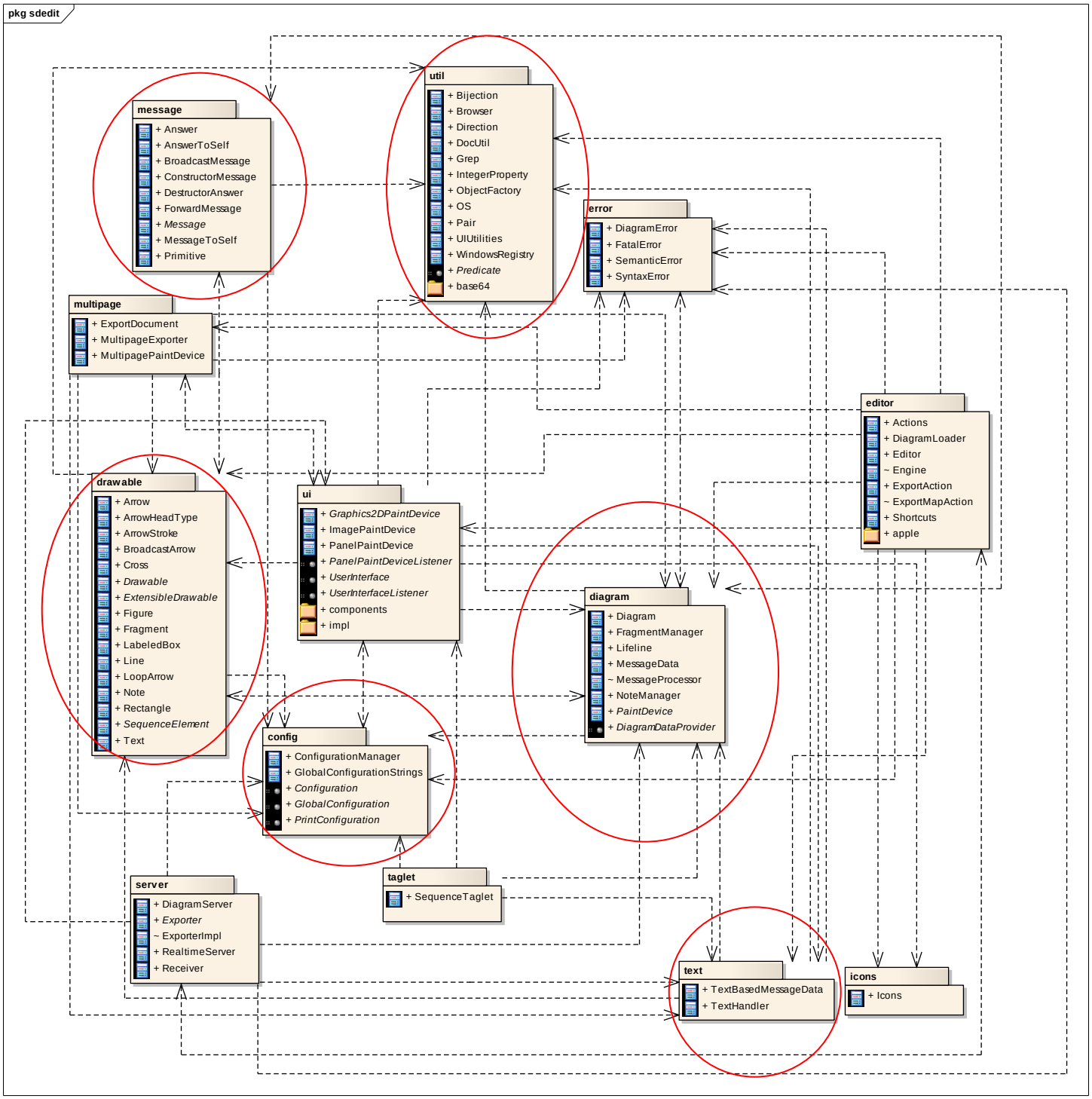

Figura 41 - Diagrama de pacotes do QSDE

O diagrama de pacotes resultante deste processo está representado na figura 42. Como é possível observar, houve um agrupamento de alguns dos pacotes do digrama anterior e remoção de outros. No QSDE as classes que directa ou indirectamente se relacionavam com a geração dos diagramas, estavam dispersas por vários pacotes que comunicavam entre si. Na nova arquitectura, elas encontram-se todas agrupadas no

pacote *diagram*. O pacote *config* passou a ser o repositório de todas as classes que são necessárias para criar e guardar as configurações relativas ao desenho de diagramas. O pacote *icons* continuou a fazer a gestão dos ícones da biblioteca, enquanto que o pacote *error* contém as classes que são necessárias para apresentar mensagens de erro apresentadas pela biblioteca.

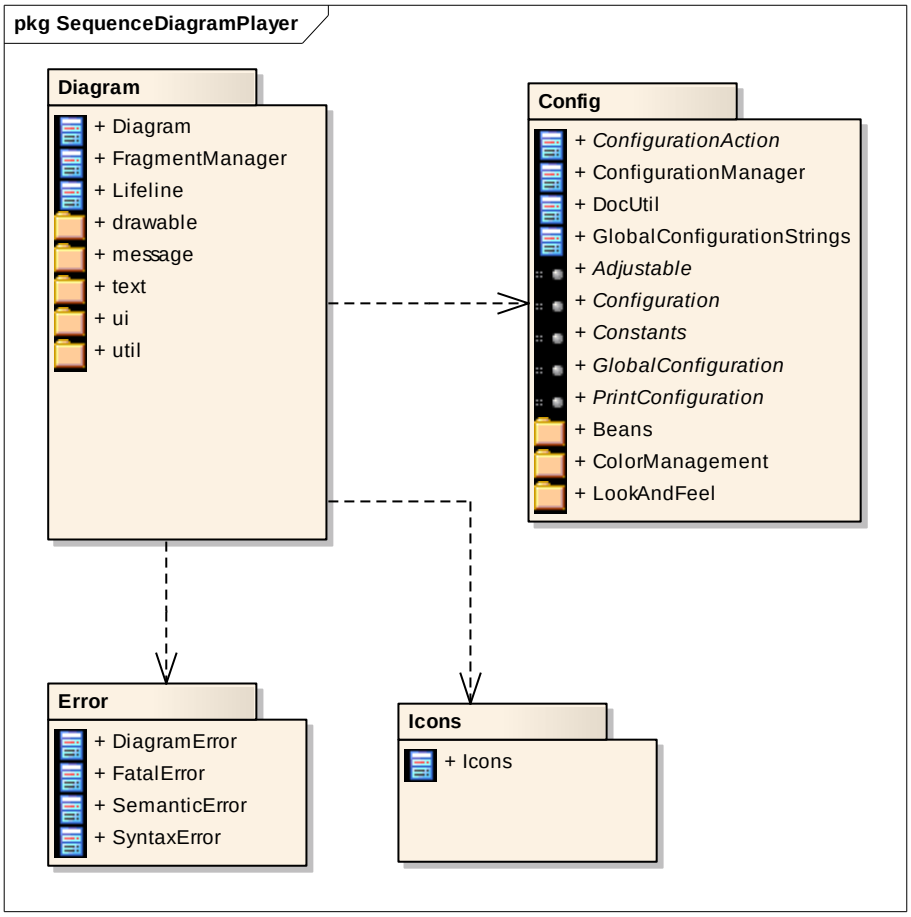

Figura 42 - Diagrama de pacotes da biblioteca de desenho de diagramas de sequência do *ReModeler*

5.3.2.1.3. Alterações efectuadas ao QSDE

Como já referido, o QSDE tinha basicamente tudo o que era necessário para criar a biblioteca de desenho de diagramas de sequência. No entanto, para as necessidades específicas do *ReModeler*, havia algumas faltas. A primeira dessas faltas foi identificada ao nível da linguagem, onde não havia modo de identificar os passos que estão presentes num cenário visual. A introdução de um novo comando, obrigou à criação de um mecanismo visual que representasse, no desenho do diagrama, os

segmentos respeitantes a cada passo. A maneira mais intuitiva de conseguir isto foi desenhar janelas de interacção coloridas, tirando partido da facilidade do QSDE em desenhar janelas de interacção monocromáticas. A cor de cada janela é então definida pela cor de vem declarada no comando da linguagem. Ex:

[c:step1 BLUE] … [/c]

[c:step2 RED] … [/c]

…

A última alteração efectuada foi a reestruturação interna da estrutura do programa, de modo separar apenas os componentes necessários e tornando-os o mais desacoplados possível.

A leitura dos comandos da linguagem, inicialmente feita com recurso a uma janela de introdução de comandos, passou a ser conseguida através de métodos internos disponibilizados na API. Isto fez com que o componente da janela, deixasse de ser necessário, e portanto dispensável. Por sua vez, a janela do desenho do diagrama de sequência passou a estar completamente independente do programa QSDE, por forma a permitir a sua reutilização. Para que tal fosse possível, a biblioteca passou a disponibilizar um método que faz o acesso à janela e que permite a sua utilização em vários contextos.

Um desenho de um diagrama de sequência tem tendência a ficar muito grande e se não existir um mecanismo adequado de escalamento, pode acontecer que ele ultrapasse os limites da janela em que se encontra. Para isto não acontecer, o desenho dos diagramas foi alterado de forma a se adaptar automaticamente à janela onde está a ser visualizado, ou seja, passou a haver um escalamento automático no momento do desenho do diagrama na janela. Isto foi conseguido através de uma serie de funções de escalamento, que já estavam desenvolvidas no QSDE, mas mesmo assim, foi disponibilizada ao utilizador a possibilidade de escalar o diagrama ao seu próprio gosto. A restante interface gráfica do QSDE foi completamente removida, por não ser necessária para a biblioteca que foi desenvolvida.

Adicionalmente foram ainda incorporadas mais duas funcionalidades na biblioteca, em virtude do programa onde esta foi baseada. A primeira dessas funcionalidades foi a exportação de diagramas para formatos externos, como pdf [52] e

a segunda foi possibilitar estender o desenho do diagrama de sequência de modo a ocupar todo o ecrã do computador.

5.3.2.1.4. A linguagem da geração dos diagramas de sequência

A linguagem presente no QSDE era bastante completa para o contexto geral de criação de diagramas de sequência. Contudo, os requisitos impostos pelo ReModeler, mais precisamente, para a criação dos cenários visuais, exigia a que houve uma modificação da mesma. A declaração dos passos de um cenário é imprescindível para garantir a coerência entre os vários elementos que constituem um cenário visual para além de melhorar a sua compreensão.

À linguagem original foi-lhe adicionado um novo comando para permitir a marcação de uma região no diagrama que representa cada passo do cenário. O formato da linguagem é bastante simples e intuitivo. Ela permite declarar objectos e contém comandos de escrita de três mensagens principais: os *creates*, os *destroy* e os *calls*. A declaração de objectos, segue o seguinte formato:

<Nome>:<Tipo> ou */<Nome>:<Tipo>*

A diferença entre as duas formas de declaração, mostrada pela introdução de "/" no inicio do comando, é a visibilidade do objecto no diagrama até este ser criado, ou seja, sempre que o comando contiver o "/" no inicio, significa que o objecto deve permanecer invisível no diagrama até que lhe seja feita uma criação explícita. Isto é necessário porque nesta linguagem os objectos têm de ser todos declarados no inicio, como tal, é necessário esconder objectos até que estes sejam precisos.

No exemplo seguinte, temos a declaração de dois objectos visíveis e um invisível.

```
obj1:Class1

obj2:Class2

/obj3:Class3
```

Os objectos 1 e 2 são logo desenhados no diagrama de sequência após a sua declaração, o objecto 3 fica à espera de receber uma mensagem de criação.

Após a declaração de todos os objectos envolvidos, seque-se a declaração das várias mensagens. A sintaxe da declaração de uma mensagem pode variar consoante o tipo de mensagem que se deseja representar. No geral uma mensagem tem o seguinte formato:

<Chamador>[Nível] :[Resposta]=<Chamado>.<Mensagem>

A resposta e o nível são opcionais. A resposta é o resultado da invocação de um método e o nível indica o número de mensagens que têm de ser respondidas antes de ser criada a mensagem que se está a declarar.

No caso de a mensagem ser de criação de um objecto, (construtor), a mensagem passada deverá ser "*new*". Do mesmo modo, se a mensagem for a destruição de um objecto, (destrutor), a mensagem passada deverá ser de "*destroy*".

Sintaxe para o construtor:

<Chamador>[Nível] :[Resposta]=<Chamado>.new

Sintaxe para o destrutor:

<Chamador>[Nível] :[Resposta]=<Chamado>.destroy

Entre cada grupo de mensagens, pode ser declarada uma janela de interacção. Como foi focado anteriormente, as janelas de interacção foram usadas para representar blocos de execução no diagrama, que correspondem a um determinado passo no cenário. A declaração destas janelas de interacção foi alterada face à versão original. Agora é possível declarar uma janela de interacção com indicação da cor que denota o passo do cenário. Note-se que a cor declarada tem de estar escrita em Inglês ou na sua representação hexadecimal. A sintaxe deste comando é a seguinte:

[c:<Passo> <Cor>]

Mensagem 1 … Mensagem N

[/c]

A declaração da janela de interacção, sem marcação de um bloco de execução, tem o seguinte formato:

[c:<Condição>]

Mensagem1 … Mensagem N

[/c]

A figura 43 mostra um exemplo de um diagrama de sequência sem marcação de passos do cenário.

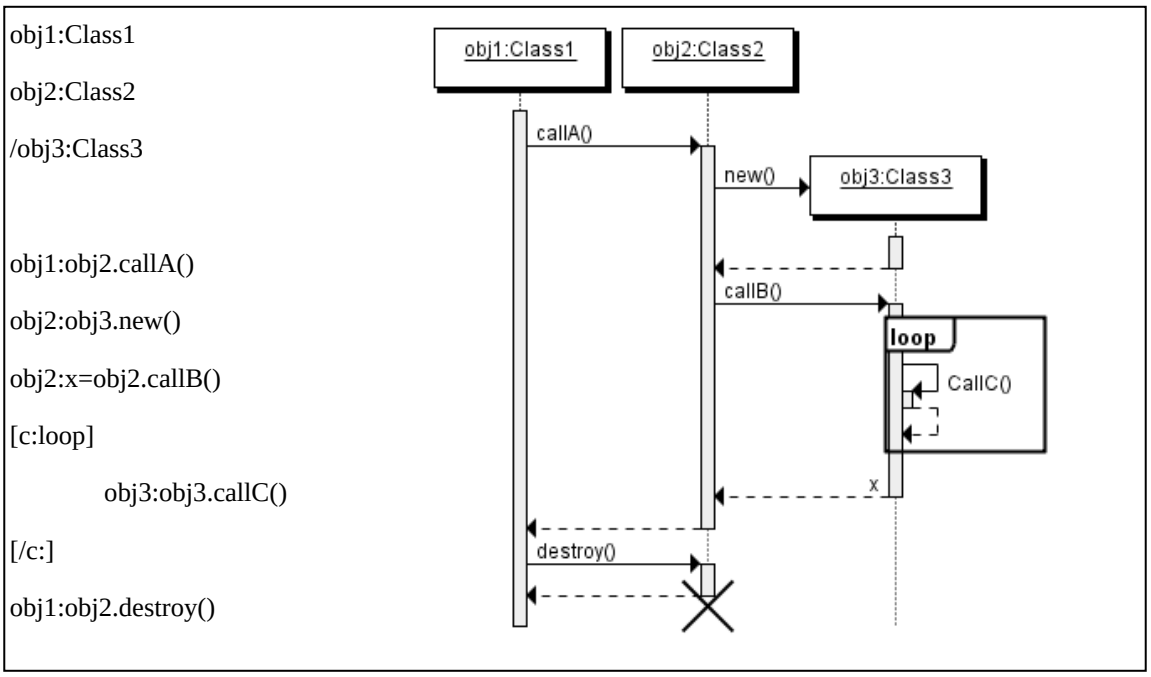

Figura 43 - Diagrama de sequência sem marcação de passos

A Figura 44 mostra um exemplo de um diagrama de sequência com marcação de passos.

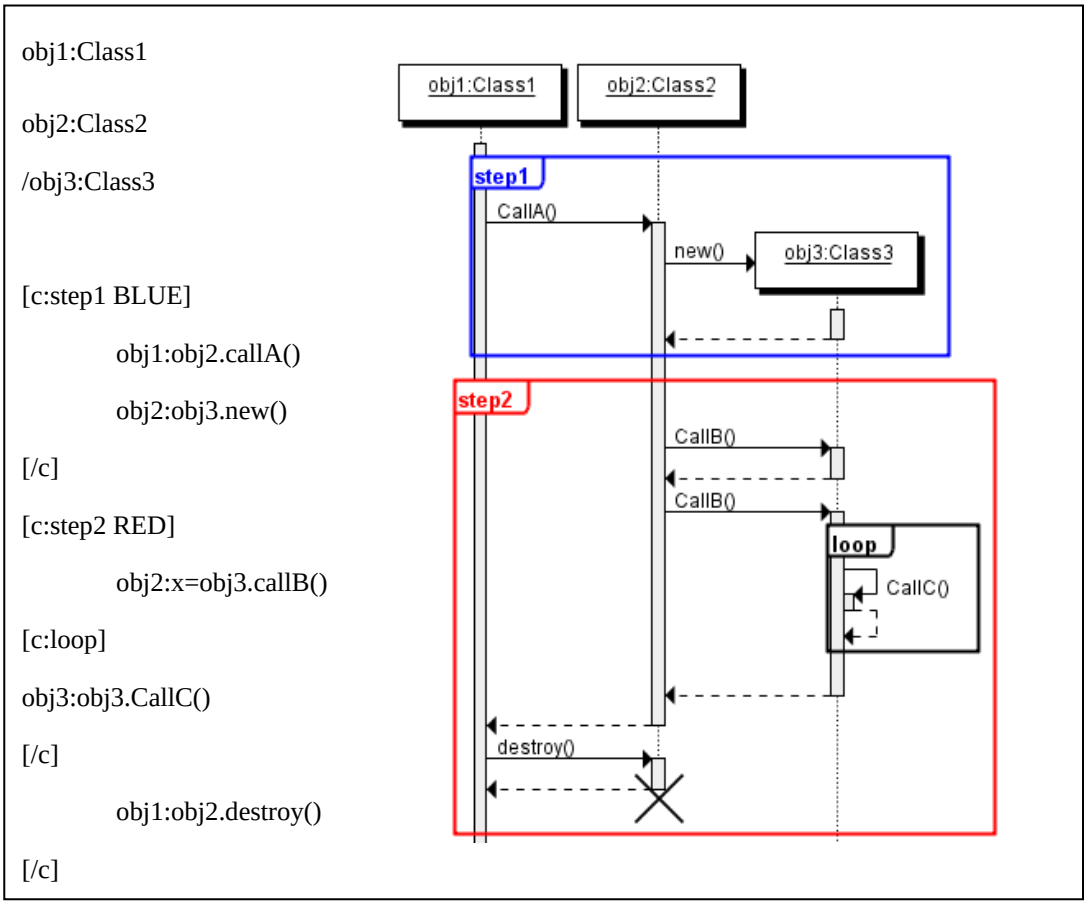

Figura 44 - Diagrama de sequência com marcação de passos de cenário

5.3.2.2. Classes do pacote *playerComponents*

Até aqui foi descrito o sub pacote do pacote *visualScenarios* que é responsável pelo desenho dos diagramas de sequência. Nesta secção vai ser apresentado o sub-pacote *playerComponents* que implementa os outros dois processos da realização do cenário visual: a sincronização dos seus elementos e a sua visualização.

A construção de um cenário visual é um processo complexo que passa pela apresentação simultânea de dois tipos de cenários: o cenário interno e o externo. O problema aqui é conseguir um mecanismo de sincronização entre ambos, pois é necessário existir um ponto de coordenação. Nesta implementação foi considerada a passagem temporal do filme do cenário externo, como ponto de referência para o aparecimento das mensagens do cenário interno. Como já foi referido anteriormente,

sempre que é feita uma captura interna, as mensagens que vão ocorrendo vão sendo marcadas com o instante em que ocorreram. Este instante vai depois permitir a sincronização temporal entre as mensagens e o próprio filme. É necessário notar que quando é feita uma captura dos dois cenários simultaneamente eles vão coincidir temporalmente com exactidão. Portanto, a passagem de tempo do filme vai servir para indicar quando devem ou não ser apresentadas as mensagens do cenário interno.

Ao nível da implementação existem classes distintas para representar cada tipo de cenário. A coordenação entre ambos deve ser realizada por outra classe separadamente. O pacote que contém tudo o que é relativo à apresentação de um cenário visual é o *playerComponents*. A figura 45 mostra o diagrama de classes deste pacote, apresentando as classes que o constituem, assim como classes que pertencem a outros pacotes, como o *visualScenarioGenerator*, o *seqDiagPlayerLibrary* e outros externos.

A reprodução de um cenário visual implica a sua criação em tempo real. A classe *VisualScenarioPlayer* está encarregue de gerir todo o processo. Em primeiro lugar, ela vai comunicar com a classe *VisualScenarioManager,* para que sejam lidas as informações da base de dados, sobre o cenário visual a reproduzir. A classe *SDEditorManager* [40] vai depois gerar os comandos da linguagem própria, que traduzem as execuções internas de um cenário visual, e que vão ser guardados na classe *visualScenarioData*. Por sua vez, as legendas do cenário externo são lidas pela classe *DatabaseUpperController* e são guardadas na classe *Subtitle*.

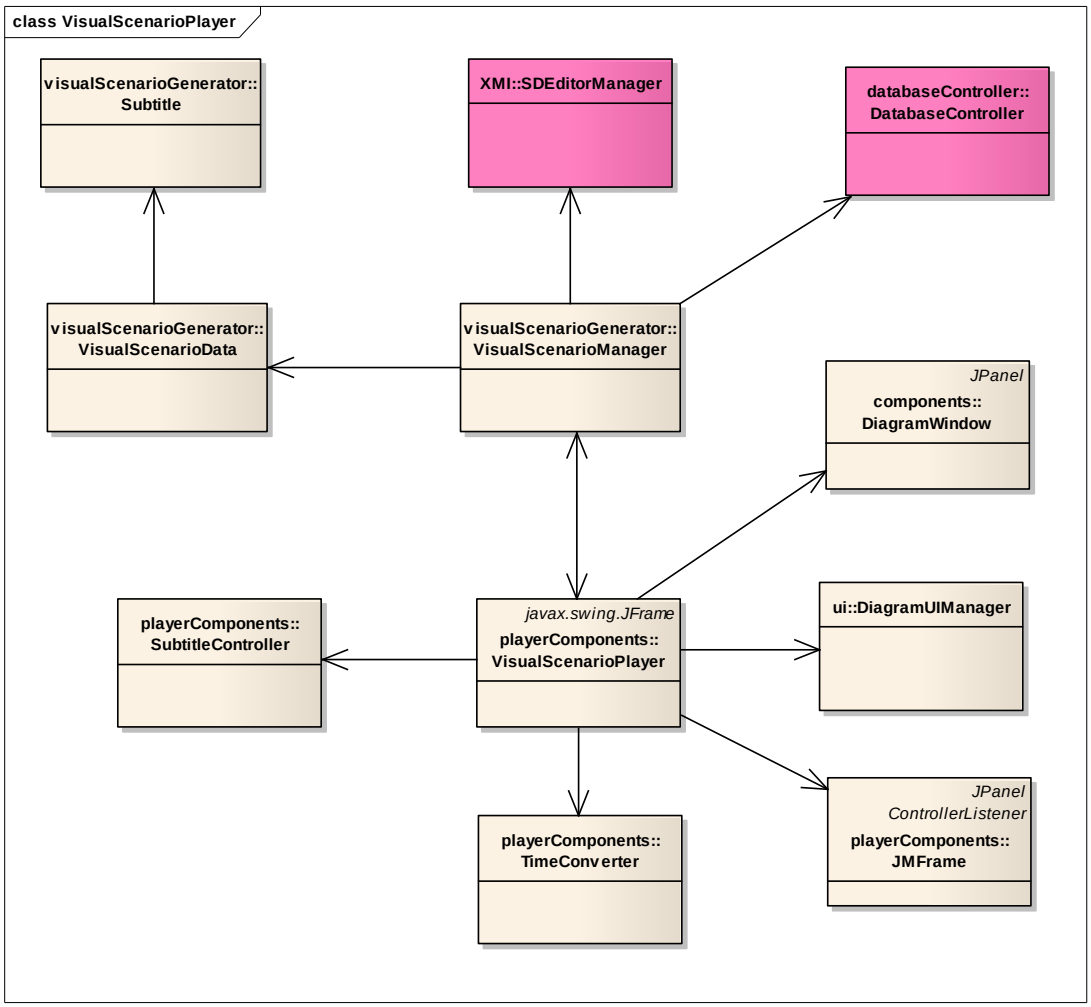

Figura 45 - Diagrama de classes do pacote *playerComponents*

Seguido este processo inicial, começa a reprodução do filme do cenário externo. Para cada nanossegundo de filme corrido, a classe *VisualScenarioManager* recebe uma invocação de um método que vai perguntar se existe algum evento a decorrer nesse intervalo. Um evento pode ser uma ou mais operações de um cenário interno e/ou o aparecimento de uma legenda no cenário externo. Quando existe uma ou mais operações de um cenário interno, estas são passadas para a classe *VisualScenarioPlayer* que por sua vez as entrega à classe *DiagramUIManager*, que é a interface da biblioteca do cenário interno. Se houver uma legenda para o instante, ela é apresentada no ecrã e é activada a classe *SubtitleController*, que é responsável por fazer a gestão do tempo de apresentação das legendas. Por último temos a classe *TimeConverter* que é responsável por fazer a conversão do tempo do filme para uma escala de nanossegundos.

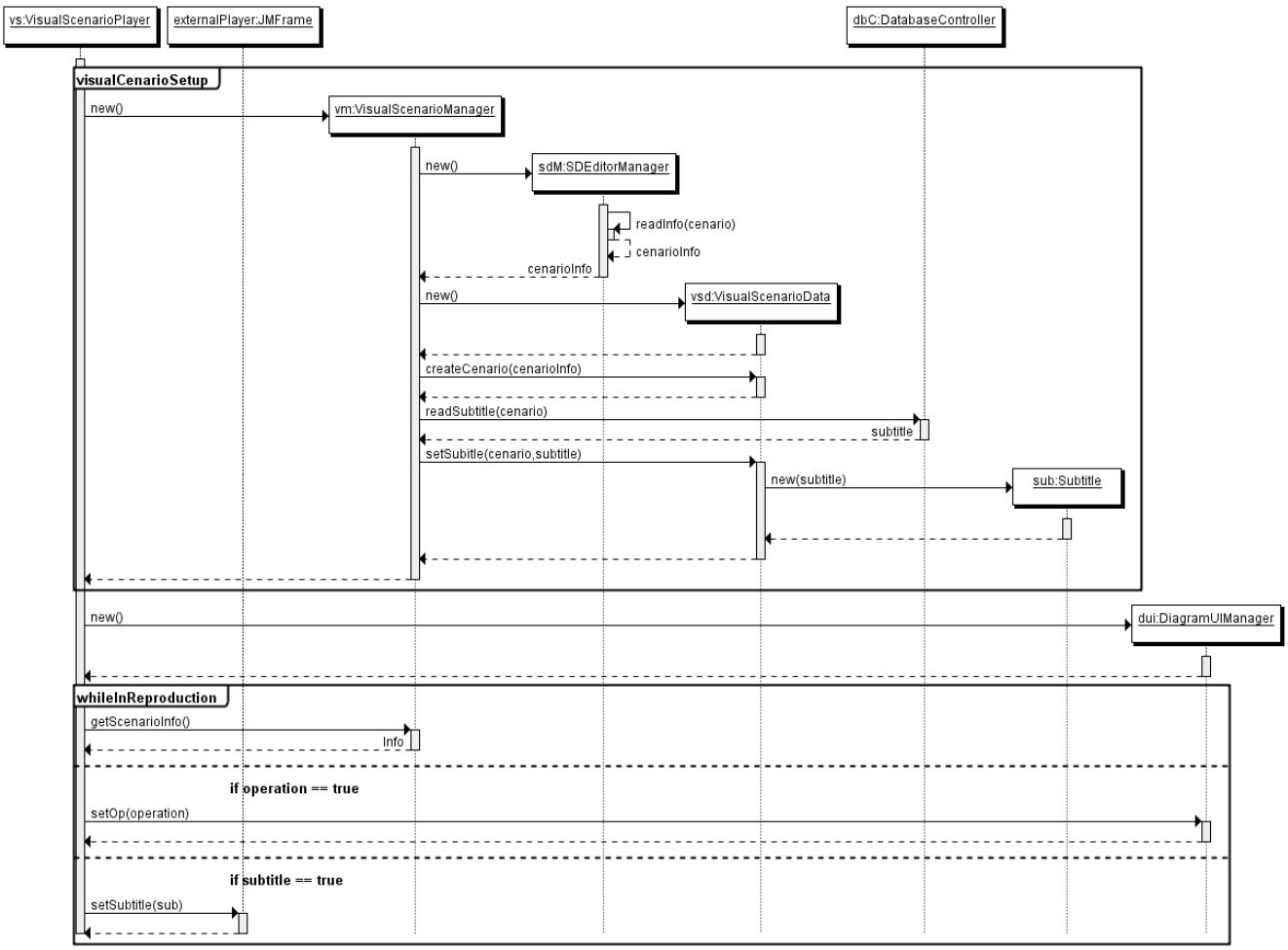

Figura 46 - Diagrama de sequência da geração e reprodução de um cenário visual

A sincronização entre ambos os cenários é garantida através dos tempos do filme do cenário externo. O filme do cenário externo é apresentado na classe *JMFrame* e o filme do cenário interno é apresentados na classe DiagramWindow. A figura 46 apresenta um excerto do diagrama de sequência das acções que ocorrem para a reprodução de um cenário visual.

5.4. Pacote *manual*

A geração do manual de utilizador automático é feita em função dos casos de utilização, dos cenários e dos respectivos cenários visuais, que estão presentes no sistema. Para se criar um manual é necessário ler da base de dados quais os casos de utilização que existem para um sistema e verificar se os mesmos já estão documentados. A documentação de um caso de utilização é constituída pelos seus cenários e descrições, e pelos respectivos cenários visuais.

A implementação da geração do manual de utilizador é conseguida pelo pacote *manual*, que por sua vez é composto por vários sub pacotes, com responsabilidades distintas. A figura 47 representa o diagrama de pacotes para o pacote *manual*.

- Pacote *manualPlayer*: responsável pela visualização do manual de utilizador;
- Pacote *manualBrowser*: responsável por fazer pesquisa sobre os manuais gerados pelo *manualGenerator*;
- Pacote *manualElements*: agrupa os vários elementos que constituem um manual

O acesso à base de dados é mais uma vez feito através do pacote *databaseUpperController*.

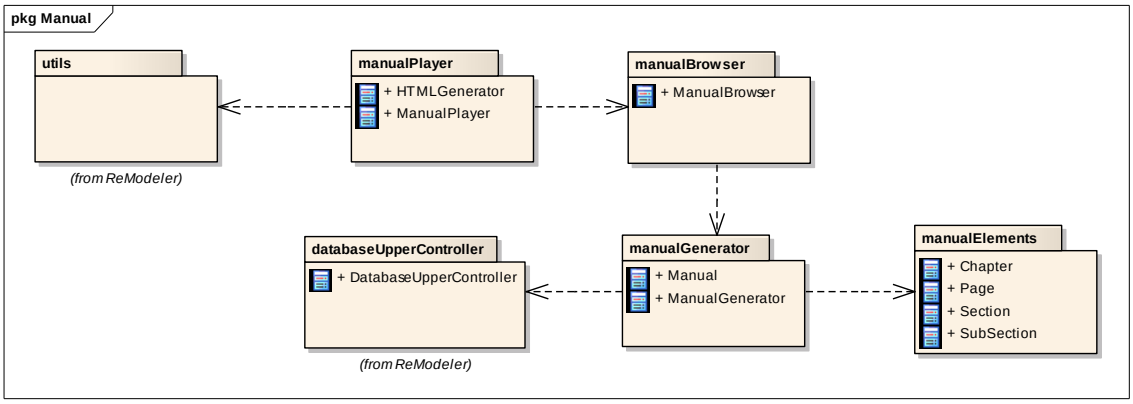

Figura 47 - Diagrama de pacotes do pacote *Manual*

As relações entre as classes que constituem todos estes pacotes podem ser representadas pelo diagrama de classes da Figura 48. Quando um utilizador activa a funcionalidade de visualização de um manual de utilizador, a classe *manualPlayer* inicia uma instância do *manualBrowser*, que por sua vez inicia uma instância do *manualGenerator*. Esta última classe vai pedir as informações que estão armazenadas na base de dados e que permitem criar o manual. Os dados fornecidos vêm em função das capturas ocorridas no *ReModeler*. Para cada diagrama de casos de utilização presente no sistema, são lidos todos os seus casos de utilização, os respectivos cenários e os passos presentes nas descrições dos mesmos. Estas informações são distribuídas pelas classes *Chapter*, *Section*, *Subsection* e *Page*, organizadas segundo uma hierarquia,

em que cada capítulo corresponde a um diagrama de caso de utilização, cada secção corresponde a um caso de utilização, cada subsecção corresponde a um cenário e por último cada passo corresponde a uma página.

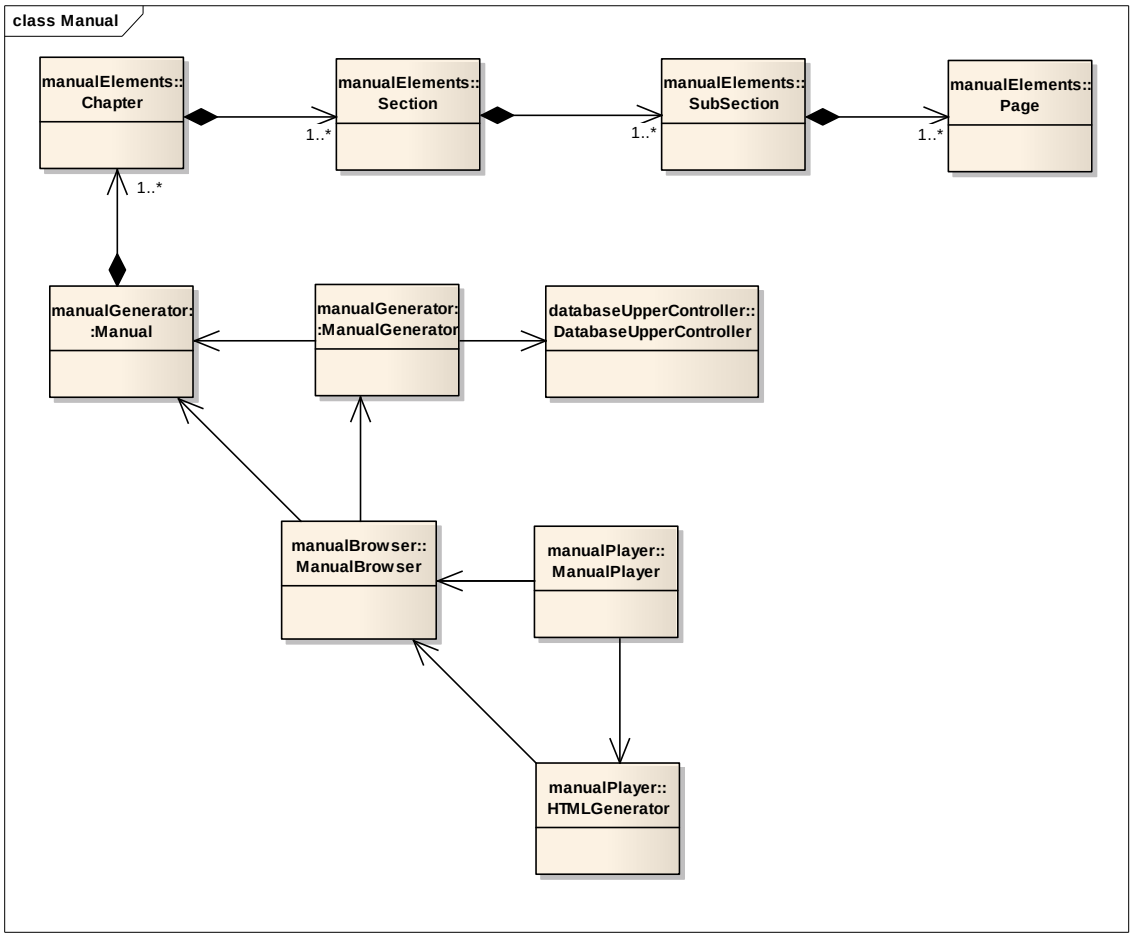

Figura 48 - Diagrama de classes do pacote *Manual*

Como já foi referido, cada passo de um cenário é marcado com um instante temporal, durante a captura de um cenário externo. Na situação inversa à sincronização dos tipos de cenários de um cenário visual, essa marcação vai servir para se identificar a imagem constituinte do filme a que corresponde o passo. Após o preenchimento dos dados do manual, ou seja, a sua criação, a classe *manualBrowser* vai disponibilizar uma interface gráfica que permite a interacção do utilizador com o manual. Esta interface permite navegar sobre os vários componentes constituintes do manual: os capítulos, secções, páginas, etc. A classe *manualPlayer* é a responsável pela obtenção das imagens do filme que constituirão as páginas do manual. A classe *HTMLGenerator* está incumbida de fazer a geração e exportação do manual para o formato HTML. Os dados presentes no ficheiro HTML vão ser os mesmos que se podem observar no

manualPlayer. A figura 49 mostra o diagrama de sequência simplificado da interacção de alguns dos componentes que fazem parte da geração do manual.

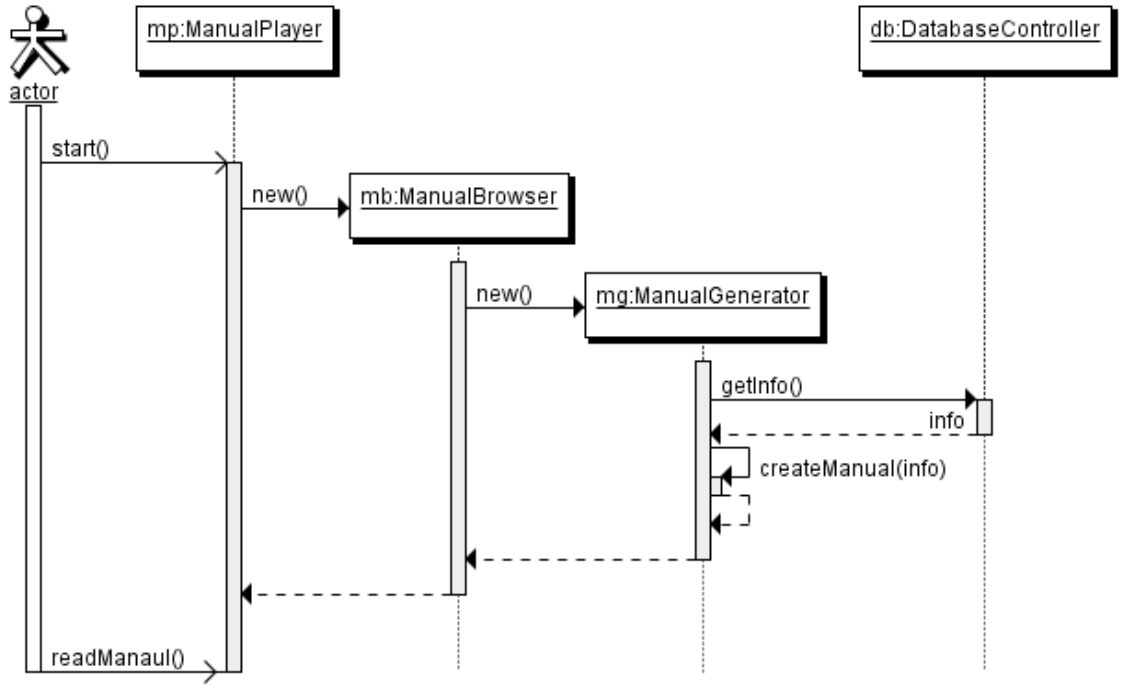

Figura 49 - Diagrama de sequência da geração e visualização do manual de utilizador

5.5. Pacote *systemInterface*

A interface do *ReModeler* foi desenvolvida em *JAVA Swing* [55] com o auxílio de uma biblioteca chamada *InfoNode* [56]. O *Java Swing* é parte integrante da *Java Foundantion Classes* [57] (JFC) e fornece uma API para a construção de interfaces gráficas para programas Java. O *InfoNode* é uma biblioteca que disponibiliza uma API em Java, para permitir a utilização de um sistema de janelas flutuantes. Esta biblioteca pode ser encontrada no *sourceforge* [50] e tem o código fonte aberto ao público.

O sistema de janelas do *ReModeler* é semelhante a qualquer IDE (*Integrated Development Environment*) corrente. Este apresenta um sistema de janelas flutuantes que permitem criar e editar casos de utilização, actores e baterias de teste. Para tal, existe uma janela dedicada aos casos de utilização, uma janela dedicada aos actores e uma janela dedicada às baterias de teste. Estas três janelas são acompanhadas por uma consola de mensagens e de uma janela de visualização de dados. O resto da interacção do utilizador com o *ReModeler* é realizado através de um sistema de janelas de diálogo, que são iniciadas pelas acções do utilizador sobre o sistema. Os componentes relativos à visualização dos cenários e do manual não estão presentes neste pacote, por serem

independentes da interface visual do *ReModeler*. O pacote que agrupa todos os elementos relativos à interface gráfica é o *systemInterface*.

O pacote *systemInterface* é bastante extenso e está dividido em vários sub pacotes. A figura 50 representa o diagrama de pacotes internos ao pacote *systemInterface*.

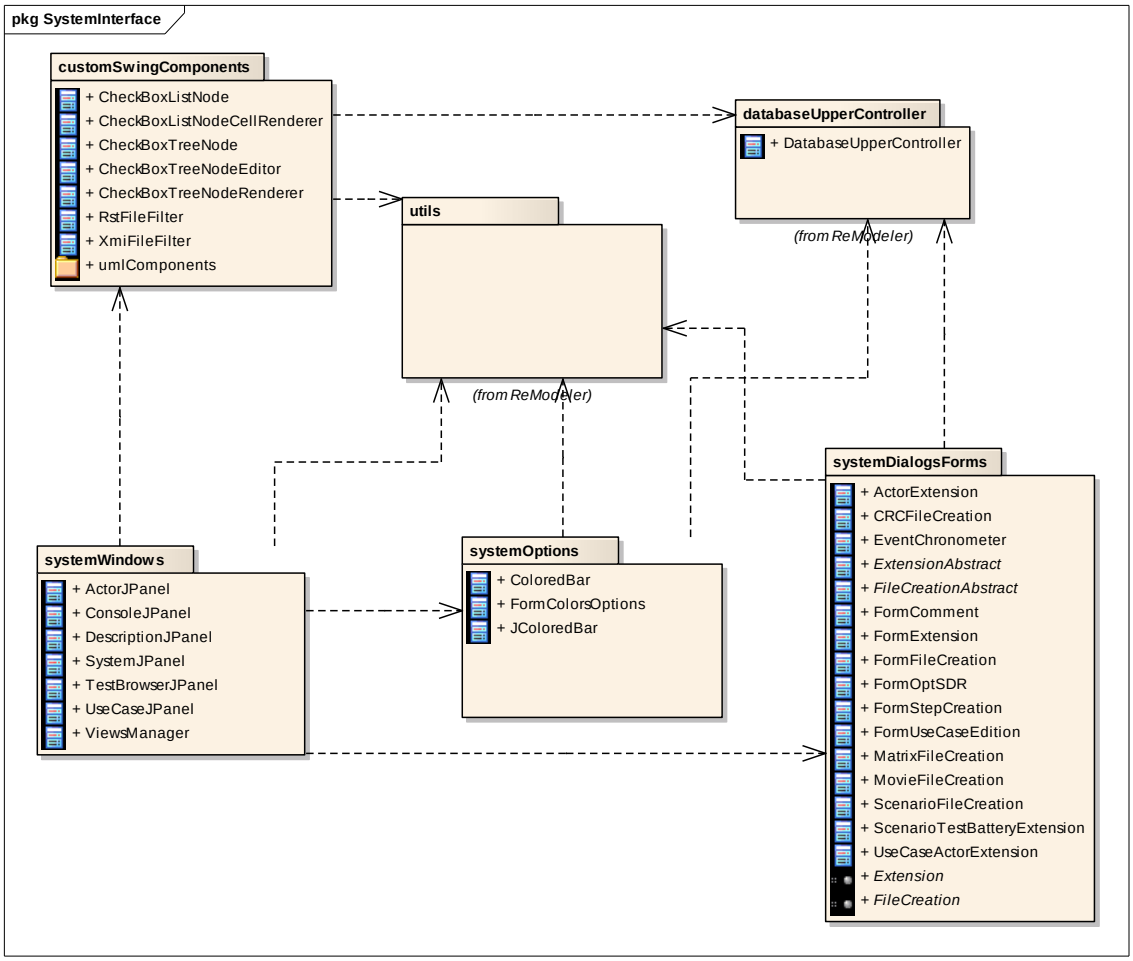

Figura 50 - Diagrama de pacotes do pacote *systemInterface*

5.5.1. Pacote *systemWindows*

O sistema de janelas do *ReModeler* está implementado no pacote *systemWindows*. A figura 51 mostra o diagrama de classes deste pacote.

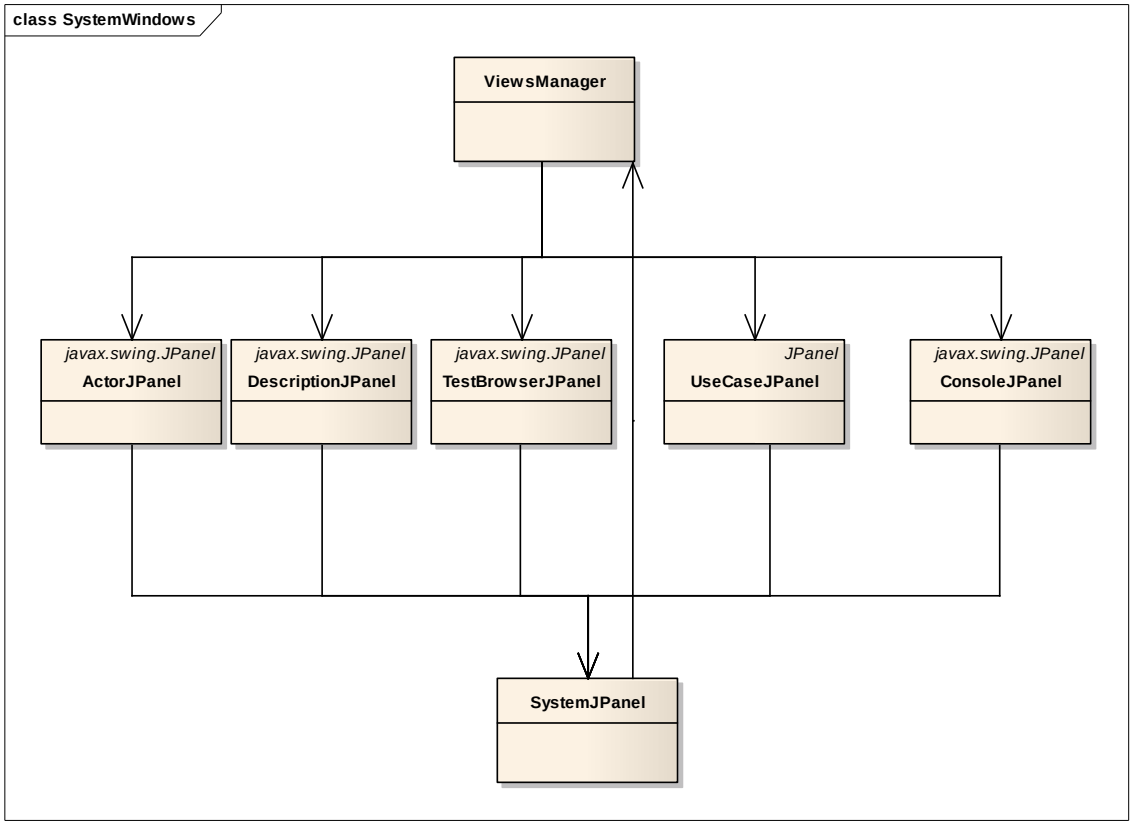

Figura 51 - Diagrama de pacotes do pacote *systemWindow*

A classe *UseCaseJPanel* e *ActorJPanel* contêm as janelas que são responsáveis pela edição e criação de casos de utilização. Nestas janelas aparecem duas estruturas em árvore onde é possível criar, editar e apagar casos de utilização e actores. A classe *ConsoleJPanel* é responsável por uma consola do *ReModeler*, onde aparecem todas as mensagens do sistema. Esta consola serve para um utilizador tomar conhecimento de possíveis erros que estejam a acorrer na utilização do *ReModeler*, ou para receber mensagens informativas resultantes da execução dos vários componentes.

A classe *DescriptionJPanel* pertence à janela que tem a função de mostrar ao utilizador os dados sobre os casos de utilização, cenários, passos, actores, baterias de cenário de teste e outros componentes gerados pelo *ReModeler* [40]. Existe ainda uma última janela no sistema, que corresponde às baterias de cenários de teste, que é criada pela classe *TestBrowserJPanel*.

A classe que faz a gestão das várias janelas existentes no sistema é a classe *ViewsManager*, sendo a comunicação entre elas uma responsabilidade da classe *SystemJPanel*. Por exemplo, se a classe *UseCaseJPanel* quiser comunicar com a *ActorJPanel*, ela vai pedir ao *SystemJPanel* para passar a informação a *ActorJPanel*.

Para que seja possível essa passagem de informação, a classe *SystemJPanel* vai pedir ao gestor das janelas *ViewsManager* que lhe forneça momentaneamente a janela necessária, passando depois a ser possível invocar os métodos apropriados da classe recebida.

5.5.2. Pacote *systemDialogForms*

As janelas de diálogo do *ReModeler* foram agrupadas no pacote *systemDialogForms*. A figura 52 mostra um excerto do diagrama de classes do pacote *systemDialogForms*, que serve para demonstrar a ideia subadjacente ao funcionamento das janelas de dialgo do *ReModeler*. As várias actividades que existem no sistema são feitas através de caixas de diálogo genéricas, que se conseguem adaptar às diferentes actividades que desenvolvem.

No diagrama de classes é possível ver a caixa de diálogo *FormExtension* que actua sobre a interface *Extension.* Este formulário permite realizar todo o tipo de extensões, como por exemplo, no caso de um actor estender outro.

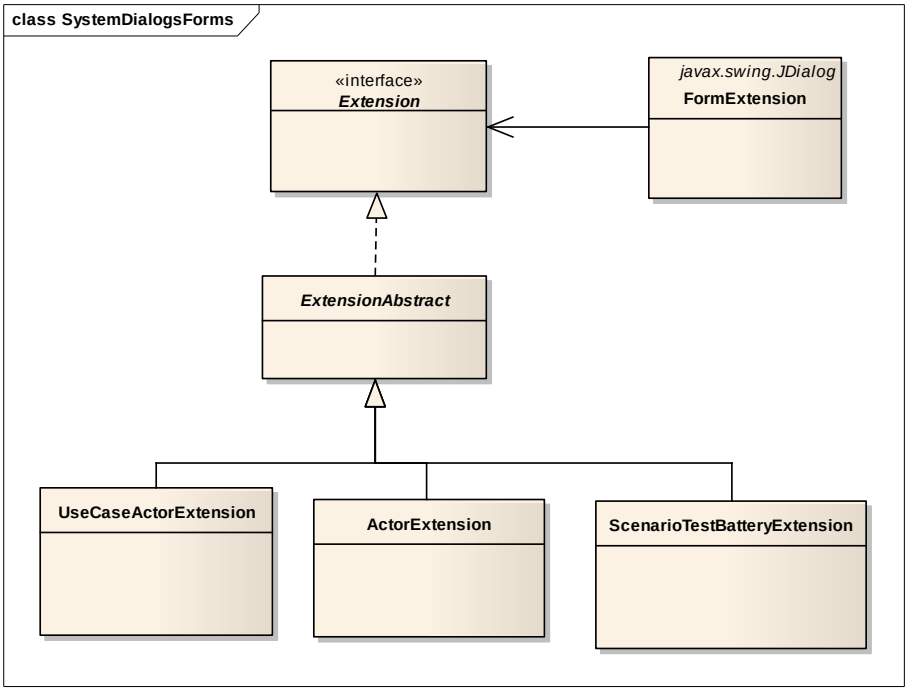

Figura 52 - Excerto do diagrama de classes do pacote *systemDialogForms*

5.5.3. Pacote *customSwingComponents*

Quer o sistema de janelas, quer as caixas de diálogo precisaram de componentes gráficos feitos à medida. Estes componentes foram agrupados no pacote *customSwingComponents*.

5.5.4. Pacote *systemOptions*

Por último temos o pacote *systemOptions*, que ainda está em desenvolvimento, e que serve para o utilizador ter acesso a opções de outros componentes gerados [40] pelo *ReModeler*.

[Esta página foi intencionalmente deixada em branco]

Capítulo 6

Aplicação do RMP a um caso de estudo

Conteúdo

Este capítulo apresenta um caso de estudo da utilização do *ReModeler Process*. No caso de estudo são descritos os vários passos do processo e apresentados os resultados obtidos em cada passo. No fim do capítulo são apresentadas as emaças que existem à validação.

6. Aplicação do RMP a um caso de estudo

Neste capítulo vai ser apresentado um caso de uso referente à aplicação do processo *ReModeler* (RMP) a um sistema real. Para uma validação fidedigna do RMP escolheu-se um sistema de dimensão média e que possibilitasse a repetição do caso de estudo. Não é muito interessante aplicar este processo a exemplos de pequena dimensão já que os sistemas reais tendem a ter dimensões muito elevadas. Assim sendo, pensou-se que um sistema de código aberto, disponível no *sourceforge*, que normalmente não apresenta nenhuma documentação disponível, seria um exemplo indicado para demonstrar as possibilidades da modelação aumentada e, particularmente, do *ReModeler Process.* O processo de documentação do sistema vai ser detalhado passo a passo e, simultaneamente, serão apresentados os resultados obtidos. Por último, no fim do capítulo vão ser apresentadas as ameaças à validação do RMP.

6.1. *Jeneo*

O sistema escolhido para a utilização do processo RMP foi o *Jeneo* [58]. Este sistema é um editor de árvores genealógicas com a capacidade de adicionar fotografias aos nós da árvore. A figura 53 mostra uma imagem do funcionamento da aplicação. Como é possível observar, este sistema é composto por uma janela principal onde é editada a árvore genealógica e por um menu que tem as várias opções que estão disponíveis para a sua utilização. Entre as opções encontramos a possibilidade de salvar a árvore editada ou até exportá-la para o formato SVG [59].

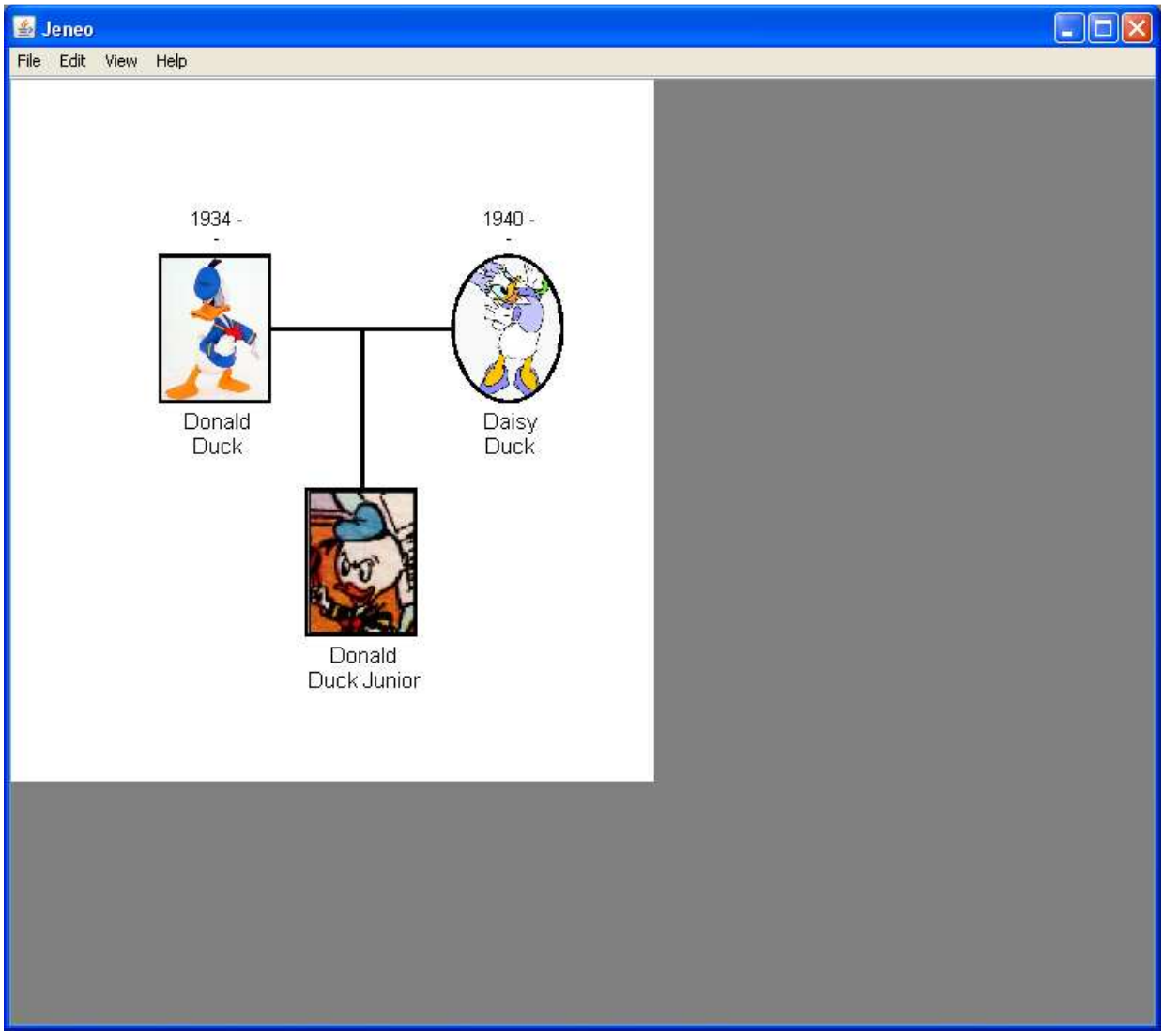

Figura 53 – Janela principal do Jeneo

6.2. Aplicação do *ReModeler Process* ao *Jeneo* – (Processo detalhado)

Perante um sistema não documentado, o primeiro passo do *ReModeler Process* é efectuado por um perito do domínio do problema. Este, para criar a documentação de um sistema, vai descrevê-lo de uma forma precisa através de casos de utilização, que por sua vez serão descritos através de cenários estruturados. No caso do *Jeneo* foram identificados alguns casos de utilização, como mostra a figura 54. De seguida, são apresentados os vários resultados obtidos através do RMP, para o caso de utilização *Inserir filha.*

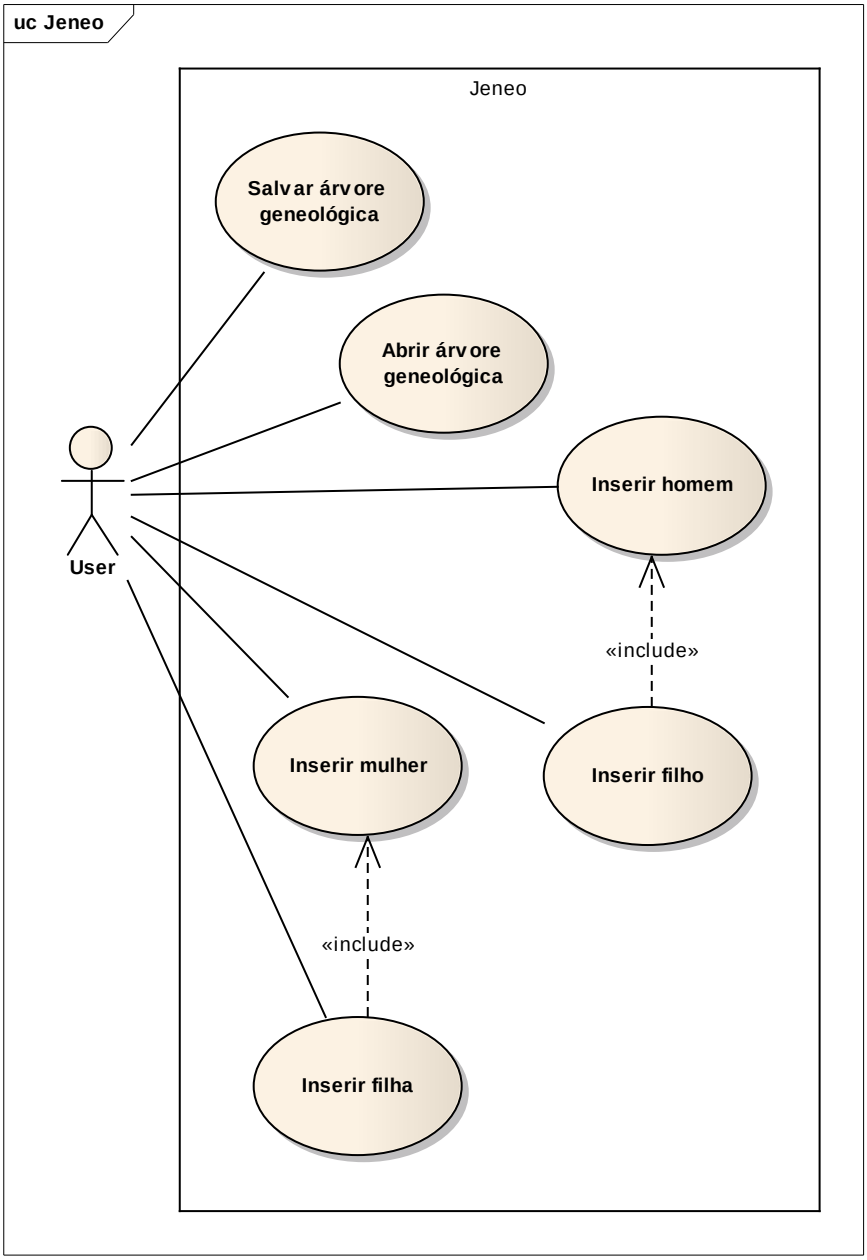

Figura 54 – Casos de utilização do Jeneo

6.2.1. Descrição estruturada de cenários do caso de utilização *Inserir filha* .

Pré-condição:

No sistema já deve ter sido previamente inserido um homem e uma mulher.

Cenário principal

1. Inclui: Inserir mulher
2. O utilizador com o rato selecciona os pais.

3. O utilizador escolhe a opção *pick child* do menu *edit.*

4. O utilizador selecciona a filha para os pais seleccionados.

5. O sistema apresenta a ligação entre os pais e a filha.

Cenário alternativo

1. No ponto 4 o utilizador não selecciona a filha, regressando ao ponto 3.

6.2.2. Execução do *ReModeler*

Com a identificação dos casos de utilização, o primeiro passo do *ReModeler process* fica concluído. Os casos de utilização podem ser descritos numa ferramenta UML e posteriormente exportados para ficheiros XMI, mas também podem ser criados directamente no *ReModeler*, aquando da sua execução.

O passo seguinte consiste em compilar o *ReModeler* com o sistema a documentar. Neste processo, o *ReModeler* é "entrelaçado" (*weaved*) com o sistema a documentar, criando-se um novo sistema conjunto. Para sistemas de uma média e grande dimensão este processo pode demorar algum tempo. Finalizada a compilação, é necessário executar o novo sistema resultante. Inicialmente aparece uma janela que pertence ao *ReModeler* (figura 55), que permite, entre outras opções, importar o diagrama de casos de utilização, em formato XMI [39].

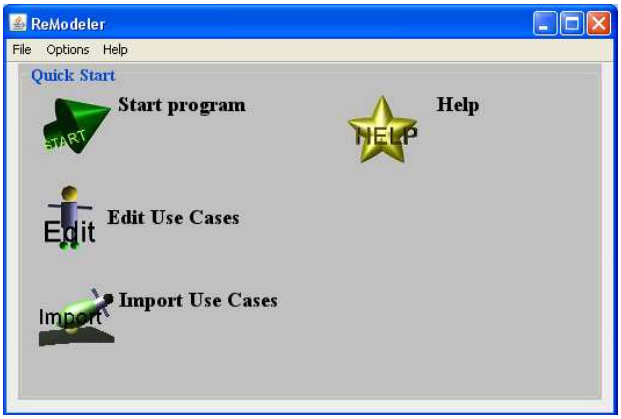

Figura 55 – Janela inicial do ReModeler.

Para este caso de estudo, optou-se por criar directamente os casos de utilização no próprio *ReModeler*. Para o fazer, accionou-se directamente a janela principal do *ReModeler* (figura 56) e criaram-se os casos de utilização anteriormente descritos.

Na janela principal *ReModeler* podemos encontrar cinco janelas flutuantes, identificadas na figura 56. As duas primeiras janelas correspondem à edição e manutenção dos casos de utilização e dos respectivos actores. Alguns dos outputs do *ReModeler* são visualizados na janela 3. É aqui que é possível ler as descrições de cenários estruturadas, criadas pelo utilizador. Algumas das mensagens que o sistema disponibiliza para o utilizador são apresentadas na janela 4, como por exemplo erros internos que possam ocorrer durante a execução do *ReModeler*. Por último, na janela 5, temos a possibilidade de edição e criação de baterias de cenários de teste.

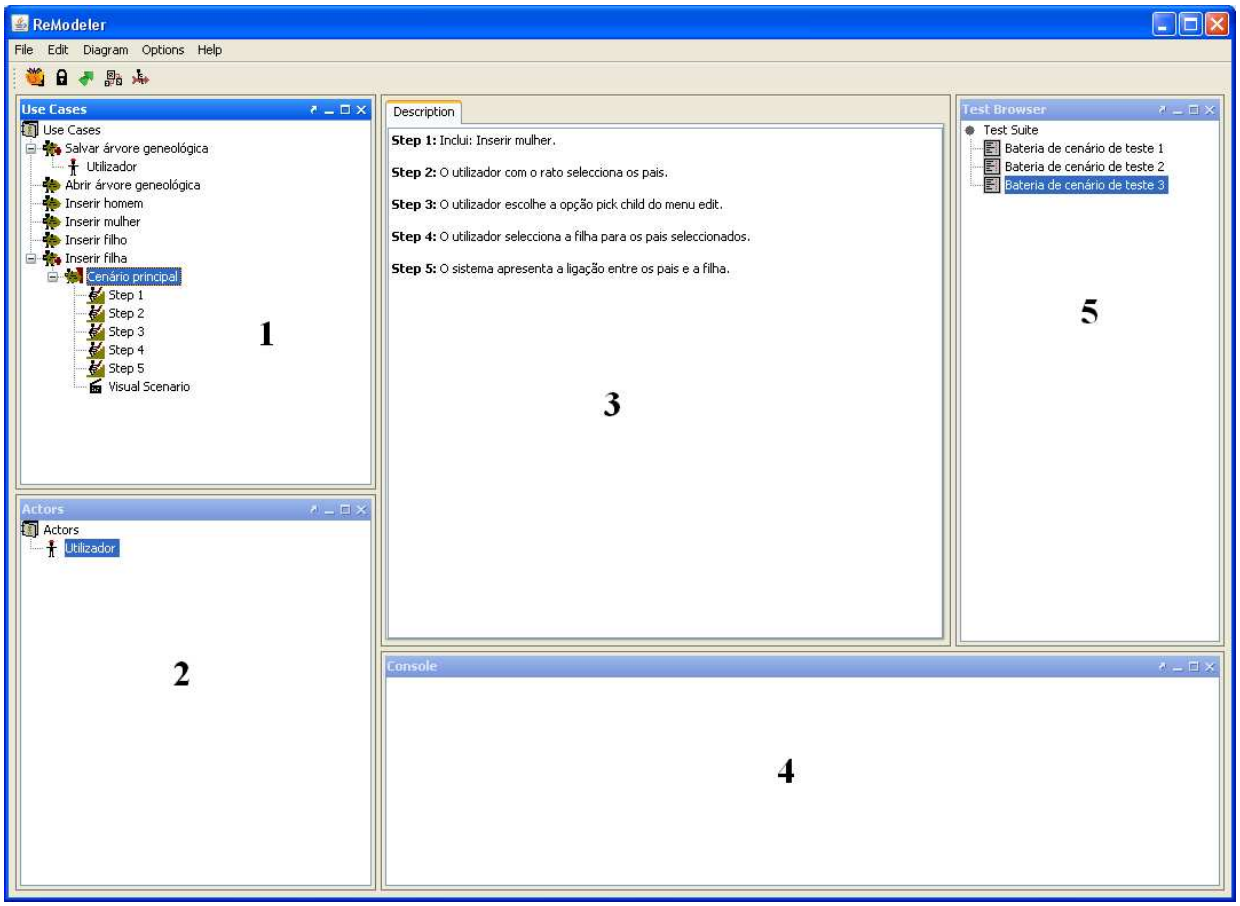

Figura 56 – Janela principal do *ReModeler*.

6.2.3. Criação dos casos de utilização e dos respectivos cenários no RMP

Para a criação e edição dos casos de utilização, o *ReModeler* disponibiliza uma árvore. Nessa mesma árvore encontram-se os seus cenários, juntamente com os respectivos passos. Na figura 57 temos a edição do caso de utilização *Inserir filha*, e na figura 58 temos a edição dos seus cenários. O nível de detalhe atingido com este processo é muito importante para a posterior concretização do mesmo e é raramente encontrado nas ferramentas UML, onde normalmente os cenários são descritos simplesmente como notas UML. Na figura 59 vemos o resultado deste processo. É possível observar a descrição dos cenários criada pelo utilizador na janela central do *ReModeler*.

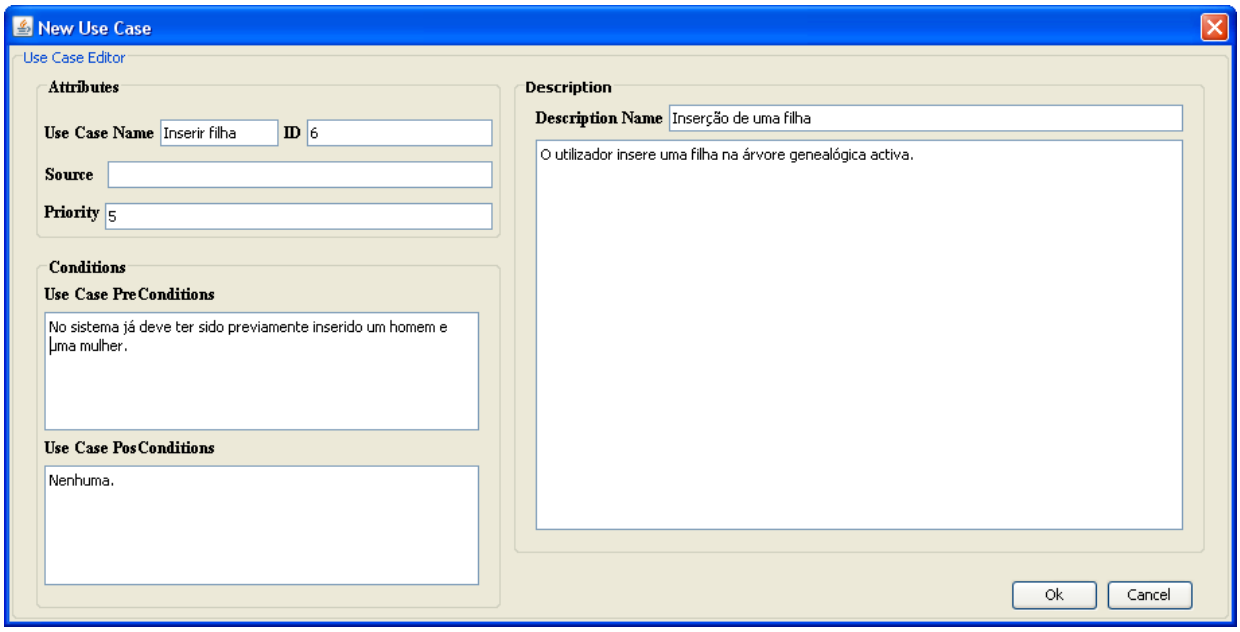

Figura 57 – Edição dos casos de utilização do *Jeneo* no *ReModeler*.

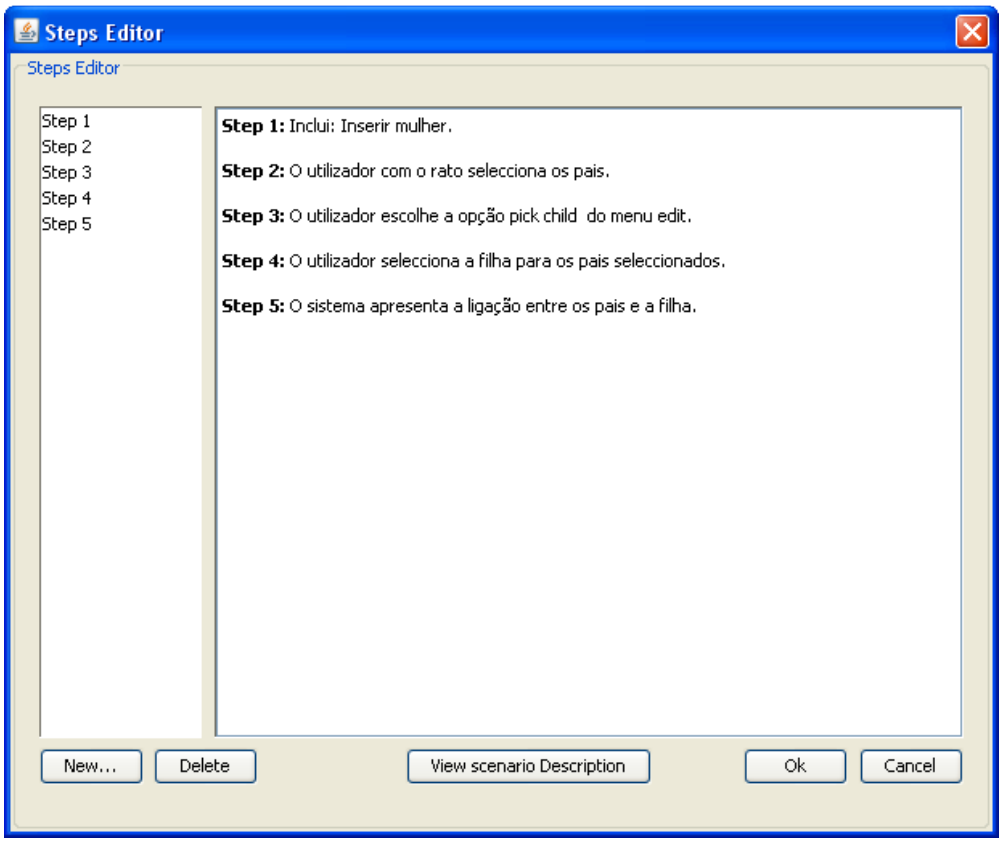

Figura 58 – Edição dos passos do cenário principal do caso de utilização *Inserir filha*.

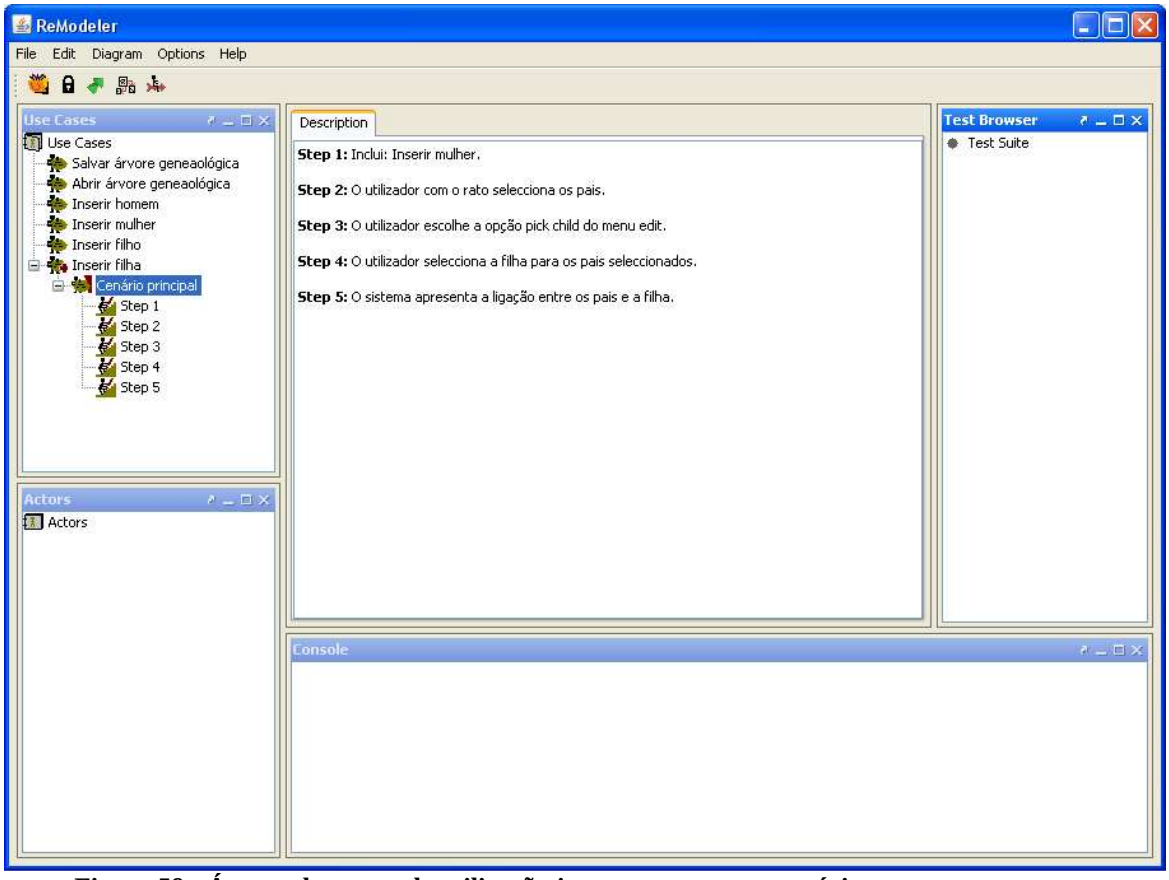

Figura 59 - Árvore dos casos de utilização juntamente com os cenários e os seus passos.

6.2.4. Criação dos cenários visuais

Seguidamente, o *Jeneo* foi posto a correr através do *ReModeler*. Nesta altura aparecem duas janelas no ecrã, uma relativa ao *Jeneo* e outra relativa ao *ReModeler* (figura 60). O passo seguinte do *ReModeler process* consiste na criação dos cenários visuais. Como já foi referido, os cenários visuais são um conceito de modelação aumentada que permite melhorar a nossa compreensão sobre um sistema de informação. Estes permitem-nos estabelecer ligações de rastreabilidade entre os requisitos de um sistema e o seu modelo arquitectural. No *ReModeler process* a sua criação passa por executar cada cenário do sistema separadamente, ou seja, executar o sistema para cada cenário, passo a passo, marcando numa janela à parte o passo que se está a executar em cada momento. Inicialmente aparece uma janela (figura 61) onde é possível escolher a localização do cenário visual e os cenários (externo e interno) que se querem capturar. Para se obter um cenário visual é preciso fazer uma captura de ambos, simultaneamente.

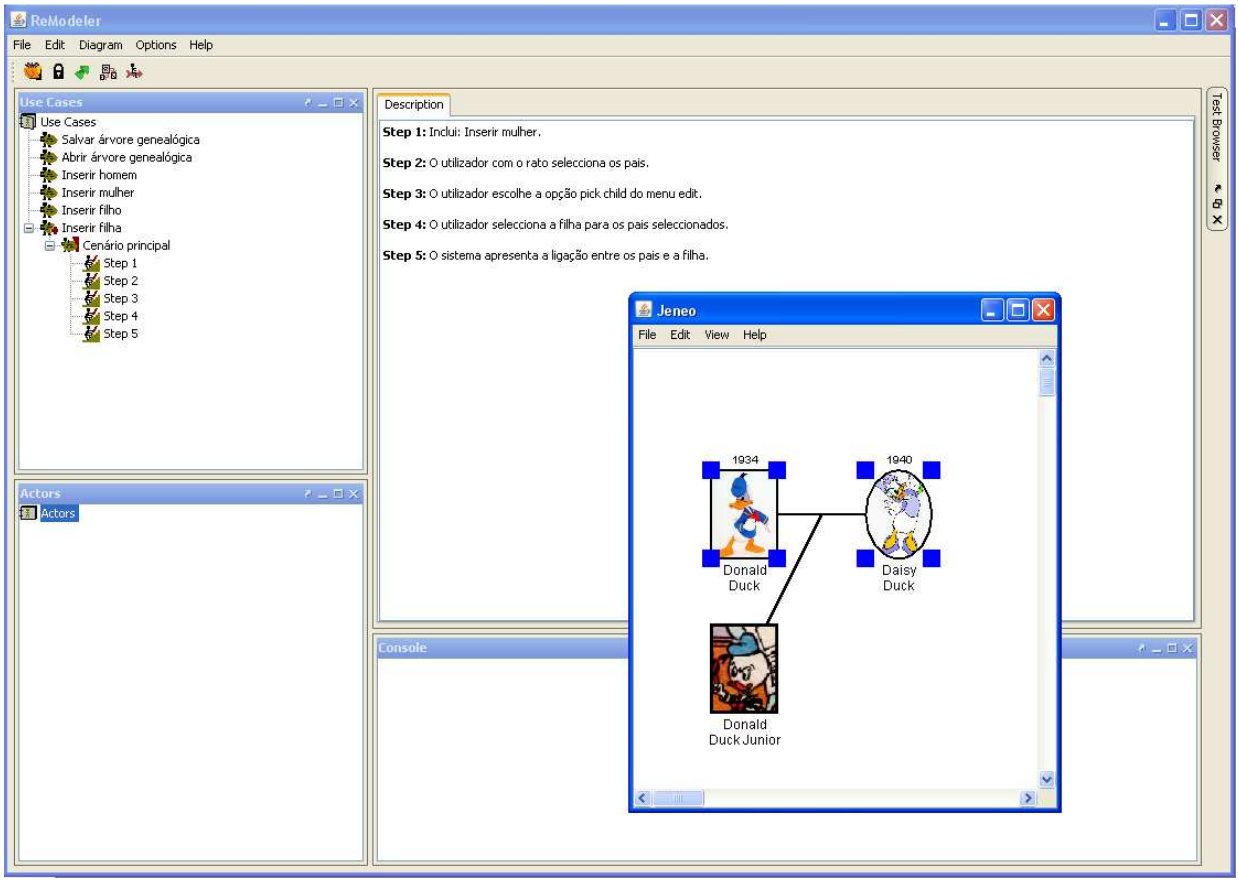

Figura 60 - *ReModeler* juntamente com *Jeneo*.

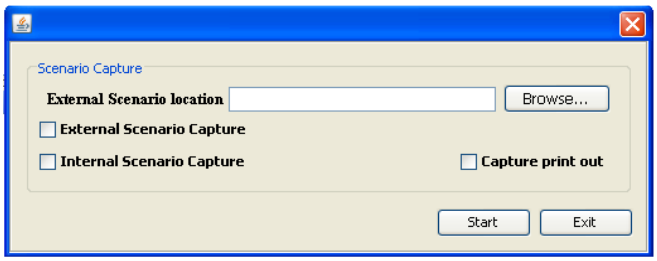

Figura 61 - Inicio da captura de um cenário visual

A figura 62 mostra a captura do cenário principal do caso de utilização *Inserir filha*. A janela do lado direito mostra onde é feita a marcação dos vários passos dos cenários. Considera-se que os passos de um cenário são sequenciais, por isso sempre que o utilizador quer marcar um passo, só tem carregar no botão *mark step* (na figura marcado por uma circunferência) para este ficar automaticamente marcado. No final da captura de cada cenário, o cenário visual correspondente fica automaticamente criado.

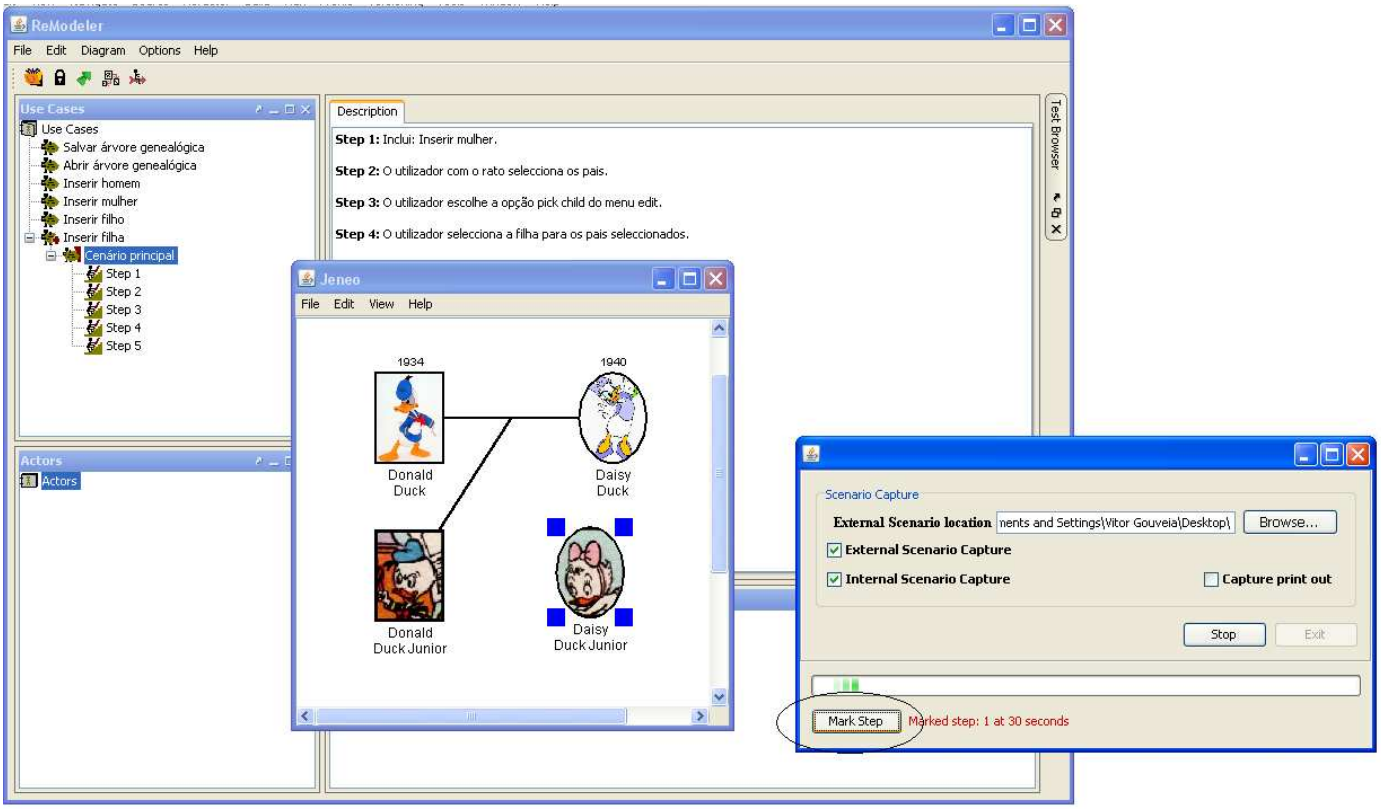

Figura 62 - Marcação de passos do cenário principal do caso de utilização *Inserir filha.*

6.2.5. Reprodução de um cenário visual

Após a criação de um cenário visual este pode ser reproduzido quantas vezes for necessário no *player* do *ReModeler*. O cenário visual encontra-se na árvore de casos de utilização do *ReModeler* debaixo do cenário que lhe é respectivo. Para este ser reproduzido basta seleccioná-lo, carregar no botão direito do rato e seleccionar a opção *Play* (figura 63). Antes de começar a reprodução, surge uma janela de filtragem (figura 64). Esta é extremamente importante pois vai permitir que sejam escolhidas as classes e os métodos que queremos reproduzir num cenário interno [40]. Visto que os diagramas de sequência do cenário interno tendem a crescer muito, é importante existir esta opção. Neste exemplo, estamos a retirar alguns dos componentes gráficos do *Jeneo*. Por último, falta referir que o cenário interno é gerado de acordo com o que está descrito em [40]. Aqui é explicado como é feita a captura interna e como é validado o diagrama de sequência gerado.

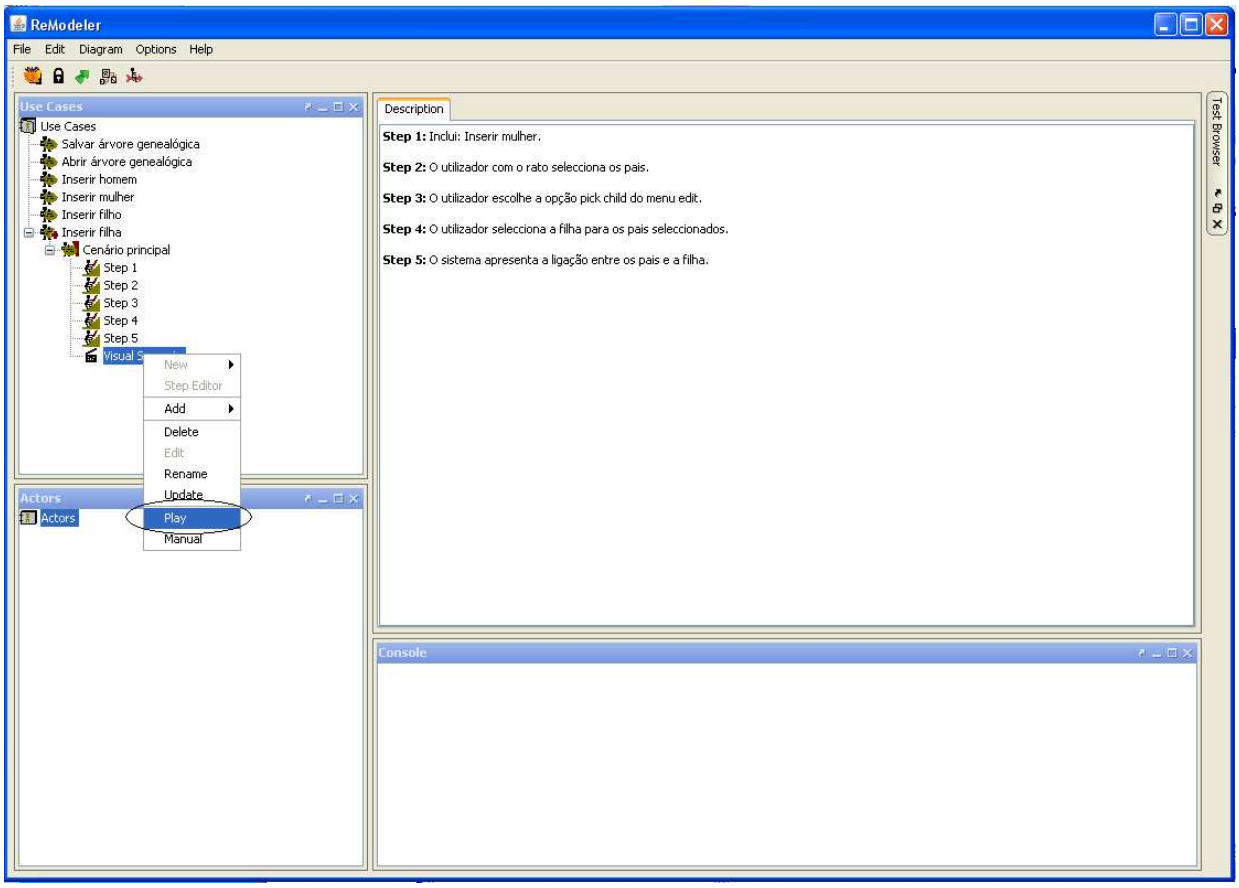

Figura 63 – Inicio da reprodução de um cenário visual.

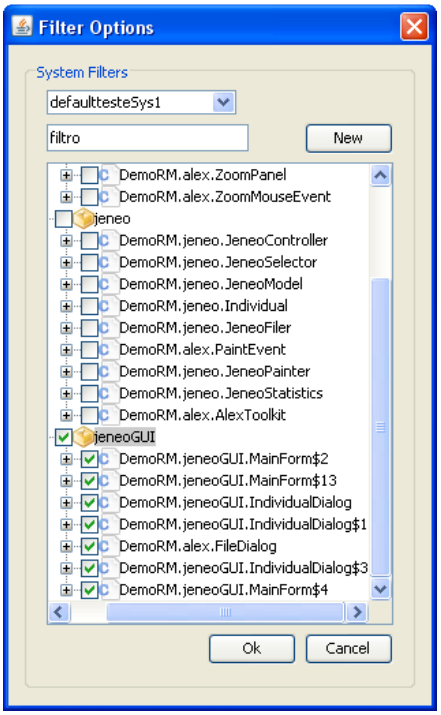

Figura 64 - Janela de filtragem

Na figura 65 é possível observar o *player* dos cenários visuais. Aqui encontramos as opções básicas (identificadas pela seta verde) que se encontram em qualquer *player* de vídeos: começar a reprodução, parar, pausar e andar para a frente e para trás no filme. É ainda possível exportar o diagrama do cenário interno para um ficheiro externo, como por exemplo pdf (seta laranja) ou visualizá-lo em *full screen* (seta azul). No lado esquerdo, ao alto, temos uma barra temporal que permite aumentar ou diminuir a velocidade de reprodução. O cenário externo é reproduzido na janela identificada com o número 1, a legenda do cenário externo, que corresponde ao passo marcado, aparece na janela identificada com o número 2, o cenário interno é visualizado na janela identificada com o número 3. Os passos da descrição de cenários estão identificados no cenário interno com janelas coloridas. Estas envolvem as invocações que ocorreram nesse passo. Por último, por baixo da janela das legendas encontramos uma barra temporal do cenário visual.

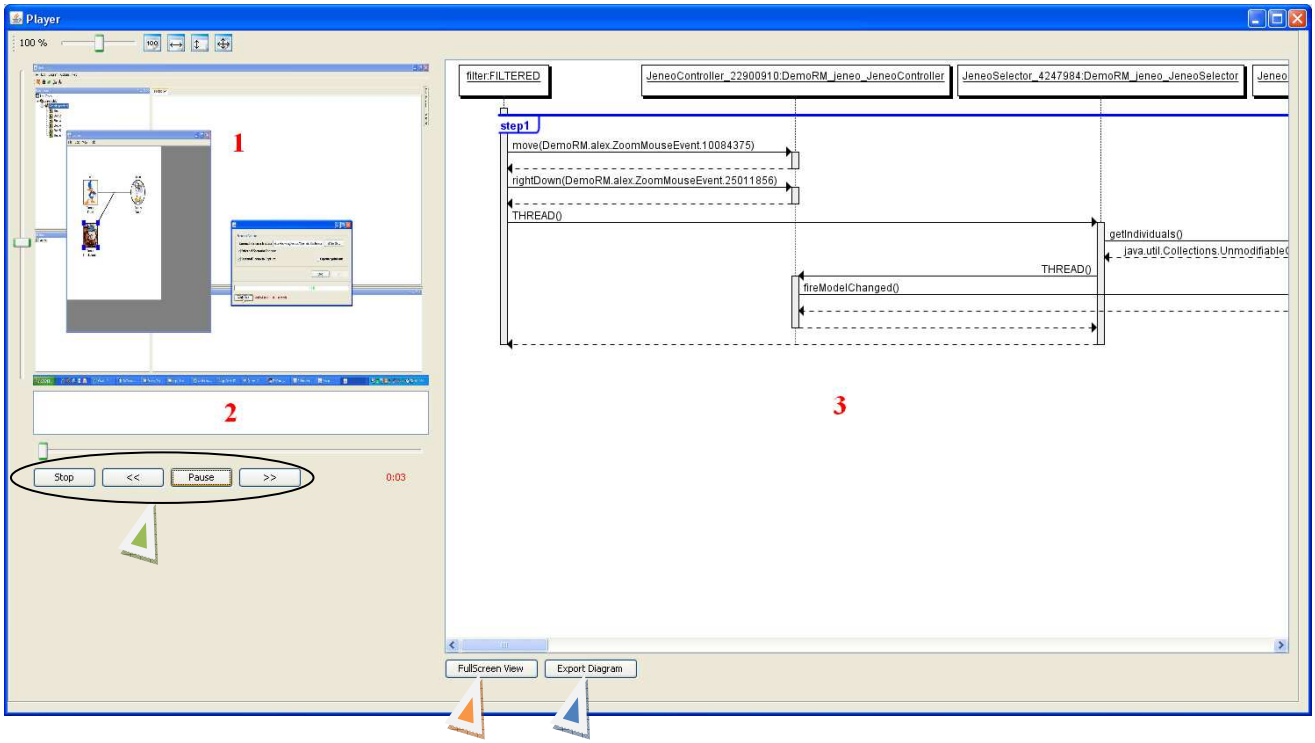

Figura 65 - *Player* dos cenários visuais do *ReModeler*

6.2.5.1. Sincronização entre os dois mundos

Durante a reprodução de um cenário visual temos dois mundos a correr em simultâneo, o exterior e o interior de uma execução de um software. A sincronização entre ambos é garantida através do tempo do cenário externo. Durante a reprodução de um cenário visual basta utilizar o tempo do cenário externo para saber quais as mensagens do cenário interno que devem ser mostradas. Estas mensagens, aquando da captura, foram marcadas temporalmente para puderem serem sincronizadas com o cenário externo. Como os cenários são capturados simultaneamente consegue-se uma boa aproximação entre o que acontece exteriormente e interiormente. Na figura 66 e figura 67 vemos o que acontece na transição entre as cenas do filme (passos da descrição de cenário marcados previamente pelo utilizador). No diagrama de sequência do cenário interno é criada uma moldura para cada cena do filme. Esta permite saber que métodos foram invocados em cada cena do filme. Por último, na janela de

legendagem, vai aparecer a legenda correspondente à cena marcada. Como já referido a legenda é o passo da descrição do cenário previamente criada.

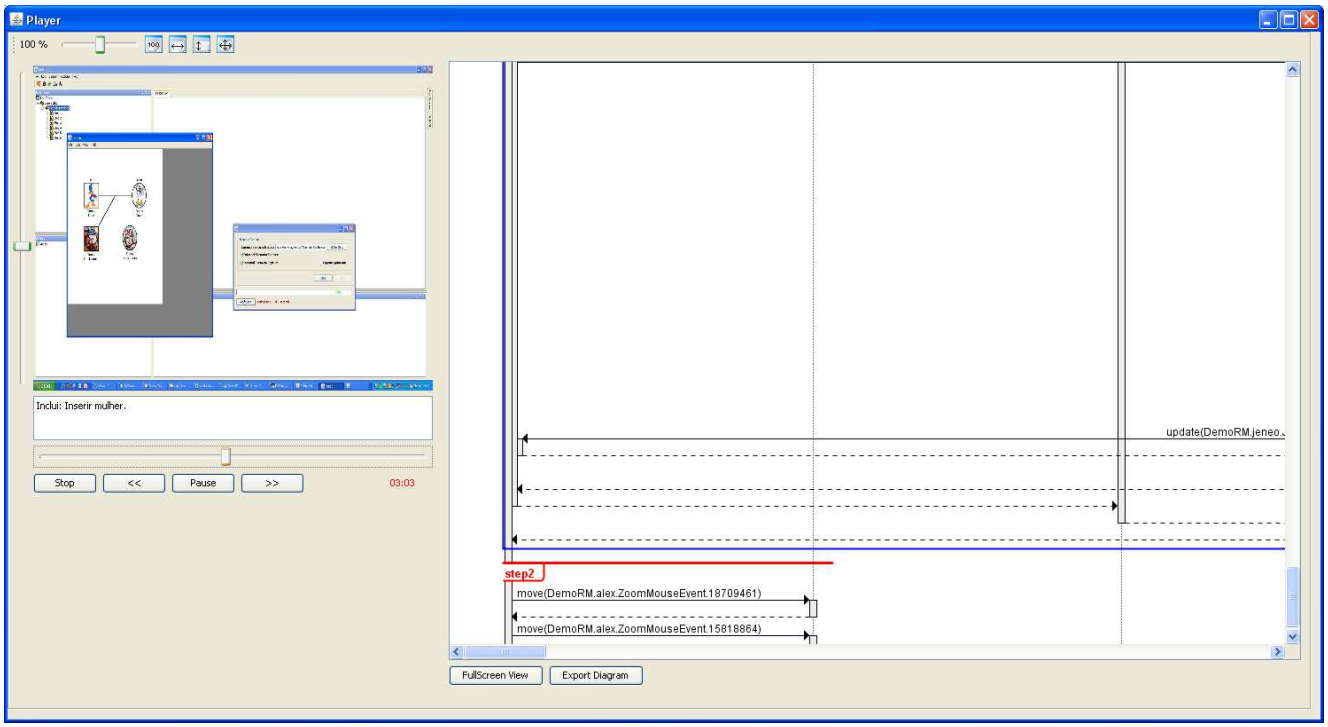

Figura 66 - Fim do passo 1 e início do passo 2

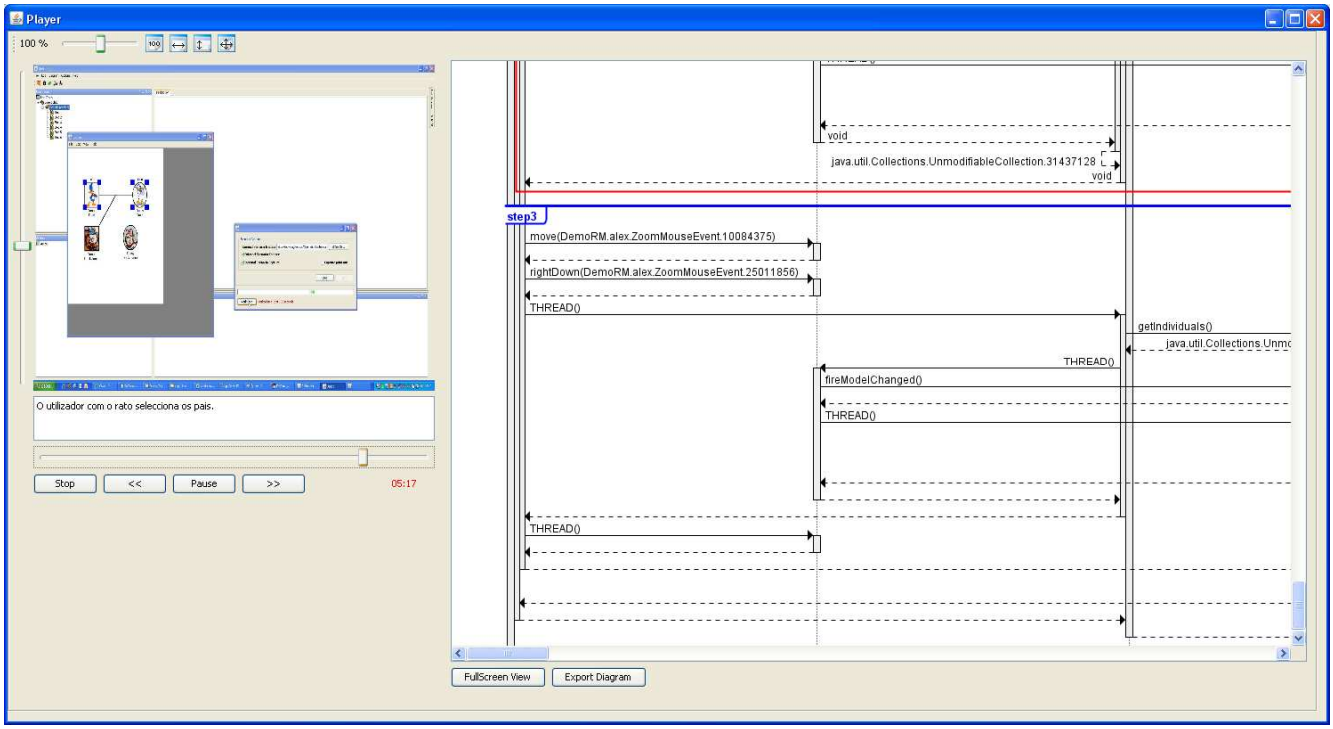

Figura 67 - Fim do passo 2 e início do passo 3

6.2.6. Geração do manual de utilizador

Simultaneamente com o processo das capturas é gerado o manual de utilizador. As figuras 68, 69 e 70 mostram páginas do manual de utilizador gerado para o *Jeneo*, visto através do *player* do *ReModeler*. Neste *player* a navegação entre as várias páginas do manual é feita através de botões, que permitem ver a página seguinte e a anterior. No lado esquerdo temos escrito o caso de utilização corrente, o cenário corrente, qual o passo que estamos a visualizar e a sua descrição. No lado direito temos a imagem que ilustra o passo e que foi marcada pelo utilizador, durante a criação do cenário visual. O *player* do manual de utilizador tem a possibilidade de exportar o manual para um formato não proprietário (HTML). As figuras 71 e 72 são um exemplo do manual de utilizador nessa versão.

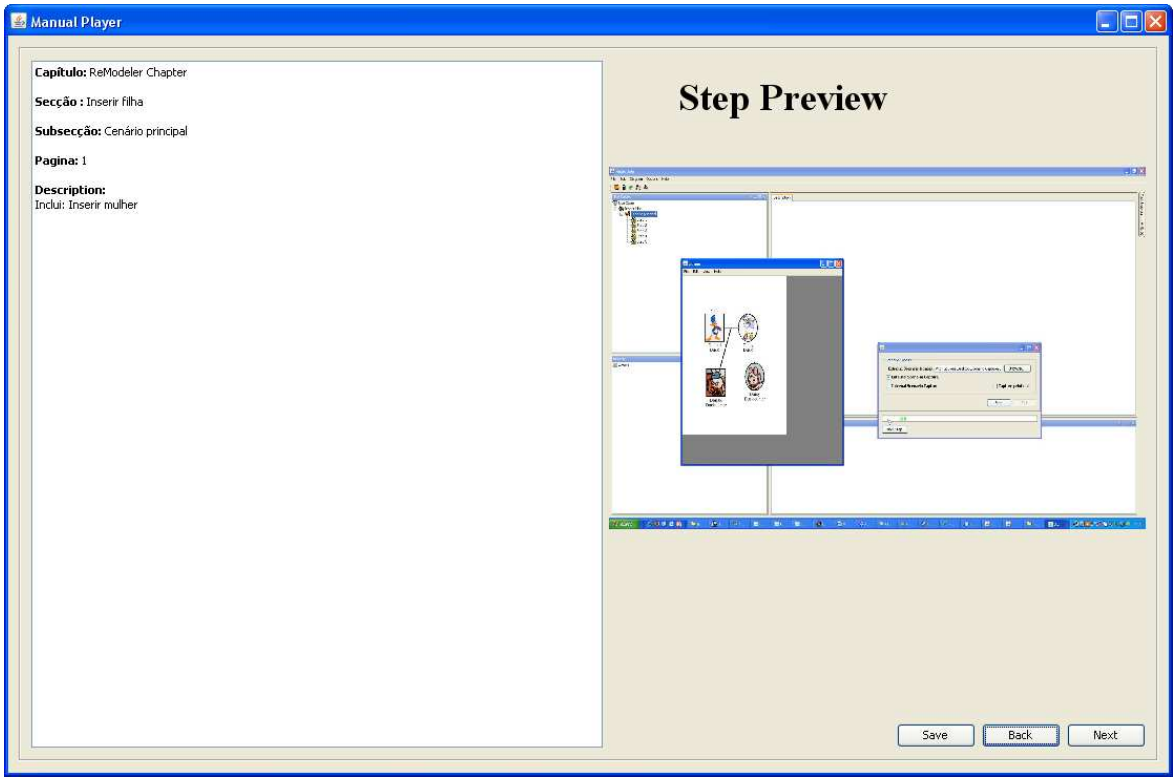

Figura 68 – Página do manual de utilizador relativa ao primeiro passo do cenário principal do caso de utilização *Inserir filha*.

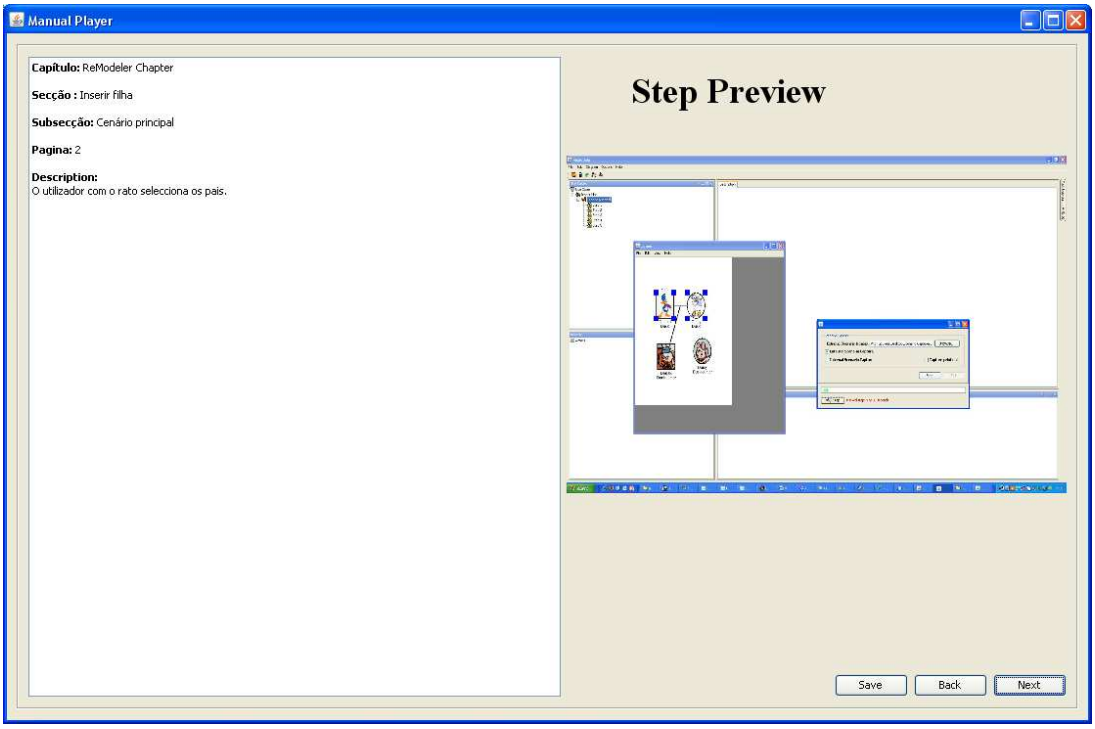

Figura 69 – Página do manual de utilizador relativa ao segundo passo do cenário principal do caso de utilização *Inserir filha.*

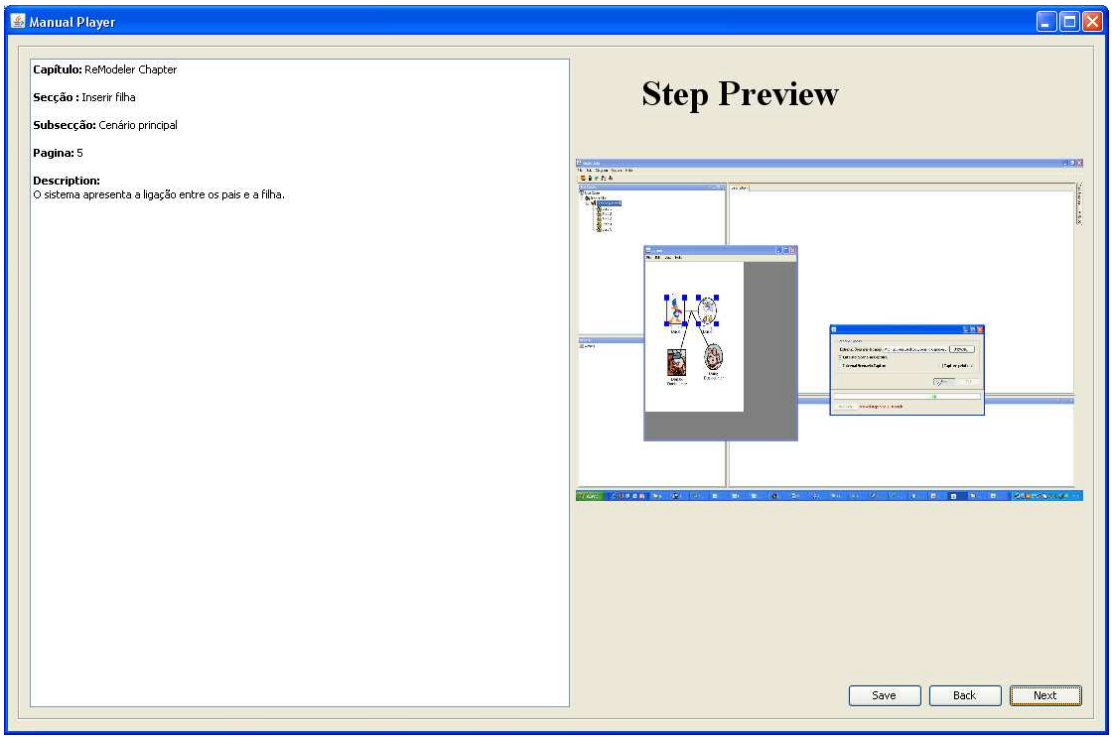

Figura 70 - Página do manual de utilizador relativa ao passo final do cenário principal do caso de utilização *Inserir filha.*

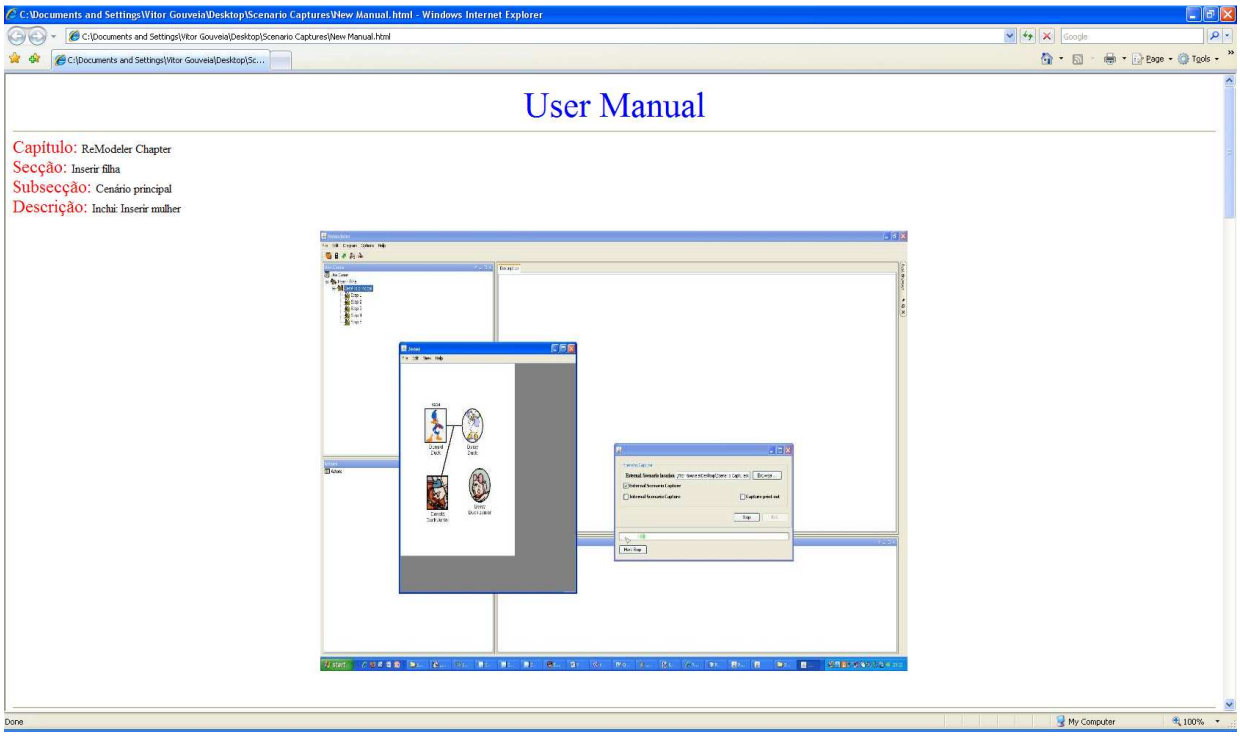

Figura 71 - Versão HMTL do manual de utilizador (1)

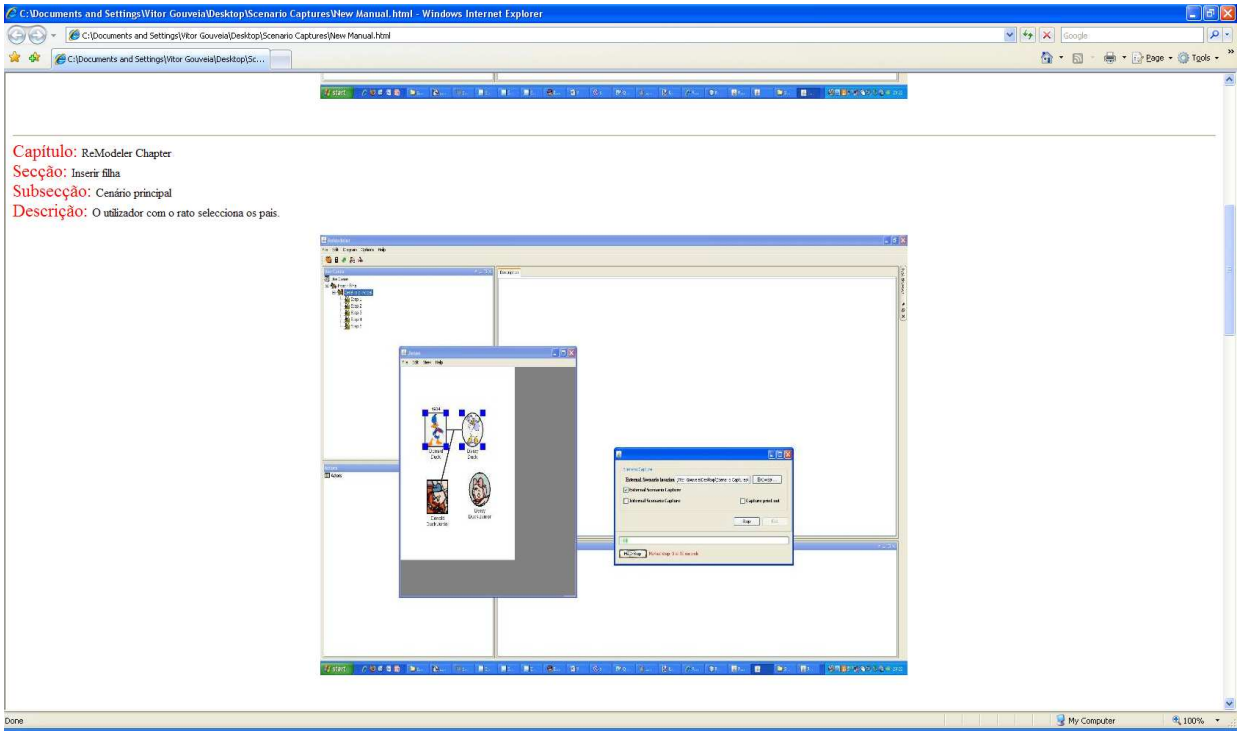

Figura 72 - Versão HTML do manual de utilizador (2)

6.2.7. Criação de baterias de cenários de teste

O último passo do *ReModeler Process* consiste na criação de baterias de cenários de teste. O *ReModeler* permite que os vários cenários declarados sejam escolhidos para fazerem parte de uma ou mais baterias de cenários de teste (figura 73). A criação destas baterias permite que mais tarde possam ser feitos testes de cobertura aos cenários de teste. Esta fase do RMP apenas permite que os cenários sejam agrupados em baterias de cenários de teste. A sua utilização em testes de cobertura não está incluída no âmbito desta dissertação.

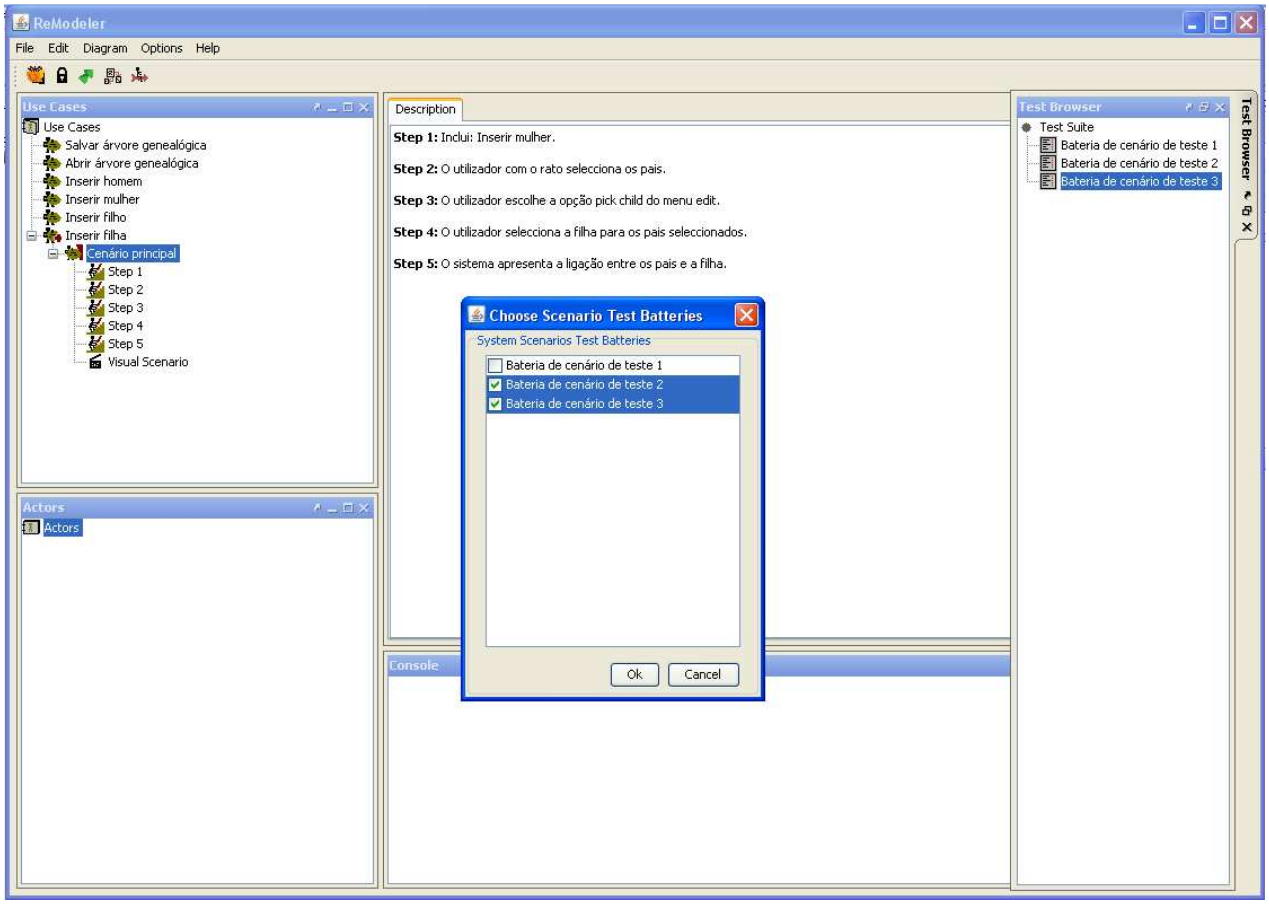

Figura 73 - Inserção dos cenários nas baterias de cenários teste.

6.3. Ameaças à validação

Neste capítulo, foi exposto um caso de estudo, onde foram apresentados os resultados obtidos com a aplicação do RMP. É fácil reparar que não existe nenhuma comparação com processos alternativos. Este ponto apresenta-se como a maior ameaça

à validação desta proposta. Idealmente, deviam-se ter executado várias alternativas e devia ter sido feita uma comparação quantitativa com o RMP, baseada em indicadores temporais, retirados de cada alternativa testada. Por outro lado, é de notar que este processo ainda não foi aplicado num contexto empresarial. Na realidade, apenas com sistemas reais será possível demonstrar os verdadeiros ganhos que advêm da sua utilização. Acredita-se no entanto, que este processo traga muitos benefícios aquando da sua utilização, uma vez que é totalmente automático, após a identificação e descrição dos casos de utilização.

O RMP pretende ser uma abordagem que se estende a todo o tipo de sistemas. Neste momento, apenas temos resultados para sistemas Java, porque os componentes responsáveis pela captura dos cenários visuais estão escritos nessa mesma linguagem. Como tal, é necessário perceber como é possível estender a sua utilização a outro tipo de sistemas.

Notou-se que o sistema que resulta da combinação do sistema a documentar com o *ReModeler*, apresenta alguma degradação ao nível do desempenho. Esta situação, é também uma ameaça à validação deste processo, visto que não é possível determinar que impacto pode haver em sistemas reais.

Outro ponto importante do RMP é que este depende fortemente da ferramenta *ReModeler* e como tal sofre as limitações que esta apresenta. A primeira limitação que se pode apontar é relativa ao próprio editor dos casos de utilização. Este ainda se apresenta muito simples, sem opções importantes, como o *Include* e o *Extends* para os casos de utilização.

Notou-se também que os diagramas de sequência e do cenário interno tendem a crescer muito. Isto poderá também ser uma emaça ao RMP visto que o utilizador deixa de conseguir perceber a realidade subadjacente ao mesmo. Apesar de existir filtragem este problema não está resolvido, uma vez que não são identificados ciclos que possam ocorrer na execução do código, e compactar a representação do diagrama. Assim sendo, os diagramas continuam a apresentar uma dimensão relativamente grande o que torna difícil a sua compreensão.

A última ameaça encontrada é relativa à reprodução de um cenário visual. O cenário interno, composto por um diagrama de sequência, tende a crescer em dois

sentidos: horizontal e vertical. Isto poderia não ser um problema se os diagramas gerados fossem de dimensão reduzida, mas como tal não acontece, este crescimento em dois eixos tende a dificultar a visualização do diagrama de sequência, sendo necessário ao utilizador, parar muitas vezes a execução para saber o que está a acontecer no momento que está a ser visualizado.

No capítulo das conclusões estão apresentados os passos do trabalho futuro que visam responder às ameaças aqui descritas.

Capítulo 7

Conclusões e Trabalho Futuro

Conteúdo

Este capítulo encerra a dissertação. São apresentadas as conclusões obtidas e de que forma foram atingidas as contribuições da dissertação. A encerrar é traçado o caminho para o trabalho futuro que fica aberto por esta dissertação.

7. Conclusões e Trabalho Futuro

7.1. Conclusões

Nesta dissertação apresentam-se uma série de novos conceitos de forma a melhorar a rastreabilidade dos requisitos de software. Um dos maiores obstáculos enfrentados foi o tempo necessário para a realização do protótipo e posterior validação do mesmo. Seguidamente, são recapituladas as contribuições previstas desta dissertação e de que forma elas foram atingidas. No final, é apresentado o caminho traçado para o trabalho futuro.

7.1.1. Revisão das contribuições da dissertação

1. Mitigar o problema dos sistemas legados

   Os sistemas legados não são novidade no panorama actual do desenvolvimento de sistemas de informação. Na realidade, um sistema legado pode ser definido como "qualquer sistema de informação que dificilmente resiste à mudança e à evolução [7]", o que por si só é bastante fácil de encontrar. Normalmente estes sistemas para além de estarem em operação (o que por si só levanta problemas vários à sua evolução), crescem de forma descontrolada ao longo de um período de tempo alargado, o que leva a que a sua modificação acarrete elevados custos [7]. A sua modificação costuma ser muito dispendiosa, porque a sua documentação não existe e as tentativas de perceber o sistema são muito demoradas. No entanto, foram já apresentadas algumas soluções para o problema dos sistemas legados [7]. Normalmente as soluções caem em três categorias distintas: voltar a escrever o sistema, criar novas interfaces de comunicação entre os componentes para os tornar mais acessíveis e, por último, migrar o sistema para um ambiente mais flexível

mantendo os dados e as funcionalidades do sistema legado. O que se propôs nesta dissertação não cai em nenhuma das categorias anteriormente referidas. O que se apresentou foi a possibilidade de facilitar a compreensão um sistema legado através da sua execução. O processo passa por entrelaçar o sistema legado com a ferramenta desenvolvida usando a tecnologia dos aspectos e executá-lo. Através dos cenários visuais, que têm a capacidade de apresentar duas visões diferentes e complementares sobre o mesmo mundo e dos outros artefactos da modelação aumentada, é possível compreender o funcionamento do sistema.

2. Introduzir uma nova técnica da rastreabilidade dos requisitos

   No segundo capítulo foram apresentadas algumas das técnicas de rastreabilidade dos requisitos actuais, mas que, de uma maneira ou de outra, apresentam algumas lacunas. Com os cenários visuais introduzo uma nova perspectiva sobre o problema. De uma forma automática, com o auxílio dos modelos UML e com uma intervenção mínima por parte do utilizador, é estabelecida uma ligação de rastreabilidade entre os casos de utilização, cenários e o código fonte do sistema. No editor de cenários visuais temos uma aliança cuja existência não se encontra nas ferramentas actuais: os casos de utilização estão directamente ligados à descrição estruturada de cenários. Os cenários visuais ampliam ainda mais o nosso conhecimento sobre um cenário de um caso de utilização, pois permitem, perceber, de uma forma simples e aparentemente eficaz, como um cenário está implementado. Com a combinação das duas realidades de um cenário visual, temos uma ligação fortíssima entre requisitos e implementação, que se releva muito útil, principalmente para um sistema legado. A criação de ligações de rastreabilidade para este tipo de sistema passa a ser tão simples como executá-lo, marcando e descrevendo abreviadamente o que se vai fazendo em cada passo.

3. Facilitar a validação dos requisitos de um sistema de software

   Como referido no terceiro capítulo, o manual do utilizador é onde estão descritas as funcionalidades de um sistema e a maneira de realizá-las. A combinação dos cenários visuais com o manual de utilizador automático vai possibilitar uma validação mais fácil dos requisitos perante um utilizador de um sistema. No manual

automático, temos não só a descrição dos requisitos, como também os passos para a sua realização. Perante isto, para um utilizador validar um sistema bastar-lhe-á ler o manual e executar os cenários visuais respectivos.

No manual temos a descrição de todas as funcionalidades de um sistema e nos cenários visuais temos um filme animado da sua execução. Estes dois, em conjunto, permitem a validação de um sistema, mas também permitem ensinar o funcionamento do mesmo a um novo utilizador

4. Gerar automaticamente manuais de utilizador de sistemas em execução

A criação de manuais de utilizador não costuma ser uma tarefa fácil, nem muito rápida. Estes acabam muitas vezes por ser desenvolvidos apenas quando um sistema está completo e nem sempre demonstram a realidade do mesmo [35]. Como foi exposto ao longo desta dissertação, é possível, através dos cenários visuais, criar automaticamente manuais de utilizador. Isto pode ser aproveitado, por exemplo, para demonstrar funcionalidades de um sistema, durante o seu desenvolvimento. Demonstrar funcionalidades à medida que estas vão sendo criadas é uma boa forma de as validar e proceder à sua alteração, caso seja necessário.

5. Enriquecer especificações de requisitos

Muitas vezes os casos de utilização são a única especificação de requisitos criada de um sistema de informação, ficando as descrições de cenários deixadas para segundo plano. Nesta dissertação apresento uma forma de combinar os casos de utilização com as descrições de cenários, que é posteriormente enriquecida com cenários visuais e com o manual de utilizador. Desta maneira as especificações de requisitos são uma combinação de diagramas de casos de utilização, descrições estruturadas de cenários, cenários visuais e do manual de utilizador.

6. Melhorar o desenvolvimento através dos modelos

A utilização de modelos UML em todo o ciclo de vida do software não é prática habitual nas empresas. O mais vulgar é estes serem usados apenas nas fases de análise e desenho.

Com o auxílio dos cenários visuais é possível estender a utilização de modelos UML aos testes de software, e às fases de manutenção. Os cenários visuais permitem elevar o nível de abstracção da análise fina (*debugging*, *profiling*), habitualmente feita ao nível do código fonte, para o nível dos modelos, permitindo avaliações mais abrangentes do comportamento da aplicação.

As novas perspectivas criadas pela junção dos cenários visuais com as baterias de cenários de teste trazem os requisitos de um sistema para o mundo dos testes e elevam o patamar dos testes para o nível dos modelos.

7.1.2. Alternativas ao uso do *ReModeler Process*

Foi efectuada uma pesquisa ao mercado de ferramentas para se encontrar uma alternativa ao *ReModeler Process*. Não se encontrou nada que possibilitasse a captura simultânea dos dois cenários (externo e interno) de um cenário visual. Existem no entanto, alternativas para a criação de ambos em separado. Por exemplo, para fazer capturas de cenários externos existem programas como o *Camtasia Studio* [60], que permitem gravar filmes de execuções de programas. Para as capturas internas já existem algumas aplicações que permitem gravar diagramas de sequência de execuções de um programa, como por exemplo, o *MaintainJ* [61]. Nestas ferramentas, os diagramas gerados são estáticos, sem animação nem ligação a um filme das acções do utilizador.

Caso se quisesse criar um cenário visual com as ferramentas que existem no mercado, teríamos de usar duas ferramentas distintas e o que obteríamos seria apenas uma pequena aproximação. Teríamos um filme e um diagrama, mas sem nenhuma ligação entre eles e sem possibilidade de ter as duas visões sincronizadas e a executar simultaneamente.

A geração do manual de utilizador depende fortemente da criação dos cenários visuais. No mercado actual não foi encontrado nada que permita obter o que se está a propor. Existem propostas de como se criar um manual de utilizador, mas não estão disponíveis ferramentas para automatizar a sua geração.

7.2. Trabalho Futuro

O trabalho realizado no âmbito desta dissertação abre perspectivas para futuras evoluções, quer a nível do protótipo desenvolvido, quer no que diz respeito a inovações ao nível da abordagem da modelação “aumentada”.

7.2.1. Sistema de captura de cenários

A primeira evolução que está prevista para o protótipo vai no sentido de o tornar mais independente do sistema a documentar. Neste momento, quando queremos documentar um sistema, este tem de ser compilado juntamente com o *ReModeler*. Este último é actualmente também um sistema completo, constituído por vários módulos. De entre esses módulos é possível destacar uma interface gráfica, componentes para gerar documentação de sistemas e componentes para fazer a captura da execução do sistema a documentar. A junção de todos estes componentes com o sistema a documentar torna muitas vezes o sistema final relativamente pesado. O que está previsto fazer para mitigar este problema é separar o módulo das capturas do resto do *ReModeler*, tornando-o numa pequena aplicação independente. Esta sim, é compilada com o sistema a documentar e tem a tarefa de executar as capturas (externa e interna) tal como estão a ser feitas actualmente.

7.2.2. Editor de casos de utilização “aumentado”

Outra evolução prevista para o protótipo é relativa à edição de casos de utilização e de cenários. Neste momento, as opções que estão disponíveis no editor de casos de utilização são relativamente simples, mas será uma mais-valia tornar o processo de documentação mais intuitivo, para melhorar a compreensão do sistema. Para o conseguir seria interessante criar um editor gráfico hiperligado. Actualmente o ReModeler disponibiliza uma árvore de casos de utilização que passaria a estar associada ao editor gráfico, semelhante ao que encontramos em ferramentas UML. Este editor, além de possuir as propriedades semelhantes a um qualquer editor gráfico de casos de utilização, vai ter a possibilidade de se puder clicar nos casos de utilização e estes apresentarem os respectivos cenários, passos e cenários visuais. A ideia é dar ao utilizador uma visão “aumentada” dos modelos de casos de utilização.

7.2.3. Editor de cenários “aumentado”

A descrição estruturada de cenários é um ponto muito importante no RMP. No entanto, actualmente o editor que está disponível para o efeito é muito trivial, permitindo apenas descrever sequencialmente os vários passos que constituem cada descrição. Dado que esta situação não é aceitável nem realista, está prevista uma reestruturação desse mesmo editor, de modo a que passe a disponibilizar opções para criar relações de "*include*" e "*extend*", de forma automática com o auxílio de hiperligações. Estas hiperligações vão permitir que se navegue entre as várias descrições de cenários, o que permite uma melhor compreensão das mesmas. A edição dos passos da descrição também será enriquecida com opções de formatação e correcção ortográfica para facilitar a sua criação

7.2.4. Editor de cenários visuais

Os cenários visuais assumem um papel importante na documentação de sistemas, mas actualmente não é possível qualquer edição nos mesmos. Está prevista a criação de um editor onde seja possível criar hiperligações para os passos do cenário. O utilizador, usando o rato, poderá marcar zonas no cenário externo, associando essa zona a um passo de cenário. Posteriormente, estas hiperligações devem ficar activas durante a reprodução, demonstrando com mais eficiência, a que se referem os passos da descrição de cenário no contexto de um cenário externo.

O cenário externo também tem algumas evoluções esperadas. O filme do cenário externo é um conjunto de imagens retiradas da totalidade do ecrã, mas seria mais correcto que as imagens pertencessem unicamente às janelas do sistema a documentar.

7.2.5. Exportação para vários formatos do manual de utilizador

Existe algum trabalho que falta realizar relativamente ao manual de utilizador. O único formato de exportação actual é o HTML. Este ponto poderá vir ser uma limitação à sua utilização por parte de terceiros. Por isso, está considerado a exportação do mesmo para outros formatos, tais como pdf [52].

7.2.6. Validação Empresarial do RMP

Relativamente à experimentação do protótipo no mundo real, sabemos que esta está longe de estar concluída: os testes de validação estão ainda no início. Pretende-se fazer mais testes de forma a comprovar que, de facto, a modelação "aumentada" e os conceitos nela presentes, auxiliam o desenvolvimento de software. Os testes passarão por deslocações a empresas, onde possa ser feito o levantamento do processo de desenvolvimento da mesma, retirando vários indicadores dos recursos envolvidos, como o esforço gasto a realizar as tarefas do ciclo de vida. De seguida, deverá ser introduzido o ReModeler e os mesmos indicadores devem voltar a ser medidos. A validação é conseguida com a comparação dos resultados, que espero que venham reforçar as conclusões dos benefícios a que já se chegou.

7.2.7. Organização das actividades do trabalho futuro

Em termos de plano de trabalho futuro podemos identificar uma ordem de trabalhos: inicialmente deve ser feita a separação do componente da captura do resto do sistema *ReModeler* e deve ser melhorado o editor de casos de utilização; seguidamente deve ser criado o editor de cenários visuais hiperligado e devem ser acrescentados novos formatos de exportação do manual de utilizador; por último, devem ser realizados os testes de validação finais ao protótipo concluído.

Existe também um espaço para a introdução de novos conceitos no âmbito da modelação aumentada, já que apenas se começou agora a desenvolver os presentes nesta dissertação. Um pequeno passo acabou de ser dado com esta dissertação. Um novo caminho está aberto para a documentação automática de sistemas.

[Esta página foi intencionalmente deixada em branco]

Bibliografia

1. Pressman, R.S., *Software Engineering: A Practitioner's Approach*. 6th ed. 2005: McGraw-Hill Book Company.
2. Silva, A. and C. Videira, *UML, Metodologias e Ferramentas CASE*. 2ª ed. Vol. 1. 2005: Centro Atlântico.
3. OMG, *Unified Modeling Language Specification version 2.1.2*. 2007, OMG: http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML.
4. Jacobson, I., et al., *Object-Oriented Software Engineering- A Use Case Driven Approach*. 1992, Reading, MA, USA / Wokingham, England: Addison-Wesley / ACM Press.
5. Google. *Google search Engine*. 2008 [cited 2008 12-1]; Available from: http://www.google.com.
6. Nunes, M. and H. O'Neill, *Fundamental de UML*. 3ª ed. 2001: FCA.
7. Bisbal, J., et al., *Legacy information systems: issues and directions.* Software, IEEE, 1999. **16**(5): p. 103-111.
8. Cheng, B.H.C. and J.M. Atlee. *Research Directions in Requirements Engineering*. in *Future of Software Engineering, 2007. FOSE '07*. 2007.
9. Gotel, O.C.Z. and C.W. Finkelstein. *An analysis of the requirements traceability problem*. in *Requirements Engineering, 1994., Proceedings of the First International Conference on*. 1994.
10. *IEEE guide to software requirements specifications.* IEEE Std 830-1984, 1984.
11. George, S., *Plausible and adaptive requirement traceability structures*, in *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. 2002, ACM: Ischia, Italy.
12. Cleland-Huang, J., et al., *Best Practices for Automated Traceability*. Computer, 2007. **40**(6): p. 27-35.
13. Mark, G., S.M. Kathryn, and E.P. Dewayne, *Recovering and using use-case-diagram-to-source-code traceability links*, in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. 2007, ACM: Dubrovnik, Croatia.
14. Neumuller, C. and P. Grunbacher. *Automating Software Traceability in Very Small Companies: A Case Study and Lessons Learne*. in *Automated Software Engineering, 2006. ASE '06. 21st IEEE/ACM International Conference on*. 2006.
15. Egyed, A., *A scenario-driven approach to trace dependency analysis.* Software Engineering, IEEE Transactions on, 2003. **29**(2): p. 116-132.
16. Haumer, P., et al., *Improving reviews of conceptual models by extended traceability to captured system usage.* Interacting with Computers, 2000. **13**(1): p. 77-95.
17. Dorfman, M. and H. Thayer Richard, *Standards, Guidelines and Examples : System and Software Requirements Engineering*. 1990: IEEE Computer Society Press.

18. Wikipedia. *Traceability matrix*. 2008 [cited 2008 29-1]; Available from: http://en.wikipedia.org/wiki/Traceability_matrix.
19. Jane, C.-H., *Toward improved traceability of non-functional requirements*, in *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*. 2005, ACM: Long Beach, California.
20. Jackson, J. *A keyphrase based traceability scheme*. in *Tools and Techniques for Maintaining Traceability During Design, IEE Colloquium on*. 1991.
21. Antoniol, G., et al., *Recovering traceability links between code and documentation.* Software Engineering, IEEE Transactions on, 2002. **28**(10): p. 970-983.
22. Cleland-Huang, J., C.K. Chang, and M. Christensen, *Event-based traceability for managing evolutionary change.* Software Engineering, IEEE Transactions on, 2003. **29**(9): p. 796-810.
23. Leila, N., et al., *Using scenarios to support traceability*, in *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*. 2005, ACM: Long Beach, California.
24. Wikipedia. *Information retrieval*. 2008 [cited 2008 5-1]; Available from: http://en.wikipedia.org/wiki/Information_retrieval.
25. Wikipedia, *Stop words*. 2008: http://en.wikipedia.org/wiki/Stop_word, accessed on January 5.
26. *Thales Group*: http://www.thalesonline.com/, accessed on January 8.
27. Alspaugh, T.A., *Scenario ML*: http://www.ics.uci.edu/~alspaugh/ScenarioML.html.
28. Leite, J.C.S.d.P. and K.K. Breitman, *Experiences Using Scenarios to Enhance Traceability*, in *2nd International Workshop on Traceability in Emerging Forms of Software Engineering at the 18th IEEE Conference on Automated Sofware Engineering*.
29. Beck, K., *Extreme Programming Explained: Embrace Change*. 1ª ed. 1999: Addison-Wesley Publishing Company. 224.
30. Telelogic, *Requirements Management for Advanced Systems and Software Development* 2008: http://www.telelogic.com/products/doors/doors/index.cfm, accessed on January 15
31. Wikipedia, *Concurrent Versions System*. 2008: http://en.wikipedia.org/wiki/Concurrent_Versions_System, accessed on January 13.
32. Wikipedia, *Wiki*. 2008: http://en.wikipedia.org/wiki/Wiki, accessed on January 13.
33. Cleland-Huang, J., C.K. Chang, and G. Yujia. *Supporting event based traceability through high-level recognition of change events*. in *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*. 2002.
34. Abreu, F.B., V. Gouveia, and F. Silva, *Model-Driven Testing for Java Projects*, in *ExpoQA*. 2007: Madrid.
35. McIntosh Shand, R., *User manuals as project management tools. I. Theoretical background.* Professional Communication, IEEE Transactions on, 1994. **37**(2): p. 75-80.
36. McIntosh Shand, R., *User manuals as project management tools. II. Practical applications.* Professional Communication, IEEE Transactions on, 1994. **37**(3): p. 123-142.

37. Abran, A., et al., eds. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. 2004, IEEE Computer Society.
38. Wikipedia, *Test Case*. 2008: http://en.wikipedia.org/wiki/Test_case, accessed on January 13.
39. OMG, *MOF 2.0/XMI Mapping, Version 2.1.1*. 2007: http://www.omg.org/cgi-bin/doc?formal/2007-12-01.
40. Silva, F.P.D.d., *Recuperação Automática da Modelação Comportamental em Aplicações de Testes Baseado em Modelos*. 2008.
41. Software, R., *What is the Rational Unified Process.* 2001.
42. SUN, *Java Media Framework API Guide*. 1999.
43. Wikipedia, *Audio Interchange File Format*. 2008: http://en.wikipedia.org/wiki/Audio_Interchange_File_Format, accessed on July 30.
44. Wikipedia, *Au file format*. 2008: http://en.wikipedia.org/wiki/Au_file_format, accessed on July 30.
45. Wikipedia, *Audio Video Interleave*. 2008: http://en.wikipedia.org/wiki/AVI#External_links, accessed on July 30.
46. Wikipedia, *Musical Instrument Digital Interface*. 2008: http://en.wikipedia.org/wiki/MIDI, accessed on July 30.
47. Group, M.P.E. *The MPEG Home Page*. 2008 [cited 2008 July 30]; Available from: http://www.chiariglione.org/mpeg/.
48. Wikipedia. *QuickTime*. 2008 [cited 2008 July 30]; Available from: http://en.wikipedia.org/wiki/QuickTime#QuickTime_framework.
49. Wikipedia. *WAV*. 2008 [cited 2008 July 30]; Available from: http://en.wikipedia.org/wiki/WAV.
50. Sourceforge. *Sourceforge*. 2008 [cited 2008 February 28]; Available from: http://sourceforge.net/.
51. Strauch, M. *Quick Sequence Diagram Editor*. 2008 [cited 2008 February 28]; Available from: http://sdedit.sourceforge.net/.
52. Incorporated, A.S., *PDF Reference*. 2006.
53. Systems, S. *Sparx Systems*. 2008 [cited 2008; Available from: http://www.sparxsystems.com.au/.
54. W3C. *HTML 4.01 Specification*. 2008 [cited 2008 June 22]; Available from: http://www.w3.org/TR/REC-html40/.
55. SUN. *Swing (Java Foundation Classes)*. 2005 [cited 2008; Available from: http://java.sun.com/javase/6/docs/technotes/guides/swing/.
56. AB, N.T. *InfoNode*. 2008 [cited 2008; Available from: http://www.infonode.net/.
57. SUN. *Java Foundation Classes*. 2008 [cited 2008; Available from: http://java.sun.com/javase/technologies/desktop/.
58. Legeyda, A. *Sourceforge - Jeneo*. 2008 [cited 2008 June 20]; Available from: http://sourceforge.net/projects/jeneo/.
59. Wikipedia. *Scalable Vector Graphics*. 2008 [cited 2008 June 20]; Available from: http://en.wikipedia.org/wiki/Scalable_Vector_Graphics.
60. Corporation, T. *Camtasia Studio*. 2008 [cited 2008 June 20]; Available from: http://www.techsmith.com/camtasia.asp.
61. Inc., M. *MaintainJ*. 2008 [cited 2008 April 1]; Available from: http://www.maintainj.com/index.html.