UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA PARA AUXÍLIO NA ANÁLISE DE IMPACTO E RASTREABILIDADE DE REQUISITOS NA GESTÃO DE MUDANÇAS

JOSÉ ALBERTO ZIMERMANN

BLUMENAU
2011

2011/1-22

JOSÉ ALBERTO ZIMERMANN

FERRAMENTA PARA AUXÍLIO NA ANÁLISE DE IMPACTO E RASTREABILIDADE DE REQUISITOS NA GESTÃO DE MUDANÇAS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Marcel Hugo, Mestre - Orientador

BLUMENAU
2011

2011/1-22

FERRAMENTA PARA AUXÍLIO NA ANÁLISE DE IMPACTO E RASTREABILIDADE DE REQUISITOS NA GESTÃO DE MUDANÇAS

Por

JOSÉ ALBERTO ZIMERMANN

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Marcel Hugo, Mestre – Orientador, FURB
Membro:	Prof. Everaldo Artur Grahl, Mestre – FURB
Membro:	Profa. Joyce Martins, Mestre – FURB

Blumenau, 01 de julho de 2011

Dedico este trabalho à minha mãe, Maria Odete Pacheco, que me apoiou, incentivou, acreditou e possibilitou que eu pudesse chegar até aqui. A ela, só posso agradecer.

AGRADECIMENTOS

Ao ser superior que nos guia em nossas jornadas.

À minha mãe, Maria Odete Pacheco, que sempre esteve ao meu lado nesta difícil caminhada.

Aos meus irmãos, que, cada qual da sua maneira, me orientou, aconselhou, incentivou e fez das minhas alegrias e conquistas, as suas.

Ao meu pai, Cesar Zimermann, que mesmo não estando todos os dias ao meu lado, acreditou num futuro melhor para mim.

A todos os meus familiares que estiveram ao meu lado.

Aos meus amigos, que fizeram da sua alegria, companhia, inteligência e incentivos, fatores preponderantes para que eu tivesse vontade de chegar até o final.

Ao professor Marcel Hugo, que foi um exímio orientador, dedicado, eficiente e também por ter acreditado na ideia deste trabalho.

À professora Joyce pela dedicação e ajuda sobre árvores sintáticas abstratas.

Ao professor Roque que sempre esteve disponível durante este período, ajudando no que fosse necessário.

Aos demais professores que se disponibilizaram a contribuir para o meu crescimento profissional e pessoal.

O segredo da existência não consiste somente em viver, mas em saber para que se vive.

Fiódor Dostoievski

RESUMO

Este trabalho apresenta o EA Requisite Manager, um *plugin* para o ambiente de desenvolvimento Eclipse. O objetivo é, através do mapeamento de códigos fontes de projetos Java, possibilitar ao desenvolvedor de sistemas rastrear, analisar o impacto causado com a alteração de requisitos e quantificar o custo das alterações, através da metodologia AHP. O mapeamento é fornecido através da importação de um arquivo XML de um projeto modelado e especificado através da ferramenta CASE EA. A comunicação entre o *plugin* e o EA é feita através de bibliotecas disponibilizadas pelo fabricante, onde é possível acessar diversas funcionalidades da ferramenta. A inserção e busca de anotações nos códigos fontes é feita com o auxílio de uma árvore sintática abstrata, que permite a avaliação dos componentes de um arquivo fonte da linguagem Java. A análise de importância e quantificação dos requisitos é feita através da aplicação da metodologia AHP, usando como critérios a quantidade de classes, métodos e casos de uso que são vinculados aos requisitos de um projeto.

Palavras-chave: Gestão de mudanças. Rastreabilidade. Análise de impacto. Requisitos. AHP.

ABSTRACT

This work presents EA Requisite Manager, a plugin for the development ambient Eclipse. The objective is, through the Java code source mapping, enable the system developer trace, analyze the impact caused by the requirement modification and quantify the changes costs, through the AHP methodology. The mapping is provided through the import of an XML archive of a project designed e projected using the CASE tool EA. The communication between the plugin and EA is done through frameworks provided by the producer, where is possible to access various features of the tool. The insertions and search of annotations on source codes are done with the aid of an abstract syntax tree, which allows evaluation of the components of a Java source file. The importance of analysis and requirement quantification is done by applying the AHP methodology, using as criteria the number of classes, methods and use cases that are linked to the project's requirements.

Key-words: Change management. Traceability. Impact analysis. Requirements. AHP.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

AHP – *Analytical Hierarchy Process*

AOP – *Aspect-Oriented Programming*

API – *Application Programming Interface*

AST – Árvore Sintática absTrata

AWT - *Abstract Window Toolkit*

CASE – *Computer-Aided Software Engineering*

CSV – *Comma-Separated Values*

DTD – *Document Type Definition*

EA – Enterprise Architect

EAP - Enterprise Architect *Project*

GUI – *Graphical User Interface*

HTML – *HyperText Markup Language*

IDE – *Integrated Development Environment*

IBM – *International Business Machines*

FPA – *Function Point Analysis*

JDT – Java *Development Tool*

JRXML – Jasper Reports *eXtensible Markup Language*

MS SQL – *Microsoft Structured Query Language*

PDF – *Portable Document Format*

SGBD – Sistema de Gerenciamento de Banco de Dados

SO – Sistema Operacional

SWT - *Standard Widget Toolkit*

UCP – *Use Case Points*

UML - *Unified Modeling Language*

XMI - XML *Metadata Interchange*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A crescente busca pela qualidade de um produto exige cada vez mais que as empresas fabricantes de software busquem um processo ou um modelo no qual se siga uma metodologia de trabalho de conhecimento de todos os envolvidos. Segundo Oliveira (2006, p. 14), diversas empresas optam por processos que contribuam no gerenciamento de mudanças. O objetivo do controle de mudanças, desta forma, é assegurar que as alterações feitas em um projeto sejam consistentes e passíveis de mensurar o impacto que irão gerar. No mercado, existem diversas ferramentas que contemplam as fases de elaboração e desenvolvimento de um sistema, auxiliando inclusive a implantação e uso de processos de software bem definidos.

Comumente percebe-se que dentro da área de informática as empresas encontram dificuldades em gerir o processo de mudança, em virtude de fatores como a velocidade e dinâmica com que elas ocorrem. As mudanças no software são feitas em resposta a solicitações de modificação de requisitos de um projeto e isso deve ser feito de maneira que a estrutura fundamental já existente permaneça estável.

Sommerville (2003, p. 518) afirma que 65% da manutenção de um sistema está relacionada à implementação de novos requisitos, 18% na modificação de requisitos já existentes e 17% na correção de defeitos de um sistema. Desta maneira, a manutenção pode ser considerada como uma extensão do processo de desenvolvimento de software, com atividades associadas às de especificação, projeto, implementação e testes.

Neste sentido, dentro do processo de manutenção, Pressman (1995, p. 883) afirma que um dos problemas clássicos associados à manutenção é a dificuldade em se rastrear a evolução do software, uma vez que as mudanças não estão devidamente documentadas. Já Borges (2003, p. 59) defende que para se obter o controle e estabilidade de requisitos, durante o projeto de desenvolvimento, é necessário acompanhar quantitativamente as alterações em requisitos. Isto permite mensurar o tamanho das alterações, em termos de esforço empregado na manutenção dos artefatos.

Diante do exposto, neste trabalho foi desenvolvida uma ferramenta em forma de *plugin* para ser integrado ao ambiente de desenvolvimento Eclipse. Esta ferramenta, através de uma fonte de dados no formato *eXtensible Markup Language* (XML), gerado com o apoio de bibliotecas da ferramenta *Computer-Aided Software Engineering* (CASE) Enterprise Architect (EA), permite ao usuário efetuar o rastreamento dos requisitos de software implementados nos códigos fontes Java.

Este mapeamento[1] é realizado com a inserção de anotações, após a leitura e interpretação do arquivo XML com os dados do projeto, nos códigos fontes do sistema. Estas anotações, por sua vez, identificam quais são os requisitos que atendem determinada classe ou método. O usuário também pode incluir, alterar e remover anotações, uma vez que é disponibilizada o conjunto de anotações anteriormente descritas.

Com o desenvolvimento da ferramenta também é possível efetuar a análise de impacto que uma mudança pode gerar no projeto. É possível ponderar qual é o custo que uma determinada alteração gera para o sistema, apresentando desta forma, quais requisitos que serão alterados e qual a representação desses em relação ao projeto.

1.1 OBJETIVOS DO TRABALHO

Este trabalho teve como objetivo desenvolver uma ferramenta em forma de *plugin* para o ambiente de desenvolvimento Eclipse, que permitisse ao usuário efetuar a rastreabilidade entre os artefatos produzidos na fase de elaboração do projeto de software e os códigos fontes da aplicação, bem como possibilitasse a análise de impacto dos requisitos que serão alterados.

Os objetivos específicos podem ser detalhados da seguinte maneira:

a) disponibilizar uma funcionalidade que efetue o mapeamento dos requisitos de softwares definidos no projeto nos códigos fontes do sistema;
b) definir um conjunto de anotações que possibilitarão ao usuário marcar nos códigos fontes quais trechos do código atendem um requisito;
c) ponderar os custos que as mudanças em requisitos podem representar no projeto, através do processo *Analytical Hierarchy Process* (AHP)[2];
d) disponibilizar uma interface de consulta que permita ao usuário efetuar a análise de impacto e custo, relativo às alterações no projeto.

[1]O relacionamento entre requisitos (artefatos gerados na fase de projeto) e códigos fontes (artefatos gerados na fase de implementação) foi feito através de *links* entre os artefatos na fase de modelagem do projeto. A ferramenta EA possui funcionalidades que permitem esta ligação e cabe ao usuário efetuar o relacionamento correto entre os artefatos.

[2]Para se quantificar os valores das alterações dos requisitos é necessária a utilização de uma métrica em conjunto com o AHP, como o *Function Point Analysis* (FPA) ou *Use Case Points* (UCP). A quantificação de valores e adoção de métricas não são abrangidas pelo escopo deste trabalho.

1.2 ESTRUTURA DO TRABALHO

A parte inicial deste capítulo apresenta a introdução sobre o trabalho, sendo citados os objetivos e a estrutura do trabalho, sendo esta última, apresentada nesta seção.

O segundo capítulo apresenta a fundamentação teórica necessária para o desenvolvimento deste trabalho, abordando-se os assuntos sobre processo de desenvolvimento de software, gerência de requisitos, gestão de mudanças e a sua aplicação no gerenciamento de projetos, conceito e funcionalidades da rastreabilidade de requisitos e artefatos e metodologia AHP. Também são apresentadas as principais ferramentas utilizadas para o desenvolvimento deste trabalho. Por último, são comentados os trabalhos correlatos.

No terceiro capítulo são descritas as etapas de desenvolvimento deste trabalho, apresentando-se os requisitos funcionais e não funcionais, casos de uso, os diagramas de sequência das principais atividades, matriz de relacionamento entre requisitos e casos de uso da ferramenta e diagrama de classes da solução desenvolvida. Já na seção sobre a implementação são apresentadas as principais técnicas utilizadas, os resultados e discussões sobre a ferramenta implementada.

O quarto capítulo apresenta a conclusão sobre o trabalho e as extensões que podem ser sugeridas para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A seção 2.1 descreve as fases de um processo de desenvolvimento de software e trata sobre a gerência de requisitos. Na seção 2.2 são explicados os benefícios da gestão de mudanças em um projeto, mostrando os estágios em que ela deve ser executada. Na seção 2.3 é explicado o conceito e as funcionalidades da rastreabilidade de requisitos e artefatos. A seção 2.4 apresenta a metodologia AHP, mostrando-se o seu funcionamento e como pode ser aplicada. A partir da seção 2.5 são apresentadas as ferramentas envolvidas no desenvolvimento do trabalho.

2.1 PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

Sommerville (2003, p. 7) define um processo de software como sendo uma série de atividades e resultados que tem como finalidade um produto de software. Os processos de softwares são complexos e devem seguir uma série de procedimentos, a fim de garantir um produto que atenda as diversas necessidades do cliente. Neste sentido, existem diversos processos, que têm como características comuns as seguintes fases:

a) especificação: é onde ocorre a definição das funcionalidades do software, indicando inclusive, as suas restrições;
b) projeto e implementação de software: nesta etapa é onde ocorre o desenvolvimento do software, atendendo a especificação anteriormente descrita;
c) validação do software: o software necessita ser validado, garantindo que ele atenda as necessidades do cliente;
d) evolução do software: o software pode evoluir para atender as necessidades mutáveis do cliente.

Ao se colocar um software em uso, novos requisitos são detectados como necessários e os já existentes são alterados à medida que o sistema é utilizado. Partes dos requisitos podem ser modificadas para corrigir erros encontrados na operação ou até mesmo para melhorar o desempenho da aplicação.

É importante que o projeto de software seja atualizado com a evolução do sistema a partir da inclusão, alteração ou exclusão de requisitos, a fim de se identificar de maneira clara

quais são os componentes envolvidos nas alterações de um sistema.

2.1.1 Gerência de requisitos

Sommerville (2003, p. 117) afirma que os requisitos nos grandes sistemas estão em constante modificação.

A gerência de requisitos está associada ao processo de controle do processo de desenvolvimento tendo como referência a base de requisitos. Segundo Tocchetto (2007, p. 20), o principal objetivo do gerenciamento de requisitos é “descobrir, organizar, comunicar e administrar o impacto das mudanças de requisitos de um software”. Desta maneira, Tocchetto (2007, p. 20) afirma que os principais benefícios do gerenciamento de requisitos são:

a) maior controle em projetos de grande porte;
b) aumento da qualidade de software e maior satisfação do cliente;
c) diminuição dos custos de projeto e atrasos.

2.2 GESTÃO DE MUDANÇAS

O processo de manutenção é explicado por Sommerville (2003, p. 521) ao se iniciar um conjunto de pedidos de mudança por parte dos usuários, gerentes ou clientes. Custo e impacto devem ser analisados para se verificar quanto do sistema será afetado pela alteração e quanto poderá custar para desenvolver esta mudança.

Assim sendo, em uma situação ideal, o estágio de desenvolvimento de alterações deverá modificar especificação, projeto e implementação do sistema, com o objetivo de refletir no software. Desta maneira, os novos requisitos que refletem as mudanças no sistema são propostos, analisados e validados, para que a mudança seja consistente e não impacte de maneira negativa no restante do sistema.

Neste contexto, a gerência de mudança tem o objetivo de garantir que as mudanças sejam realizadas com sucesso, sem que haja perda da qualidade do software. Para que isso ocorra, é necessário que esta fase da manutenção do software controle as solicitações de manutenção, aprove as solicitações e a partir daí, estabeleça como a modificação será implementada e quais restrições que deverão ser respeitadas, definindo de maneira clara qual

o escopo da alteração. Segundo Sommerville (2003, p. 121), existem três estágios em um processo de gerenciamento de mudanças:

a) análise do problema e especificação da mudança;
b) análise e custo da mudança;
c) implementação de mudanças.

Nestes estágios são utilizadas informações de rastreabilidade para se estimar o tamanho e o custo da mudança. O custo da mudança é estimado em termos das modificações no documento de requisitos e, se apropriado, no projeto de sistemas e na implementação. Desta maneira, entende-se que é muito importante possuir o documento de requisitos e o projeto de software constantemente atualizados.

2.3 RASTREABILIDADE DE REQUISITOS

A rastreabilidade é uma técnica que provê o relacionamento entre diversos requisitos, projeto e a sua implementação. É ela quem auxilia no entendimento dos relacionamentos que existem entre os requisitos e o projeto de software desenvolvido. Para Hamilton e Beeby (1991, p. 1), a rastreabilidade é a habilidade de descobrir o histórico de cada funcionalidade do sistema.

Segundo Hazan e Leite (2003), a rastreabilidade permite garantir como e porque os artefatos atendem os requisitos dos clientes, sendo uma forma fundamental para entender os relacionamentos existentes entre requisitos e outros artefatos que fazem parte do processo de software. Desta maneira, a elaboração de um projeto de software deve produzir requisitos que sejam rastreáveis, ou seja, que sejam capazes de serem rastreados a partir da sua origem.

Já Santos e Aragão (2009, p. 14) afirmam que um dos maiores desafios da engenharia de requisitos é criar um mecanismo que possibilite a criação de *links* entre os requisitos e os códigos fontes. Desta maneira, um dos benefícios que a rastreabilidade oferece é a possibilidade de se cruzar as informações especificadas na fase de projeto (requisitos) e os itens desenvolvidos na fase de implementação, que são os códigos fontes.

A rastreabilidade permite que as estimativas de custos das alterações em requisitos de um projeto possam ser mais precisas. Também permite que as mudanças possam ocorrer sem a dependência do engenheiro ou programador de conhecerem as áreas afetadas por tais mudanças.

É possível afirmar que a manutenção da rastreabilidade pode ser um trabalho penoso, extenso e até mesmo inviável, quando não auxiliado por uma ferramenta especializada. Isso é explicado por Richardson e Green (2004, p. 24) que afirmam que programadores são relutantes em manter os documentos do projeto atualizados, quebrando assim, o mecanismo que possibilita ocorrer a rastreabilidade.

2.3.1 Classificação da rastreabilidade

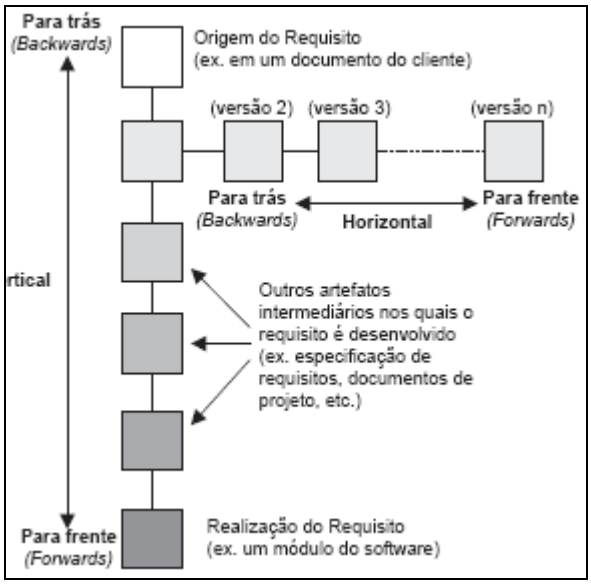
Genvigir (2009, p. 39) afirma que existem duas maneiras de se efetuar a rastreabilidade:

a) *forward*: rastreabilidade efetuada em um requisito até seus refinamentos;
b) *backward*: rastreabilidade efetuada de um refinamento até sua origem.

Ainda de acordo com Genvigir (2009, p. 39), um processo de rastreabilidade é falho caso não seja possível realizar um destes dois tipos de rastreabilidade. Isto está ligado ao fato de que estas capacidades são propriedades básicas para a realização da atividade de rastreabilidade. É possível classificar ainda a rastreabilidade quanto aos seus tipos:

a) horizontal: rastreabilidade efetuada entre versões ou variações de uma base de artefatos. Ocorre em uma determinada fase do ciclo de vida do projeto;
b) vertical: rastreabilidade que ocorre entre requisitos e artefatos produzidos pelo processo de desenvolvimento do projeto.

Uma visão sobre os tipos de rastreabilidade é apresentada na figura 1.



Fonte: Genvigir (2009, p. 40).

Figura 1 – Rastreabilidade horizontal e vertical

2.4 METODOLOGIA AHP

A metodologia AHP foi criada por Saaty (1991) com o objetivo de ponderar as características qualitativas, permitindo assim, a ponderação e priorização de cada um dos requisitos de um projeto que estão inseridos neste contexto. O funcionamento deste processo se dá pela atribuição de pesos a fatores individuais, do menos influente ao mais influente. AHP utiliza-se de uma matriz quadrada que permite avaliar a importância de uma característica sobre a outra. A partir disso, o processo permite obter o valor percentual de cada um dos requisitos e assim, mapear o valor das fases do projeto.

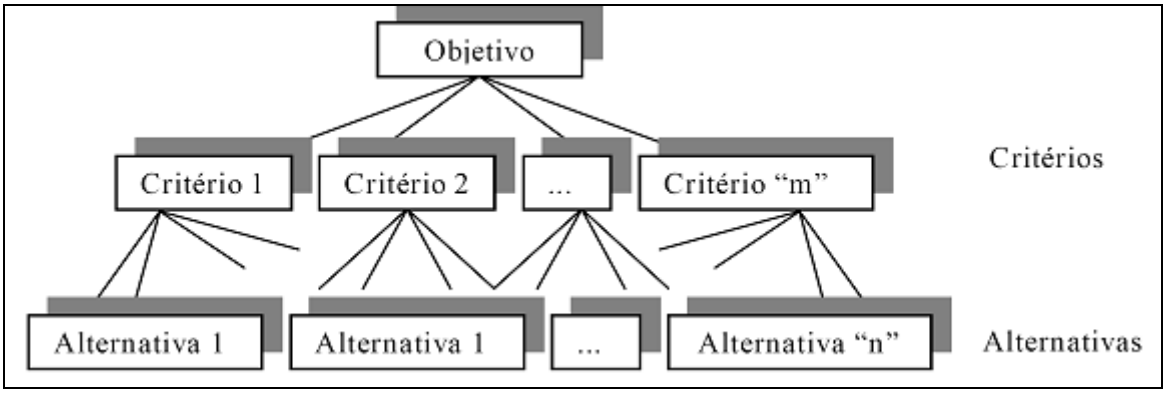
Segundo Tocchetto (2007, p. 21), o processo AHP permite obter o valor em percentual de cada requisito que faz parte do projeto, sendo possível assim, mapear os valores das fases do projeto, tendo-se desta forma, o real custo de cada um deles, a partir da aplicação de uma métrica.

Este método baseia-se no método newtoniano e cartesiano de pensar, que busca tratar a complexidade com a decomposição e divisão do problema em fatores, que podem ainda ser decompostos em novos fatores até o nível mais baixo, claros e dimensionáveis e estabelecendo relações para sintetizar (MARINS; SOUZA; BARROS, 2009, p. 1779).

2.4.1 Etapas para a construção de um pensamento analítico

Segundo Costa (2002, p. 16) é possível definir duas etapas de pensamento analítico, os quais são:

a) construção de hierarquias: no método AHP o problema é estruturado em níveis, facilitando desta forma, a melhor compreensão e avaliação deste. A figura 2 apresenta esta estrutura do pensamento analítico. Para aplicar o AHP é necessário que tanto os critérios quanto as alternativas possam ser estruturadas de forma hierárquica, sendo que no primeiro nível a hierarquia corresponde ao propósito geral do problema, o segundo e o terceiro às alternativas do problema;



Fonte: Marins, Souza e Barros (2009, p. 1780).

Figura 2 – Representação de uma estrutura hierárquica

b) definição de prioridades: fundamenta-se na habilidade do ser humano de perceber o relacionamento entre objetos e situações observadas, comparando pares, sob um mesmo foco, critério ou mesmo julgamento. Para se cumprir este princípio, é necessário que o julgamento ocorra entre um par de elementos, aplicando a escala definida no quadro 1. Esta comparação se dá através da montagem de uma matriz entre os elementos, onde `i` representa o elemento de uma linha e `j` um elemento da coluna.

Escala	Definição	Descrição
1	Menos importante	Duas atividades contribuem igualmente para o objetivo
3	Importância pequena	A experiência e o julgamento favorecem levemente uma atividade em relação à outra
5	Importância grande ou essencial	A experiência e o julgamento favorecem fortemente uma atividade em relação à outra
7	Importância muito grande	Uma atividade é fortemente favorecida. Sua dominação de importância é demonstrada na prática
9	Importância absoluta	A evidência favorece uma atividade em relação à outra com o mais alto grau de certeza
2, 4, 6, 8	Valores intermediários. Se na atividade "j" (coluna) recebe um dos valores acima, quando comparada com a atividade "j" (coluna), então "j" (coluna) tem o mesmo valor recíproco de "i" (linha)	Quando se deseja maior compromisso. É uma designação razoável
Racionais	Razão da escala	Se a consistência tiver de ser forçada para obter "n" valores numéricos para completar a matriz.

Fonte: Tocchetto (2007, p. 22).

Quadro 1 – Escala numérica da metodologia AHP

Segundo Marins, Souza e Barros (2009, p. 1780), a quantidade de julgamentos para a construção de uma matriz de julgamentos genérica `A` é `n (n-1)/2`, onde `n` é o número de elementos pertencentes a esta matriz. Os elementos de `A` são definidos pelas condições

apresentadas no quadro 2.

$$A = \begin{bmatrix} 1 & a_{12} & \cdots & a_{1n} \\ 1/a_{21} & 1 & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ 1/a_{n1} & 1/a_{n2} & \cdots & 1 \end{bmatrix}, \text{ onde:}$$

$$a_{ij} > 0 \Rightarrow positiva$$
$$a_{ij} = 1 \therefore a_{ji} = 1$$
$$a_{ij} = 1/a_{ji} \Rightarrow recíproca$$
$$a_{ik} = a_{ij} \cdot a_{jk} \Rightarrow consistência$$

Fonte: Marins, Souza e Barros (2009, p. 1780).

Quadro 2 – Condições para a definição de uma matriz de julgamento

2.4.2 O processo de aplicação da metodologia AHP

Para se entender o funcionamento do processo de cálculo do AHP, pode-se considerar o exemplo descrito a seguir. Para tal, será considerado um projeto que contém três requisitos, os quais são:

a) requisito 1;

b) requisito 2;

c) requisito 3.

O relacionamento realizado entre os requisitos do projeto é apresentado através do quadro 3.

Requisito / Requisito	Requisito 1	Requisito 2	Requisito 3
Requisito 1	1	a12	a13
Requisito 2	a21 = 1/a12	1	a23 = 1/a32
Requisito 3	a31 = 1/a13	a32	1

Quadro 3 – Matriz de importância entre os requisitos

Conforme observado por Tocchetto (2007, p. 23), as premissas que devem ser seguidas para a inserção de valores são:

a) Se $a_{ij} = \alpha$, então $a_{ji} = 1/\alpha$, α é diferente de 0;

b) Se o requisito i é julgado com igual importância ao requisito j, então $a_{ij} = 1$, $a_{ji} = 1$ e $a_{ii} = 1$.

A partir da definição desta matriz, deve-se incluir os valores para cada par de requisitos, de acordo com a escala proposta pelo autor do processo e apresentada no quadro 1.

Deve-se observar que a inserção de valores para a definição da matriz de importância é

feita a partir de definição de um critério ou julgamento. Assim, os valores são inseridos na matriz de acordo com a escala de importância estabelecida pelo julgamento. No quadro 4 é possível verificar a formação da matriz.

Requisito / Requisito	Requisito 1	Requisito 2	Requisito 3
Requisito 1	1	1/3	2
Requisito 2	3	1	5
Requisito 3	½	1/5	1

Quadro 4 – Matriz de importância

Após a definição, é necessário o cálculo do auto-vetor e a sua normalização, para cada um dos requisitos envolvidos no processo. O cálculo do auto-vetor se dá através da expressão definida no quadro 5.

$$W_i = (\prod_{j=1}^{n} a_{ij})^{1/n}$$

Fonte: Tocchetto (2007, p. 24).

Quadro 5 – Expressão para a obtenção do auto-vetor

Já a normalização do auto-vetor se dá com base na expressão apresentada no quadro 6.

$$T| = | W_1 / \sum W_i \quad W_2 / \sum W_i \quad \ldots \quad W_n / \sum W_n |$$

Fonte: Tocchetto (2007, p. 24).

Quadro 6 – Expressão para a normalização do auto-vetor

A partir da definição e conhecimento das fórmulas apresentadas nos quadros 5 e 6, pode-se exemplificá-las, aplicando as fórmulas para o Requisito 1, que é:

a) somatório da coluna do Requisito 1: T = 1 + 3 + 0,5 = 4,5;
b) primeira linha da primeira coluna normalizada: (1 * 100) / 4,5 = 22,22 / 100 = 0,22;
c) segunda linha da primeira coluna normalizada: (3 * 100) / 4,5 = 66,66 / 100 = 0,66;
d) terceira linha da primeira coluna normalizada: (0,5 * 100) / 4,5 = 11,11 / 100 = 0,11.

Este processo de normalização deve ser feito com todos os requisitos envolvidos no processo. Segundo Tocchetto (2007, p. 24), a utilização do vetor T normalizado serve para quantificar e ponderar a importância de várias características de um requisito. Pode acompanhar o resultado da normalização do vetor no quadro 7.

Requisito / Requisito	Requisito 1	Requisito 2	Requisito 3	Resultado
Requisito 1	0,22	0,21	0,25	0,68

Requisito 2	0,67	0,65	0,63	1,95
Requisito 3	0,11	0,13	0,12	0,36

Quadro 7 – Matriz com os valores normalizados

Desta maneira, o percentual de importância de cada requisito é apresentado através do quadro 8. Assim, tem-se que:

a) o requisito 1 equivale aproximadamente a 22,44% do projeto;
b) o requisito 2 equivale aproximadamente a 64,35% do projeto;
c) o requisito 3 equivale aproximadamente a 11,88% do projeto.

Fator 1/n	Requisito normalizado	Importância (%)
1/3	0,68	22,44
1/3	1,95	64,35
1/3	0,36	11,88

Quadro 8 – Importância de cada requisito

2.5 INTERPRETAÇÃO DE DADOS XML COM O JDOM

JDOM é uma simples representação em Java de um documento XML, segundo Hunter e McLaughlin (2007). JDOM oferece uma forma de representar este documento para uma leitura, manipulação e escrita de maneira fácil e eficiente.

2.5.1 Leitura de documento XML com JDOM

JDOM pode ler XML existentes em documento de arquivos, *sockets*, cadeia de caracteres ou alguma outra fonte de leitura, segundo Harold (2001). Para se efetuar a leitura de um arquivo XML, utilizando-se a biblioteca JDOM, os seguintes passos devem ser seguidos:

a) construir um objeto `org.jdom.input.SAXBuilder`;
b) invocar o método `build()` para se construir um objeto do tipo `Document`, a partir de um `Reader`, `InputStream`, `URL`, `File` ou `String`;
c) caso exista algum problema com a leitura do arquivo XML, uma exceção do tipo `IOException` será lançada. Da mesma maneira, se algum problema ocorrer

durante a construção do documento, uma exceção do tipo `JDOMException` será disparada;

d) caso contrário, será possível navegar pelo objeto `Document` utilizando-se os métodos da sua classe e da classe `Element`, além das demais classes fornecidas pela biblioteca JDOM.

A classe `SAXBuilder` representa o *parser* de um XML subjacente. Para se analisar um documento de um objeto do tipo `URL` basta criar-se um objeto `SAXBuilder` com o construtor sem parâmetros. Após, deve-se passar a `String` de um objeto `URL` para o método `build()`. Este método retornará um objeto do tipo `Document`. O quadro 9 mostra um exemplo de como efetuar a leitura de arquivo XML, utilizando a biblioteca JDOM.

```
public void buildDocument() throws JDOMException, IOException {

    // Criamos uma classe SAXBuilder que vai processar o XML
    SAXBuilder sb = new SAXBuilder();

    // Este documento agora possui toda a estrutura do arquivo.
    Document d = sb.build(xmlFile);

    // Recuperamos o elemento root
    root = d.getRootElement();
}
```

Quadro 9 – Exemplo de construção de um objeto `Document`

2.5.2 Navegação sobre os elementos de um XML utilizando JDOM

Após efetuar o *parser* do XML e construir um objeto do tipo `Document`, a navegação nos elementos que fazem parte da estrutura XML pode ser feitas através da classe `Element`. É possível obter-se todos os filhos de um determinado nó, utilizando-se o método `getChildren()`, que retorna um objeto do tipo `java.util.List`. O quadro 10 mostra um exemplo de como se pode obter os dados de um XML, a partir da navegação entre os nós da árvore construída.

```
public void buildDocument() throws JDOMException, IOException {

      // Criamos uma classe SAXBuilder que vai processar o XML
      SAXBuilder sb = new SAXBuilder();

      // Este documento agora possui toda a estrutura do arquivo.
```

```
        Document d = sb.build(xmlFile);

        // Recuperamos o elemento root
        root = d.getRootElement();
}
```

Quadro 10 – Navegação nos elementos de um XML

2.6 AMBIENTE DE DESENVOLVIMENTO ECLIPSE

Eclipse é um projeto de código fonte aberto cujo objetivo é criar um ambiente de desenvolvimento. Este ambiente é composto por várias partes extensíveis e também por ferramentas para possibilitar a construção e implantação de um ciclo de desenvolvimento de programas. Originalmente, foi criado pela *International Business Machines* (IBM). Em 2004 se tornou uma fundação sem fins lucrativos com a intenção de apoiar o desenvolvimento de uma comunidade em torno do projeto Eclipse (ECLIPSE FOUNDATION, 2011).

Segundo Burnette (2006, p. 12), a *Integrated Development Environment* (IDE) Eclipse funciona nos principais sistemas operacionais do mercado, dependendo da máquina virtual Java para funcionar. Uma das características da plataforma é que não existem programas de instalação, configuração ou mesmo valores de registro a serem alterados para o seu funcionamento.

2.6.1 Estendendo o Eclipse

A plataforma Eclipse foi concebida e projetada para ser extensível. Assim, é possível adicionar funcionalidades ao ambiente de desenvolvimento, através de pontos de extensão. Um ponto de extensão define um ponto onde pode ser adicionada uma determinada funcionalidade. Já a extensão é a implementação da funcionalidade para um ponto de extensão específico.

Para Henkels (2007, p. 33), um *plugin* é um conjunto de funcionalidades desenvolvidas para serem integradas ao Eclipse. Este conjunto de funcionalidades deve ser acompanhado de um arquivo chamado `plugin.xml`, que orienta a IDE em como deverá ser feita a integração das funcionalidades. No quadro 11 pode-se observar a estrutura deste arquivo.

```
<plugin>
   <extension
         point="org.eclipse.ui.actionSets">                    -> ponto de extensão utilizado
      <actionSet
            label="Ação"                                       -> definição e identificação do ponto de
            visible="true"                                        extensão
            id="plugin-demo.actionSet">
         <menu                                                 -> criação de um menu para extensão
               label="Menu Simples"
               id="sampleMenu">
            <separator
                  name="sampleGroup">
            </separator>
         </menu>                                               -> implementação da ação do ponto de extensão
         <action
               label="Menu Simples"
               icon="icons/sample.gif"
               class="plugindemo.actions.SampleAction"         -> classe responsável por implementar a ação
               tooltip="Hello, Eclipse world"
               menubarPath="sampleMenu/sampleGroup"
               toolbarPath="sampleGroup"
               id="plugindemo.actions.SampleAction">
         </action>
      </actionSet>
   </extension>
</plugin>
```

Quadro 11 – Estruturação do arquivo `plugin.xml`

O Eclipse oferece diversos tipos de extensão, podendo-se agregar funcionalidades e pontos de extensão em praticamente todas as partes do ambiente. Segundo Filho (2008), os principais pontos de extensão da plataforma são:

a) assistente de criação de projeto;
b) assistente de criação de arquivo;
c) natureza do projeto;
d) perspectiva;
e) editor de arquivo;
f) *outline* do arquivo;
g) configuração de execução;
h) execução de arquivo;
i) marcadores de problema;
j) barra de tarefas;
k) *working sets*;
l) ajuda.

Além dos *plugins*, o Eclipse possui outros dois conceitos de funcionalidades: os *fragments* e *features*. *Fragments* provêm conteúdo adicional a um *plugin* existente e são utilizados para prover pacotes de linguagem e implementações específicas de plataforma. Já os *features* são entendidos como conjuntos de *plugins*.

2.6.2 Desenvolvimento de *plugins* com o *Standard Widget Toolkit* (SWT)

SWT é um *widget toolkit* para Java, de código fonte aberto e projetado para fornecer acesso eficiente e portátil para o uso e manipulação da interface gráfica do Sistema Operacional (SO) na qual é aplicada (SWT, 2011). Pode ser entendido como uma camada adicional sobre os componentes padrões do SO. A figura 3 mostra o comportamento da interface nos diversos sistemas operacionais.

Fonte: Ferreira (2009, p. 5).

Figura 3 – Componente SWT em diversas plataformas

Os componentes que fazem parte do SWT foram desenvolvidos observando as melhores características de outras bibliotecas de interface gráfica do Java, como o Swing e o *Abstract Window Toolkit* (AWT). Inicialmente concebido para o desenvolvimento de aplicações para o Eclipse, hoje o SWT pode ser utilizado sem a necessidade de rodar para uma aplicação da IDE.

Segundo Rocha (2007, p. 15), a tarefa de desenhar os componentes da *Graphical User Interface* (GUI) é uma tarefa custosa. Desta maneira, o SWT se beneficia da sua comunicação direta com o SO e repassa para este a solicitação de desenhar os componentes da interface.

Uma característica notável da biblioteca é que os seus componentes são liberados de maneira automática, não sendo necessária a chamada de um método que execute esta tarefa. Entretanto, no caso do desenvolvimento de aplicações que consomem grande quantidade de memória, é possível a liberação dos recursos alocados.

Existem diversas maneiras de se construir aplicações utilizando o SWT, de acordo Feigenbaum (2006). Os leiautes disponíveis são: `FillLayout`, `FormLayout`, `GridLayout`, `RowLayout` e `StackLayout`.

A biblioteca SWT possui uma série de controles que representam as interfaces de comunicação com o SO em que a biblioteca é utilizada. Neste sentido, o pacote `org.eclipse.swt.widgets` define os componentes descritos no quadro 12.

Controle	Característica
`Button`	Controle selecionável que recebe notificações quando é pressionado e/ou liberado
`Canvas`	Controle que fornece uma superfície de desenhos gráficos. É frequentemente utilizado para o desenho de outros controles
`Caret`	Controle utilizado como um ponto de inserção de texto
`Combo`	Controle selecionável que permite ao usuário escolher uma opção dentro de uma lista ou opcionalmente um novo valor dentro de um campo de texto editável.
`Composite`	Controle capaz de conter outros *widgets*
`CoolBar`	Controle composto que permite que os usuários reposicionem dinamicamente os `CoolItem` contidos na barra
`CoolItem`	Objeto de interface selecionável posicionado de forma dinâmica em uma área da barra
`Group`	Controle composto de grupos de outros elementos, envolto por uma borda ou até mesmo um `Label`
`Label`	Controle não-selecionável que mostra textos ou imagens
`List`	Controle selecionável que permite ao usuário escolher um texto a partir de uma lista
`Menu`	Interface do usuário que contém lista de itens de menu
`MenuItem`	Controle selecionável que represente um item de menu
`ProgressBar`	Controle não-selecionável que mostra o progresso de uma ação do usuário
`Sash`	Controle selecionável que permite ao usuário arrastar o contorno de um objeto para a janela pai
`Scale`	Controle selecionável que representa um intervalo de valores numéricos
`ScrollBar`	Controle selecionável que representa um intervalo de valores de posições numéricas
`Shell`	Janela gerenciada pelo SO. `Shell` pode ser pai de um `Display` ou de outras janelas `Shell` secundárias
`Slider`	Controle selecionável que representa um intervalo de valores numéricos. É distinguido de um controle `Scale` por prover a funcionalidade de arrastar que pode ajustar o valor atual do componente
`TabFolder`	Controle composto que agrupa páginas e podem ser selecionados pelo usuário através dos seus `Labels`
`TabItem`	Controle selecionável de um respectivo `TabFolder` que representa uma página
`Table`	Controle selecionável que mostra uma lista de itens que podem ser selecionados pelo usuário. Itens são apresentados em linhas que mostram múltiplas colunas
`TableColumn`	Controle selecionável que representa uma coluna em um `Table`
`TableItem`	Controle selecionável que representa o item em uma `Table`
`Text`	Controle editável que permite ao usuário informar textos
`ToolBar`	Controle composto que suporta leiautes de `ToolItem` selecionáveis
`ToolItem`	Controle selecionável que representa um item de `ToolBar`
`Tree`	Controle selecionável que mostra uma lista de `TreeItem` de maneira hierárquica
`TreeItem`	Controle selecionável que representa um item de uma `Tree`

Quadro 12 – Componentes da biblioteca SWT

2.7 MODELAGEM DE PROJETOS ATRAVÉS DO EA

O EA é uma ferramenta CASE baseada na *Unified Modeling Language* (UML) utilizada para a modelagem e construção de projetos de sistemas de software (SPARX SYSTEMS, 2010, p. 3). Segundo Batista (2007, p. 29), a UML permite descrever de maneira visual, as necessidades que existem na construção e elaboração de um projeto de software,

através de mapas ou modelos. Neste sentido, a ferramenta EA abrange todas as fases do ciclo de desenvolvimento de software, desde o levantamento e elicitação de requisitos, detalhamento de casos de uso, desenvolvimento (com suporte a geração de código a partir dos diagramas de classes) e manutenção.

2.7.1 Exportação de projetos do EA

O EA oferece a possibilidade de efetuar a exportação de projetos. Segundo a Sparx Systems (2010, p. 288), pacotes podem ser exportados e importados, melhorando a flexibilidade e robustez dos modelos do EA e permitindo que analistas e modeladores exteriorizem elementos do projeto em um formato XML *Metadata Interchange* (XMI).

Ainda de acordo com a Sparx Systems (2010, p. 289), a exportação de um modelo do EA, para outros modelos, permite o desenvolvimento de atividades distribuídas, controle de versão manual, entre outros benefícios. A exportação dos dados do projeto, através do EA, está limitada para ferramentas que atendam os seguintes requisitos:

a) UML 2.1 XMI *standard* ou;
b) UML 1.4 XMI 1.2 *standard* ou;
c) UML 1.3 XMI 1.1 / XMI 1.0 *standard*.

Para efetuar a exportação de um pacote XMI, os seguintes passos podem ser executados:

a) selecionar o pacote a exportar, no menu `Project Browser`;
b) com o botão direito do mouse, selecionar a opção `Export model to XMI`.

Desta maneira, será exibida uma caixa de diálogo com o usuário, que poderá informar o nome do arquivo destino (que conterá o arquivo XML) e configurar os demais detalhes do processo de exportação. A figura 4 exibe os detalhes do diálogo que será exibido.

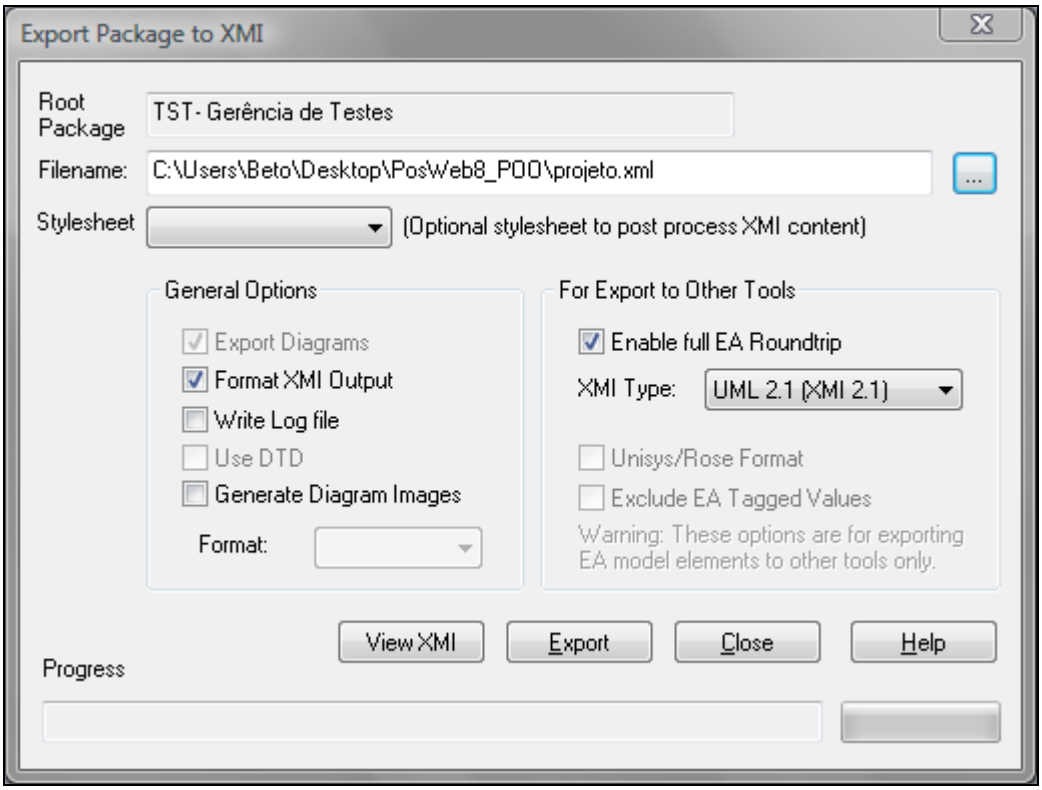


Figura 4 – Diálogo de exportação do arquivo XMI de um projeto modelado no EA

2.7.2 Criação de *links* entre elementos de um projeto

O EA oferece uma funcionalidade de criação de conexões entre determinadas partes do projeto. Essas conexões, ou *links*, são guardadas em cada elemento conectado e são úteis para mostrar ao usuário como determinadas partes do documento de especificação do projeto estão atreladas.

Para efetuar o processo de conexão entre elementos, os seguintes passos podem ser seguidos:

a) clicar com o botão direito em um elemento que será criado a conexão, selecionando a opção `Add | Create Link`;

b) será apresentado um diálogo, com os elementos que poderão ser relacionados ao objeto selecionado. Após efetuar a seleção, deve-se confirmar a conexão através do botão `OK`.

A figura 5 mostra a caixa de diálogo que é apresentada para o usuário.

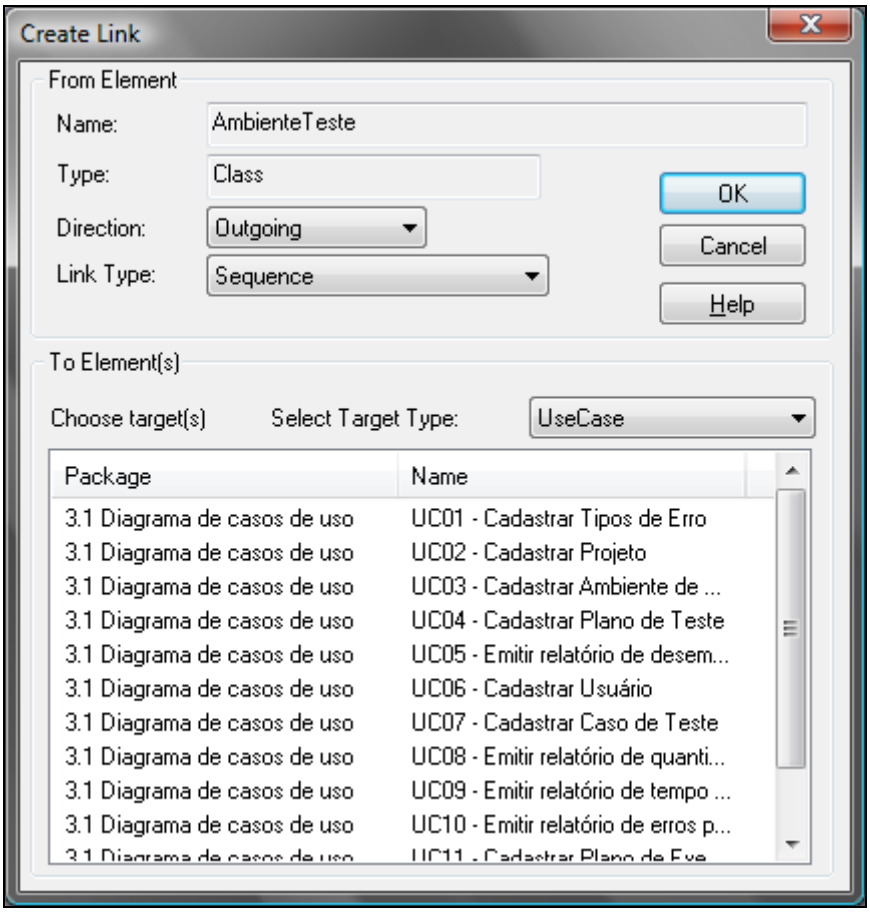


Figura 5 – Conexão de elementos de um projeto do EA

2.7.3 Integração do EA com uma aplicação Java

A interface de automação, segundo Batista (2007, p. 29), é um recurso disponibilizado pelo EA que permite que outras ferramentas acessem e gerenciem os seus elementos. Nesta biblioteca para automação são disponibilizados códigos fontes, que permitem o acesso a várias classes e métodos, que manipulam funcionalidades da ferramenta CASE.

Os ambientes de desenvolvimento que suportam ActiveX COM *clients* são capazes de se conectarem ao *Enterprise Architect Automation Interface.* Conforme a Sparx Systems (2010, p. 1667) é possível se conectar com a interface utilizando-se o Microsoft Visual Basic 6.0, Borland Delphi 7.0, Microsoft C#, Java e também Visual Basic.

Para efetuar a conexão com a interface do EA em uma aplicação Java basta copiar os arquivos `SSJavaCom.dll` e `eaapi.jar`, que se encontram geralmente no diretório `/Sparx Systems/EA`, para dentro do `classpath` do projeto Java.

Desta maneira, é possível efetuar o acesso às funcionalidades do EA, bastando efetuar a importação das classes Java, para dentro do projeto. As interfaces de comunicação estão no pacote `org.sparx`. O quadro 13 exibe um exemplo de comunicação feita com o EA, através da linguagem Java.

```
public void importFileXML(String pathEAPProject,
                  String pathDestinationFolder) throws Exception {

      // cria um novo objeto do tipo repositório
      Repository repository = new Repository();

      try {

            // abre o projeto passado por parâmetro
            repository.OpenFile(pathEAPProject);

            // não será aberto o client do EA
            repository.ShowWindow(0);

            // busca o GUID do projeto
            String guid = getGUIDOfPackage(repository);

            // busca a interface do projeto do repositório
            Project project = repository.GetProjectInterface();

            // transforma o guid para XML (identificador)
            String guidXML = project.GUIDtoXML(guid);

            // chama a função de exportação
            project.ExportPackageXMI(guidXML, EnumXMIType.xmiEA21, 0, -1,
1, -1, StringUtils.replaceBar(pathDestinationFolder));

      } catch (Exception e) {
            Log.logMessage(e, Level.Erro, getClass());
      } finally {
            repository.CloseFile();
      }
}
```

Quadro 13 – Acesso das funcionalidades do EA através do Java

A Sparx Systems (2010, p. 1669) afirma que a integração do EA e outras ferramentas é feita sobre um único repositório de dados, permitindo que as informações sejam atualizadas entre as ferramentas em tempo real. O repositório de dados engloba toda a coleção de modelos, pacotes, elementos, entre outros.

2.8 GERADOR DE RELATÓRIOS JASPERREPORTS

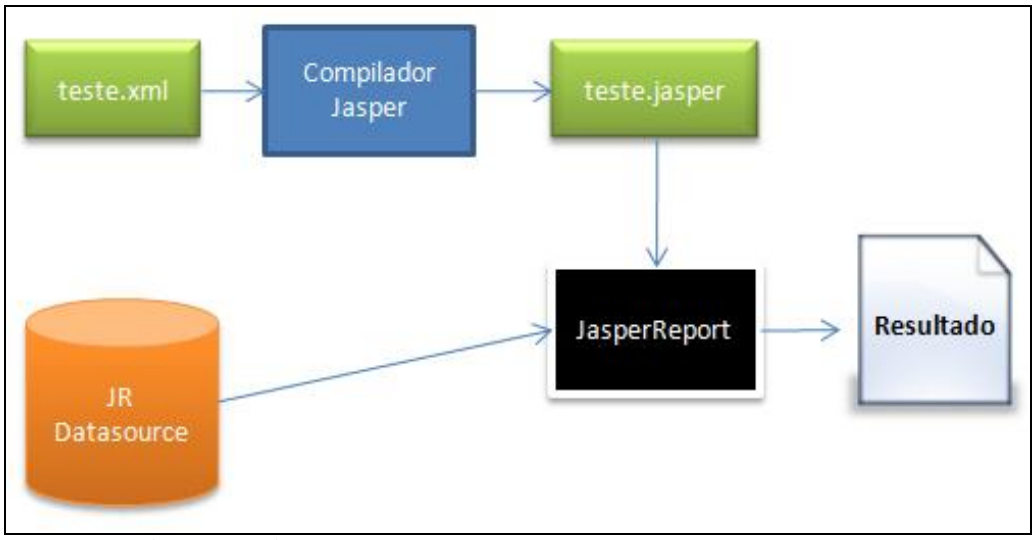
JasperReports é uma biblioteca *open source* para a geração de relatórios. Escrito em Java, essa biblioteca apresenta grande habilidade em organização e apresentação de conteúdo,

permitindo a geração dinâmica de relatórios em vários formatos, como *Portable Document Format* (PDF), *HyperText Markup Language* (HTML), *Comma-Separated Values* (CSV), entre outros.

A biblioteca é capaz de interpretar os dados através de um arquivo no formato XML que possui todo o mapeamento de campos e dados do relatório criado pelo usuário. Este arquivo XML obedece a uma determinada estrutura, vocabulário e restrições declaradas em um arquivo *Document Type Definition* (DTD). O processo de geração do relatório se utiliza de vários artefatos que devem ser gerados para a obtenção do relatório:

a) JasperDesign: representa a definição do relatório;
b) JasperReport: é o JasperDesign compilado;
c) JasperPrint: é o objeto gerado.

A figura 6 mostra o funcionamento da biblioteca JasperReports.



Fonte: Lopes (2008, p. 36).

Figura 6 – Funcionamento da biblioteca `JasperReports`

Os dados apresentados no relatório são comunicados através da interface `net.sf.jasperreports.engine.JRDataSource` da biblioteca. Existem classes que implementam a comunicação de diversos modos: desde uma conexão com um SGBD até mesmo a interpretação de dados de uma *collection* da linguagem Java.

2.8.1 Desenvolvimento de relatórios com o auxílio da interface IReport

O desenvolvimento de relatórios utilizando simplesmente a biblioteca JasperReports exige um grau de conhecimento da biblioteca demasiadamente alto, além de requerer um

grande conhecimento em codificação XML. Segundo Lopes (2008, p. 24), para solucionar este problema foram criadas diversas ferramentas com interface gráfica, sendo o IReport, a mais difundida.

O IReport disponibiliza uma interface de comunicação entre o usuário e a biblioteca JasperReports que possibilita ao usuário a modelagem de relatórios de maneira intuitiva. Além disso, o software fornece uma ferramenta para visualizar e testar o relatório criado, antes mesmo de utilizá-lo na aplicação.

Ao criar um relatório através do IReport, é gerado um arquivo no formato Jasper Reports *eXtensible Markup Language* (JRXML), que representa um arquivo XML que obedece as definições do DTD da biblioteca JasperReports. Após a compilação do projeto, é gerado um arquivo no formato JASPER, que poderá receber os parâmetros e dados da aplicação que se utiliza do relatório. A figura 7 apresenta a interface de modelagem oferecida pelo IReport.

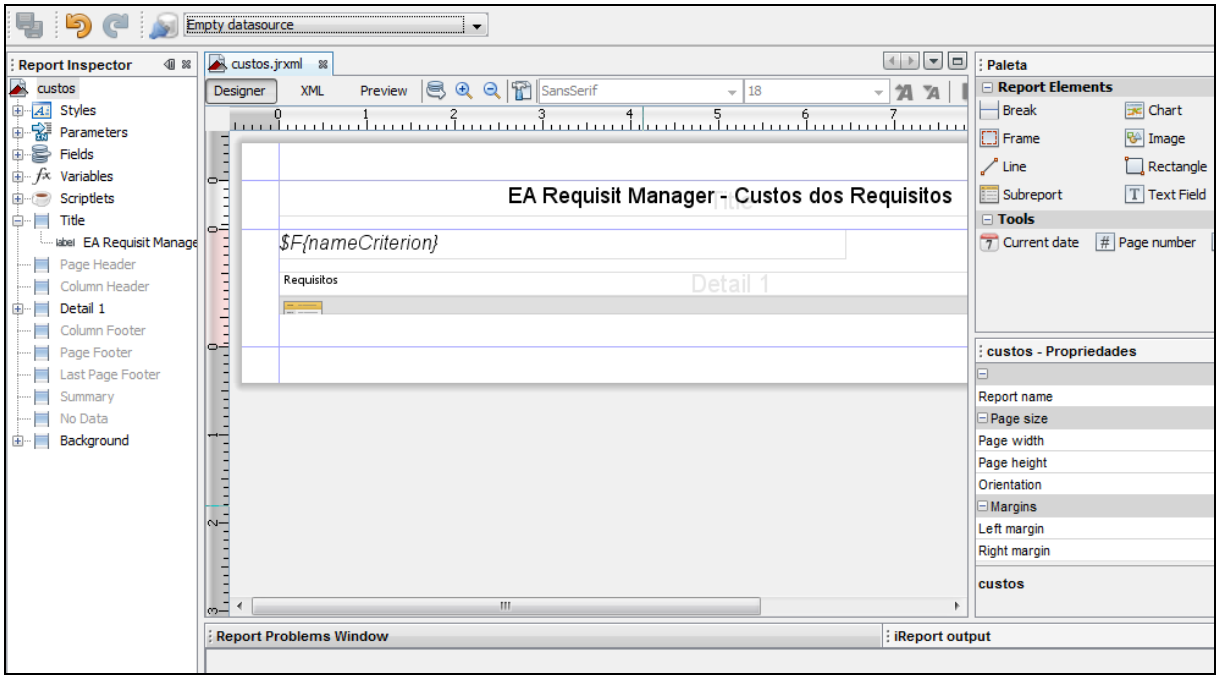


Figura 7 – Interface de modelagem de relatórios oferecida pelo IReport

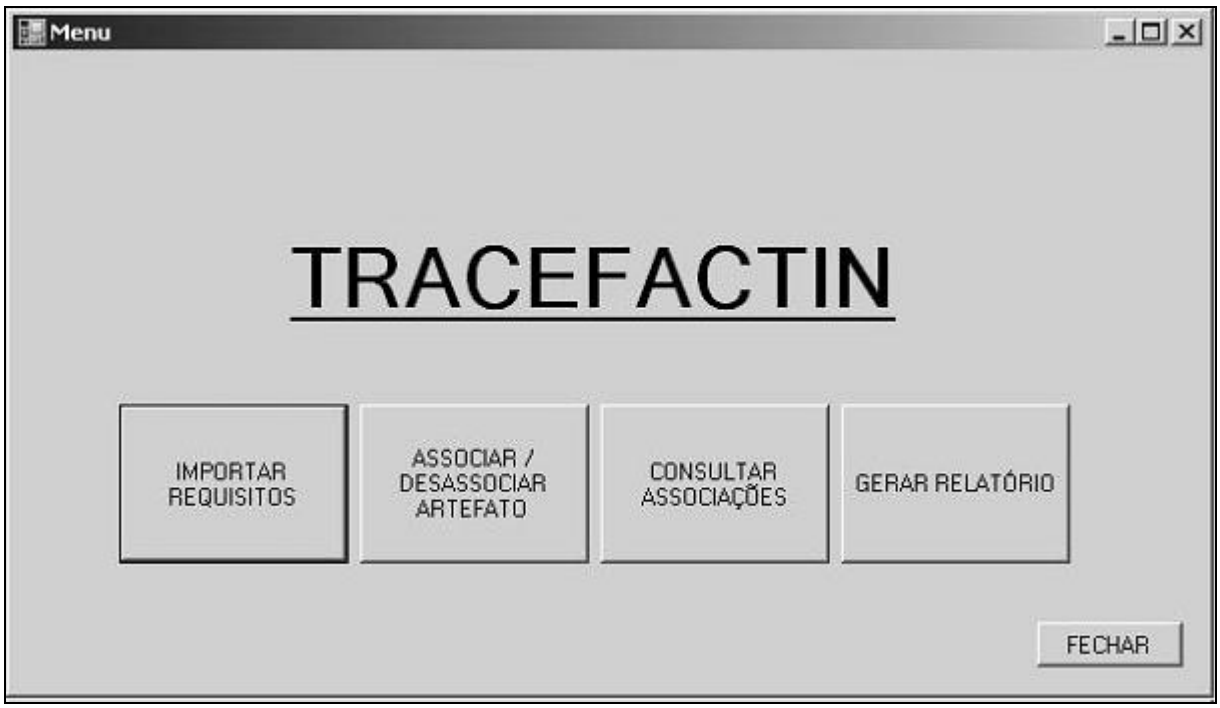
2.9 TRABALHOS CORRELATOS

Existem disponíveis ferramentas comerciais e acadêmicas que abordam a temática da rastreabilidade e análise de impacto com a mudança de requisitos, baseados em uma métrica. Desta maneira, é possível encontrar desde trabalhos de conclusão de curso até teses de

doutorado. Dentre estes trabalhos, foi selecionada a ferramenta para o auxílio à rastreabilidade de Santos e Aragão (2009), a ferramenta CASE de gerência de Batista (2007) e o ambiente de acompanhamento de custos de requisitos proposto por Tocchetto (2007), os quais são descritos a seguir.

2.9.1 TraceFact-In: ferramenta para auxílio à rastreabilidade de artefatos de software

A partir da identificação de que no processo de criação de um software era necessário associar os artefatos (códigos fontes) aos requisitos descritos na fase de projeto, Santos e Aragão (2009) desenvolveram uma ferramenta (chamada de *Add-In*) que se integra com o ambiente de desenvolvimento Visual Studio 2008 (figura 8). Nesta, o documento de especificação é escrito através da ferramenta Rational Requisite Pro, a qual pertence a IBM.



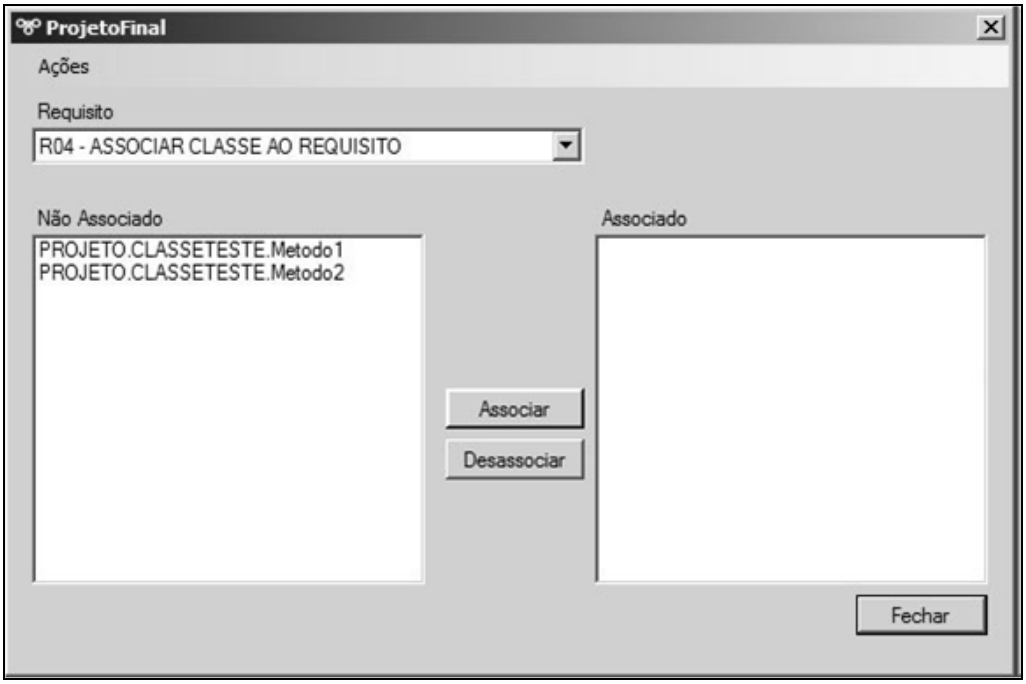
Fonte: Santos e Aragão (2009, p. 36).

Figura 8 – Tela principal da ferramenta TraceFact-In

Para que ocorra o mapeamento entre o documento de especificação e os códigos fontes (*links* de rastreabilidade) é necessário que o usuário importe os requisitos especificados manualmente para a ferramenta.

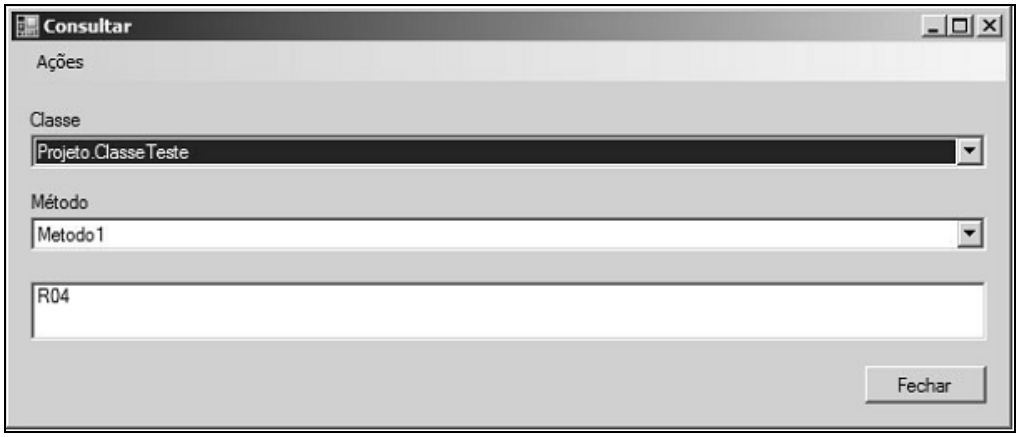
Após, o usuário pode associar os requisitos importados com os artefatos do projeto (figura 9), onde o nível de abstração é feito até os métodos de uma classe. Os dados da

associação são gravados em um banco de dados *Microsoft Structured Query Language* (MS SQL) *Server*. Também é disponibilizada uma interface de consulta (figura 10), na qual é possível verificar quais requisitos estão associados aos métodos de uma classe.



Fonte: Santos e Aragão (2009, p. 39).

Figura 9 – Tela para associar e desassociar artefato



Fonte: Santos e Aragão (2009, p. 37).

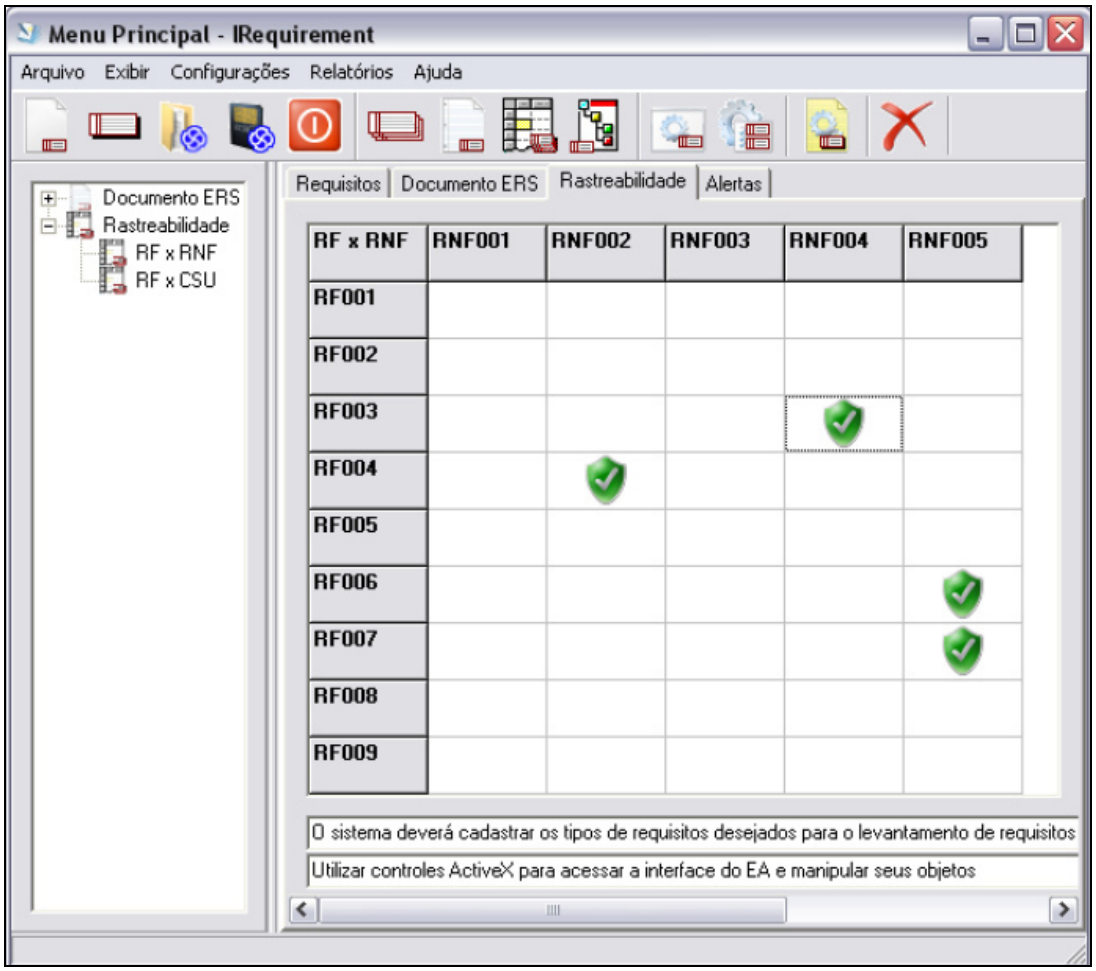
Figura 10 – Tela para consulta de artefatos

2.9.2 IRequirement: ferramenta CASE de gerência de requisitos de software integrada com EA

A ferramenta proposta por Batista (2007) tem como objetivo o gerenciamento de requisitos de software, propondo que se tenha um controle mais eficiente dos requisitos,

focando na produção de um documento de especificação completo. A ferramenta possui integração com o EA, dando a possibilidade de se incorporar requisitos desenvolvidos por esta ferramenta.

Um dos diferenciais da ferramenta CASE em relação a outras, que possuem as mesmas finalidades, é a possibilidade de vinculação de requisitos com outros requisitos e casos de uso, provendo desta maneira a rastreabilidade entre os elementos do projeto (figura 11). A ferramenta possui também uma funcionalidade que possibilita a configuração de diversos tipos de matriz de rastreabilidade, como a referência cruzada entre tipos de requisito e casos de uso.



Fonte: Batista (2007, p. 51).

Figura 11 – Tela de relacionamento de elementos em uma matriz de rastreabilidade

A linguagem de programação utilizada para a criação da ferramenta CASE foi o Borland Delphi 6.

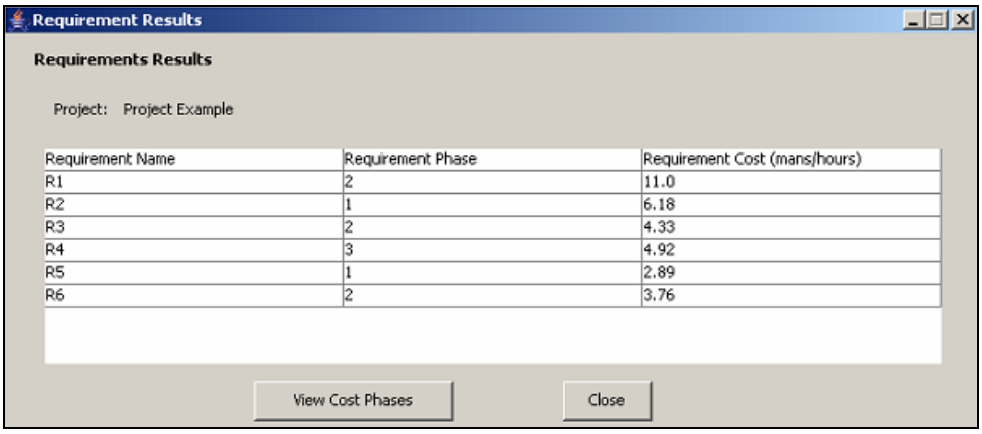
2.9.3 AspectCost: ambiente de gerência e acompanhamento de custos e requisitos baseados em *Aspect-Oriented Programming* (AOP)

Tocchetto (2007) propõe uma metodologia e disponibiliza uma ferramenta para estimar o custo total de um projeto, mensurar e controlar os custos dos requisitos no paradigma de desenvolvimento orientado a aspectos, estendendo o modelo de estimativa UCP.

As características da metodologia de Tocchetto (2007, p. 22) são explicadas da seguinte maneira:

a) estimativa do custo total do projeto: o autor engloba os casos de uso aspectuais para propiciar uma estimativa de projeto para os que utilizam a programação orientada a aspectos;
b) estimativa dos custos de requisitos: é baseado no processo AHP, proposto por Saaty (1991). O autor utiliza este processo para ponderar as características qualitativas, no caso, os requisitos. Como resultado final, é obtido o valor percentual do custo que cada requisito representa no projeto;
c) controle dos custos dos requisitos: é utilizada uma tabela comparativa de relacionamentos dos requisitos e os fatores de relacionamento entre os mesmos para controlar as alterações de valores ao longo dos requisitos, e por consequência, do projeto.

Desta maneira, é possível verificar que a metodologia proposta por Tocchetto (2007) permite efetuar o controle dos valores percentuais dos requisitos, bem como efetuar o controle dos custos de um projeto (figura 12), partindo da ideia que a evolução dos requisitos é constante.



Fonte: Tocchetto (2007, p. 72).

Figura 12 – Tela de apresentação do resultado do cálculo de custos dos requisitos

3 DESENVOLVIMENTO

Neste capítulo são apresentados os conceitos utilizados neste trabalho, os requisitos do *plugin*, a especificação, a implementação e os resultados obtidos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Nesta seção são apresentadas as funcionalidades necessárias pesquisadas para o funcionamento do *plugin* elaborado neste trabalho. Os resultados são demonstrados nas figuras 13 e 14, representados através de requisitos funcionais e requisitos não-funcionais.

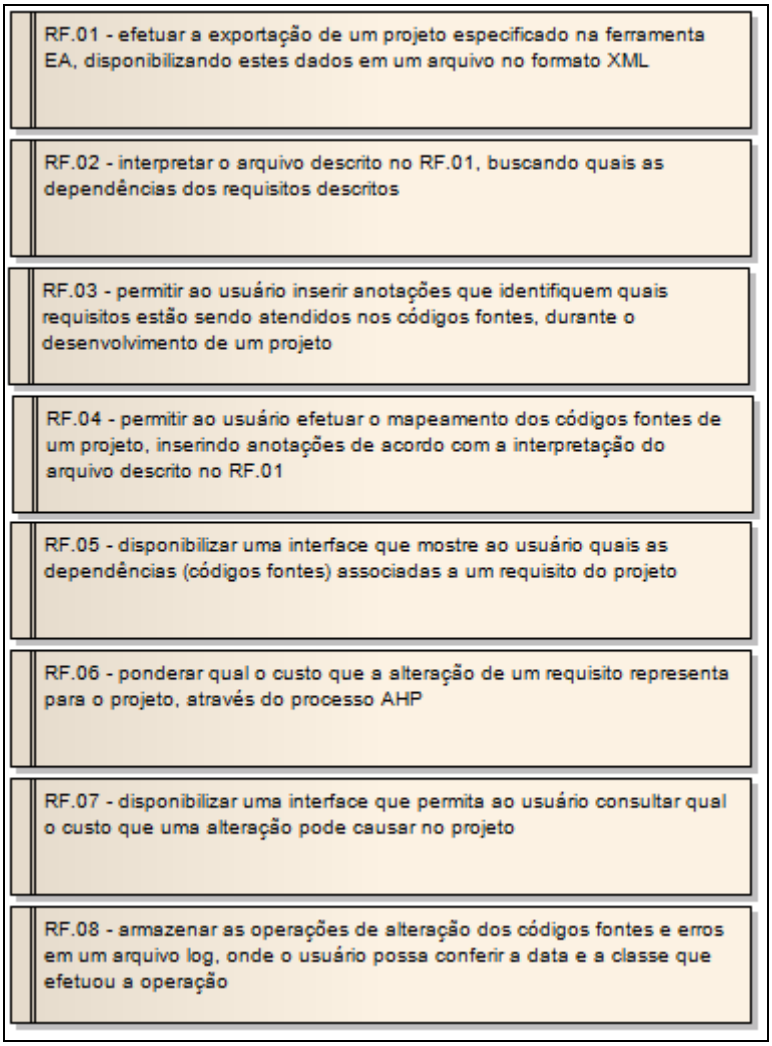


Figura 13 – Requisitos Funcionais do *plugin* desenvolvido

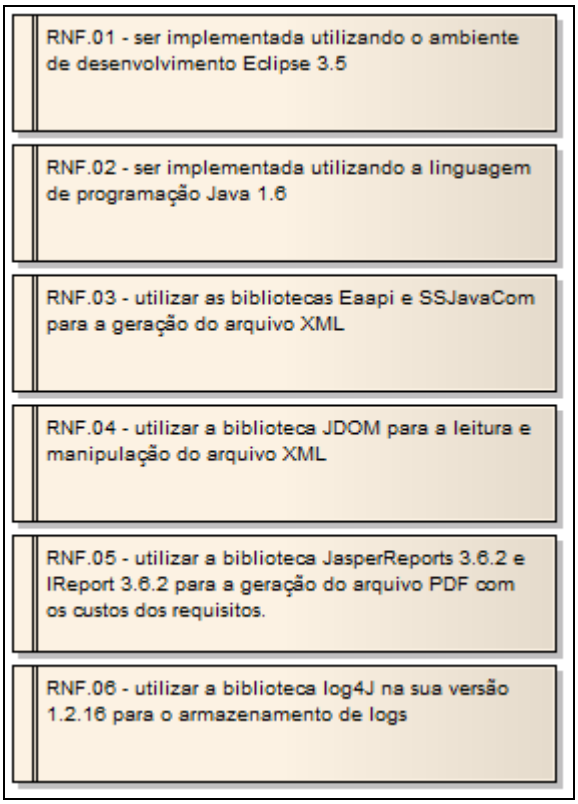


Figura 14 – Requisitos não-funcionais do *plugin* desenvolvido

3.2 ESPECIFICAÇÃO

O *plugin* foi especificado empregando o processo de análise e projeto orientado a objetos. A ferramenta Enterprise Architect 7.0 foi utilizada para o desenvolvimento dos diagramas de caso de uso, sequência e classes da *Unified Modeling Language* (UML). Também encontra-se neste capítulo o detalhamento dos casos de uso e uma matriz de relacionamento entre requisitos (funcionais e não-funcionais). O nome do *plugin* foi definido como EA Requisite Manager.

3.2.1 Diagrama de casos de uso

Na figura 15 é apresentado o diagrama de casos de uso do projeto, assim como o detalhamento de cada caso de uso que compõe a especificação do projeto (quadros 14 a 18).

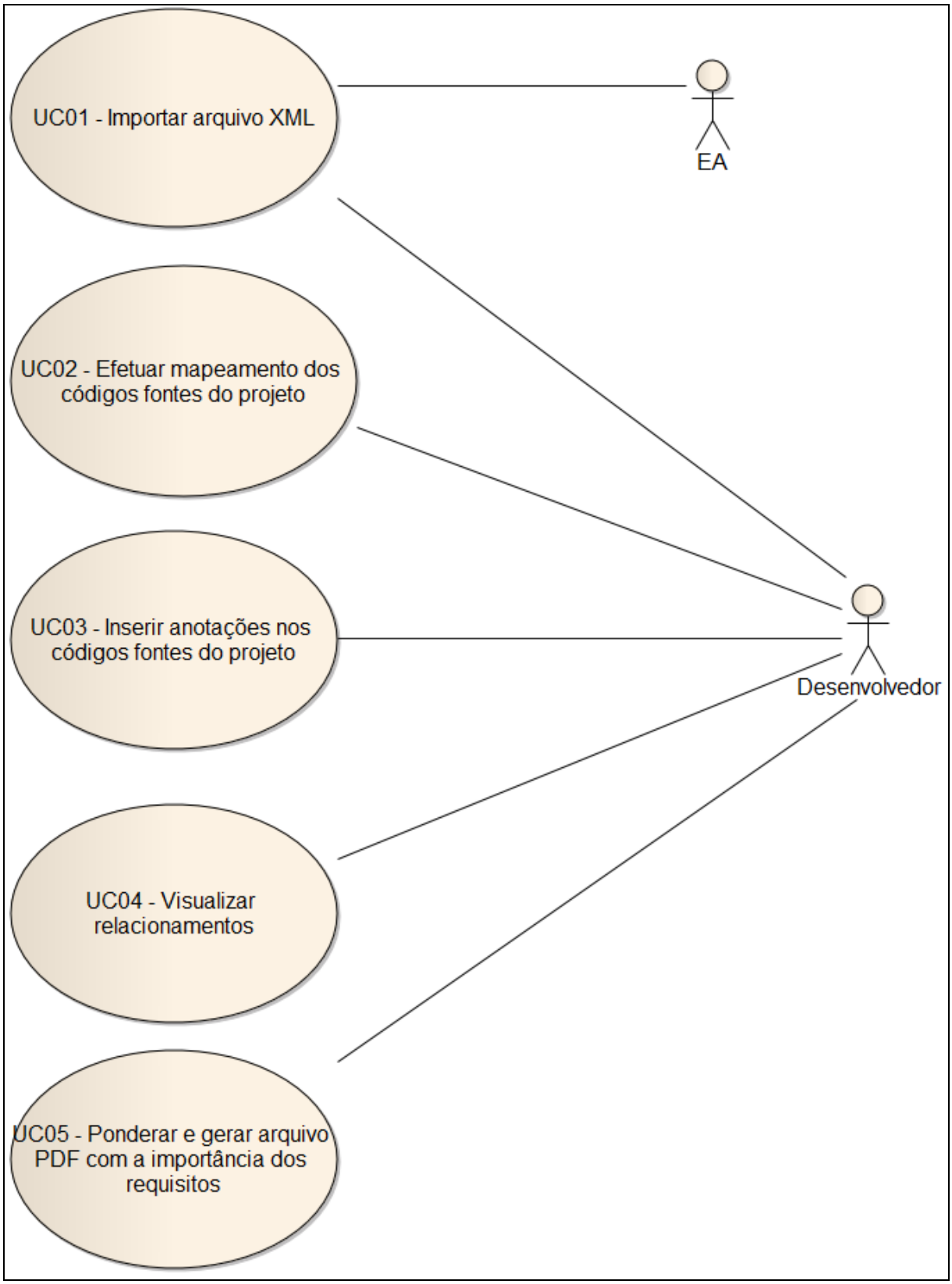


Figura 15 – Diagrama de casos de uso da ferramenta

UC01 - Importar arquivo XML
Descrição: Relacionado aos requisitos funcionais RF.01 e RF.08 e aos requisitos não-funcionais RNF.01, RNF.02, RNF.03 e RNF.06.

Pré-condições:
1) Executar o ambiente de desenvolvimento Eclipse;
2) EA Requisite Manager deve estar devidamente instalado no ambiente.

Cenário principal:
1) Desenvolvedor clica com o botão direito em cima de um projeto, na view "Package Explorer" do Eclipse;
2) Desenvolvedor seleciona o menu principal "EA Requisite Manager";
3) Desenvolvedor seleciona o submenu da funcionalidade: "Importar XML EA...";
4) Ambiente apresenta uma janela com dois campos: "Arquivo .EAP" e "Destino XML";
5) Desenvolvedor informa os dois campos;
6) Desenvolvedor pressiona o botão "Importar";
7) Ambiente apresenta mensagem: "O processo de importação foi iniciado e pode demorar alguns minutos!";
8) Desenvolvedor pressiona o botão “OK”;
9) Enterprise Architect apresenta uma janela "Export package to XMI", referente ao processo de exportação;
10) Ambiente apresenta mensagem: "O processo de importação foi finalizado com sucesso!";
11) Ambiente cria arquivo XML no diretório informado pelo Desenvolvedor.

Cenário alternativo:
No passo 5 do cenário principal "Importar XML da ferramenta CASE EA":
1) Desenvolvedor fecha a janela de diálogo apresentada, sem pressionar o botão "Importar";
2) Ambiente não cria arquivo XML para o projeto selecionado.

Cenário alternativo:
No passo 6 do cenário principal "Importar XML da ferramenta CASE EA":
1) Desenvolvedor clica no botão importar sem preencher os campos “Arquivo .EAP” ou “Destino ML”;
2) Ambiente apresenta mensagem: “É necessário informar um arquivo válido!”;
3) Ambiente não cria arquivo XML para o projeto selecionado.

Quadro 14 – Detalhamento do caso de uso UC01

UC02 - Efetuar mapeamento dos códigos fontes do projeto
Descrição: Relacionado aos requisitos funcionais RF.02, RF.04 e RF.08 e aos requisitos não-funcionais RNF.01, RNF.02, RNF.04 e RNF.06.

Pré-condições:
1) Executar o ambiente de desenvolvimento Eclipse;
2) EA Requisite Manager deve estar devidamente instalado no ambiente;
3) Desenvolvedor deverá ter exportado o arquivo XML do projeto selecionado.

Cenário principal:
1) Desenvolvedor clica com o botão direito em cima de um projeto, na view "Package Explorer" do Eclipse;
2) Desenvolvedor seleciona o menu principal "EA Requisite Manager";
3) Desenvolvedor seleciona o submenu da funcionalidade: "Mapear códigos fontes...";
4) Ambiente apresenta uma janela com um campo para o Desenvolvedor informar o arquivo XML com os relacionamentos e um label indicando o caminho do projeto selecionado;
5) Desenvolvedor preenche o campo “Arquivo XML”;
6) Desenvolvedor pressiona o botão “Mapear”;
7) Ambiente apresenta mensagem: "O processo de mapeamento foi iniciado e pode demorar alguns minutos!”;
8) Desenvolvedor pressiona o botão “OK”;
9) Ambiente efetua a inserção de anotações nos códigos fontes do projeto indicado;
10) Ambiente apresenta mensagem: “O processo de mapeamento foi finalizado com sucesso”;

Cenário alternativo:
No passo 5 do cenário principal “Efetuar mapeamento dos códigos fontes do projeto, a partir de um arquivo XML de um projeto do EA”:
1) Desenvolvedor fecha a janela de diálogo apresentada, sem pressionar o botão "Mapear";
2) Ambiente não realiza mapeamento nos códigos fontes do projeto selecionado.

Cenário alternativo:
No passo 6 do cenário principal " Efetuar mapeamento dos códigos fontes do projeto, a partir de um arquivo XML de um projeto do EA":
1) Desenvolvedor clica no botão mapear sem preencher os campos “Arquivo XML”;
2) Ambiente apresenta mensagem: “É necessário informar um arquivo válido!”;
3) Ambiente não realiza mapeamento nos códigos fontes do projeto selecionado.

Cenário alternativo:
No passo 9 do cenário principal “Efetuar mapeamento dos códigos fontes do projeto, a partir de um arquivo XML de um projeto do EA”:
1) Ambiente efetua o parser do arquivo XML e não encontra relacionamentos para determinada classe;
2) Ambiente não insere anotações nos códigos fontes, uma vez que não possuem relacionamentos com requisitos.

Cenário alternativo:
No passo 9 do cenário principal “Efetuar mapeamento dos códigos fontes do projeto, a partir de um arquivo XML de um projeto do EA”:
1) Ambiente detecta que a classe ou método já possui anotação inserida;
2) Ambiente remove anotações antigas e insere as novas.

Quadro 15 – Detalhamento do caso de uso UC02

UC03 – Inserir anotações nos códigos fontes do projeto
Descrição: Relacionado aos requisitos funcionais RF.03 e RF.08 e aos requisitos não-funcionais RNF.01, RNF.02 e RNF.06.

Pré-condições:
1) Executar o ambiente de desenvolvimento Eclipse;
2) EA Requisite Manager deve estar devidamente instalado no ambiente;
3) Desenvolvedor deverá possuir a biblioteca "ea.requisit.manager.commons" no classpath do projeto.

Cenário principal:
1) Desenvolvedor seleciona uma classe Java de um projeto, abrindo-o no editor do Eclipse;
2) Desenvolvedor insere anotação "Requisitos", no código fonte, indicando quais requisitos estão relacionados;
3) Desenvolvedor salva as alterações no código fonte.

Quadro 16 - Detalhamento do caso de uso UC03

UC04 - Visualizar relacionamentos
Descrição: Relacionado aos requisitos funcionais RF.05 e RF.08 e aos requisitos não-funcionais RNF.01, RNF.02 e RNF.06.

Pré-condições:
1) Executar o ambiente de desenvolvimento Eclipse;
2) EA Requisite Manager deve estar devidamente instalado no ambiente;
3) A perspectiva "Requisitos x Códigos Fontes" deverá estar fechada.

Cenário principal:
1) Desenvolvedor clica com o botão direito em cima de um projeto, na view "Package Explorer" do Eclipse;
2) Desenvolvedor seleciona o menu principal "EA Requisite Manager";
3) Desenvolvedor seleciona o submenu da funcionalidade: "Requisitos x Códigos Fontes";
4) Ambiente efetua a leitura de todos os relacionamentos entre os códigos fontes e os requisitos;
5) Ambiente apresenta uma view com o título "Requisitos x Códigos Fontes", disponibilizando uma árvore com o título "Requisitos". Os filhos da árvore deverão ser formados pelo nome do requisito. Já os sub-nós deverão ser compostos pelas classes e métodos que se relacionam com o requisito do nó.

Cenário alternativo:
No passo 4 do cenário principal "Visualizar relacionamentos entre requisitos e classes de um projeto de software":
1) Ambiente efetua a leitura de todos os relacionamentos entre os códigos fontes e os requisitos e não encontra classes ou métodos com anotações;
2) Ambiente apresenta uma view com o título "Requisitos x Códigos Fontes", sem informações de relacionamento.

Cenário alternativo:
No passo 5 do cenário principal "Visualizar relacionamentos entre requisitos e classes de um projeto de software":
1) Ao efetuar a apresentação da view, caso ela já esteja aberta, o ambiente não deverá apresentar as informações dos relacionamentos.

Quadro 17 - Detalhamento do caso de uso UC04

UC05 - Ponderar a importância dos requisitos e gerar arquivo PDF
Descrição: Relacionado aos requisitos funcionais RF.06, RF.07 e RF.08 e aos requisitos não-funcionais RNF.01, RNF.02, RNF.05, RNF.06.

Pré-condições:
1) Executar o ambiente de desenvolvimento Eclipse;
2) EA Requisite Manager deve estar devidamente instalado no ambiente.

Cenário principal:
1) Desenvolvedor clica com o botão direito em cima de um projeto, na view "Package Explorer" do Eclipse;
2) Desenvolvedor seleciona o menu principal "EA Requisite Manager";
3) Desenvolvedor seleciona o submenu da funcionalidade: "Visualizar custos de alterações";
4) Ambiente apresenta uma janela com dois campos: "Projeto" e "Destino PDF";
5) Desenvolvedor informa os dois campos;
6) Desenvolvedor pressiona o botão "Visualizar";
7) Ambiente busca os relacionamentos entre: classes e requisitos (códigos fontes mapeados), métodos e requisitos (códigos fontes mapeados) e casos de uso e requisitos (arquivo XML informado);
8) Ambiente aplica as técnicas da metodologia AHP, gerando os percentuais de importância para os três critérios estabelecidos: classes, métodos e casos de uso;
9) Ambiente gera o arquivo PDF, através do gerador de relatórios JasperReports, salvando o arquivo no diretório informado. O arquivo PDF deverá conter três seções, com os seguintes títulos: “Critério de Classes”, “Critério de Métodos” e “Critério de Casos de Uso”. Para cada seção, deverão ser apresentadas duas colunas com os nomes: “Requisitos” e “Valor (%)”.

Cenário alternativo:
No passo 5 do cenário principal “Ponderar a importância dos requisitos e gerar arquivo PDF, de acordo com os critérios: classe, métodos e casos de uso”:
1) Desenvolvedor fecha a janela de diálogo apresentada, sem pressionar o botão "Visualizar";
2) Ambiente não gera arquivo PDF.

Cenário alternativo:
No passo 5 do cenário principal “Ponderar a importância dos requisitos e gerar arquivo PDF, de acordo com os critérios: classe, métodos e casos de uso”:
1) Desenvolvedor clica no botão visualizar sem preencher os campos “Projeto” ou “Destino PDF”;
2) Ambiente apresenta mensagem: “É necessário informar um arquivo válido!”;
3) Ambiente não gera arquivo PDF.

Cenário alternativo:
No passo 7 do cenário principal “Ponderar a importância dos requisitos e gerar arquivo PDF, de acordo com os critérios: classe, métodos e casos de uso”:
1) Ambiente não encontra relacionamentos no projeto indicado;
2) Ambiente não gera arquivo PDF e é apresentada a seguinte mensagem: “Não foram encontrados relacionamentos para o projeto indicado”.

Cenário alternativo:
No passo 7 do cenário principal “Ponderar a importância dos requisitos e gerar arquivo PDF, de acordo com os critérios: classe, métodos e casos de uso”:
4) Ambiente não encontra relacionamentos para algum(s) do(s) critério(s);
5) Ambiente não gera arquivo PDF, porém apresentando somente as seções em que os critérios possuem relacionamento.

Quadro 18 - Detalhamento do caso de uso UC05

3.2.2 Matriz de relacionamento entre requisitos e caso de usos

Para demonstrar o cumprimento de todos os requisitos funcionais, é apresentado no quadro 19 uma matriz de relacionamento, que exibe as vinculações feitas entre os requisitos e os casos de uso que os implementam.

Caso de uso / Requisito	**UC01**	**UC02**	**UC03**	**UC04**	**UC05**
RF.01	**X**				
RF.02		**X**			
RF.03			**X**		
RF.04		**X**			
RF.05				**X**	
RF.06					**X**
RF.07					**X**
RF.08	**X**	**X**	**X**	**X**	**X**

Quadro 19 – Matriz de relacionamento entre requisitos e casos de uso

3.2.3 Diagrama de sequência

A seguir nas figuras 16 a 19 são apresentados os diagramas de sequência dos casos de uso especificados para a realização do *plugin*. Serão apresentados os diagramas de sequência gerados a partir dos casos de uso considerados mais complexos, onde existe uma grande quantidade de mensagens entre os componentes envolvidos.

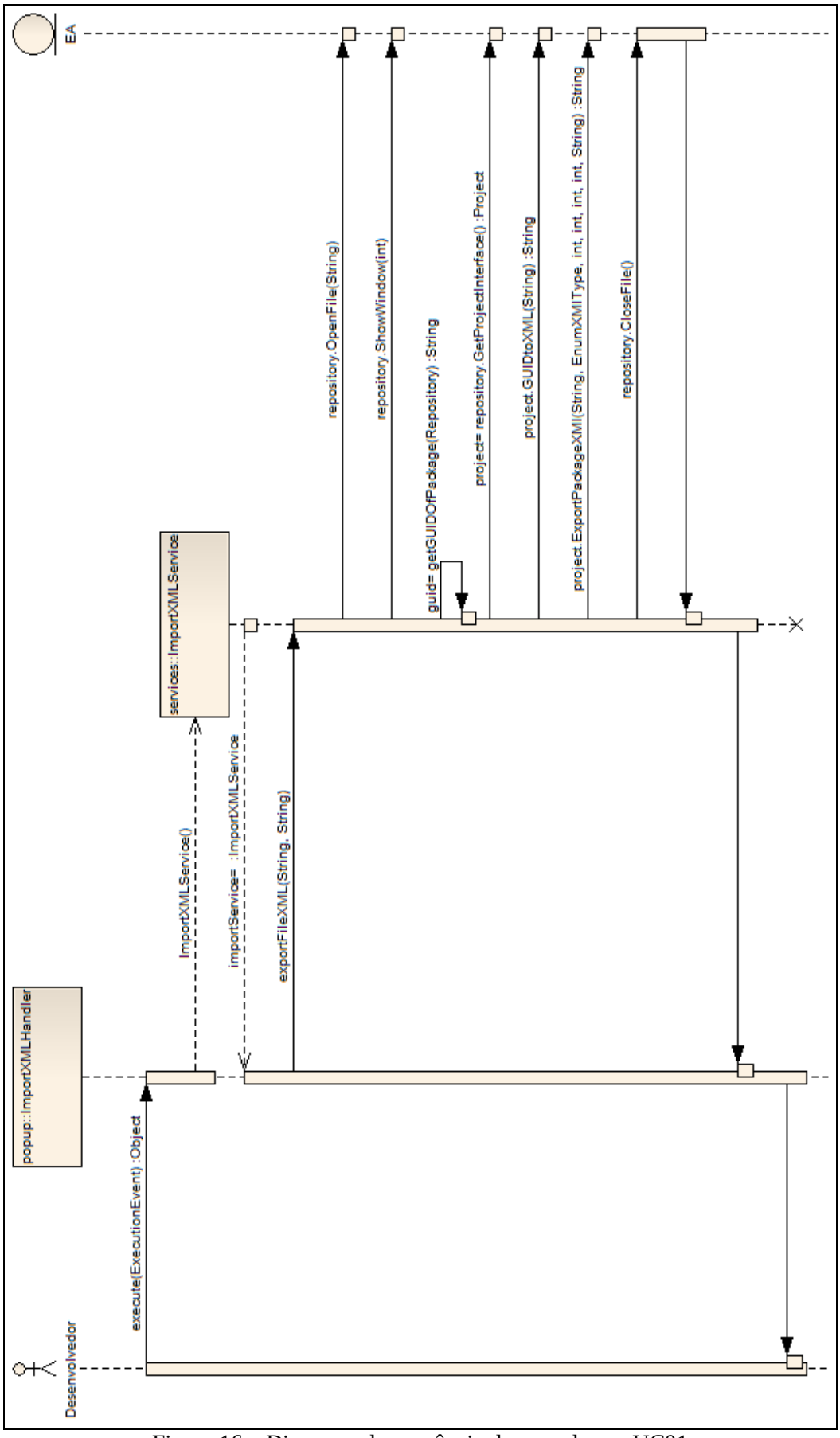


Figura 16 – Diagrama de sequência do caso de uso UC01

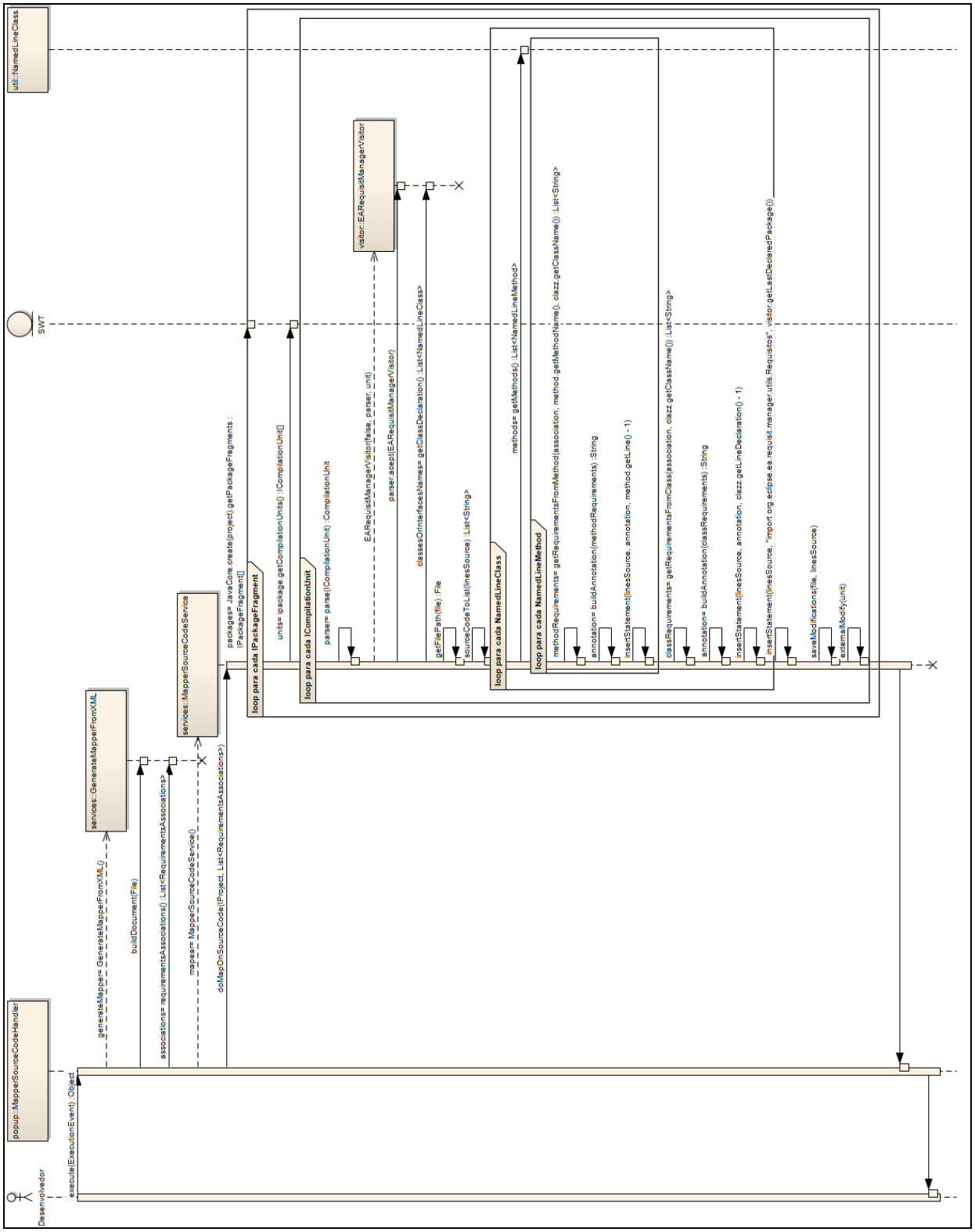


Figura 17 – Diagrama de sequência do caso de uso UC02

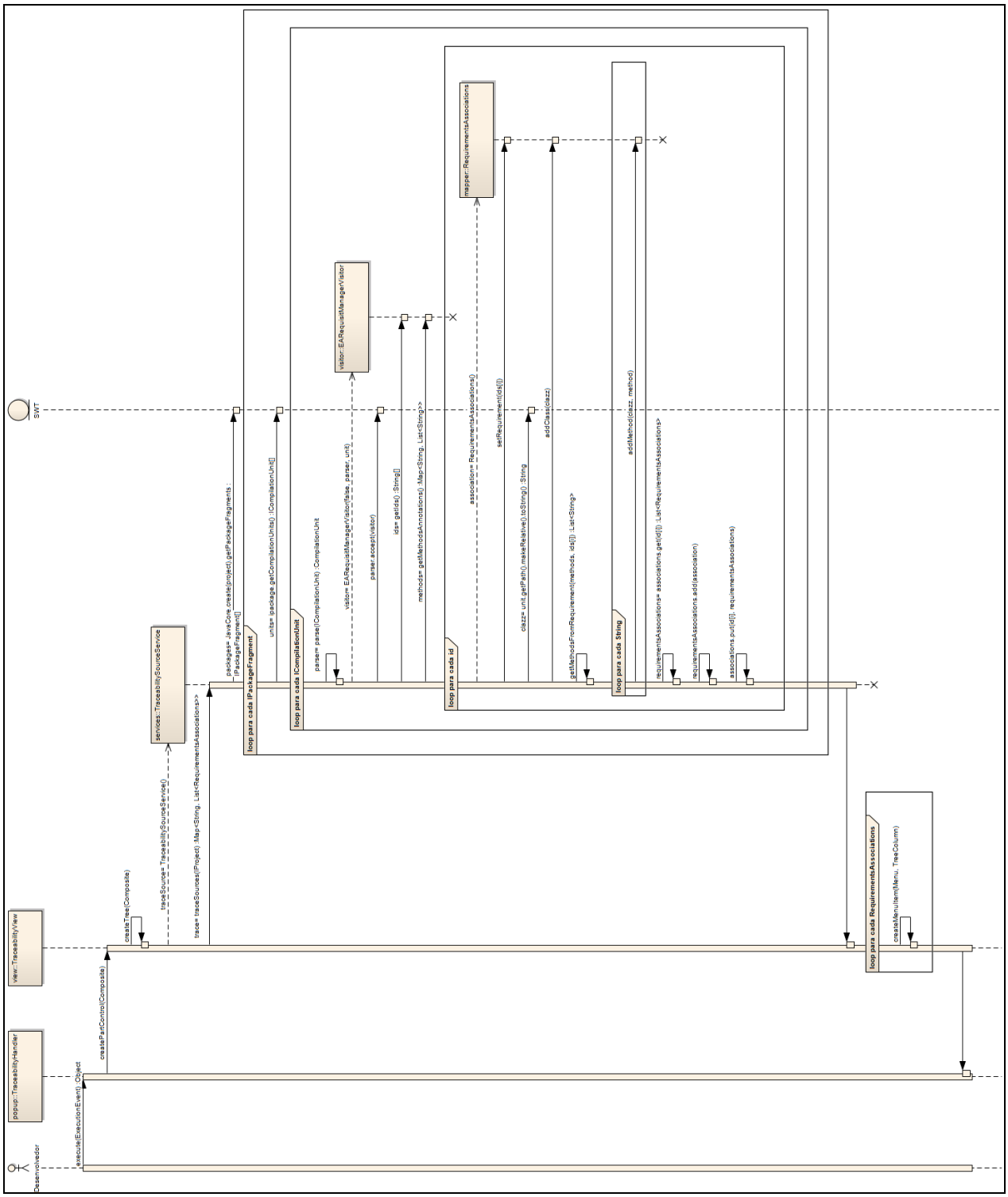


Figura 18 – Diagrama de sequência para o caso de uso UC04

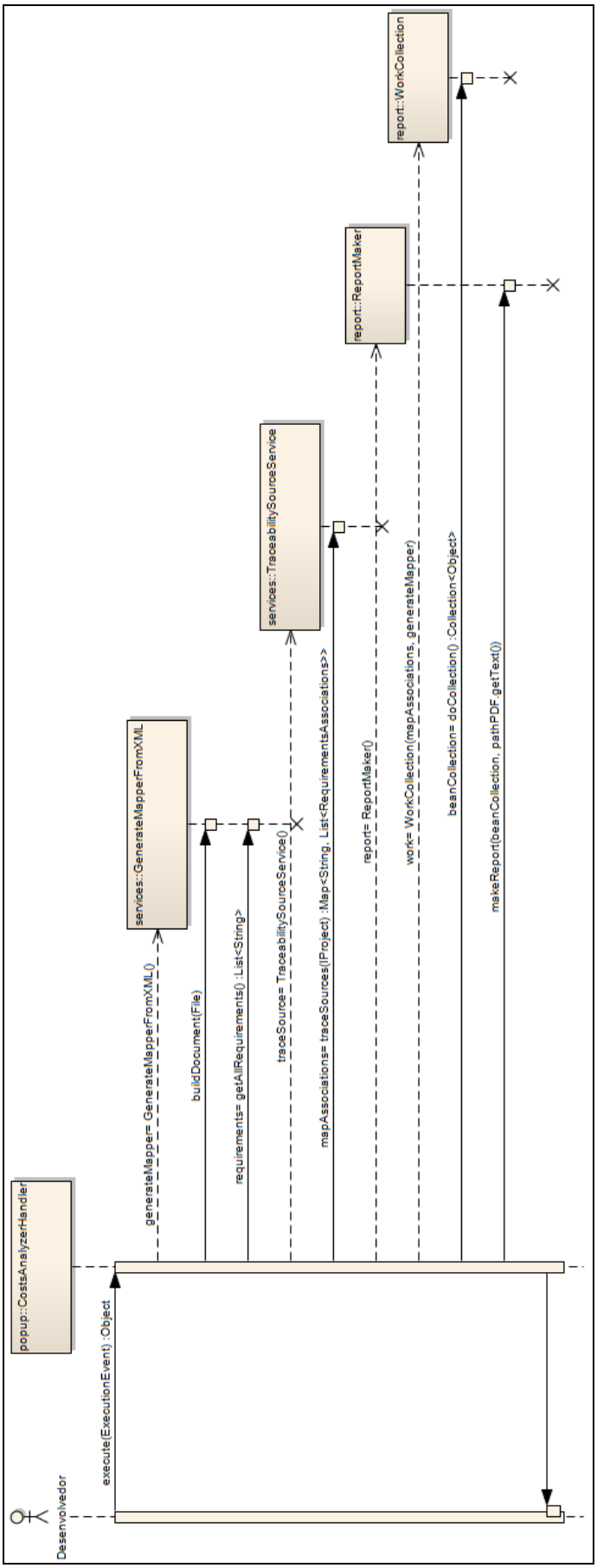


Figura 19 – Diagrama de sequência para o caso de uso UC05

3.2.4 Diagrama de classes do *plugin*

Com a intenção de facilitar a leitura, entendimento e interpretação do diagrama de classes da ferramenta, são apresentados os diagramas de acordo com o pacote que o conjunto de classes representa. Nos próximos tópicos, os diagramas contém o relacionamento entre as classes, exibindo por sua vez, somente os métodos que fazem parte de cada uma.

3.2.4.1 Pacote parser.visitor

O pacote `parser.visitor` é responsável pelo *parser* e visitação dos nós que compõe um código fonte Java. O único componente que compõe o pacote é a classe `EARequisitManagerVisitor` (figura 20).

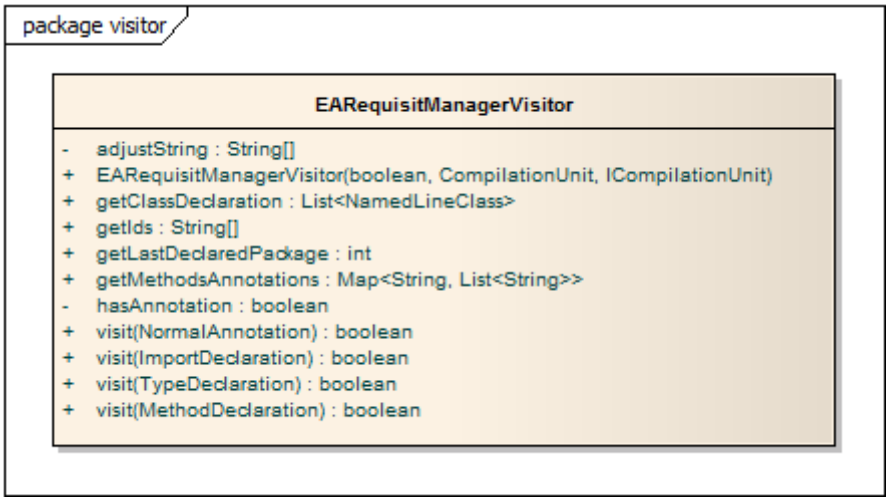


Figura 20 – Diagrama do pacote `parser.visitor`

A classe `EARequisitManagerVisitor` é responsável pela identificação dos elementos que fazem parte da estrutura do código fonte de um determinado arquivo Java. `EARequisitManagerVisitor` estende da classe `org.eclipse.jdt.core.dom.ASTVisitor`.

Vale observar que o Eclipse, a partir do Eclipse Java *Development Tool* (JDT), disponibiliza a implementação de uma Árvore Sintática absTrata (AST). No pacote `org.eclipse.jdt.core.dom` o JDT disponibiliza um conjunto de classes que representa o código fonte através de uma AST. Também é possível acessar cada parte do código fonte, através da visitação de nós, que são objetos oriundos de subclasses de `org.eclipse.jdt.core.dom.ASTNode`. A visitação de elementos do tipo `ASTNode` pode ser feito pela implementação e extensão da classe `ASTVisitor`, que através do padrão Visitor,

permite o acesso a cada elemento que compõe a estrutura do código fonte de um componente Java.

Neste sentido, a classe `EARequisitManagerVisitor` efetua a análise dos seguintes elementos de um código fonte: pacotes declarados, anotações, declaração de tipos e declaração de métodos.

O método `visit(ImportDeclaration)` é responsável por analisar a declaração de um elemento `import` no código fonte. Este método armazena a linha em que a declaração do `import` ocorre.

Por sua vez, a análise de anotações de um componente Java ocorre através do método `visit(NormalAnnotation)`. Neste método é verificado se existem anotações do tipo `Requisitos` na declaração de uma classe ou método. A partir desta análise, é guardado o valor armazenado na anotação, que identificam quais são os requisitos de um projeto que um método ou classe atende.

Já os métodos `visit(TypeDeclaration)` e `visit(MethodDeclaration)` são responsáveis por identificar qual o nome do elemento da declaração do seu tipo e em qual linha ocorre. Estes valores são guardados em duas listas (cada qual representando as respectivas classes e métodos) que são atributos da classe `EARequisitManagerVisitor`.

3.2.4.2 Pacote popup

O pacote `popup` representa as ações que são disponibilizadas no menu da ferramenta (figura 21). Cada ação implementa uma funcionalidade do *plugin* e é responsável pela sua comunicação com as classes de negócio, detalhadas nos outros tópicos.

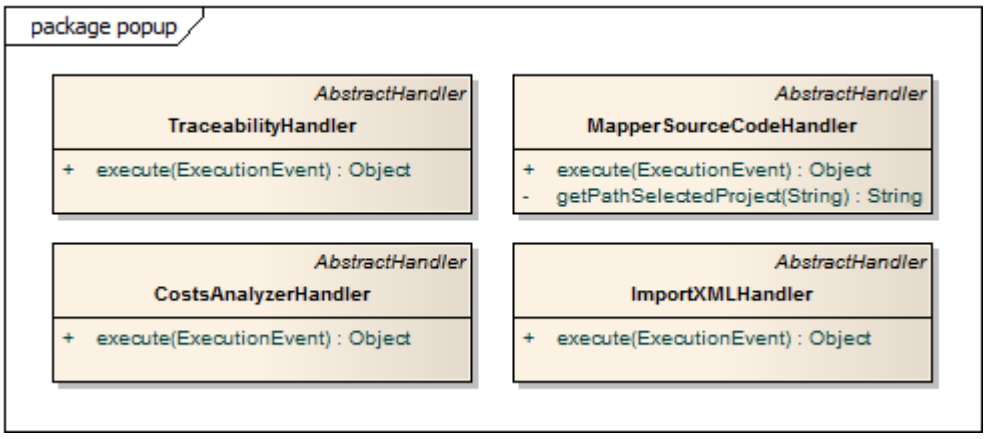


Figura 21 - Diagrama do pacote `popup`

As classes `CostsAnalyzerHandler`, `ImportXMLHandler`, `MapperSourceCodeHandler` e `TraceabilityHandler` são mapeadas no arquivo `plugin.xml`, responsável por orientar a integração das funcionalidades com o *plugin.* Cada classe é mapeada através da *tag* `<extension>`, indicando que a classe será um ponto de extensão para a exibição de determinada funcionalidade.

Cada uma destas classes é responsável pela implementação da interface gráfica da respectiva funcionalidade, apresentando as janelas de comunicação entre o usuário e o *plugin.* Todas as janelas e diálogos apresentados através destas classes utilizam a biblioteca SWT, que é utilizada e padronizada com a IDE Eclipse.

As classes deste pacote estendem a classe `AbstractHandler`, do pacote `org.eclipse.core.commands`, que é a responsável por fornecer a implementação de um comando de menu para o *plugin*. Desta forma, estas classes implementam o método `execute(ExecutionEvent)`, que é chamado quando o usuário seleciona uma opção de menu da ferramenta.

A classe `CostsAnalyzerHandler` é responsável por representar o ponto de entrada do *plugin* para a aplicação da metodologia AHP e geração do documento PDF. É através desta funcionalidade que o usuário poderá visualizar as informações e resultados referentes à importância dos requisitos.

Por sua vez, a classe `ImportXMLHandler` executa a comunicação da interface do usuário com a camada de serviço responsável pela geração do arquivo XML. É a partir desta classe que é iniciado o processo de comunicação da ferramenta com o EA.

Já a classe `MapperSourceCodeHandler` é responsável por iniciar o processo de mapeamento dos códigos fontes do projeto selecionado. Para se identificar o caminho do projeto, é utilizado o método `getPathSelectedProject(String)`, que possui como função buscar a localização completa do projeto selecionado pelo usuário. É também nesta classe que a ferramenta se comunica com o serviço responsável por efetuar a leitura e interpretação do arquivo XML, gerado através do EA.

Por último, `TraceabilityHandler` é a classe é responsável por instanciar a perspectiva que exibe o relacionamento entre códigos fontes e requisitos de um projeto. Esta perspectiva é explicada na seção 3.2.4.2.1.

3.2.4.2.1 Pacote view

O pacote `view` é responsável por conter as classes que apresentam a perspectiva com os relacionamentos entre requisitos, classes e métodos de um projeto (figura 22). Estas classes geram um componente *view* para o usuário, composta por uma árvore de requisitos.

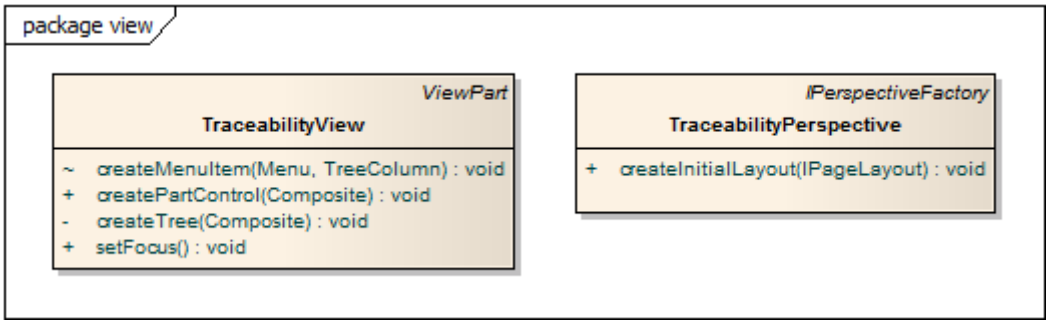


Figura 22 – Diagrama do pacote `view`

A classe `TraceabilityPerspective` é responsável por registrar a perspectiva juntamente com as demais *views* que o Eclipse possui, sendo chamada ao se iniciar a IDE, através do método `createInitialLayout(IPageLayout)`. Esta classe implementa a interface `org.eclipse.ui.IPerspectiveFactory`.

Já a classe `TraceabilityView` estende a classe `org.eclipse.ui.part.ViewPart` e é mapeada no arquivo `plugin.xml`, informando assim, que o *plugin* possui uma perspectiva que deve ser instanciada quando o usuário executa uma ação (comando) de menu. Neste caso, a ação é capturada através da classe `TraceabilityHandler`, que por sua vez, chama esta classe. Esta, por sua vez, é responsável por criar a árvore que exibe os requisitos e se comunicar com a camada de negócios, representada pela classe `TraceabilitySourceService`. A árvore é criada através do método `createTree(Composite)` e `createMenuItem(Menu, TreeColumn)`.

3.2.4.3 Pacote services

O pacote `services` representa os serviços e operacionalidades que o *plugin* executa. É através deste pacote que a interface gráfica passa os parâmetros relacionados à escolha do usuário e que a ferramenta efetuará as funcionalidades selecionadas. Este pacote é composto por quatro classes, cada uma representando determina funcionalidade do *plugin* (figuras 23 e 24).

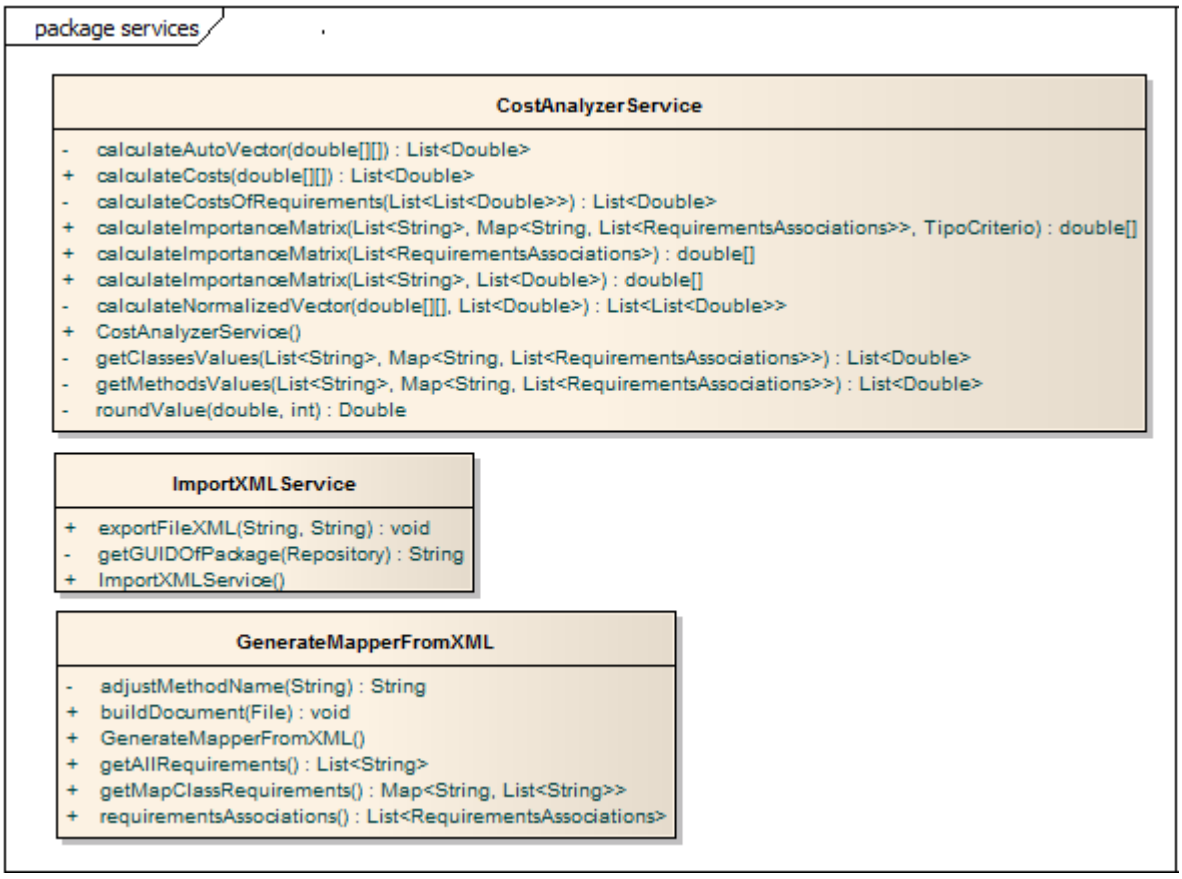


Figura 23 – Diagrama parcial do pacote `services`

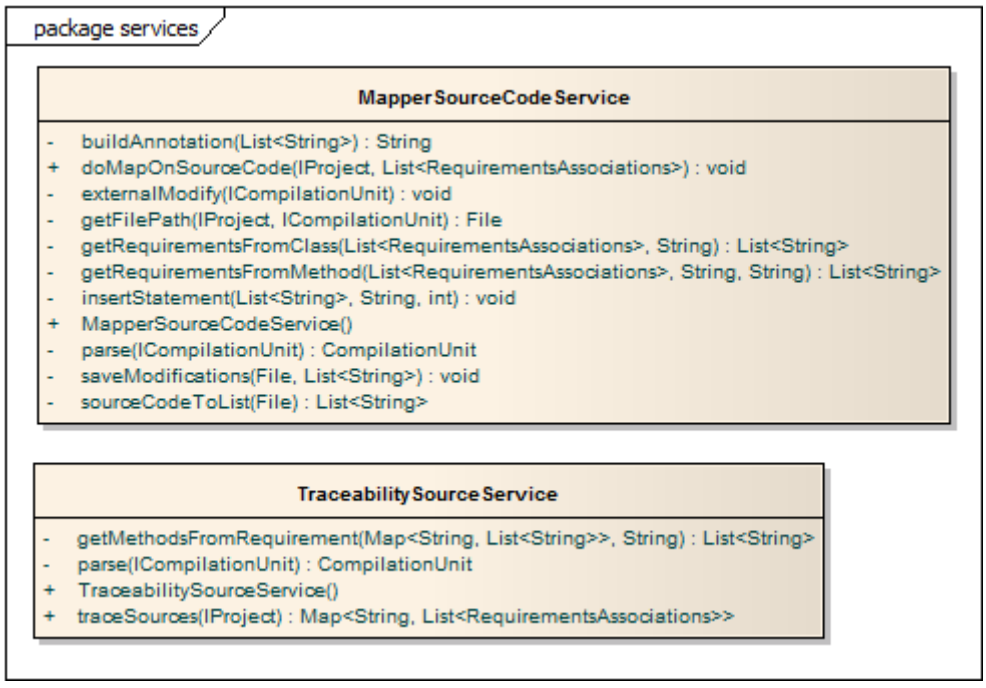


Figura 24 – Diagrama parcial do pacote `services`

A classe `ImportXMLService` é responsável por se comunicar com as bibliotecas disponibilizadas pelo EA, acessando as suas funcionalidades. Nesta classe é gerado um arquivo XML com as informações do projeto selecionado. O método que gera o arquivo é `exportFileXML(String, String)`. Os parâmetros que devem ser passados para este método

são: localização do arquivo no formato Enterprise Architect *Project* (EAP) de um projeto especificado e modelado através do EA e o caminho do arquivo XML que será gerado.

Por sua vez, a classe `GenerateMapperFromXML` é responsável por buscar todas as associações e mapeamentos disponíveis no arquivo XML. Nesta classe são buscados, através do método `requirementsAssociations()`, os relacionamentos entre os requisitos e casos de uso, casos de uso e diagrama de sequência e classes associadas a requisitos. Também é possível buscar todos os requisitos especificados, através do método `getAllRequirements()`.

A classe `MapperSourceCodeService` é responsável por efetuar a inserção de anotações no códigos fontes de um projeto selecionado na árvore *Package Explorer*, da IDE Eclipse. Através do método `doMapOnSourceCode(IProject, List<RequirementsAssociations>)`, são inseridas anotações nas classes e métodos que possuem relacionamentos com requisitos. Este relacionamento é provido pela classe `GenerateMapperFromXML`.

Já a classe `TraceabilitySourceService` é responsável por ler os relacionamentos existentes entre os requisitos, classes e métodos dos códigos fontes. Nesta classe é aplicado o conceito de leitura do código a partir da análise de uma AST. O método responsável por fornecer o mapeamento do código fonte, com a relação de todas as classes e métodos vinculados aos requisitos é o `traceSources(IProject)`.

Por último, a classe `CostAnalyzerService` é responsável por aplicar os conceitos da metodologia AHP, a partir dos dados de relacionamento obtidos entre os requisitos e as classes e métodos rastreados. O método `calculateImportanceMatrix(List<String>, List<RequirementsAssociations>>, TipoCriterio)` possui como objetivo a geração de uma matriz quadrada com a importância de um requisito em relação a outro. Após a obtenção desta informação, é possível efetuar o cálculo de ponderação, através do método `calculateCosts(double[][])`. Este método retorna uma lista de valores percentuais, que representam o tamanho da importância de um requisito, de acordo com um critério anteriormente informado.

3.2.4.4 Pacote `mapper`

O pacote `mapper` é composto pelas classes que efetuam a leitura, comunicação e navegação em um arquivo XML com as informações de um projeto especificado e modelado

através do EA. Neste pacote é utilizada a biblioteca JDOM, identificando-se os relacionamentos presentes no projeto.

A classe `ElementVisitor` (figura 25) é a principal classe de comunicação com o arquivo XML. Nesta classe foram desenvolvidas rotinas para se recuperar as informações dos requisitos, classes, métodos e casos de uso do projeto. Para se utilizar os demais métodos desta classe, é necessário que, após a criação do objeto, seja chamado o método `buildDocument()`, que irá permitir que o arquivo XML seja navegado.

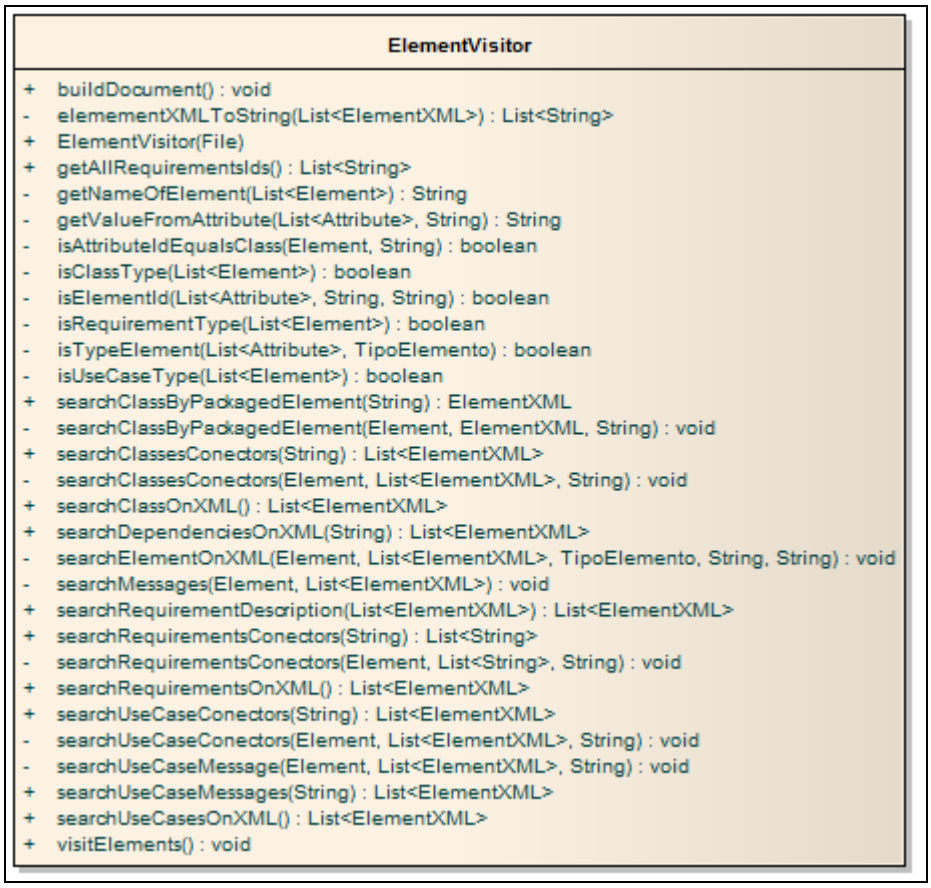


Figura 25 – Classe `ElementVisitor`, pertencente ao pacote `mapper`

Por sua vez, a classe `RequirementAssociations` (figura 26) representa os relacionamentos que existem entre os requisitos e os demais artefatos do projeto. Neste sentido, seguindo este modelo, os requisitos poderão ter as seguintes associações: lista de casos de uso e um mapa que possui como chave uma classe e como valor uma lista de métodos. É possível que existam chaves, sem que existam valores para ela.

É importante observar que os métodos e classes são lidos das seguintes maneiras:

a) relacionamento entre classe e requisitos: são lidas todas as conexões de requisitos em determinada classe;
b) relacionamento entre caso de uso e requisitos: são lidas todas as conexão de requisitos em determinado casos de uso;

c) relacionamento entre casos de uso e diagrama de sequência: são lidos todos os diagramas de sequência vinculados a determinado caso de uso. Assim, obtém-se a relação entre diagrama de sequência e requisitos.

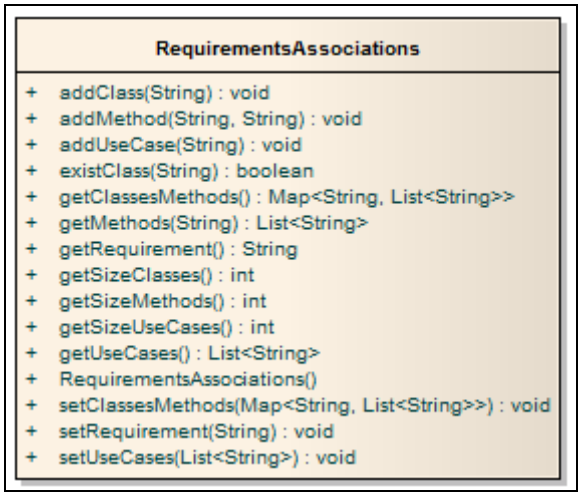


Figura 26 – Classe `RequirementAssociations`, pertencente ao pacote `mapper`

3.2.4.5 Pacote `report`

O pacote `report` é responsável por conter as classes que efetuam a comunicação entre o *plugin* e o gerador de relatórios JasperReports (figura 27). A partir destas classes é que será gerado o arquivo PDF contendo as informações de importância dos requisitos do projeto, de acordo com os critérios disponibilizados.

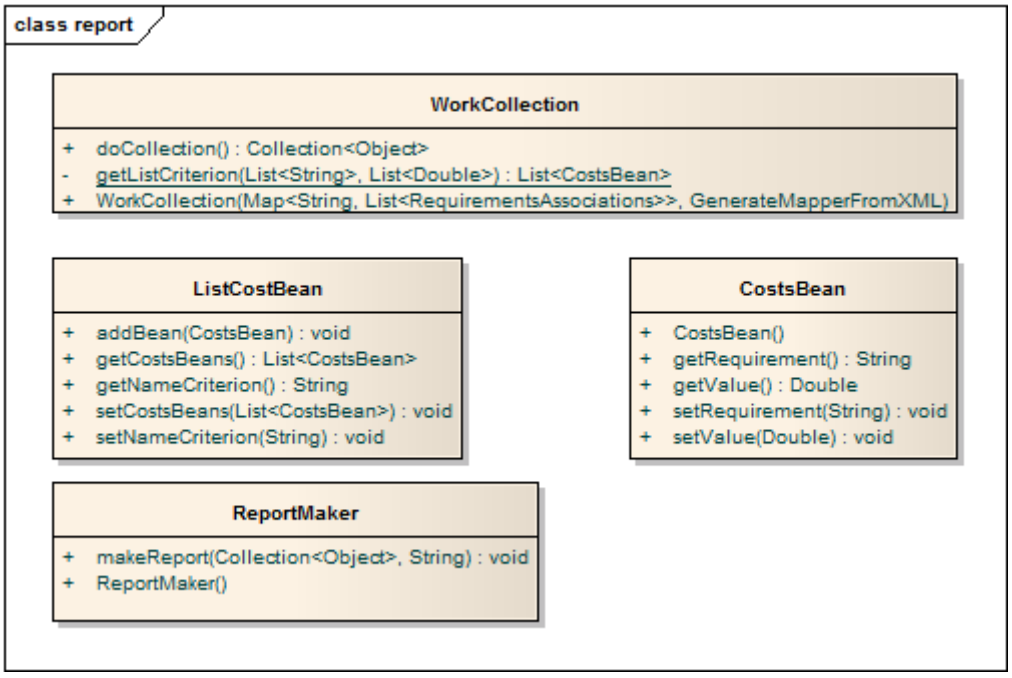


Figura 27 – Diagrama do pacote `report`

A classe `WorkCollection` é responsável por fornecer os dados que serão exibidos no relatório. Nesta classe, através do método `doCollection()`, é gerada uma lista contendo os valores de importância dos requisitos. Os critérios adicionados na lista são: classes, métodos e casos de uso.

Já a classe `ReportMaker` é responsável por efetuar a comunicação com o JasperReports, gerando o arquivo PDF com os dados fornecidos pela classe `WorkCollection`.

Por sua vez, as classes `CostsBean` e `ListCostBean` representam os *beans* que serão utilizados pelo gerador de relatório para a leitura e interpretação dos dados. Os dados que são mostrados, basicamente, são os nomes dos requisitos e seus respectivos percentuais de importância.

3.2.4.6 Pacote `log`

O pacote `log` tem como função permitir ao usuário armazenar as operações realizadas no sistema (figura 28). A classe `Log` permite que o usuário insira as informações de erro e de operações realizadas no sistema, bastando informar a mensagem, o nível da operação e a classe em que o evento ocorre. As informações são armazenadas em um arquivo texto, gravado no `workspace` de projetos, possibilitando que o usuário efetue a audição do funcionamento e operações realizadas pelo *plugin*.

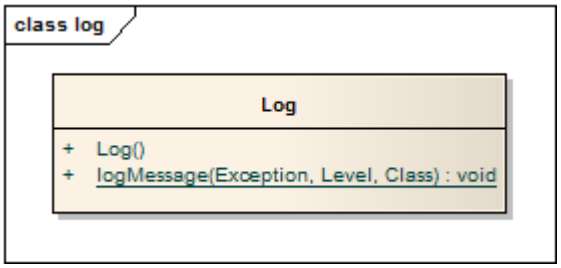


Figura 28 – Diagrama do pacote `log`

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

O EA Requisite Manager foi desenvolvido na linguagem Java, utilizando o ambiente de desenvolvimento Eclipse (ECLIPSE FOUNDATION, 2011) na sua versão 3.5.

O desenvolvimento do leiaute utilizado para a modelagem e geração do arquivo PDF, com os dados de importância e ponderação dos requisitos, foi realizado com o auxílio do IReport (JASPERFORGE, 2011), na versão 3.6.2.

3.3.1.1 Código desenvolvido

O desenvolvimento da ferramenta iniciou com a definição de como funcionaria a interface de integração da ferramenta com a IDE Eclipse. Desta maneira, o primeiro passo para a construção do *plugin* foi a criação de um projeto específico, buscando-se especificar as ações de menu envolvidas e as *views* da IDE que seriam utilizadas. Esta definição ocorreu através da adição de funcionalidades ao arquivo XML de configuração do *plugin*: o arquivo `plugin.xml`.

Foram definidas quatro extensões para a ferramenta (quadro 20), representadas no arquivo com a *tag* `<extension>`, onde o atributo indicando o tipo de extensão foi representado com valor `point="org.eclipse.ui.commands"`. Para cada extensão, que representava uma funcionalidade da ferramenta, adicionou-se uma sub-*tag* `<command>`, indicando desta maneira, a classe de interface com o usuário que seria chamado quando uma ação de menu fosse executada.

```
 <extension     -> extensão para a funcionalidade "Importar XML EA"
       point="org.eclipse.ui.commands">
    <command
          defaultHandler="org.eclipse.ea.requisit.manager.popup.ImportXMLHandler"
          id="org.eclipse.ea.requisit.manager.importXML"
          name="ImportXMLHandler">
    </command>
</extension>

<extension   -> extensão para a funcionalidade "Visualizar custos de alteração"
       point="org.eclipse.ui.commands">
    <command
          defaultHandler="org.eclipse.ea.requisit.manager.popup.CostsAnalyzerHandler"
          id="org.eclipse.ea.requisit.manager.costsAnalyzer"
          name="CostsAnalyzer">
    </command>
</extension>

<extension  ->extensão para a funcionalidade "Mapear códigos fontes"
       point="org.eclipse.ui.commands">
    <command
          defaultHandler="org.eclipse.ea.requisit.manager.popup.MapperSourceCodeHandler"
          id="org.eclipse.ea.requisit.manager.mapperSourceCode"
          name="MapperSourceCode">
    </command>
</extension>

<extension   -> extensão para a funcionalidade "Requisitos X Códigos Fontes"
       point="org.eclipse.ui.commands">
    <command
          defaultHandler="org.eclipse.ea.requisit.manager.popup.TraceabilityHandler"
          id="org.eclipse.ea.requisit.manager.traceability"
          name="Traceability">
    </command>
</extension>
```

Quadro 20 – Definição dos pontos de extensão da ferramenta

Com a definição dos pontos de extensão e quais classes que implementariam cada ação, foram definidas a localização das chamadas das funcionalidades e os comandos de interface que o usuário visualizaria. Assim, optou-se por disponibilizar a ferramenta para a *view* `PackageExplorer`. Para cada extensão, foi adicionado um menu, através de uma *tag* `<menu>`. Foi definido que o menu ficaria visível somente quando o usuário selecionar um projeto Java na árvore de projetos da IDE, clicando através do botão direito. Junto ao menu, foi definido um comando, representado pela *tag* `<command>`, onde foi definido o nome da funcionalidade que o usuário visualizaria e a vinculação com o ponto de extensão. O quadro 21 apresenta a implementação da localização do menu e a implementação de um comando.

```
<extension point="org.eclipse.ui.menus">  -> definição da localização da extensão
  <menuContribution locationURI="popup:org.eclipse.jdt.ui.PackageExplorer">

      <menu  -> criação do menu
            id="org.eclipse.ea.requisit.manager"
            label="EA Requisit Manager"
            icon="icon/manager.png">
            <visibleWhen> -> criação de restrição para o aparecimento do
               <with        menu
                     variable="activeMenuSelection">
                  <iterate>
                     <adapt
                           type="org.eclipse.jdt.core.IJavaProject">
                     </adapt></iterate>
               </with>
            </visibleWhen>
         <command  -> definição do comando e vinculação com o ponto de extensão
               commandId="org.eclipse.ea.requisit.manager.costsAnalyzer"
               label="Visualizar custos de alterações..."
               style="push"
               icon="icon/cost.png">
               <visibleWhen>
               <with
                     variable="activeMenuSelection">
                  <iterate>
                     <adapt
                           type="org.eclipse.jdt.core.IJavaProject">
                     </adapt></iterate>
               </with>
            </visibleWhen>
         </command>
```

Quadro 21 – Definição da localização e comandos da ferramenta

Após a definição da operacionalidade da ferramenta, iniciou-se o desenvolvimento das classes que implementam as funcionalidades do *plugin.*

A primeira ação implementada foi a importação de um arquivo XML com os dados de um projeto modelado e especificado na ferramenta CASE EA, representado no menu da ferramenta pelo sub-menu: `Importar XML EA`.

Para a implementação do comando, foi criada a classe `ImportXMLHandler`, que estende da classe `org.eclipse.core.commands.AbstractHandler`. Com a criação da classe, foi implementado o método `execute(ExecutionEvent)`, que é responsável pela execução da ação de menu selecionada pelo usuário (quadro 22).

```
public class ImportXMLHandler extends AbstractHandler {

	private Text textArquivoEAP;
	private Text textDestinoXML;

	/**
	 *
	 * Este método é responsável por exibir uma dialog para o usuário
	 * informar o caminho do arquivo .EAP que será analisado e o arquivo
	 * .XML que será resultante do processo de importação
	 *
	 * @author José Alberto Zimermann
	 * @since 07/04/2011
	 */
	@Override
	 public Object execute(ExecutionEvent e) throws ExecutionException {
```

Quadro 22 – Método `execute` da classe `ImportXMLHandler`

Este método, por sua vez, é responsável por efetuar a comunicação com o pacote de serviços da ferramenta, que tem como função a implementação efetiva da funcionalidade. Neste caso, a classe `ImportXMLService` foi instanciada para poder efetuar a comunicação com as bibliotecas responsáveis pela exportação do arquivo XML. O quadro 23 exibe a codificação da chamada da implementação do serviço da interface. O método que implementa a ação é o `importFileXML(String, String)`.

```
// botão de importação
Button btn = new Button(shell, SWT.PUSH);
btn.setText("Importar");
btn.addSelectionListener(new SelectionAdapter() {

	// ação de importar foi selecionada
	public void widgetSelected(SelectionEvent event) {

		//busca os caminhos dos arquivos apontados
		String pathProjetoEAP = textArquivoEAP.getText();
		String pathDestino = textDestinoXML.getText();

		// verifica se os caminhos apontados não são nulos ou brancos
		if (StringUtils.isValid(pathProjetoEAP) &&
StringUtils.isValid(pathDestino)) {

		// esconde a tela
		shell.setVisible(false);

		// chama o serviço de exportação
		ImportXMLService importService = new ImportXMLService();

		try {

		// avisa ao usuário que a importação iniciou
		MessageDialog.openInformation(shell, "EA Requisite Manager",
		"O processo de importação foi iniciado e pode demorar alguns
minutos!");

		// efetua a importação
		importService.importFileXML(pathProjetoEAP, pathDestino);
```

Quadro 23 – Chamada do método responsável pelo serviço de importação de XML

A geração do arquivo XML é possível através da invocação do método `ExportPackageXMI`, disponível na classe `org.sparx.Project`. Esta classe e as demais utilizadas para efetuar a comunicação com a ferramenta CASE EA são fornecidas pela biblioteca `eaapi.jar`.

É válido observar que a obtenção do arquivo XML se dá pela identificação do `guid` de um projeto, que é localizado neste caso, no elemento raiz do repositório. Esta identificação é feita a partir do objeto `Repository` de um determinado arquivo EAP. Pode-se observar o processo de identificação realizado através do quadro 24.

```
private String getGUIDOfPackage(Repository repository) throws Exception {

	Collection<Package> packages = repository.GetModels();

	String guid = "";

	//o primeiro pacote do projeto é o pacote raiz, com os dados de todo
o projeto

	if(packages != null && packages.GetCount() > 0){
		guid =  packages.GetAt((short) 0).GetPackageGUID();
	}

	return guid;
}
```

Quadro 24 – Identificação do `guid` de um repositório

Para a implementação da funcionalidade de mapeamentos dos códigos fontes, foi implementada a classe no pacote `view`, chamada de `MapperSourceCodeService`. Nesta classe, o método `execute(ExecutionEvent)` é responsável por duas fases:

a) primeira fase: buscar o mapeamento e relacionamentos existentes no arquivo XML;
b) segunda fase: efetuar o mapeamento nos códigos fontes, de acordo com o relacionamento buscado.

Na primeira fase, para o desenvolvimento da leitura dos relacionamentos providos pelo arquivo, foi utilizada a biblioteca `jdom-1.0.jar`, que permitiu o acesso ao arquivo XML, leitura e navegação nos elementos que compunham a estrutura. Para efetuar o acesso aos elementos, foram implementados os trechos de códigos apresentados no quadro 25. A classe `GenerateMapperFromXML` é a classe responsável por efetuar a leitura dos mapeamentos disponíveis no arquivo XML. A classe `ElementVisitor` é a classe de comunicação com o arquivo XML.

```
// efetua o mapeamento
GenerateMapperFromXML generateMapper = new GenerateMapperFromXML();
generateMapper.buildDocument(new File(pathXML));

// mapeamentos da classes, métodos e casos de uso
List<RequirementsAssociations> associations =
generateMapper.requirementsAssociations();
```

Quadro 25 – Busca das associações de requisitos no arquivo XML

O processo de leitura do mapeamento foi definido e implementado da seguinte maneira:

a) obter todos os requisitos especificados em um projeto: a identificação de elementos do tipo requisito se deu considerando todos os elementos da *tag* `<element>` em que um dos seus atributos possuíssem o valor `uml:Requirement`. O quadro 26 apresenta a chamada do método `searchRequirementsOnXML` da classe `ElementVisitor`;

```
public List<ElementXML> searchRequirementsOnXML() throws JDOMException,
                  IOException {

      // lista de classes do arquivo
      List<ElementXML> requirements = new ArrayList<ElementXML>();

      // Busca as classes a partir do nó raiz
      searchElementOnXML(root, requirements, TipoElemento.Requirement,
null, null);

      return requirements;
}
```

Quadro 26 – Busca de requisitos de um projeto

Por sua vez, o método `searchRequirementsOnXML` efetua a busca dos requisitos com o auxílio do método `searchElementOnXML`, que tem como função buscar os elementos no arquivo XML, com base no parâmetro `TipoElemento`. Neste caso, serão buscados os elementos do tipo `Requirement`. No quadro 27, pode-se verificar a implementação do método `searchElementOnXML`.

```
private void searchElementOnXML(Element root, List<ElementXML> elementos,
                  TipoElemento tipo, String id, String searchFor) throws
JDOMException, IOException {

  // Recuperamos os elementos filhos (children)
  List<Element> elements = root.getChildren();
  Iterator<Element> i = elements.iterator();

  // Iteramos com os elementos filhos, e filhos do dos filhos
  while (i.hasNext()) {
    Element e = i.next();
    searchElementOnXML(e, elementos, tipo, id, searchFor);

    if (element.getName().equals("element")) {

      List<Attribute> atributos = (List<Attribute>) e.getAttributes();

      if (isTypeElement(atributos, tipo) && isElementId(atributos, id,
searchFor)) {

        ElementXML e = new ElementXML();

        for (Attribute attribute : atributos) {

          if (attribute.getName().equals("id")) {
            e.setId(attribute.getValue());
          }

          if (attribute.getName().equals("idref")) {
            e.setId(attribute.getValue());
          }

          if (attribute.getName().equals("name")) {
            e.setDescription(attribute.getValue());
          }
        }

        // para os requisitos, procura-se atributo "alias"
        // que fica dentro da tag <properties> do <element>
        if(tipo == TipoElemento.Requirement) {

          String alias = searchAliasFromRequirement(element);

          if(StringUtils.isValid(alias)) {
            e.setAlias(alias);
          }
        }

        elementos.add(e);
      }
    }
  }
}
```

Quadro 27 – Implementação da obtenção elementos de um projeto

b) buscar as classes associadas diretamente com os requisitos: este processo buscou inicialmente, todas as classes especificadas em um diagrama de classes, no projeto. O método `searchClassOnXML` foi encarregado de efetuar a busca, conforme é possível observar no quadro 28. Neste método, é chamado o método

searchElementOnXML, exibido no quadro 27, passando como parâmetro o tipo TipoElemento.Class;

```
public List<ElementXML> searchClassOnXML() throws JDOMException,
                  IOException {

      // lista de classes do arquivo
      List<ElementXML> classes = new ArrayList<ElementXML>();

      // Busca as classes a partir do nó raiz
      searchElementOnXML(root, classes, TipoElemento.Class, null, null);

      return classes;
}
```

Quadro 28 – Busca de classes de um projeto

Após, foram buscados todos os requisitos vinculados às classes obtidas, através do método searchRequirementsConectors. Este método foi responsável por buscar todos os elementos do tipo <connector>, onde o atributo idref fosse igual ao id da classe procurada. Assim, foi possível obter os requisitos *linkados*. Os quadros 29 e 30 demonstram os métodos implementados para efetuar esta busca.

```
public List<String> searchRequirementsConectors(String idClasse) {

      List<String> conectores = new ArrayList<String>();
      searchRequirementsConectors(root, conectores, idClasse);
      return conectores;
}
```

Quadro 29 – Busca de requisitos vinculados a uma classe

```
private void searchRequirementsConectors(Element root, List<String>
conectores, String idClasse) {

	// Recuperamos os elementos filhos (children)
	List<Element> elements = root.getChildren();
	Iterator<Element> it = elements.iterator();

	// Iteramos com os elementos filhos, e filhos do dos filhos
	while (it.hasNext()) {
		Element element = it.next();
		searchRequirementsConectors(element, conectores, idClasse);

		if (element.getName().equals("connector")) {

			// busca os filhos do conector
			List<Element> filhos = element.getChildren();

			for (int i = 0; i < filhos.size(); i++) {

				if (i < filhos.size() - 1) {

					Element source = filhos.get(i);
					Element target = filhos.get(i + 1);

				   if (source.getName().equals("source")
				      && isAttributeIdEqualsClass(source, idClasse)
				      && isRequirementType(target.getChildren())) {

				     conectores.add(getNameOfElement(target.getChildren()));
				     break;
					}
				    }
			}
		}
	}
}
```

Quadro 30 – Método auxiliar para a busca de requisitos vinculados a uma classe

c) buscar todos os casos de uso vinculados aos requisitos: a partir da `id` de um requisito, foram buscados todos os elementos que possuíssem a *tag* `<connector>`, onde os elementos subordinados desta *tag* fossem: `<source>` e `<target>`. Também foi verificado se um dos atributos do elemento `<target>` era do tipo `UseCase`. O quadro 31 apresenta a verificação feita para a identificação de um elemento do tipo caso de uso. Os casos de usos foram obtidos com auxílio do método `searchUseCaseConectors`, da classe `ElementVisitor`;

```
private boolean isUseCaseType(List<Element> children) {

	for (Element element : children) {

		if (element.getName().equals("model")) {

			List<Attribute> atributos = element.getAttributes();

			for (Attribute attribute : atributos) {

				if (attribute.getName().equals("type")
				&& attribute.getValue().equals("UseCase")) {
					return true;
				}
			}

		}
	}

	return false;
}
```

Quadro 31 – Teste para identificação de elemento para casos de uso

d) buscar as associações entre casos de uso e diagramas de sequência: para se estabelecer estas relações, foram buscados todos os elementos do tipo `<packagedElement>`, onde o `id` daquele elemento fosse igual ao casos de uso procurado. Dentro deste elemento `<packagedElement>`, foram buscados todos os elementos `<message>`, que representavam as mensagens trocadas entre as entidades envolvidas no diagrama de sequência. A partir desta identificação, foi possível obter o nome da classe e método envolvido em cada operação do diagrama. Nos quadros 32 e 33 é possível verificar o código desenvolvido para esta busca. Os métodos responsáveis por efetuar estas operações são `searchUseCaseMessages` e `searchClassByPackagedElement`, da classe `ElementVisitor`.

```
private void searchUseCaseMessage(Element root, List<ElementXML> messages,
String idUseCase) {

  // Recuperamos os elementos filhos (children)
  List<Element> elements = root.getChildren();
  Iterator<Element> it = elements.iterator();

  // Iteramos com os elementos filhos, e filhos do dos filhos
  while (it.hasNext()) {

    Element e = it.next();
    searchUseCaseMessage(e, messages, idUseCase);

    // verifica se o element é um
    if (element.getName().equals("packagedElement")) {

     String idElement = getValueFromAttribute(e.getAttributes(), "id");

     if(idElement.equals(idUseCase)){

        // busca os filhos do conector
        List<Element> filhos = element.getChildren();

        for (int i = 0; i < filhos.size(); i++) {

          // buscar messages a partir do nó package
          searchMessages(filhos.get(i), messages);
        }

      }
   }
 }
}
```

Quadro 32 – Método responsável pela identificação das mensagens de um diagrama de sequência vinculado a um caso de uso

```
private void searchMessages(Element root, List<ElementXML> messages) {

  // Recuperamos os elementos filhos (children)
  List<Element> elements = root.getChildren();
  Iterator<Element> it = elements.iterator();

  // Iteramos com os elementos filhos, e filhos do dos filhos
  while (it.hasNext()) {

    Element element = it.next();
    searchMessages(element, messages);

    // verifica se o element é um
    if (element.getName().equals("message")) {

      List<Attribute> messageAttributes = element.getAttributes();

      String name = this.getValueFromAttribute(messageAttributes, "name");

      String signature = this.getValueFromAttribute(messageAttributes,
"signature");

       if(StringUtils.isValid(name) && StringUtils.isValid(signature)){

          ElementXML elementXML = new ElementXML();
          elementXML.setDescription(name);
          elementXML.setId(signature);
          messages.add(elementXML);
       }
    }
  }
}
```

Quadro 33 – Método auxiliar utilizado para identificação de mensagens de um diagrama de sequência

Finalmente, após a apresentação de todos os passos necessários para buscar os relacionamentos entre os requisitos, classes e métodos de um projeto, o método responsável por efetuar chamadas é apresentado no quadro 34. O método foi implementado na classe `GenerateMapperFromXML`.

```
try {

  List<ElementXML> requirements = v.searchRequirementsOnXML();
  Map<String, List<String>> mcr = getMapClassRequirements();

  for (ElementXML elementXML : requirements) {

    // associações de um caso de uso
    RequirementsAssociations ra = new RequirementsAssociations();

    String name = "";
    if(StringUtils.isValid(elementXML.getAlias())){
      name = elementXML.getAlias();
    } else {
      name = elementXML.getDescription();
    }

    ra.setRequirement(name);

    // busca todos os casos de uso associados ao requisito
    List<ElementXML> useCases =
v.searchUseCaseConectors(elementXML.getId());

    // busca relacionamento: caso de uso x diagrama de sequência
    for (ElementXML useCase : useCases) {

     ra.addUseCase(useCase.getDescription());

     List<ElementXML> messages = v.searchUseCaseMessages(useCase.getId());

       for (ElementXML message : messages) {
         ElementXML elementClass =
v.searchClassByPackagedElement(message.getId());
         String className = elementClass.getDescription();
         requirementsAssociations.addClass(className);
         requirementsAssociations.addMethod(className,
adjustMethodName(message.getDescription()));
       }
    }

    // busca todas as classes associadas com requisitos
    Set<Entry<String, List<String>>> entrySet = mcr.entrySet();
    Iterator<Entry<String, List<String>>> it = entrySet.iterator();

    while(it.hasNext()){

      Entry<String, List<String>> entry = it.next();
      List<String> listRequirement = entry.getValue();

      for (String requirement : listRequirement) {
        if(requirement.equals(elementXML.getDescription())){
          requirementsAssociations.addClass(entry.getKey());
          break;
        }
      }
   }
   associations.add(requirementsAssociations);
}
```

Quadro 34 – Apresentação do método `requirementsAssociations()`

Após efetuar a busca dos relacionamentos providos pelo arquivo XML, a segunda fase da implementação da funcionalidade demandou a utilização da biblioteca `org.eclipse.jdt.core_3.5.2.jar`, que pode ser encontrada junto à instalação da IDE Eclipse. Nesta biblioteca, foram utilizadas as classes de *parser* disponíveis, que permitiram o acesso a uma AST, contendo todos os elementos que compõem a formação e estrutura de um código fonte implementado na linguagem Java.

Identificar cada elemento da estrutura da linguagem Java, bem como a sua sintaxe, foi necessária para poder se saber em qual região do código fonte poderia se inserir as anotações, sem que isso causasse erros de compilação ou removesse o código desenvolvido.

A AST disponibilizada nesta biblioteca permitiu a identificação do nome dos nós componentes, dos seus respectivos valores e a linha em que estava declarado ou implementado. No quadro 35 é apresentado um trecho de código onde é verificado em qual linha está declarado um elemento `import`.

```
public boolean visit(ImportDeclaration node) {

  try {

    if (node.getName().getFullyQualifiedName().equals(
       "org.eclipse.ea.requisit.manager.utils.Requisitos")) {
            this.imports.add(node);
    }

    this.lastDeclaredPackage = parser.getLineNumber(node.
getStartPosition());

    } catch (Exception e) {
      Log.logMessage(e, Level.Erro, getClass());
    }

    return super.visit(node);
}
```

Quadro 35 – Identificação da localização de um elemento no código fonte

Neste sentido, a classe `EARequisitManagerVisitor`, que estende de `org.eclipse.jdt.core.dom.ASTVisitior`, permitiu a identificação dos elementos: anotação, declaração de `import`, declaração de tipo e declaração de método. É importante salientar que, se a classe `ASTVisitor` permite sobrescrever outros métodos, onde é possível efetuar a identificação de outras partes de elementos do código fonte Java.

Após a identificação dos relacionamentos para as classes e métodos e onde deveriam ser inseridas as anotações dentro do código fonte Java, foram realizados os seguintes passos para a inserção da anotação e declaração de `import` da anotação:

a) o arquivo `.java`, onde a classe ou método estava declarado, foi aberto utilizando-se a classe `java.io.file.File`;

b) o texto do arquivo foi transferido para um objeto `List<String>`, onde cada elemento da lista representava uma linha do arquivo. A leitura foi realizada com o auxílio da classe `java.io.BufferedReader`;

c) para cada método foi adicionado uma anotação, contendo todos os requisitos relacionados, na linha acima da sua declaração;

d) adicionada uma anotação para a classe, contendo todos os requisitos relacionados, na linha acima onde esta é declarada;

e) adicionada a declaração do `import` da anotação `Requisitos`;

f) a lista foi transferida para o arquivo `.java` novamente, com o auxílio da classe `java.io.BufferedWriter`;

g) foi enviada uma mensagem para a IDE Eclipse, atualizando e sincronizando os códigos fontes.

No quadro 36 é possível verificar a implementação do método responsável pela inserção de anotações para cada arquivo fonte.

```
List<String> linesSource = sourceCodeToList(file); -> transferência do código fonte para uma lista de String
try {
    // para cada método
    for (NamedLineClass clazz : classesOrInterfacesNames) {

        List<NamedLineMethod> methods = clazz.getMethods();
        Collections.sort(methods, new MethodComparator());
        for (NamedLineMethod method : methods) {
            // requisitos atendidos por um método
            List<String> methodRequirements = getRequirementsFromMethod(association, method.getMethodName(),
                    clazz.getClassName());

            if(methodRequirements.size() > 0) {
                // inserir os requisitos para o método
                String annotation = buildAnnotation(methodRequirements);
                insertStatement(linesSource, annotation, method.getLine() - 1); -> adição de anotação para cada método
            }
        }
        // requisitos atendidos por uma classe
        List<String> classRequirements = getRequirementsFromClass(association, clazz.getClassName());
        if(classRequirements.size() > 0) {

            // inserior os requisitos para a classe
            String annotation = buildAnnotation(classRequirements);
            insertStatement(linesSource, annotation, clazz.getLineDeclaration() - 1); -> adição de anotação para a classe
            // inserir o import da anotação
            String importSta = "import org.eclipse.ea.requisit.manager.utils.Requisitos;";
            insertStatement(linesSource, importSta, visitor.getLastDeclaredPackage()); -> adição de anotação para o import
        }
    }
} catch (Exception e) {
    Log.logMessage(e, Level.SEVERE, getClass());
} finally {
    saveModifications(file, linesSource); -> gravação das alterações
    externalModify(unit);-> sincronização dos fontes para o Eclipse
}
```

Quadro 36 – Processo de inserção de anotações no arquivo fonte Java

Já para o desenvolvimento da funcionalidade de rastreabilidade de requisitos e códigos fontes, foi utilizada a mesma árvore sintática, desenvolvida através da classe

EARequisitManagerVisitor. Para se descobrir os mapeamentos, foram percorridos todos os arquivos do projeto, verificando-se as classes e métodos declarados com as anotações do tipo Requisitos. No quadro 37 é possível verificar a implementação desenvolvida para buscar as anotações relacionadas aos requisitos.

```
public boolean visit(NormalAnnotation node) {

  try {
    if (node.getParent().getClass() == TypeDeclaration.class) {

      if (node.getTypeName().getFullyQualifiedName().equals("Requisitos"))
{

        List<MemberValuePair> values = node.values();

          for (MemberValuePair memberValuePair : values) {
            this.ids = memberValuePair.getValue().toString().split(",");
            this.ids = adjustString();
          }

          this.annotations.add(node);
        }
    }

    if(node.getParent().getClass() == MethodDeclaration.class) {

      MethodDeclaration method = (MethodDeclaration) node.getParent();

      if(method != null &&
         node.getTypeName().getFullyQualifiedName().equals("Requisitos"))
{

         List<MemberValuePair> values = node.values();

         for (MemberValuePair memberValuePair : values) {
            String[] ids = null;
            ids = memberValuePair.getValue().toString().split(",");
            ids = adjustString();
            methodsAnnotations.put(
                  method.getName().getFullyQualifiedName().toString(),
                  Arrays.asList(ids));
         }

         this.annotations.add(node);
      }
    }

  } catch (Exception e) {
    Log.logMessage(e, Level.Erro, getClass());
  }

  return super.visit(node)
}
```

Quadro 37 – Identificação das anotações de métodos e classes

O desenvolvimento da geração do arquivo PDF, contendo os valores de importância dos requisitos foi feito utilizando-se a biblioteca jasperreports-3.6.2. Além desta

biblioteca, para a geração do relatório, o JasperReports exigiu a inclusão de outras bibliotecas ao projeto. São elas: `commons-beanutils-1.8.0.jar`, `commons-collections-3.2.1.jar`, `commons-digester-1.7.jar`, `commons-logging-1.1.jar`, `groovy-all-1.5.5.jar` e `iText-2.1.0.jar`. A modelagem do leiaute do relatório foi feita através do IReport Designer, na versão 3.6.2. O quadro 38 exibe o código desenvolvido para efetuar a comunicação entre a aplicação e o gerador de relatórios.

```
public void makeReport(Collection<Object> beanCollection, String path) {

  InputStream input = null;

  try{

    // criar o objeto do datasource
    JRBeanCollectionDataSource dataSource = new
JRBeanCollectionDataSource(beanCollection);

    // popular map com os parametros do relatorio
    Map<String, Object> parameters = new HashMap<String, Object>();
    parameters.put("SUBREPORT_DIR", "/report/");

    // arquivo jasper
    input = getClass().getResourceAsStream("/report/custos.jasper");

    // chamar o gerador do relatório
    JasperPrint print = JasperFillManager.fillReport(input, parameters,
dataSource);

    // salvar o resultado em um pdf
    JasperExportManager.exportReportToPdfFile(print, path);

    Log.logMessage("Arquivo PDF gerado no diretório: " + path, Level.Info,
getClass());

  } catch(Exception e){
    Log.logMessage(e, Level.Erro, getClass());
  } finally{
    FileUtils.closeInputStream(input);
  }
}
```

Quadro 38 – Geração do arquivo PDF através da aplicação

Por último, o armazenamento dos *logs* da ferramenta foi desenvolvido com o auxílio do log4j (APACHE SOFTWARE FOUNDATION, 2011), utilizando a biblioteca `log4j-1.2.16.jar`, permitindo desta maneira, controlar os fluxos de erros gerados pelo programa, bem como saber os arquivos que foram alterados com a inserção de anotações no código fonte.

3.3.1.2 Definição da anotação `Requisitos`

Com o intuito de possibilitar ao desenvolvedor usuário da ferramenta a correta inserção de anotações no código fonte, foi construído um artefato para ser adicionado no `classpath` dos projetos que utilizarão a ferramenta EA Requisite Manager.

Este artefato, disponibilizado através de um arquivo com extensão `.jar`, contém a definição da interface de uma anotação. Esta anotação, por sua vez, possui um atributo `ids()`, que permite que o usuário informe uma lista de `String`, contendo os requisitos que são atendidos. No quadro 38, pode-se verificar como esta anotação foi implementada.

```
package org.eclipse.ea.requisit.manager.utils;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(value = {ElementType.METHOD, ElementType.CONSTRUCTOR,
ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)

public @interface Requisitos {

	String[] ids();
}
```

Quadro 39 – Definição da anotação `Requisitos`

Ainda é possível observar que o uso da anotação `Requisitos` está limitado através da anotação `Target`. Nesta anotação, restringe-se a marcação de elementos do código fonte Java, como sendo os métodos, métodos construtores de classe e declaração de tipos. Por sua vez, a anotação `Retention` permite a interpretação da anotação `Requisitos` em tempo de execução, através de reflexão.

3.3.1.3 Exemplificação do uso da metodologia AHP com a utilização de critérios

Para demonstrar a aplicação do uso da metodologia AHP com a utilização de critérios, será apresentado e detalhado um caso de teste, que mostrará como são formados os valores percentuais que quantificam a importância dos requisitos em relação ao projeto. Os requisitos deste caso de teste são definidos no quadro 40.

Requisitos
RF.01
RF.02
RF.03
RF.04

Quadro 40 – Requisitos do caso de teste

Ainda neste caso de teste, têm-se a quantidade de artefatos ligados aos requisitos. Estes artefatos são: classes, métodos e casos de uso. Estas ligações podem ser obtidas através da rastreabilidade de informações presentes nos códigos fontes do projeto. No quadro 41, podem-se acompanhar os artefatos encontrados para os requisitos. Através destes artefatos, será possível estabelecer a matriz de importância dos requisitos.

Critério / Requisito	Classes	Métodos	Casos de Uso
RF.01	16	32	4
RF.02	12	40	3
RF.03	37	25	3
RF.04	5	2	1

Quadro 41 – Matriz de requisitos e critérios

Assim, para se calcular a importância de um requisito sobre outro, adotou-se a seguinte sistemática:

a) para um requisito da coluna `i` igual ao requisito da linha `j`, o valor desta célula da matriz recebe 1;
b) para o preenchimento de uma coluna `i` e linha `j` em que haja um correspondente preenchido na coluna `j` e linha `i` (célula inversa), então o valor deve ser `1/coluna[j]linha[i]`;
c) para os demais requisitos da coluna `i` e linha `j`, o valor de importância da célula é obtido através da divisão do valor correspondente ao requisito da coluna `i` sobre o valor correspondente ao requisito da coluna `j`. Caso este valor seja maior que 9, este é arredondado para 9;

A formação das matrizes de importância pode ser verificada nos quadros 42, 43 e 44, que apresentam as matrizes de acordo com os critérios acima definidos.

Requisito / Requisito	RF.01	RF.02	RF.03	RF.04
RF.01	1	16 / 12 = 1,33	16 / 37 = 0,43	16 / 5 = 3,2
RF.02	1 / 1,33 = 0,75	1	12 / 37 = 0,32	12 / 5 = 2,4
RF.03	1 / 0,94 = 1,06	1 / 0,70 = 1,43	1	17 / 5 = 3,4
RF.04	1 / 3,2 = 0,31	1 / 2,4 = 0,41	1 / 3,4 = 0,14	1

Quadro 42 – Matriz de importância para o critério classes

Requisito / Requisito	RF.01	RF.02	RF.03	RF.04
RF.01	1	32 / 40 = 0,8	32 / 25 = 1,28	32 / 2 = 16
RF.02	1 / 0,8 = 1,25	1	40 / 25 = 1,6	40 / 2 = 20
RF.03	1 / 1,28 = 0,78	1 / 1,6 = 0,62	1	25 / 2 = 12,5
RF.04	1 / 16 = 0,06	1 / 20 = 0,05	1 / 12,5 = 0,08	1

Quadro 43 – Matriz de importância para o critério métodos

Requisito / Requisito	RF.01	RF.02	RF.03	RF.04
RF.01	1	4 / 3 = 1,33	4 / 3 = 1,33	4 / 1 = 4
RF.02	1 / 1,33 = 0,75	1	3 / 3 = 1	3 / 1 = 3
RF.03	1 / 1,33 = 0,75	1 / 1 = 1	1	3 / 1 = 3
RF.04	1 / 4 = 0,25	1 / 3 = 0,33	1 / 3 = 0,33	1

Quadro 44 – Matriz de importância para o critério casos de uso

Após a definição das matrizes de importância, normalizam-se os valores em um auto-vetor. Nos quadros 45, 46 e 47, pode-se acompanhar a obtenção destes valores e o processo de calculo, onde cada quadro apresenta a obtenção dos valores para cada critério.

Somatório da coluna do RF.01		
RF.01	1 + 0,75 + 2,31 + 0,31	4,38
Normalização das linhas do RF.01*		
Linha 01		
(1 * 100) / 4,38 = 22,86 / 100		**0,23**
(0,75 * 100) / 4,38 = 17,14 / 100		**0,17**
(2,31 * 100) / 4,38 = 52,86 / 100		**0,53**
(0,31 * 100) / 4,38 = 7,14 / 100		**0,07**
(*) Este processo deve se repetir para todas as demais linhas e requisitos		

Quadro 45 – Processo de obtenção do auto-vetor normalizado para o critério classes

Somatório da coluna do RF.01		
RF.01	1 + 1,25 + 0,78 + 0,06	3,09
Normalização das linhas do RF.01*		
Linha 01		
(1 * 100) / 3,09 = 32,36 / 100		**0,32**
(1,25 * 100) / 3,09 = 40,45 / 100		**0,40**
(0,78 * 100) / 3,09 = 25,24 / 100		**0,25**
(0,06 * 100) / 3,09 = 1,94 / 100		**0,02**
(*) Este processo deve se repetir para todas as demais linhas e requisitos		

Quadro 46 – Processo de obtenção do auto-vetor normalizado para o critério métodos

Somatório da coluna do RF.01		
RF.01	1 + 0,75 + 0,75 + 0,25	2,75
Normalização das linhas do RF.01*		
Linha 01		
(1 * 100) / 2,75 = 36,36 / 100		**0,36**
(0,75 * 100) / 2,75 = 27,27 / 100		**0,27**
(0,75 * 100) / 2,75 = 27,27 / 100		**0,27**
(0,25 * 100) / 2,75 = 9,1 / 100		**0,09**
(*) Este processo deve se repetir para todas as demais linhas e requisitos		

Quadro 47 – Processo de obtenção do auto-vetor normalizado para o critério casos de uso

Com a aplicação do cálculo exemplificado acima para todas as linhas e requisitos definidos, têm-se a formação das matrizes de prioridade para cada um dos critérios, apresentadas nos quadros 48, 49 e 50.

Requisito / Requisito	RF.01	RF.02	RF.03	RF.04	Soma
RF.01	0,23	0,23	0,23	0,23	0,91
RF.02	0,17	0,17	0,17	0,17	0,69
RF.03	0,53	0,53	0,53	0,53	2,11
RF.04	0,07	0,07	0,07	0,07	0,29

Quadro 48 – Matriz de prioridade do critério classes

Requisito / Requisito	RF.01	RF.02	RF.03	RF.04	Soma
RF.01	0,32	0,32	0,32	0,32	1,29
RF.02	0,40	0,40	0,40	0,40	1,62
RF.03	0,25	0,25	0,25	0,25	1,01
RF.04	0,02	0,02	0,02	0,02	0,08

Quadro 49 – Matriz de prioridade do critério métodos

Requisito / Requisito	RF.01	RF.02	RF.03	RF.04	Soma
RF.01	0,36	0,36	0,36	0,36	1,45
RF.02	0,27	0,27	0,27	0,27	1,09
RF.03	0,27	0,27	0,27	0,27	1,09
RF.04	0,10	0,10	0,10	0,10	0,36

Quadro 50 – Matriz de prioridade do critério casos de uso

Por último, os quadros 51, 52 e 53 apresentam os percentuais finais da quantificação dos requisitos, para cada um dos critérios estabelecidos.

Requisito		Valor (%)
RF.01	(0,25 * 1,28) * 100	22,86%
RF.02	(0,25 * 1,6) * 100	17,14%
RF.03	(0,25 * 1) * 100	52,86%
RF.04	(0,25 * 0,8) * 100	7,14%

Quadro 51 – Quantificação final dos valores do requisito para o critério classes

Requisito		Valor (%)
RF.01	(0,25 * 1,29) * 100	32,32%
RF.02	(0,25 * 1,62) * 100	40,40%
RF.03	(0,25 * 1,01) * 100	25,25%
RF.04	(0,25 * 0,08) * 100	2,02%

Quadro 52 – Quantificação final dos valores do requisito para o critério métodos

Requisito		Valor (%)
RF.01	(0,25 * 0,91) * 100	36,36%
RF.02	(0,25 * 0,69) * 100	27,27%
RF.03	(0,25 * 2,11) * 100	27,27%
RF.04	(0,25 * 0,29) * 100	9,09%

Quadro 53 – Quantificação final dos valores do requisito para o critério casos de uso

É possível notar que, de acordo com o critério estabelecido e a quantidade de artefatos que um critério possui associado, a importância dos requisitos varia. Desta maneira, é possível efetuar uma avaliação do projeto sob vários aspectos.

Sob o critério das classes, nota-se que o RF.03 possui 52,86% de importância, uma vez que possui uma grande quantidade de classes relacionadas a ele, comparando-se com os demais requisitos. Por sua vez, de acordo com o critério de métodos, nota-se que o RF.02 possui maior relevância, valendo 40,40% do total do projeto. Por último, ao avaliar-se a importância dos requisitos considerando-se o critério de casos de uso, o RF.01 possui maior importância, ocupando 36,36% de importância no projeto.

3.3.2 Operacionalidade da implementação

A demonstração apresentada nesta seção utilizou o projeto e os códigos fontes da implementação da própria ferramenta. Optou-se por este estudo de caso, levando-se em consideração o projeto bem documentado, com a definição de requisitos, detalhamento dos

casos de uso, criação de diagramas de sequência e relacionamento entre os artefatos do projeto. O projeto foi modelado e especificado através do EA, permitindo a obtenção do arquivo XML. Também justifica-se a utilização dos códigos fontes do *plugin* por terem sido desenvolvidos na linguagem de programação Java.

Após a inicialização da IDE Eclipse, é possível verificar se o *plugin* EA Requisite Manager está instalado, através do menu `Help > About Eclipse Platform > Installation Details`, que exibe uma tela com informações sobre a versão ambiente do Eclipse. Na aba `Plugins` é apresentada uma listagem de todos os plugins instalados na IDE. A figura 29 apresenta a tela exibida.

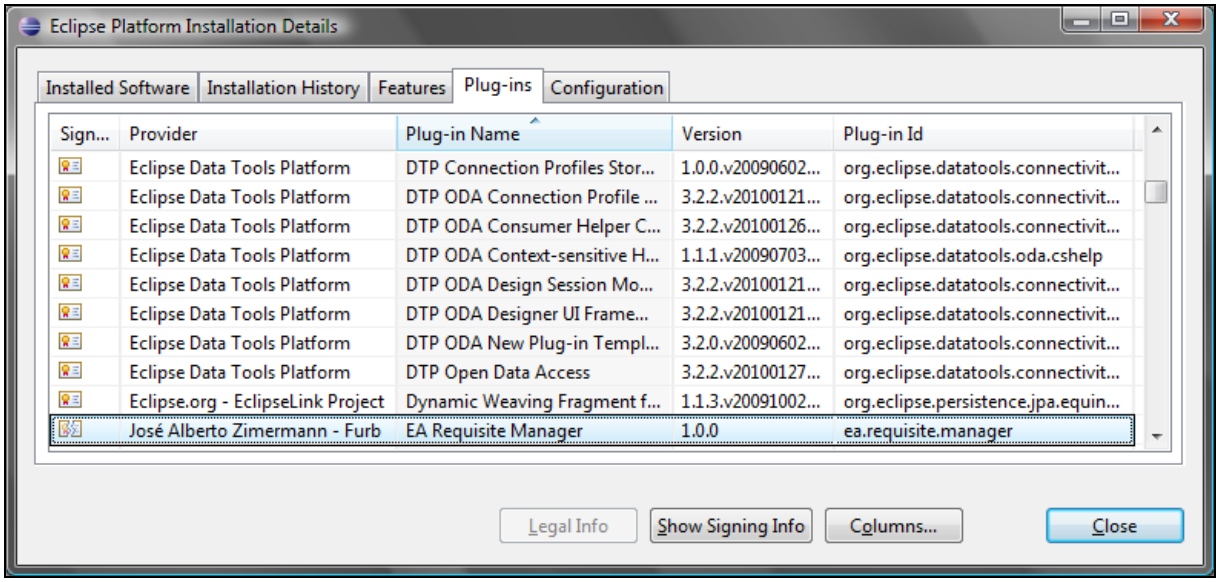


Figura 29 – Tela de detalhes dos *plugins* instalados no Eclipse

O `EA Requisite Manager` é disponibilizado para o usuário através da *view* da IDE Eclipse `Package Explorer`, conforme apresentado na figura 30. Ao clicar com o botão direito em um projeto Java, é disponibilizado um menu do *plugin*, no qual é possível acessar as funcionalidades da ferramenta. Para os demais itens disponibilizados na árvore de projetos da *view*, como arquivos fonte, bibliotecas ou pastas, o menu não é apresentado.

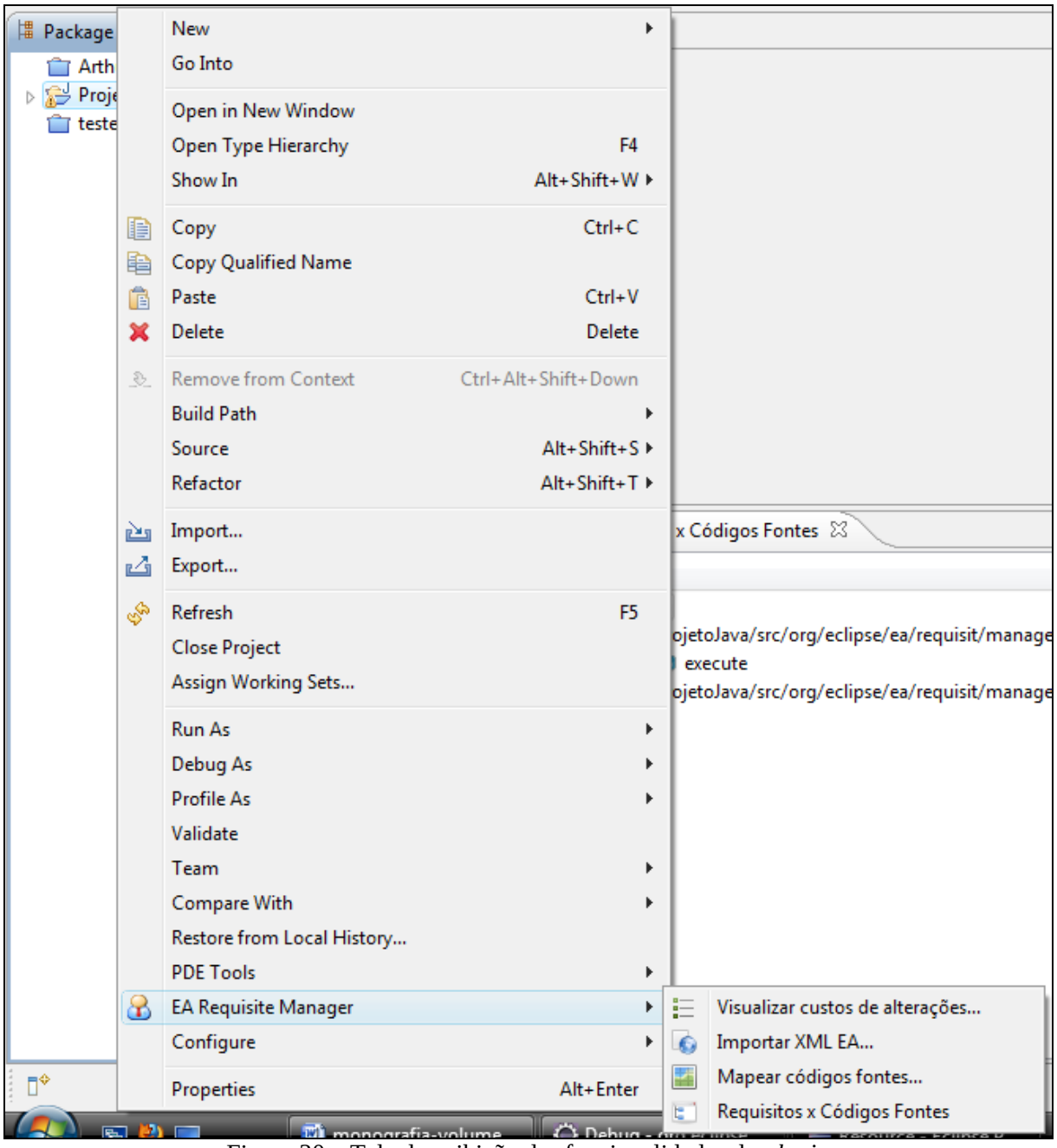


Figura 30 – Tela de exibição das funcionalidades do *plugin*

O usuário pode efetuar a importação de um arquivo XML de um projeto especificado e modelado através do EA utilizando a opção: `Importar XML EA`. Ao selecionar esta opção, será exibida uma janela, apresentada na figura 31, em que o usuário deverá informar o arquivo EAP que contém o repositório do projeto EA e também o destino do arquivo XML que será gerado através da importação.

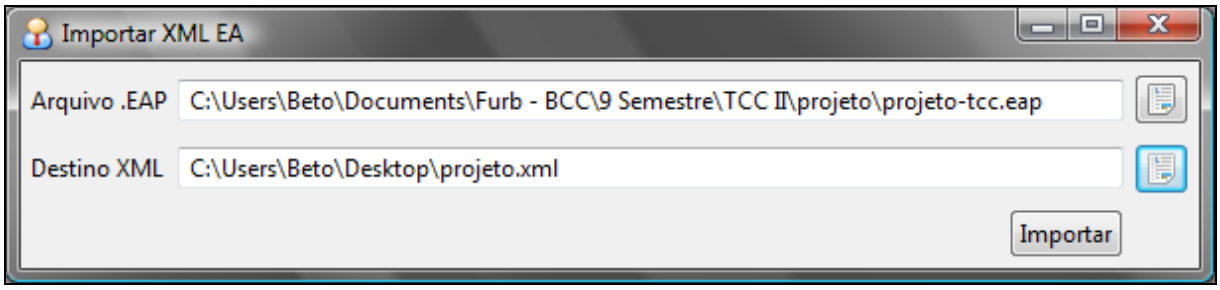


Figura 31 – Janela de configuração dos dados para a importação de XML

Após informar os dados e pressionar o botão `Importar`, o sistema exibirá uma mensagem para o usuário, informando sobre o início do processo de importação. A figura 32 exibe a tela informativa que é exibida.

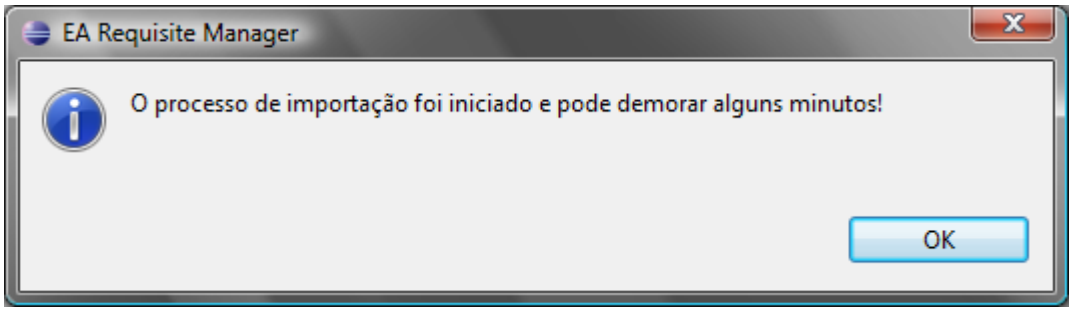


Figura 32 – Mensagem informativa do início do processo de importação

Ao confirmar o início da operação, a ferramenta fará a comunicação do *plugin* com o EA, acessando a funcionalidade de geração de arquivo XML. Assim, será exibida uma tela em que será exibido o progresso do processo de importação. Na figura 33, é exibida a tela de processamento do arquivo.

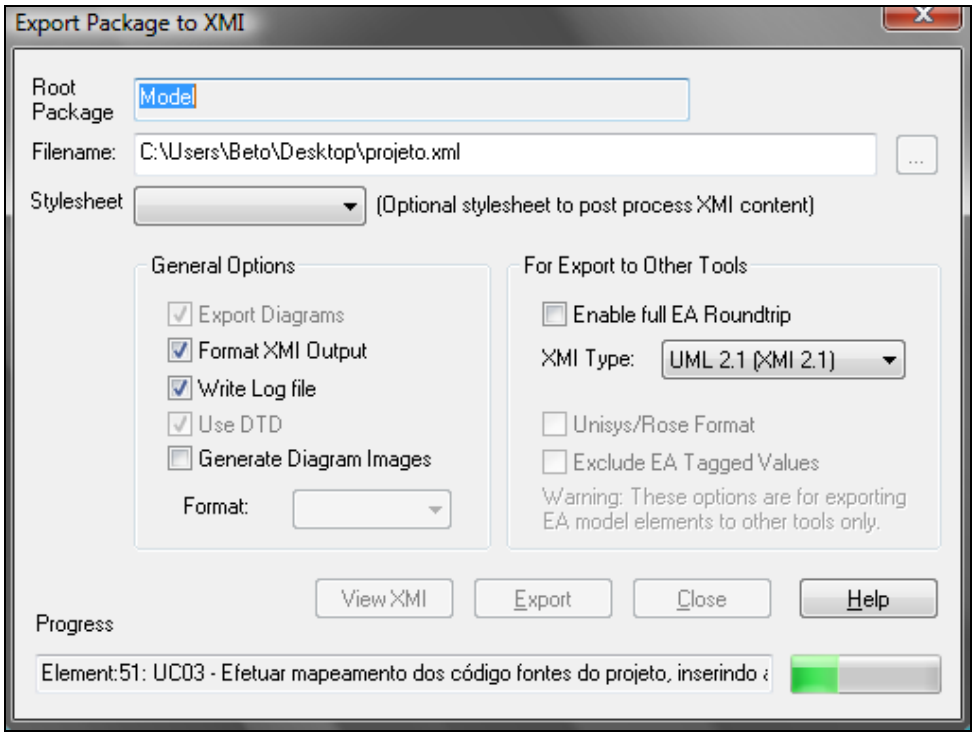


Figura 33 – Tela de progresso da importação do arquivo XML

Ao finalizar o processo, a tela representada através da figura 33 é fechada e é exibida uma mensagem informativa, confirmando a finalização do processo de importação. A figura 34 exibe a mensagem apresentada ao usuário.

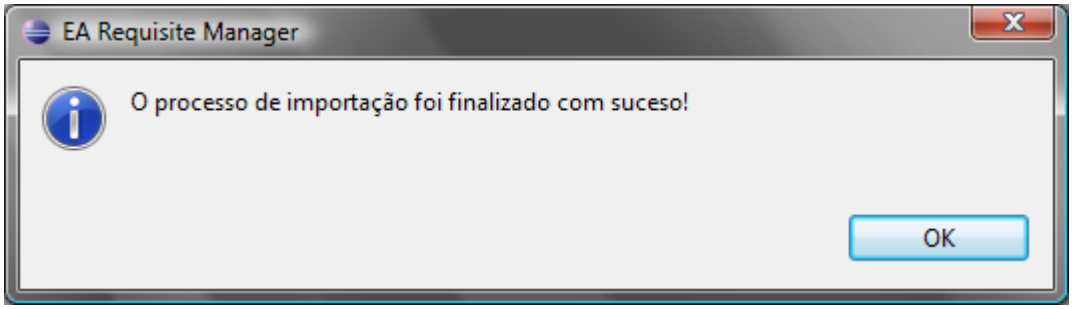


Figura 34 – Tela informativa da finalização do processo de importação

Após a importação do arquivo XML de um projeto do EA, o usuário pode acessar as demais funcionalidades da ferramenta, que demandam a utilização deste arquivo.

Na funcionalidade `Mapear códigos fontes`, a ferramenta irá avaliar um arquivo para inserir anotações nos códigos fontes do projeto, com as informações de requisitos. Desta maneira, ao acessar a funcionalidade, será exibida uma janela onde o usuário deverá informar o arquivo XML que contém as informações de um projeto especificado no EA. Este é o mesmo arquivo gerado através da funcionalidade `Importar XML EA`. Nesta janela também é exibido o caminho do projeto selecionado pelo usuário. Na figura 35, é exibida a tela apresentada ao usuário.

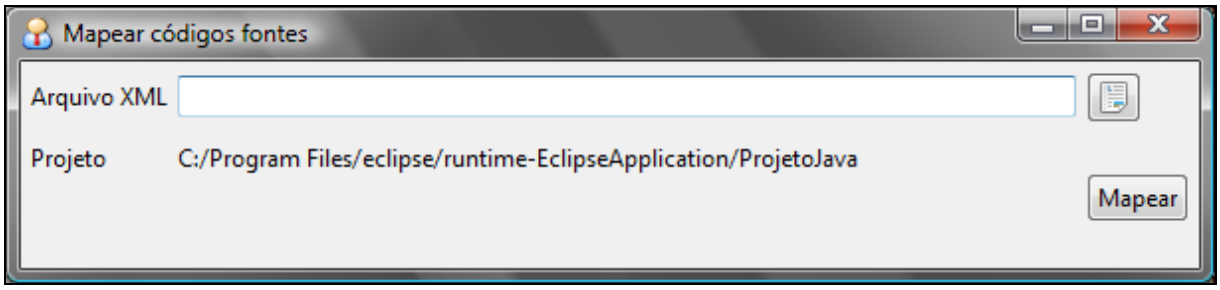


Figura 35 – Janela de configuração da funcionalidade de mapeamento

Ao executar o início do processo de mapeamento do projeto, será exibida uma mensagem informativa, alertando o usuário que a execução da operação poderá demorar alguns minutos, conforme exibido na figura 36.

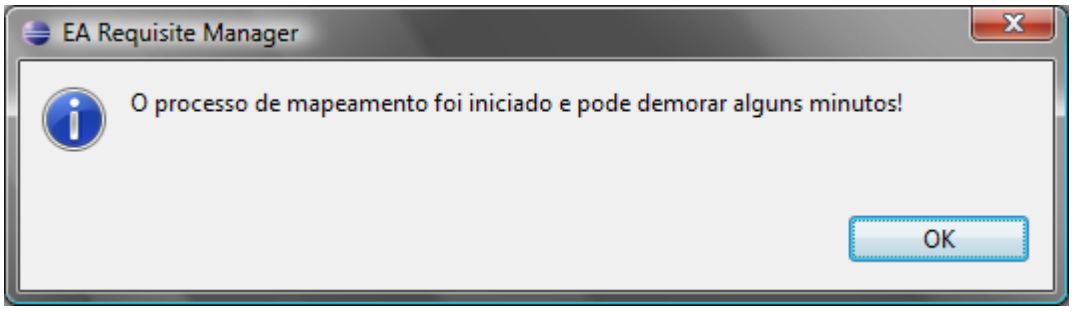


Figura 36 – Mensagem informativa sobre o início do processo de mapeamento

No término da operação, ainda é apresentada uma mensagem informando sobre a finalização do mapeamento dos códigos fontes. A figura 37 apresenta a mensagem exibida ao usuário.

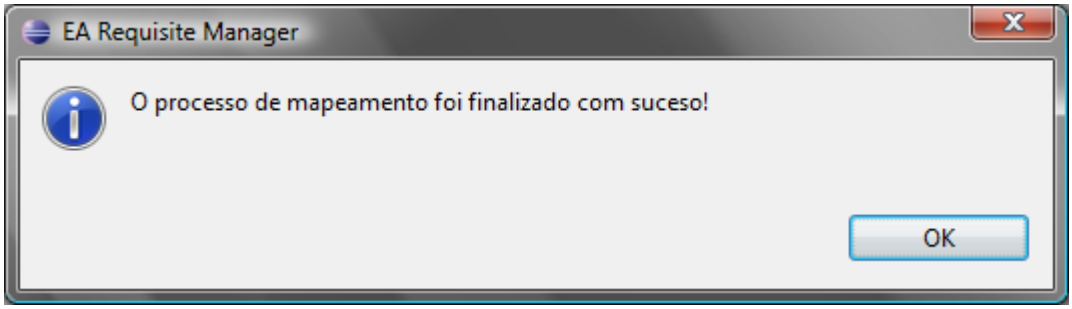


Figura 37 – Tela com a mensagem de finalização do mapeamento

Os códigos fontes, após a execução da funcionalidade, apresentarão anotações, conforme os relacionamentos existentes no projeto Java, onde é possível observar os requisitos que uma classe ou método atende. Na figura 38, é exibida uma classe Java, que possui relacionamento com um requisito.

```
import java.util.Map;
import java.util.Set;
import java.util.logging.Level;

import org.eclipse.core.resources.IProject;
import org.eclipse.ea.requisit.manager.parser.visitor.EARequisitManagerVisitor;
import org.eclipse.ea.requisit.manager.services.mapper.RequirementsAssociations;
import org.eclipse.ea.requisit.manager.utils.log.Log;
import org.eclipse.jdt.core.ICompilationUnit;
import org.eclipse.jdt.core.IPackageFragment;
import org.eclipse.jdt.core.IPackageFragmentRoot;
import org.eclipse.jdt.core.JavaCore;
import org.eclipse.jdt.core.dom.AST;
import org.eclipse.jdt.core.dom.ASTParser;
import org.eclipse.jdt.core.dom.CompilationUnit;
import org.eclipse.ea.requisit.manager.utils.Requisitos; -> import adicionado para a Anotação

@Requisitos(ids={"RF.05", "RF.06", "RF.07", "RF.08"}) -> Anotação na declaração da classe
public class TraceabilitySourceService {

@Requisitos(ids={"RF.05", "RF.06", "RF.07", "RF.08"}) -> Anotação na declaração do método
    public TraceabilitySourceService() {
        super();

    }

@Requisitos(ids={"RF.05", "RF.06", "RF.07", "RF.08"})
    public Map<String, List<RequirementsAssociations>> traceSources(IProject project){
```

Figura 38 – Mapeamento realizado pela ferramenta em um código fonte

Além do mapeamento efetuado pela ferramenta, o usuário tem a opção de inserir anotações em classes ou métodos, de acordo com o seu próprio entendimento. Para isso, é necessário que o usuário efetue a importação do tipo `org.eclipse.ea.requisit.manager.utils.Requisitos`, caso ele ainda não esteja

presente no código fonte.

Para ambos os casos, é necessário que o usuário tenha vinculado em seu `classpath`, a biblioteca `org.eclipse.ea.requisit.manager.commons`, que contém a definição da anotação. Na figura 39, é exibida a vinculação desta biblioteca em um projeto.

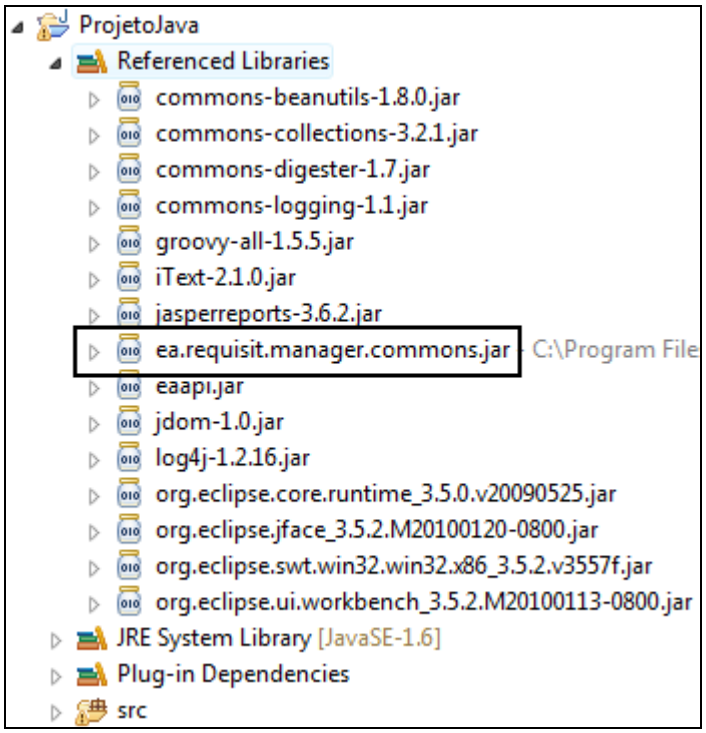


Figura 39 – Vinculação da biblioteca `ea.requisit.manager.commons` ao projeto

Com os códigos fontes mapeados, é possível ao usuário verificar quais são as dependências que um requisito possui, em relação às classes e métodos de um projeto. Desta maneira, ao selecionar a opção `Requisitos X Códigos Fontes`, será apresentada uma *view*, onde é exibida uma árvore com todos os requisitos mapeados no código fonte do projeto. A figura 40 exibe a *view* que é criada com as informações de rastreabilidade dos requisitos.

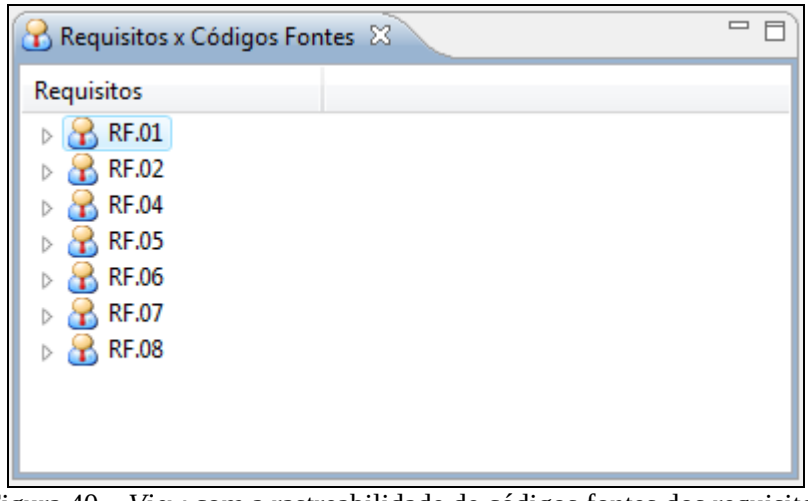


Figura 40 – *View* com a rastreabilidade de códigos fontes dos requisitos

Na figura 41 é apresentada a *view*, na qual o primeiro nível abaixo de um requisito é apresentada todas as classes com a anotação `Requisitos`.

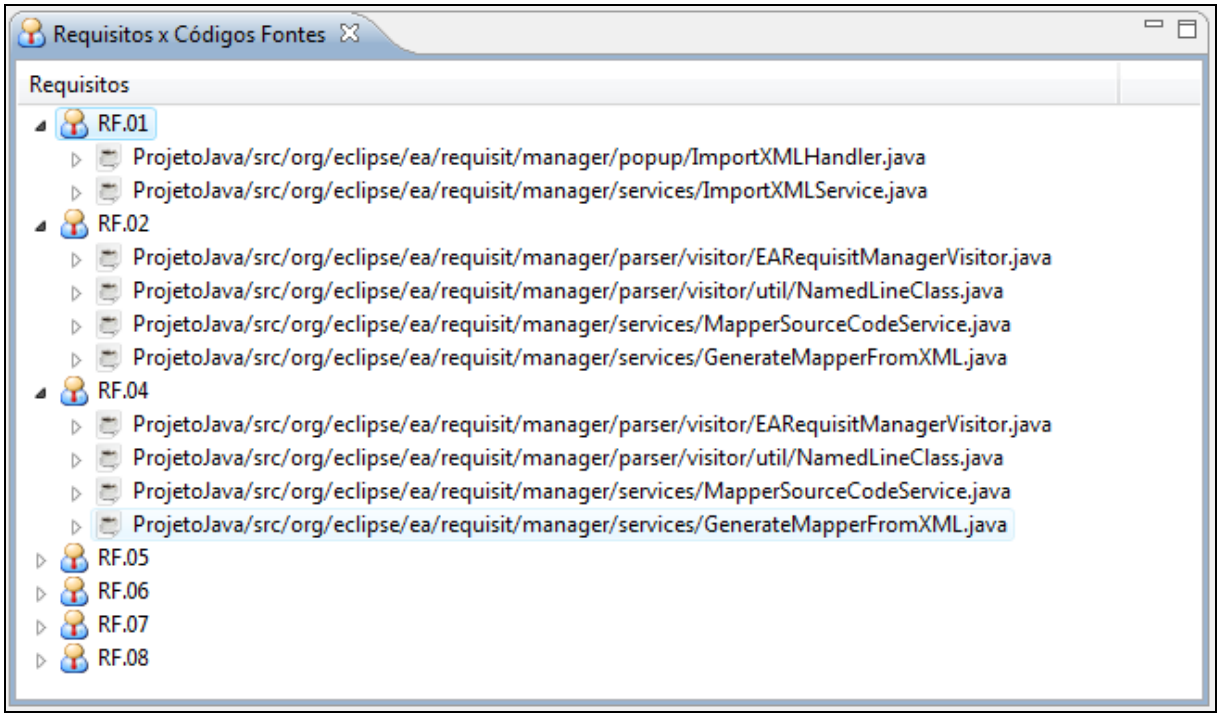


Figura 41 – Primeiro nível de informação para o mapeamento de um requisito: classe

Já no segundo nível, são apresentados os métodos com a anotação `Requisitos`. Os métodos apresentados foram obtidos com base no diagrama de sequência dos casos de uso, vinculados ao requisito em questão. Na figura 42, é possível observar como são dispostas as informações.

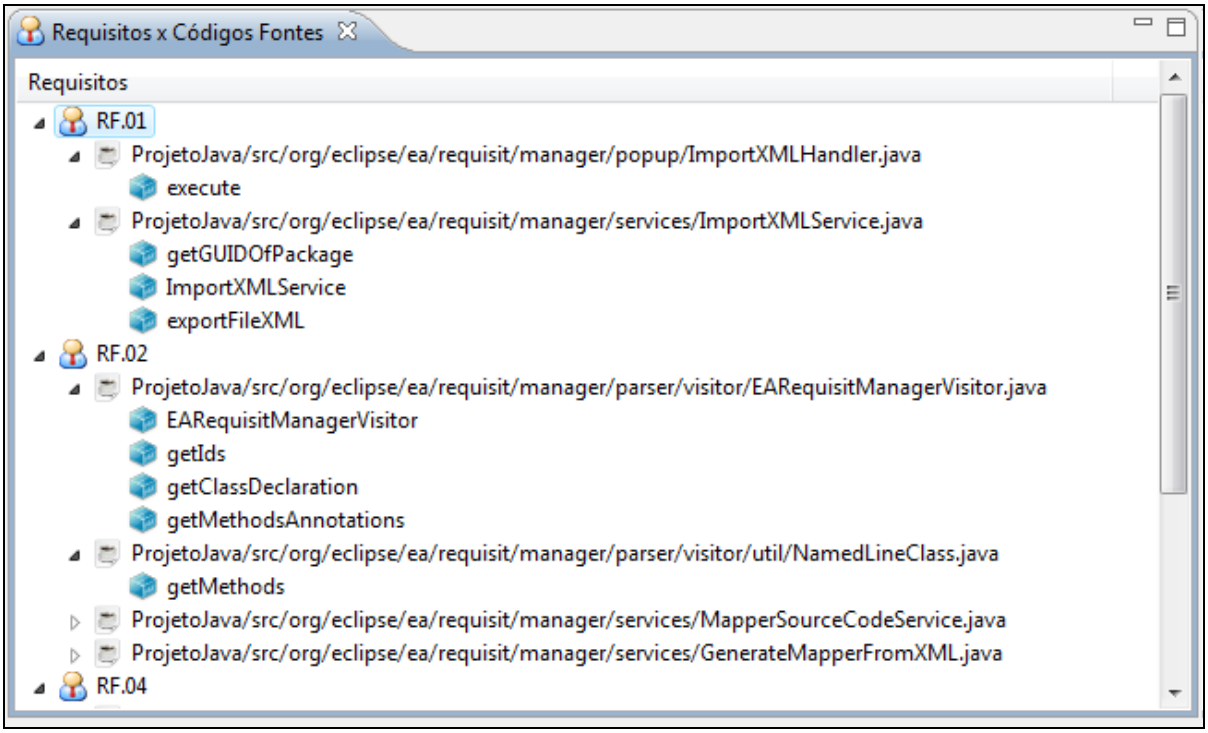


Figura 42 – Segundo nível de informação para o mapeamento de um requisito: método

Por último, o usuário tem a possibilidade de verificar qual a importância dos requisitos de um projeto. Na opção `Visualizar custos de alterações`, é exibida a tela demonstrada através da figura 43, na qual o usuário deve informar o arquivo XML do projeto que será analisado e também o destino do arquivo PDF que será gerado.

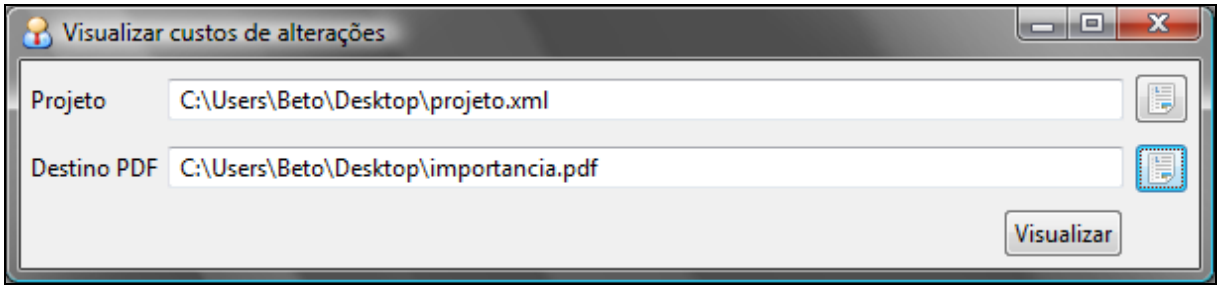


Figura 43 – Configuração da geração do arquivo PDF

Para a geração do arquivo PDF é aplicada a metodologia AHP, que avalia a representatividade e importância dos requisitos, para os critérios: classes, métodos e casos de uso. No arquivo PDF gerado, cada critério é apresentado de maneira separada, onde são exibidos os requisitos e o valor correspondente para aquele critério. Na figura 44 é apresentado o arquivo gerado com as informações.

EA Requisite Manager - Custos dos Requisitos

Critério de Classes

Requisitos	Valor (%)
RF.01	27.16
RF.02	13.58
RF.04	13.58
RF.05	13.58
RF.06	13.58
RF.07	13.58
RF.08	4.93

Critério de Métodos

Requisitos	Valor (%)
RF.01	40.89
RF.02	12.23
RF.04	12.23
RF.05	10.56
RF.06	9.88
RF.07	9.88
RF.08	4.29

Critério de Casos de Uso

Requisitos	Valor (%)
RF.01	16.12
RF.02	16.12
RF.04	16.12
RF.05	16.12
RF.06	16.12
RF.07	16.12
RF.08	3.22

Figura 44 – Arquivo PDF gerado com as informações de importância dos requisitos

3.4 RESULTADOS E DISCUSSÃO

Em relação aos trabalhos correlatos apresentados na fundamentação teórica, o Quadro 39 apresenta o comparativo entre o *plugin* desenvolvido e os trabalhos correlatos.

Ferramenta / Característica	**TraceFact-In**	**IRequirement**	**AspectCost**	**EA Requisite Manager**
Rastreabilidade de requisitos	Sim	Sim	Não	Sim
Integração com ferramenta CASE	Requisite Pro	EA	StarUML e Rational Rose	EA
Finalidade de uso	Acadêmico	Acadêmico	Acadêmico	Acadêmico
Aplicação de método para quantificação de dados	Não	Não	Sim	Sim
Banco de dados	MS SQL Server	-	XML	-
Análise de impacto com a alteração de requisitos	Sim	Sim	Sim	Sim
Linguagem utilizada para desenvolvimento	C#	Borland Delphi 6	Java	Java
Integração com ambiente de desenvolvimento	Microsoft Visual Studio 2008	-	-	Eclipse

Quadro 54 – Comparativo entre as características da ferramenta e trabalhos correlatos

A ferramenta desenvolvida alcançou os objetivos e agregou novas funcionalidades, em relação às ferramentas correlatas, permitindo tanto a rastreabilidade de requisitos e análise de impacto, quanto a aplicação de uma metodologia para quantificação de dados de importância de um projeto de software.

Um dos diferenciais da ferramenta desenvolvida se dá quanto à forma de permitir a rastreabilidade de artefatos de um projeto. No *Add-In* desenvolvido por Santos e Aragão (2009), é necessário que o usuário efetue a vinculação de artefatos manualmente, através de uma interface que permite a associação de classes e métodos com requisitos. Já na ferramenta desenvolvida por Batista (2007), o tipo de rastreabilidade existente se dá somente entre os artefatos do projeto, não sendo possível a associação de códigos fontes gerados a partir da especificação de um sistema. O EA Requisite Manager, por sua vez, permite através das suas funcionalidades, a leitura e mapeamento sem a interferência do usuário, tornando este processo mais ágil.

Outro ponto diferencial da rastreabilidade de requisitos desenvolvida na ferramenta desenvolvida é a possibilidade do desenvolvedor obter com precisão quais são os códigos fontes que serão impactados com a alteração de um determinado requisito. Neste trabalho, foi possível identificar métodos e classes que estão vinculados aos requisitos de um projeto, a partir do diagrama de classes e sequência especificado na fase de projetos.

Quanto à integração do *plugin* com a integração com outras ferramentas CASE de modelagem de projetos, notou-se a limitação dos recursos, uma vez que somente é permitida a análise de projetos do EA. O AspectCost, desenvolvido por Tocchetto (2007), por sua vez, é integrável com o StarUML e Rational Rose, proporcionando maior flexibilidade para os diversos usuários que a ferramenta pode atingir. Da mesma maneira, as ferramentas desenvolvidas por Santos e Aragão (2009) e Batista (2007) são limitados à integração com o Requisite Pro e EA, respectivamente.

Já em relação à finalidade de uso, é possível afirmar que todas as ferramentas analisadas foram concebidas para uso acadêmico, o que restringe a gama de funcionalidades que uma ferramenta CASE de gerenciamento de requisitos pode ter.

Um fator que pode sofrer melhorias em futuros trabalhos se dá em relação à visualização do cálculo de ponderação de importância de requisitos. Na ferramenta desenvolvida por Tocchetto (2007), o usuário pode acompanhar a formação dos valores do cálculo, a partir do preenchimento da matriz de comparação, onde o usuário pode fornecer o seu próprio julgamento em relação à importância dos requisitos. Entretanto, um aspecto positivo do *plugin* desenvolvido se dá em relação à apresentação de três critérios de avaliação de importância, onde o usuário pode comparar o impacto de cada critério, sob a ótica de uma perspectiva (classe, método ou casos de uso). Esta avaliação foi disponibilizada através da geração de um arquivo PDF, permitindo ao usuário utilizar os valores conforme a sua necessidade.

É possível destacar também que a integração da ferramenta com o ambiente de desenvolvimento no qual o desenvolvedor trabalha é um facilitador. Esta característica se difere do IRequirement, desenvolvido por Batista (2007) e Tocchetto (2007), uma vez que ambas as ferramentas são aplicações externas à IDE de desenvolvimento. A soma desta característica e a linguagem de programação escolhida para o desenvolvimento da ferramenta tornam o EA Requisite Manager diferente das demais ferramentas. Entretanto, é válido observar que não cabe ao contexto do trabalho discutir a eficiência e estabelecer comparações entre cada linguagem ou ambiente para quais as ferramentas foram desenvolvidas.

Durante a concepção deste trabalho e elaboração de sua proposta, pensou-se ser necessário o uso da biblioteca AspectWerkz (BON; VASSEUR, 2005) para realizar a tarefa de inserção de anotações do código fonte, marcando-os com os requisitos que determinada classe ou método atenderia. Contudo, no desenvolvimento da ferramenta e construção do conhecimento a cerca das tecnologias que seriam necessárias para a realização dos objetivos do trabalho, optou-se pela não utilização da biblioteca AspectWerkz, uma vez que não

apresentava relevância e utilidade para a implementação das funcionalidades necessárias para contemplar a ideia do trabalho. Esta biblioteca não possibilita a definição de um tipo de anotação específico (seja via API ou XML), no qual seja possível inserir os valores dos atributos de maneira dinâmica. Assim, optou-se pelas bibliotecas disponibilizadas pela IDE Eclipse, que forneceram os recursos necessários para a obtenção dos objetivos finais.

Outro ponto importante relacionado ao resultado final da implementação foi a opção em não se gerar um *template* para o EA, no qual o usuário informaria o valor de complexidade de um determinado requisito. Notou-se que esta ideia não supriria a necessidade de obtenção do relacionamento entre os requisitos do projeto. Desta maneira, optou-se pela definição de relacionamentos levando-se em consideração a quantidade de artefatos relacionados aos requisitos.

Já em relação à escolha da leitura e interpretação do arquivo XML, notou-se que esta opção permitiu que todos os dados necessários fossem buscados. Isto foi possível uma vez que bastou interpretação dos dados disponibilizados pelo processo de exportação do EA. Neste sentido, o acesso direto ao repositório do EA não fez diferença para a completude do acesso às informações.

Entretanto, é válido observar que o acesso direto ao repositório do EA pode tornar a ferramenta mais prática e intuitiva, uma vez que não necessitaria que o usuário informasse o arquivo XML toda vez que fosse mapear os códigos fontes de um determinado projeto.

Com o desenvolvimento da ferramenta também fica claro a importância e relevância da fase de projetos, dentro de um processo de desenvolvimento de software. A constante evolução de um sistema, a partir da inclusão, alteração e remoção de requisitos exige que a documentação e projeto sejam bem elaborados e também atualizados, com o intuito de se obter um maior controle do impacto que as possíveis mudanças podem causar.

Neste sentido, a elaboração de um projeto que incorpora a descrição e detalhamento dos artefatos permite que seja possível mensurar, controlar e conferir o que foi solicitado em termos de funcionalidade, por parte do usuário, e o que foi de fato, realizado com a implementação do projeto.

4 CONCLUSÕES

Auxiliar na automatização de um processo ou parte dele requer o estudo de todas as partes que o envolvem, do início ao fim. A análise da problemática que envolve a rastreabilidade de requisitos de software e análise de impacto na gestão de mudanças permite afirmar que, quando estes acontecem de maneira manual, tornam altos os custos e riscos de imprecisão numa decisão final. Isto pode levar uma empresa à tomada de decisão incorreta, levantamento errôneo na avaliação de impacto e custos e análise de alterações no código fonte imprecisa.

Portanto, o desenvolvimento da referida ferramenta auxiliou na automatização do processo de rastreabilidade e análise de impacto, disponibilizando informações para que a tomada de decisão ocorra de maneira mais precisa e segura, contribuindo para que o gerenciamento de requisitos controle com maior eficiência a evolução dos requisitos em função da manutenção.

Por sua vez, o mapeamento dos requisitos de um projeto nos códigos fontes, com a utilização da ferramenta, ocorre sem a interferência do usuário. Este processo só foi possível, uma vez que a fase de projetos e elaboração forneceu a vinculação dos requisitos, com as classes e métodos envolvidos. Desta maneira, é imprescindível afirmar que a leitura, interpretação e busca dos dados do arquivo XML, fornecido pela ferramenta EA, foi essencial para a concepção e desenvolvimento de todas as demais funcionalidades do *plugin*. A importância da fase de projetos de software se deu pelo fornecimento da visão dos elementos que envolvem o desenvolvimento de software, a partir da análise de diagramas de classe e sequência.

Da mesma forma, o auxílio de uma AST disponibilizada por biblioteca da IDE Eclipse permitiu e facilitou a inserção de anotações nos códigos fontes. O estabelecimento destas anotações, contendo os requisitos que um fragmento de código atende, foi determinante para se estabelecer o funcionamento e operacionalidade da ferramenta, uma vez que a interpretação destas anotações forneceu a rastreabilidade de artefatos.

Em paralelo à avaliação dos benefícios da rastreabilidade de requisitos e análise de impacto gerado pela ferramenta, deve-se observar a necessidade que existe em quantificar os valores de importância dos requisitos. Desta maneira, a aplicação da metodologia AHP permitiu estabelecer os valores correspondentes à importância relativa dos requisitos em um projeto, dimensionando desta maneira, o impacto que uma alteração pode gerar em um

projeto. Ainda relativo à quantificação dos valores dos requisitos, a adoção de critérios baseados em artefatos de um projeto, permitiu se estabelecer um comparativo, no qual é apresentada a importância dos requisitos, conforme cada critério estabelecido.

Assim sendo, o estudo dos conceitos e técnicas aplicadas ao trabalho proporcionou o desenvolvimento desta ferramenta, atendendo com sucesso os objetivos inicialmente estabelecidos como preponderantes para a realização do *plugin*.

4.1 EXTENSÕES

Algumas sugestões de extensão para este trabalho são:

a) disponibilizar uma funcionalidade para a remoção das anotações nos códigos fontes, sem que seja necessário o usuário efetuar este processo em todos os arquivos;
b) melhorar e aperfeiçoar o mecanismo de inserção de anotações nos códigos fontes, utilizando para isso, bibliotecas da IDE Eclipse;
c) criar um mecanismo extensível, que permita a integração do *plugin* com outras ferramentas CASE de modelagem de projetos;
d) criar uma interface que permita ao usuário acessar e analisar o *log* gravado;
e) disponibilizar um cadastro de tipos de critério, onde seja possível ao usuário estabelecer quais são os critérios que servirão como comparativo para a metodologia AHP e quais são os valores a serem atribuídos a cada critério/requisito;
f) disponibilizar uma interface em que o usuário consiga acompanhar o passo a passo da quantificação de requisitos.

REFERÊNCIAS BIBLIOGRÁFICAS

APACHE SOFTWARE FOUNDATION. **Apache log4j 1.2**. [S.l.], 2011. Disponível em: <http://logging.apache.org/log4j/1.2//>. Acesso em: 20 maio 2011.

BATISTA, Raphael M. **Ferramenta de gerência de requisitos de software integrada com Enterprise Architect.** 2007. 67 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://campeche.inf.furb.br/tccs/2007-I/2007-1raphaelmarcosbatistavf.pdf>. Acesso em: 08 set. 2010.

BELTRÃO FILHO, Mauro F. de H. **Gingway –** uma ferramenta para criação de aplicações Ginga-NCL interativas para TV digital**.** 2008. 59 f. Monografia (Bacharelado em Ciência da Computação) – Centro de Informática, Universidade Federal de Pernambuco, Recife. Disponível em: <http://www.cin.ufpe.br/~tg/2008-2/mfhbf.pdf>. Acesso em: 08 maio 2010.

BORGES, Eduardo P. **Um modelo de medição para processos de desenvolvimento de software**. 2003. 154 f. Dissertação (Mestrado em Ciência da Computação) – Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte. Disponível em: <http://homepages.dcc.ufmg.br/~wilson/pesquisa/DissertacaoEduardo.pdf>. Acesso em: 05 maio 2010.

BON, Jonas; VASSEUR, Alexandre. **AspectWerkz:** plain Java AOP 2.0 API. [S.l.], 2005. Disponível em: <http://aspectwerkz.codehaus.org>. Acesso em: 24 maio 2011.

BURNETTE, Ed. **Eclipse IDE –** guia de bolso**.** Tradução João Torello. Porto Alegre: Bookman, 2006.

COSTA, Helder G. **Introdução ao método de análise hierárquica:** análise multicritério no auxílio à decisão**.** Niterói: Universidade Federal Fluminense, 2002.

ECLIPSE FOUNDATION. **About the eclipse foundation.** [S.l.], 2011. Disponível em: <http://www.eclipse.org/org/>. Acesso em: 07 maio 2011.

FEIGENBAUM, Barry. **SWT, Swing or AWT:** which is right for you? [S.l.], 2006. Disponível em: <http://www.ibm.com/developerworks/grid/library/os-swingswt/>. Acesso em: 08 maio 2011.

FERREIRA, Felype S. **Implementação e análise de uma linha de produtos de software.** 2009. 67 f. Projeto de Graduação (Bacharelado em Ciência da Computação) – Centro de Informática, Universidade Federal de Pernambuco, Recife. Disponível em: <http://www.cin.ufpe.br/~tg/2009-2/fsf2.pdf>. Acesso em: 08 maio 2010.

GENVIGIR, Elias C. **Um modelo para rastreabilidade de requisitos de software baseado em generalização de elos e atributos.** 2009. 200 f. Teste de Doutorado do Curso de Pós Graduação em Computação Aplicada – Instituto Nacional de Pesquisas Espaciais, São José dos Campos. Disponível em: <http://mtc-m18.sid.inpe.br/col/sid.inpe.br/mtc-m18%4080/2009/03.02.14.17/doc/publicacao.pdf>. Acesso em: 14 jul. 2011.

HAMILTON, Vivien L.; BEEBY, Martin. Issues of traceability in integration tools. In: IEE COLLOQUIUM ON TOOLS AND TECHNIQUES FOR MAINTAINING TRACEABILITY DURING DESIGN, 1., 1991, Londres. **Procedings…** Londres: IEEE, 1991. p. 4/1-4/2. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=182212>. Acesso em: 30 ago. 2010.

HAROLD, Elliotte R. **Processing XML with Java.** [S.l.], 2001. Disponível em: <http://www.cafeconleche.org/books/xmljava/>. Acesso em: 08 maio 2011

HAZAN, Claudia; LEITE, Julio C. S. P. Indicadores para a gerência de requisitos. In: WORKSHOP EM ENGENHARIA DE REQUISITOS, 6., 2003, Piracicaba. **Anais eletrônicos...** Rio de Janeiro: PUC-RIO, 2003. Não paginado. Disponível em: <http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER03/claudia_hazan.pdf>. Acesso em: 30 ago. 2010.

HENKELS, André. **Drawcode:** um plugin do eclipse para geração de código a partir de diagramas de classe e diagramas N-S**.** 2007. 101 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

HUNTER, Jason; MCLAUGHLIN, Brett. **JDOM**: FAQ. [S.l.], 2007. Disponível em: <http://www.jdom.org/docs/faq.html#a0000>. Acesso em: 08 maio 2011.

JASPERFORGE. **IReport: JasperForge**. [S.l.], 2011. Disponível em: <http://jasperforge.org/projects/ireport>. Acesso em: 30 maio 2011.

LOPES, Luiz H. C. **Sistema web para gestão de pautas e atas de reuniões.** 2008. 55 f. Monografia (Curso de Especialização em Informática Empresarial) – Universidade Estadual Paulista, Guaratinguetá. Disponível em: <http://www.feg.unesp.br/ceie/Monografias-Texto/CEIE0805.pdf>. Acesso em: 08 maio 2011.

MARINS, Cristiano S.; SOUZA, Daniela de O.; BARROS, Magno da S. O uso do método de análise hierárquica (AHP) na tomada de decisões gerenciais: um estudo de caso. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL - PESQUISA OPERACIONAL NA GESTÃO DO CONHECIMENTO, 41., 2009, Porto Seguro. **Anais eletrônico...** Rio de Janeiro: Universidade Federal Fluminense, 2009. p. 1778-1788. Disponível em: <http://www.ic.uff.br/~emitacc/AMD/Artigo%204.pdf>. Acesso em: 15 maio 2011.

OLIVEIRA, Fabricio. **Software de apoio à gerência de solicitação de mudanças.** 2006. 72 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

PRESSMAN, Roger S. **Engenharia de software.** Tradução José Carlos Barbosa dos Santos. São Paulo: Makron Books, 1995.

RICHARDSON, Julian; GREEN, Jeff. Automating traceability for generated software artifacts. In: INTERNACIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGENIREERING, 19., 2004, Moffett Field. **Procedings...** Moffett Field: IEEE, 2004. p. 24-33. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1342721>. Acesso em: 30 ago. 2010.

ROCHA, Antônio F. **Configuração dinâmica de interface com o usuário.** 2007. 41 f. Trabalho de Conclusão de Curso (Engenharia da Computação) – Deparamento de Sistemas Computacionais, Escola Politécnica de Pernambuco, Recife. Disponível em: <http://dsc.upe.br/~tcc/20071/Monografia_TCC_Fernando_20071_Final.pdf>. Acesso em: 09 maio 2011.

SAATY, Thomas L. **Método de análise hierárquica.** Tradução Wainer da Silveira e Silva. São Paulo: McGraw-Hill, 1991.

SANTOS, Eder M. dos; ARAGÃO, João P. B. de. **Rastreabilidade de artefatos de software.** 2009. 43 f. Trabalho de Conclusão de Curso (Bacharelado em Informática) – Universidade Católica de Salvador, Salvador. Disponível em: <http://info.ucsal.br/banmon/Arquivos/Mono_12122009.pdf>. Acesso em: 30 ago. 2010.

SOMMERVILLE, Ian. **Engenharia de software.** 6. ed. Tradução Maurício de Andrade. São Paulo: Addison Wesley, 2003.

SPARX SYSTEMS. **Enterprise Architect:** user guide. Version 8.0. [S.l.], 2010. Documento eletrônico disponibilizado com o Ambiente Enterprise Architect 8.0.

SWT. **SWT:** the standard widget toolkit**.** [S.l.], 2011. Disponível em: <http://www.eclipse.org/swt/>. Acesso em: 08 maio 2011.

TOCCHETTO, André L. **AspectCost:** um ambiente de gerência e acompanhamento de custos de requisitos baseados em AOP. 2007. 90 f. Dissertação (Mestrado em Computação Aplicada) – Programa Interdisciplinar de Pós-Graduação em Computação Aplicada, Universidade do Vale do Rio dos Sinos, São Leopoldo. Disponível em: <http://bdtd.unisinos.br/tde_arquivos/1/TDE-2007-06-27T110125Z-306/Publico/aspectcost.pdf>. Acesso em: 30 ago. 2010.