Geração Automática de um Grafo de Requisitos a partir do Código da Aplicação

Marcelo M. Vieira[1], **Fabio Tirelo**[1], **Humberto T. Marques Neto**[1]

[1]Departamento de Ciência da Computação
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
30.535-901 - Belo Horizonte - Brasil

mmvieira@sga.pucminas.br, {ftirelo,humberto}@pucminas.br

Resumo. *Os requisitos de um software podem sofrer manutenções significativas durante o processo de desenvolvimento. Essas manutenções podem apresentar um custo elevado na medida em que o processo evolui, tanto por conta da complexidade em manter a rastreabilidade entre os requisitos, quanto por causa do impacto que uma alteração em um deles possa causar nos demais. Este trabalho propõe a geração automática de um grafo de requisitos através do mapeamento do grafo de dependências de módulos de uma aplicação e os requisitos por eles satisfeitos. Com isso, é possível utilizar os relacionamentos entre os módulos para manter a rastreabilidade entre os requisitos, permitindo apontar o impacto que a alteração em um deles possa causar nos demais. Um estudo de caso com 34 módulos e 24 requisitos é apresentado para exemplificar a técnica proposta.*

Abstract. *The software requirements change significantly during the software process. These changes may result in high cost, while the software process evolves, due the complexity in maintaining the traceability among requirements and because of the impact caused by changes in these requirements. This paper proposes an automatic generation of a requirements graph, by mapping the dependency graph of modules of an application and the requirements associated with them. In this way, it is possible to use the relationship among modules in order to maintain the requirements' traceability. This allows the assessment of the impact that a change in one requirement may cause in others. A case study with 34 modules and 24 requirements is presented to illustrate the proposed technique.*

1. Introdução

O processo de desenvolvimento de um software é iniciado com a elicitação dos seus requisitos, que podem variar em quantidade e complexidade de acordo com as necessidades dos interessados em sua implementação [Sommerville 2007]. A definição do conjunto de requisitos é, em geral, obtida de maneira incremental, partindo-se de uma especificação geralmente inconsistente, que é refinada junto aos interessados, até que se obtenha um conjunto de requisitos bem definidos [Cheng and Atlee 2007]. Ao longo desse processo de refinamento, os requisitos podem sofrer modificações, justificadas por mudanças identificadas a partir, por exemplo, da melhor compreensão do propósito do sistema. Logo, em função da volatilidade dos requisitos, organizações que desenvolvem software adotam processos de engenharia de requisitos, para que seja possível especificar, validar e manter o controle sobre sua evolução [Pressman 2006].

A engenharia de requisitos é definida como o conjunto de tarefas responsáveis por descobrir e registrar o propósito de um software, interagindo com os interessados em sua implementação e avaliando suas expectativas em relação ao sistema proposto [Nuseibeh and Easterbrook 2000]. Para que essas tarefas possam ser executadas, organizações que desenvolvem software podem adotar práticas que auxiliam na definição dos responsáveis por essa execução e pela gerência dos requisitos [Pressman 2006]. Nesse contexto, a capacidade de manter o controle dos relacionamentos entre requisitos de software, torna-se uma condição importante, para que seja possível observar o impacto que alterações em um deles possam causar nos demais [Egyed 2003].

A rastreabilidade de requisitos é a capacidade de seguir a evolução de um requisito desde sua identificação e elicitação até a sua implementação e de seguir o seu histórico de alterações, permitindo ir de sua implementação até sua especificação [Gotel and Finkelstein 1994]. A rastreabilidade auxilia tanto na execução da gerência de requisitos, no que se refere ao controle de sua evolução, quanto no estabelecimento de relacionamentos entre eles. Conforme argumentado em [Hayes et al. 2003], existem duas motivações para a utilização da rastreabilidade entre requisitos: (1) permitir que o sistema satisfaça os requisitos especificados e (2) permitir a análise de impactos que mudanças em sua especificação possam causar.

No entanto, a prática da rastreabilidade de requisitos é bem complexa, pois a manutenção dos seus relacionamentos é uma tarefa que geralmente demanda grande esforço por parte dos responsáveis por sua execução. Isso se justifica por conta da constante evolução e da quantidade de requisitos a ser gerida. O grau de dificuldade encontrado nas práticas de rastreabilidade pode variar dependendo da técnica adotada. Essas técnicas influenciam a capacidade de automatização da geração de relacionamentos entre requisitos, a precisão desses relacionamentos e a quantidade de requisitos gerenciados pelo procedimento [Cleland-Huang et al. 2004].

Como os requisitos de um sistema tendem a estar relacionados, a alteração ou exclusão de um deles pode afetar os demais [Cheng and Atlee 2007]. Logo, o uso de técnicas de rastreabilidade é recomendado quando se deseja controlar essas alterações ou possibilitar a análise do impacto que elas possam causar nos demais elementos do sistema [Nuseibeh and Easterbrook 2000]. A partir desse tipo de análise, o responsável pela gerência de requisitos pode tomar decisões sobre a viabilidade da execução da alteração de um determinado requisito. Pode-se, por exemplo, entender que, em função da complexidade dos relacionamentos verificados entre os requisitos, uma manutenção é inviável, pelo impacto que ela causaria.

Este trabalho propõe uma técnica para a construção automática de grafos de requisitos, por meio do mapeamento entre o grafo de dependências de módulos de uma aplicação e os requisitos satisfeitos por esses módulos. Com base no grafo construído é possível manter a rastreabilidade entre requisitos, estabelecida pelos relacionamentos identificados automaticamente entre eles e, assim, apontar o impacto que alterações em um deles possam causar nos demais. Esse mapeamento pode ser realizado, dado que um requisito pode ser satisfeito por um ou mais módulos da aplicação existindo, portanto, uma relação entre código e requisito. Um estudo de caso é apresentado para exemplificar a técnica proposta para construção automática do grafo de requisitos de um sistema de *petshop* on-line.

O restante deste artigo está organizado da seguinte forma: na Seção 2, é feita a análise sobre o estado da arte. Na Seção 3, é descrito o procedimento de geração dos relacionamentos entre os requisitos sob análise. O estudo de caso para validação do procedimento de criação de relacionamentos entre requisitos é descrito na Seção 4. Na Seção 5, apresentam-se as conclusões deste trabalho e os apontamentos para trabalhos futuros.

2. Referencial Teórico

A rastreabilidade de requisitos e, por consequência, o estabelecimento de relacionamentos entre eles têm sido foco de diversos trabalhos. Uma das técnicas mais comumente utilizadas consiste na criação de uma matriz de rastreabilidade [Sommerville 2007]. Assim, dado um conjunto de requisitos $\{R_1, R_2, ..., R_n\}$, a matriz de rastreabilidade é uma matriz binária $A = [a_{ij}]_{n \times n}$, tal que a_{ij} é igual a 1, se os requisitos R_i e R_j estão relacionados. Um exemplo de aplicação desenvolvida para a utilização dessa matriz pode ser encontrado no RequisitePro [IBM 2010]. A principal vantagem dessa técnica está no conceito simples e de fácil entendimento por parte dos envolvidos em sua utilização. No entanto, a construção e a manutenção dessas matrizes podem se tornar atividades trabalhosas e sujeitas a erros, o que pode não ser adequado a projetos de médio e grande porte [Konrad and Gall 2008].

Técnicas de rastreabilidade também foram propostas com base na geração de relacionamentos semanticamente recuperados entre os requisitos de um sistema [Hayes et al. 2003]. Nesse tipo de abordagem, os requisitos são registrados em uma base de dados, onde são recuperados por meio de consultas contendo termos que são comparados com as palavras-chave da especificação de cada requisito, definidas pelos envolvidos nessa especificação. Os relacionamentos que permitem a realização da rastreabilidade de requisitos são gerados dinamicamente por meio da análise de similaridade entre os termos dessas consultas e os termos da especificação. Dois ou mais requisitos estarão relacionados se a similaridade calculada foi maior do que um limiar estabelecido empiricamente. A principal vantagem desse tipo de abordagem está na automatização da criação dos relacionamentos entre requisitos, ao contrário do que acontece com a utilização de matrizes de rastreabilidade que, em geral, demandam atualização manual a cada modificação na especificação.

A aplicação de técnicas de recuperação semântica de relacionamentos entre requisitos pode resultar na obtenção de relacionamentos que não existem [Asuncion et al. 2010]. Isso pode ocorrer quando termos lexicamente similares são utilizados na especificação de diversos requisitos, mas seus significados são diferentes. Um modelo de rastreabilidade que tem por objetivo evitar a obtenção de falsos relacionamentos é proposto por [Cleland-Huang et al. 2009]. O objetivo desse modelo é definir os relacionamentos entre requisitos e artefatos de software em três camadas: uma lógica, responsável por definir as características desses relacionamentos, uma de dados, que representa a base de conhecimento onde os requisitos são especificados, e uma de apresentação, responsável por exibir ao usuário os relacionamentos construídos. Dessa forma, é possível qualificar os relacionamentos, refinando a informação contida na camada lógica e melhorando a forma como eles serão exibidos. No entanto, essa abordagem também demanda intervenção manual frequente para o estabelecimento dos relacionamentos entre os requisitos e os artefatos de software. Ou seja, a capacidade de automatização é reduzida, assemelhando-se às técnicas baseadas em matrizes de rastreabilidade.

Segundo [Cheng and Atlee 2007], os fatores que mais influenciam a eficiência de uma técnica de rastreabilidade de requisitos são a capacidade de automatização da geração de relacionamentos entre requisitos e a precisão desses relacionamentos. Nesse contexto, modelos baseados em grafos de rastreabilidade têm sido propostos com o objetivo de encontrar uma solução de compromisso entre esses fatores. Um meta-modelo de rastreabilidade baseado em um grafo é proposto por [Ramesh and Jarke 2001]. Nesse meta-modelo, os vértices representam elementos conceituais, como por exemplo, requisitos, e também podem representar um interessado no sistema ou um artefato utilizado para fins de documentação, como por exemplo, o documento de especificação de requisitos. Uma aresta entre dois vértices representa a existência de uma relação entre os elementos representados por esses vértices. Um exemplo possível dessa relação ocorre quando um requisito é relacionado ao responsável por sua especificação. Visto que o objetivo dessa proposta é a definição de um meta-modelo, com a caracterização do que são vértices e arestas no grafo resultante, não são apresentadas formas de automatização da construção desse grafo.

Outra abordagem para a construção de um grafo de requisitos é proposta por [Cleland-Huang et al. 2005]. Nessa abordagem os vértices do grafo representam os objetivos de um sistema. Cada aresta é inicialmente criada de forma manual e em seguida, elas são utilizadas como um conjunto hipotético de relacionamentos entre objetivos do sistema. Durante a evolução dos requisitos, essas arestas são refinadas e recuperadas automaticamente, utilizando-se modelos probabilísticos, tais como os de recuperação semântica de relacionamentos entre requisitos. O usuário deve, analisando os relacionamentos recuperados, decidir se uma mudança proposta nos requisitos pode ser feita. Essa abordagem sugere a união de técnicas de construção de grafos de requisitos e de recuperação semântica de relacionamentos entre requisitos. Ela pode ser adequada, principalmente, quando o objetivo é a gestão de relacionamentos entre requisitos não-funcionais.

O meta-modelo proposto por [Ramesh and Jarke 2001] foi adotado por [Egyed 2003] para propor a construção semi-automatizada de um grafo de requisitos, construído por meio da análise dinâmica (i.e., em tempo de execução) do código de uma aplicação e do conjunto de casos de teste associados a ela. O processo de construção do grafo recebe como entrada a definição hipotética dos relacionamentos entre os requisitos da aplicação, construída manualmente pelos envolvidos em sua especificação. Em seguida, os módulos da aplicação são monitorados em tempo de execução, durante a avaliação de casos de teste, para que sejam coletadas informações sobre quais desses módulos foram utilizados na realização de cada teste. Visto que esses casos de teste são especificados de acordo com os requisitos do sistema, [Egyed 2003] propõe a construção do grafo de requisitos, refinando os relacionamentos hipotéticos, fornecidos inicialmente, utilizando as informações obtidas no monitoramento da aplicação durante a execução dos casos de teste.

Das técnicas descritas nesta Seção, a que mais se assemelha à apresentada neste trabalho é a proposta por [Egyed 2003]. No entanto, essa técnica usa um procedimento semi-automatizado, com intervenção manual para a criação de relacionamentos hipotéticos entre requisitos. Além disso, o monitoramento da execução dos casos de teste pode não fornecer um conjunto apurado de informações, uma vez que alguns módulos da aplicação podem não ser verificados, dependendo-se das condições de teste.

Outra importante diferença entre a técnica proposta por [Egyed 2003] e as demais baseadas em grafos de requisitos, está no fato de que o foco dessas abordagens é a rastreabilidade entre requisitos e artefatos de software. Neste trabalho, explora-se a rastreabilidade existente entre requisitos e o código da aplicação para a construção automatizada de um grafo de requisitos que permita identificar os relacionamentos existentes entre eles, possibilitando assim, manter a rastreabilidade e avaliar o impacto que a alteração em um requisito possa causar nos demais.

3. Mapeamento de Módulos em Requisitos de Software

Esta Seção descreve a construção do grafo de relacionamentos entre requisitos proposto neste trabalho. Como pré-condição para esta construção, está a definição dos relacionamentos de rastreabilidade entre os requisitos e os módulos de uma aplicação. Esses relacionamentos podem ser obtidos utilizando-se algumas técnicas disponíveis na literatura, como por exemplo, as citadas em [Cleland-Huang et al. 2005], [Cleland-Huang et al. 2009], [Egyed 2003] e [Ramesh and Jarke 2001]. Recursos presentes em linguagens de programação atuais, tais como anotação[1] em Java e C#, também podem ser usados como alternativa para simplificar o processo de definição desses relacionamentos.

A construção do grafo de requisitos proposta neste trabalho é realizada de forma automatizada e ocorre em três etapas. Na primeira etapa, descrita na Seção 3.1, o grafo de dependências entre módulos da aplicação, que será utilizado para mapear as dependências em relacionamentos entre requisitos, é construído. Na segunda etapa, descrita na Seção 3.2, os requisitos satisfeitos por um mesmo módulo são relacionados caso contribuam com a especificação que deu origem à lógica implementada nesse módulo. Na terceira e última etapa, descrita na Seção 3.3, o grafo de dependências entre módulos e o grafo de relacionamentos entre requisitos, obtido na segunda etapa, são utilizados para identificar os relacionamentos entre requisitos que estejam associados em função de dependências que foram identificadas entre módulos distintos.

3.1. Geração do Grafo de Dependências entre Módulos

Esta etapa começa com a geração automatizada do grafo de dependências entre módulos da aplicação. Cada vértice desse grafo representa um módulo e uma aresta entre dois vértices existirá se os módulos representados por eles possuírem alguma relação de dependência. Um módulo C_1 dependerá de um módulo C_2 se, em qualquer trecho do código de C_1, existir uma referência ao módulo C_2. Essa referência pode ser a declaração em C_1 de atributos ou parâmetros de métodos cujo tipo seja C_2 ou a declaração, em qualquer trecho do código de $C1$, de uma variável cujo tipo seja definido por $C2$.

Sendo identificada a dependência de um módulo $C1$ com um módulo $C2$, o grafo resultante será composto por dois vértices, um representando C_1 e outro representando C_2, de modo que exista uma aresta direcionada partindo do vértice que representa C_1 para o vértice que representa C_2. Esse procedimento é automatizado, sendo executado por meio da análise estática[2] do código de cada módulo.

[1] Uma anotação é um meio de apresentar, no código da aplicação, informações relevantes de forma padronizada e estruturada, passível de processamento automatizado por ferramentas [Arnold et al. 2005].

[2] Análise executada no código de uma aplicação, sem que ela esteja em execução.

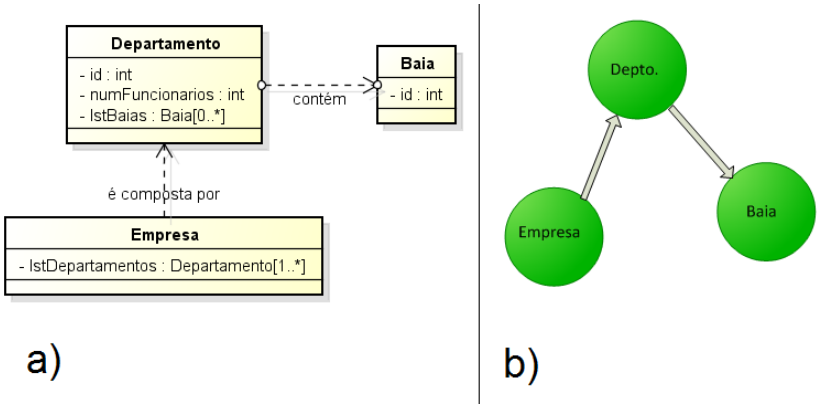

Figura 1. (a) Exemplo de dependências entre módulos e (b) o grafo resultante, após verificação dessas dependências.

A Figura 1 ilustra um exemplo de dependências entre três módulos e o grafo resultante. Nessa figura, o módulo *Departamento* depende do módulo *Baia*, uma vez que ele possui um atributo cujo tipo é definido pelo módulo *Baia*. Já o módulo *Empresa* depende do módulo *Departamento*, pois ele possui um arranjo de elementos cujos tipos são definidos pelo módulo *Departamento*. Logo, o grafo de dependências resultante contém uma aresta direcionada, partindo do vértice que representa o módulo *Departamento* para o vértice que representa o módulo *Baia* e uma aresta partindo do vértice que representa o módulo *Empresa* para o módulo *Departamento*.

3.2. Requisitos Satisfeitos pelo mesmo Módulo da Aplicação

As especificações dos requisitos de um sistema definem elementos que devem ser implementados no código dos módulos desse sistema. No entanto, um módulo pode implementar características do sistema que estejam definidas em mais de um requisito e também pode possuir uma relação de dependência com outros módulos. Essa relação pode existir, dado que a funcionalidade do sistema que um módulo visa implementar pode ser complementada por outras funcionalidades que estejam definidas em outros módulos, que por sua vez, foram construídos para satisfazer outros requisitos do sistema. Dessa forma, a especificação dos requisitos pode determinar a existência de diversas relações de dependência entre os módulos da aplicação, permitindo que seja feito um mapeamento entre essas dependências e os requisitos especificados.

Esta etapa de geração do grafo de requisitos consiste em analisar as dependências entre os módulos da aplicação, identificadas na etapa anterior, para gerar os relacionamentos entre os requisitos satisfeitos por esses módulos. No grafo resultante, um vértice representa um requisito do sistema e uma aresta entre dois vértices representa o relacionamento entre os requisitos representados por esses vértices.

A geração dos relacionamentos entre requisitos nessa etapa é dividida em dois passos. No primeiro, para cada módulo da aplicação, é gerada a lista de requisitos satisfeitos por esse módulo, verificando-se os relacionamentos definidos pela rastreabilidade entre esses elementos. Em seguida, é definida uma aresta bi-direcional entre dois requisitos quaisquer de uma mesma lista. A justificativa para esse procedimento se dá pelo fato de que, o módulo sob análise, foi construído para implementar o que está definido na especificação formada pelos requisitos aos quais ele esteja associado. Assim, a alteração

em um requisito que faça parte dessa especificação, poderá causar impacto nos demais, dado que a união desses requisitos forma a especificação em questão. Isso pode gerar a necessidade de revisão desses requisitos.

Sejam R_1 e R_2, dois requisitos satisfeitos por um módulo C_1. Se forem efetuadas alterações em R_1, que mudem seu significado, então pode ser necessário adequar o módulo C_1 à nova especificação de R_1. No entanto, ao refletir as modificações ocorridas em R_1, C_1 pode deixar de satisfazer adequadamente R_2. Dessa forma, R_2 pode necessitar de revisão ou pode invalidar a alteração em R_1. Assim, R_2 passa a depender de R_1 e essa dependência é representada por meio de uma aresta direcionada partindo do vértice que representa R_2 para o vértice que representa R_1. A determinação de que R_1 depende de R_2 é realizada em processo análogo, resultando em uma aresta bi-direcional entre os vértices de R_1 e R_2.

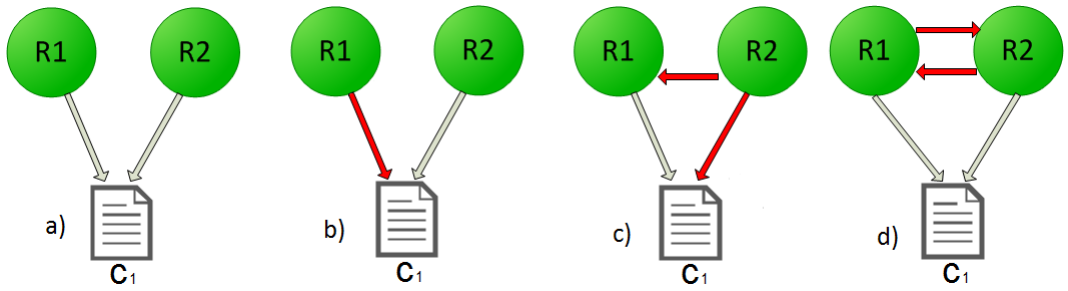

Figura 2. Exemplo do primeiro passo de geração de relacionamentos entre requisitos. (a) Dois requisitos satisfeitos por um mesmo módulo. (b) A alteração em R_1 impacta o módulo C_1. (c) Alteração propagada a R_2 e dependência criada. (d) Processo análogo a (c) para aresta partindo de R_1.

A Figura 2 ilustra o primeiro passo de geração de relacionamentos entre requisitos. Nela está definido um módulo C_1, construído para satisfazer os requisitos R_1 e R_2. No instante (b) é efetuada uma alteração em $R1$ que é propagada a C_1. No entanto, no instante (c), a relação entre C_1 e R_2 se torna inválida, em função da alteração em C_1. Logo, a modificação em R_1 gera a necessidade da revisão de R_2, sendo, portanto, criada uma aresta partindo de R_2 para R_1. O instante (d) ilustra os passos finais desse procedimento, com a criação da aresta partindo de R_1 para R_2, gerada de forma análoga.

3.3. Requisitos Satisfeitos por Módulos Distintos

A terceira etapa do processo de geração de relacionamentos entre requisitos é executada para identificar relacionamentos entre requisitos satisfeitos por mais de um módulo. Nesse passo, uma aresta direcionada de R_1 para R_2 será construída se R_1 for satisfeito por um módulo C_3, R_2 for satisfeito por um módulo C_4 e o módulo C_3 depender do módulo C_4. A Figura 3 ilustra esses relacionamentos.

O tipo de aresta construída nesta etapa se justifica, visto que alterações em R_2 que mudem seu significado, poderão implicar em alterações em C_4. No entanto, de acordo com o exemplo dado, ilustrado na Figura 3, o módulo C_3 possui uma relação de dependência com o módulo C_4 de modo que, ele também poderá sofrer modificações. Ocorrendo essas modificações, C_3 poderá não mais satisfazer R_1 de maneira adequada. Logo, nesse cenário, R_1 poderia sofrer alterações em função da alteração iniciada em R_2, ou invalidá-la.

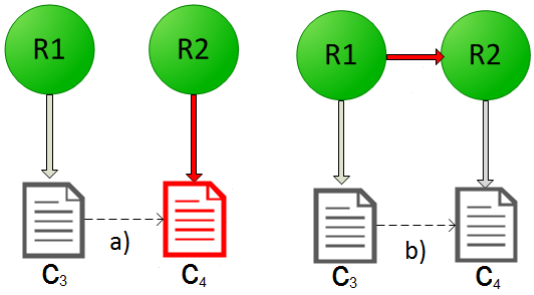

Figura 3. Exemplo do terceiro passo de geração de relacionamentos entre requisitos. (a) Alteração em R_2 propagada a C_4. (b) Alteração em C_4 propagada a C_3, criando a aresta de R_1 para R_2.

O procedimento descrito para construção do grafo proposto neste trabalho permite elevar o nível de abstração das dependências entre módulos ao nível de especificação do sistema, possibilitando manter a rastreabilidade entre requisitos, por conta da associação existente entre eles e o código da aplicação. Dessa forma, a proposta de construção do grafo de requisitos permite associar, de maneira automatizada, dois tipos de informação bem conhecidos em um processo de Engenharia de Software [Cheng and Atlee 2007]: o relacionamento entre módulos e os requisitos por eles satisfeitos, possibilitando assim a utilização dessa informação para a manutenção da rastreabilidade entre requisitos. Na Seção 4 é apresentado um estudo de caso elaborado para exemplificar o procedimento referido acima e sugerir como a análise de impacto na alteração de um requisito pode ser feita utilizando-se o grafo construído.

4. Estudo de Caso: Um Petshop On-line

Para exemplificar o procedimento de geração de relacionamentos entre requisitos descritos na Seção 3 utilizou-se a especificação e a implementação do sistema de um *petshop* on-line disponível em [IBM 2003]. O objetivo desse sistema é permitir que um usuário que queira comprar um animal de estimação possa fazê-lo por meio de uma aplicação na web. O sistema permite que os usuários cadastrem animais para venda, definindo espécie, foto, descrição e preço, e também permite que solicitações de compra de animais possam ser efetuadas. Para isso, o comprador precisa buscar pelo animal que deseja, selecioná-lo na lista de animais retornados pela pesquisa efetuada e prosseguir para a compra, gerando um pedido junto a um sistema externo de crédito, com o qual o *petshop* on-line mantém uma interface de comunicação. Por meio dessa interface, o *petshop* on-line recebe os dados da validação da compra efetuada e registra, em sua base de dados, o pedido correspondente, discriminando os itens associados. O usuário também pode pesquisar por animais disponíveis para venda e classificá-los de acordo com sua intenção de compra.

A documentação disponível do *petshop* on-line contém a especificação de requisitos, o código-fonte da aplicação, o documento de visão do sistema e o modelo arquitetural empregado, utilizando-se a plataforma JEE [Oracle 2010]. Por meio da verificação desses documentos, foram amostrados 24 requisitos, extraídos da especificação. Também foram utilizados 34 módulos da aplicação para compor o exemplo descrito nesta Seção. A escolha desses requisitos se deu em função de sua abrangência no sistema, pois, agregam desde características não-funcionais, como por exemplo, segurança, até regras de negócio. Na Tabela 1 é possível ver a relação de requisitos escolhidos.

Tabela 1. Requisitos Extraídos da Especificação do *Petshop* On-line

Identificador do requisito	Nome do Requisito
1	Validação Cartão Expirado
2	Cálculo Preço Extendido
3	Itens Preferidos Usuário
4	Dicas Navegação Sistema
5	Interface Pesquisa Itens
6	Paginação Relatório Pesquisa Itens
7	Validação Campos Obrigatórios
8	Tela Cadastro Usuários
9	Cadastro de Usuários
10	Interface PayPal
11	Gerência Dados Catálogo Animais
12	Identificação Única Usuário
13	Acesso Anônimo Sistema
14	Redirecionamento Acesso Inválido
15	Política Acesso Conta Usuário
16	Desempenho e Segurança na Rede
17	Auditoria de Logs
18	Carga de trabalho
19	Dados Categoria Animal
20	Dados Animais
21	Dados Perfil Usuário
22	Configuração Localidade
23	Dados Item Compra
24	Gerência Dados Tipo Compra

Na documentação disponível para o *petshop* on-line não estão definidos os relacionamentos entre requisitos e os módulos da aplicação. Como esses relacionamentos precisaram ser verificados, para serem utilizados como entrada no processo de construção do grafo de requisitos, conforme descrito na Seção 3, eles foram cadastrados em uma ferramenta desenvolvida como parte deste trabalho. Esse cadastro consistiu em definir, para cada requisito, quais módulos da aplicação foram construídos para satisfazê-lo. Os relacionamentos foram definidos por meio da análise do texto da especificação de cada requisito e da funcionalidade implementada em cada módulo. No entanto, esse cadastro não seria necessário, caso a especificação do *petshop* on-line apresentasse algum recurso que permitisse o mapeamento entre os módulos e os requisitos, como por exemplo, o uso de anotações. Os relacionamentos identificados podem ser visualizados na Tabela 2.

Uma vez estabelecidos os relacionamentos entre os requisitos e os módulos implementados para o *petshop* on-line, é possível prosseguir para a etapa de construção do grafo que modela esses relacionamentos. O procedimento de construção desse grafo é feito de forma automatizada e o resultado obtido pode ser visualizado na Figura 4. Os requisitos são representados pelos vértices e uma aresta existirá entre dois vértices, se for identificado um relacionamento entre os requisitos representados por esses vértices.

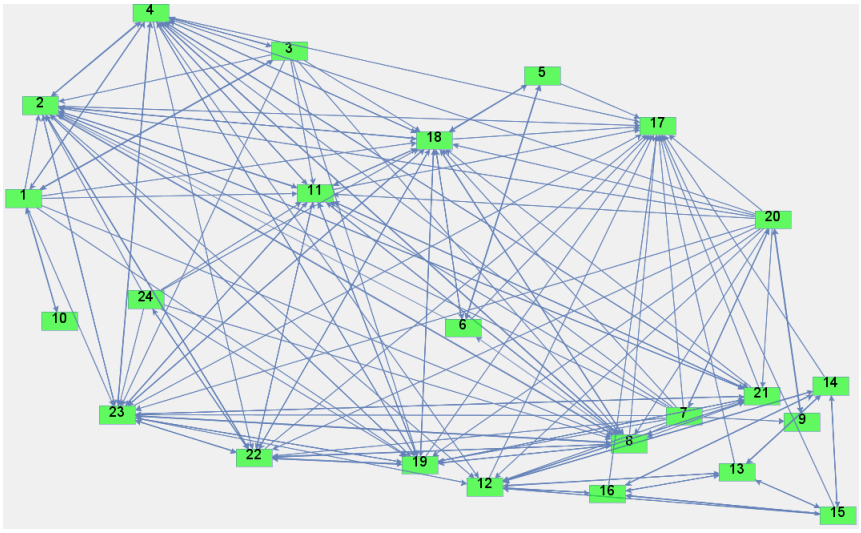

Figura 4. Grafo de relacionamentos entre requisitos do *petshop* on-line

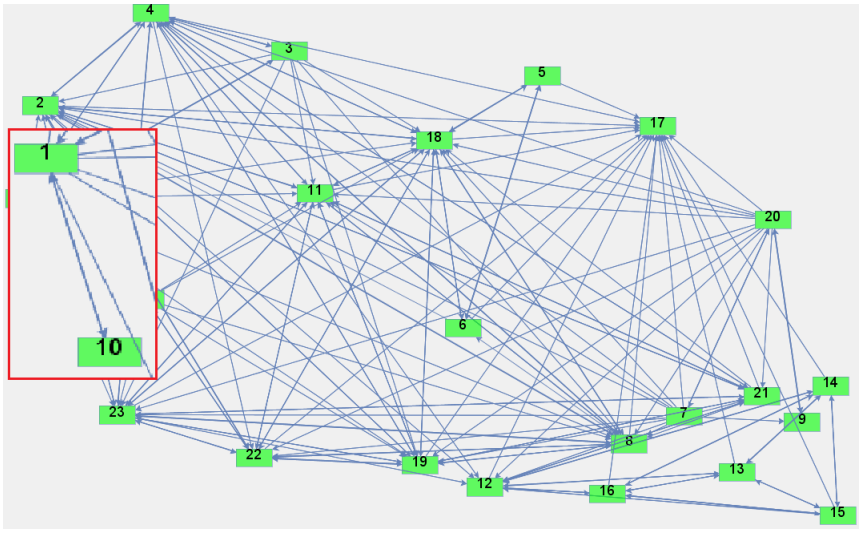

Figura 5. Um exemplo de aresta bi-direcional entre dois requisitos

Em função do tamanho do grafo gerado, alguns exemplos foram destacados e ampliados para a discussão que se segue. Na Figura 5 é possível verificar a existência de uma aresta bi-direcional entre os requisitos com identificadores 10 e 1. O requisito 1 especifica que uma compra não pode ser finalizada, caso o cartão de crédito utilizado esteja com a data de validade vencida. Já o requisito 10 define as regras para validação, junto ao sistema de crédito, do cartão utilizado em uma compra. Verificando-se os módulos associados a esses requisitos, é possível observar que ambos são satisfeitos pelo módulo *PayPalBean*, cuja função é prover as rotinas para comunicação com o sistema de crédito, com o qual o *petshop* on-line mantém uma interface. A aresta bi-direcional existente entre esses dois requisitos se justifica pelo fato de que se ocorresse uma alteração na especificação do requisito 1, permitindo o envio de uma data inválida de expiração do cartão, possivelmente um conflito com a forma de validação especificada no requisito 10 seria identificado.

Tabela 2. Relacionamentos entre Requisitos e Módulos *Petshop* On-line

1	RatingBean, PayPalBean
2	CatalogFacade, CatalogXmlAction, Item
3	RatingBean
4	Item, RatingBean
5	SearchIndex, SearchBean
6	SQLParser, SearchIndex, SearchBean
7	FileUploadBean
8	CatalogFacade, MapBean, AddressBean, ZipLocation, Address
9	FileUploadResponse
10	PayPalBean
11	CatalogXmlAction, Category, CatalogFacade, MapBean
12	SellerContactInfo, EntryFilter
13	EntryFilter
14	EntryFilter
15	EntryFilter
16	EntryFilter, CaptchaValidateFilter, RandomString, CaptchaAction, SimpleCaptcha, CaptchaSingleton
17	PetstoreConstants, PetstoreUtil
18	CatalogFacade, CatalogXmlAction, Category
19	SearchBean, Tag, CatalogFacade, SearchIndex, HTMLParser, SQLParser, UpdateIndex, HTMLParser$CallbackHandler, Indexer, IndexDocument
20	ImageAction, FileUploadResponse, ImageScaler, FileUploadBean
21	CatalogXmlAction, SellerContactInfo
22	Address, ZipLocation, AutocompleteBean
23	CatalogXmlAction, CatalogFacade, Item, Product
24	AutocompleteBean

Para que esse conflito fosse resolvido, o requisito 10 necessitaria de uma adequação em sua especificação, para os casos em que as datas de expiração não estivessem válidas, pois, a alteração proposta no requisito 1 não poderia ser executada. Logo, o requisito 1 pode influenciar o requisito 10. Por outro lado, a influência do requisito 10 no requisito 1 pode ser analisada de forma análoga, considerando-se, por exemplo, a adição de novas regras para verificação de datas na especificação do requisito 10.

O segundo exemplo, mostrado na Figura 6, permite a discussão sobre requisitos com grande número de relacionamentos. Nessa figura é possível visualizar o requisito 11, que especifica os dados que o usuário pode manipular no cadastro de animais, e o requisito 18, que especifica quais são os atributos de uma determinada espécie de animal. Esses requisitos, destacados na Figura 6, especificam elementos que estão diretamente relacionados às regras de negócio do sistema. Logo, eles contribuem para a especificação de vários outros requisitos, tais como os relacionados à geração de pedidos de compra.

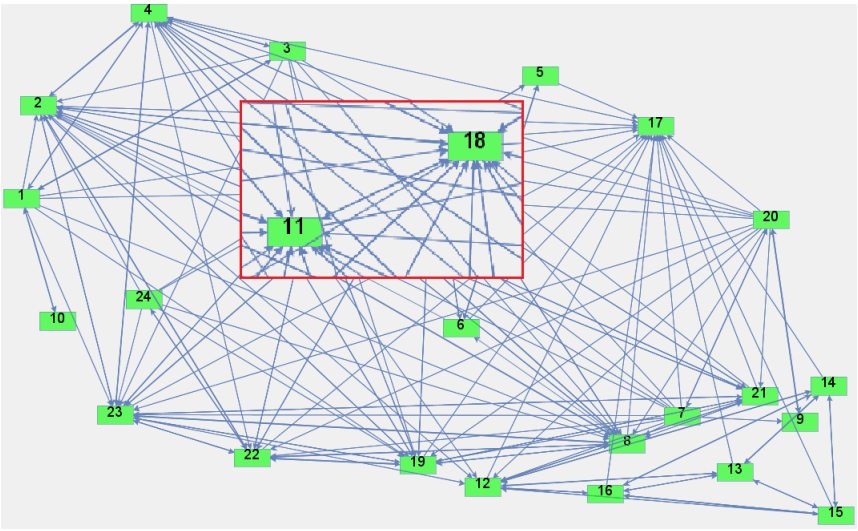

Figura 6. Exemplo de dois requisitos críticos ao sistema

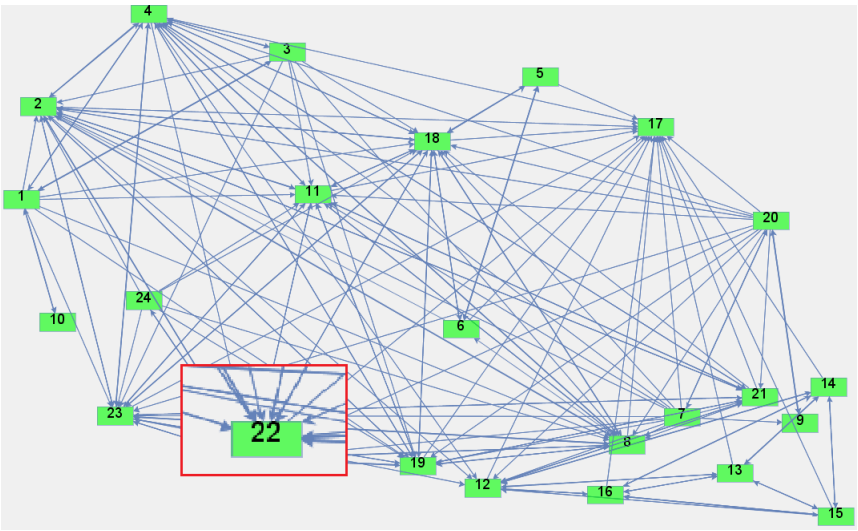

Figura 7. Exemplo de um requisito com alta probabilidade de modificação

A quantidade de arestas chegando até os requisitos 11 e 18, observada na Figura 6, permite concluir que vários outros requisitos dependem deles e que uma alteração possivelmente será propagada a um grande conjunto de requisitos do sistema. Uma vez que esses requisitos estão relacionados a diversos módulos da aplicação, isso corresponderá à necessidade de adequação desses módulos às novas especificações, implicando em um novo ciclo de desenvolvimento e testes da aplicação. Logo, esses requisitos podem ser classificados pelos responsáveis pela gerência de requisitos como sendo críticos, no sentido em que o impacto de sua alteração pode possuir custo muito elevado, em função do esforço necessário para aplicá-la.

O terceiro e último exemplo a ser discutido pode ser visto na Figura 7. Nela é possível verificar a situação do requisito de identificador 22 que especifica as formas de validação de dados, fornecidos nos campos de preenchimento obrigatório das telas de cadastro do sistema.

Uma vez que o requisito 22 está relacionado a um grande número de campos de entrada de dados, que por sua vez são especificados por diversos requisitos do sistema, ele pode ser considerado um requisito com alta probabilidade de modificação. Isso se justifica pelo fato de que basta a alteração em qualquer um dos requisitos aos quais o requisito 22 esteja relacionado, para que ele demande uma revisão em sua especificação.

Conforme pode ser verificado na Figura 7, existem diversas arestas que partem do requisito 22 para vários outros. Requisitos que sejam dependentes de um grande número de outros requisitos em um sistema podem apresentar uma manutenção de alto custo. Logo, a informação verificada na Figura 7, auxilia o responsável pela gestão do projeto desse sistema na identificação dos requisitos que podem causar maior impacto financeiro, tecnológico e no tempo de desenvolvimento da aplicação [Charette 2005].

Embora outros exemplos possam ser extraídos do grafo gerado, os três destacados acima mostram características importantes sobre o relacionamento entre os requisitos de um sistema. A partir da identificação visual desses relacionamentos, é possível executar uma avaliação mais bem estruturada sobre como um requisito influencia no desenvolvimento de um software. Se o objetivo dessa avaliação for identificar os requisitos cuja alteração possa afetar vários outros requisitos, ela pode começar por identificar, no grafo construído, aqueles requisitos que sejam representados por vértices com um grau de entrada maior do que um limiar definido empiricamente ou como resultado de uma análise histórica de projetos de software já finalizados.

Por outro lado, se o objetivo for verificar aqueles requisitos com maior probabilidade de alteração, isso pode ser feito visualizando-se no grafo construído, aqueles requisitos representados por vértices com um grau de saída elevado. Esse tipo de análise pode ser utilizado para indicar a necessidade da redução da conectividade entre os módulos associadas aos requisitos com maior chance de alteração e os demais módulos do sistema. Essa indicação se torna possível, uma vez que os relacionamentos entre os requisitos do grafo construído neste trabalho são resultado do mapeamento entre as dependências de módulos da aplicação, e o conjunto de requisitos que a especificam. Logo, a redução da conectividade pode ser sugerida, dado que módulos de requisitos altamente voláteis, possivelmente apresentarão essa natureza volátil.

A técnica de construção do grafo de requisitos proposta neste trabalho, e exemplificada neste estudo de caso, possui vantagens em relação às técnicas discutidas na Seção 2, porque ela é executada de forma automatizada, como resultado do mapeamento entre os módulos da aplicação e a especificação dos requisitos. No exemplo apresentado nesta Seção, utilizou-se o cadastro manual dos relacionamentos entre requisitos e módulos do *petshop* on-line, pois, na aplicação, não estava disponível recurso que permitisse recuperar esses relacionamentos automaticamente, como por exemplo, a utilização de anotações. Além disso, a técnica de construção de grafos de requisitos proposta neste trabalho dispensa a necessidade da avaliação de outros artefatos de software, tais como casos de teste, modelos arquiteturais e diagramas de classe. Cabe ressaltar ainda que a visualização dos relacionamentos entre requisitos, como por exemplo, a proposta neste trabalho, pode facilitar a capacidade cognitiva de compreensão do sistema, por meio de sua especificação [Nuseibeh and Easterbrook 2000].

5. Conclusão e Trabalhos Futuros

Em um processo de engenharia de requisitos, a manutenção da rastreabilidade de requisitos apresenta custo elevado na medida em que o desenvolvimento do software evolui. Como resultado da incorreta manutenção dessa rastreabilidade, está a redução na capacidade de observar o impacto que a alteração de um requisito possa causar nos demais. Neste trabalho foi proposta a construção automatizada de um grafo de requisitos, que visa contribuir com a resolução desses problemas. A construção realizada se deu por meio do mapeamento entre o grafo de dependências de módulos de uma aplicação e os requisitos por eles satisfeitos, permitindo manter a rastreabilidade de requisitos e apontar o impacto que a alteração em um deles poderia causar nos demais.

Por meio do estudo de caso descrito, foi possível, utilizando-se o grafo construído neste trabalho, elevar o nível de abstração das dependências entre os módulos de um *petshop* on-line, a um patamar equivalente ao da especificação funcional do sistema. Com isso, tornou-se viável utilizar a informação sobre essas dependências para identificar relacionamentos entre os requisitos do *petshop* on-line e indicar alguns daqueles requisitos que podem ser críticos ao sistema.

Essa criticidade pôde ser indicada, visualizando-se no grafo construído, que requisitos representados por vértices com grau de saída elevado, possivelmente terão alta probabilidade de sofrer alterações, uma vez que cada aresta que sai representa uma dependência do requisito de origem para o requisito de destino. Também foi possível identificar como requisitos críticos ao sistema, aqueles com alto grau de saída dos vértices que os representam, dado que a alteração nesses requisitos pode ser propagada a um grande número de requisitos do sistema. Dessa forma, a visualização do grafo de relacionamentos de requisitos do *petshop* on-line, pôde ser utilizada para um apontamento sobre os impactos que a modificação na especificação de um requisito possa causar nos demais.

Como trabalho futuro propõe-se a geração e a aplicação, no grafo modelado, de métricas da teoria de redes complexas para que seja possível avaliar quantitativamente o impacto que um requisito possa causar nos demais. A partir dessas métricas também será possível caracterizar formalmente o conjunto de requisitos críticos ao sistema no que se refere à grandeza do impacto por ele causado, resultante de uma possível modificação. A implementação da análise de códigos para geração do grafo de dependências para aplicações desenvolvidas em outras linguagens de programação diferentes do Java, utilizada no exemplo discutido no estudo de caso apresentado, também é um trabalho futuro que pode explorar mais a técnica proposta. Outro objetivo futuro é a validação da técnica proposta neste trabalho em um estudo de caso que contenha um número maior de requisitos e módulos, para avaliar sistemas de médio e grande porte.

Referências

Arnold, K., Gosling, J., and Holmes, D. (2005). *Java(TM) Programming Language, The (4th Edition)*. Addison-Wesley Professional.

Asuncion, H. U., Asuncion, A. U., and Taylor, R. N. (2010). Software traceability with topic modeling. In *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pages 95–104, New York, NY, USA. ACM.

Charette, R. (2005). Why software fails [software failure]. *Spectrum, IEEE*, 42(9):42 – 49.

Cheng, B. H. C. and Atlee, J. M. (2007). Research directions in requirements engineering. In *FOSE '07: 2007 Future of Software Engineering*, pages 285–303, Washington, DC, USA. IEEE Computer Society.

Cleland-Huang, J., Hayes, J. H., and Domel, J. M. (2009). Model-based traceability. In *TEFSE '09: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*, pages 6–10, Washington, DC, USA. IEEE Computer Society.

Cleland-Huang, J., Settimi, R., BenKhadra, O., Berezhanskaya, E., and Christina, S. (2005). Goal-centric traceability for managing non-functional requirements. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 362–371, New York, NY, USA. ACM.

Cleland-Huang, J., Zemont, G., and Lukasik, W. (2004). A heterogeneous solution for improving the return on investment of requirements traceability. In *RE '04: Proceedings of the Requirements Engineering Conference, 12th IEEE International*, pages 230–239, Washington, DC, USA. IEEE Computer Society.

Egyed, A. (2003). A scenario-driven approach to trace dependency analysis. *IEEE Trans. Softw. Eng.*, 29(2):116–132.

Gotel, O. and Finkelstein, C. (1994). An analysis of the requirements traceability problem. In *Requirements Engineering, 1994., Proceedings of the First International Conference on*, pages 94 –101.

Hayes, J. H., Dekhtyar, A., and Osborne, J. (2003). Improving requirements tracing via information retrieval. In *RE '03: Proceedings of the 11th IEEE International Conference on Requirements Engineering*, page 138, Washington, DC, USA. IEEE Computer Society.

IBM (2003). Developing pet store using rup and xde. `http://www.ibm.com/developerworks/rational/library/1072.html`.

IBM (2010). Ibm rational requisitepro software. `http://www-01.ibm.com/software/awdtools/reqpro/`.

Konrad, S. and Gall, M. (2008). Requirements engineering in the development of large-scale systems. In *International Requirements Engineering, 2008. RE '08. 16th IEEE*, pages 217 –222.

Nuseibeh, B. and Easterbrook, S. (2000). Requirements engineering: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 35–46, New York, NY, USA. ACM.

Oracle (2010). Java ee 6 at a glance. `http://www.oracle.com/technetwork/java/javaee/overview/index.html`.

Pressman, R. S. (2006). *Engenharia de Software*. MgGraw-Hill, São Paulo, 6 edition.

Ramesh, B. and Jarke, M. (2001). Toward reference models for requirements traceability. *IEEE Trans. Softw. Eng.*, 27(1):58–93.

Sommerville, I. (2007). *Engenharia de Software*. Pearson Addison-Wesley, São Paulo, 8 edition.