UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

FELIPE BARDEN

Uma Ferramenta de Traceability para Documentos

Trabalho de Graduação.

Prof. Dr. Sérgio Luis Cechin
Orientador

Porto Alegre, julho de 2013.

AGRADECIMENTOS

Dedico o trabalho aos meus pais, sempre prontos a fazer todo o possível para que eu alcance meus objetivos. Também gostaria de agradecer a minha namorada, Larissa, por toda compreensão e apoio que recebi.

Agradeço ao meu irmão por ser um exemplo de interesse e dedicação profissional e à minha irmã por sempre se preocupar comigo.

Agradeço ao meu orientador, Prof. Dr. Sérgio Luis Cechin, por ter aberto portas para novas possibilidades e dado oportunidade para que eu pudesse seguir em frente, pois esse trabalho não teria sido realizado sem esse apoio.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

TIM	*Traceability Information Model*
UML	*Unified Modeling Language*

LISTA DE FIGURAS

RESUMO

Neste trabalho é realizada a proposta e implementação de uma ferramenta para auxiliar as tarefas de rastreabilidade dos documentos de um projeto e de suas cláusulas. O método utilizado para captura de requisitos e funcionalidades desejadas foi baseando-se em artigos envolvendo problemas na rastreabilidade de requisitos de projeto, sendo essa nossa principal fonte de informações.

Para um melhor entendimento do trabalho, primeiro é feita uma introdução aos conceitos necessários, como rastreabilidade de requisitos e o modelo de informações de rastreabilidade. Também são apresentados os problemas encontrados em processos atuais e que motivaram a criação de uma nova ferramenta para suprir essa necessidade.

Após esse nivelamento inicial, é então descrita a ferramenta, com seus conceitos específicos (como os níveis de visualização), suas funcionalidades, pontos fortes e limitações.

Por fim, é feita uma avaliação da ferramenta e pontos que podem ser melhor desenvolvidos para facilitar a interação do usuário e seu acesso às informações do projeto.

Palavras-Chave: rastreabilidade de requisitos, especificação de requisitos, modelo de informações de rastreabilidade, rastreabilidade de documentos.

ABSTRACT

In this work, it is proposed and implemented a tool to assist the project's documents traceability job, as well as its clauses. The method used for gathering requirements and desired features was based on papers about projects's requirements traceability problems, becoming that the main source of informations.

For better understanding of this work, an introduction is done in the first part, where some necessary concepts, like requirements traceability and the traceability information model are presented. Some current and often found problems on the traceability process are also presented. They are the reason and the motivation that led to the development of this new tool and the main needs that the tool intends to meet.

After this initial leveling, the tool is then described, with their their specific concepts (such as viewing levels), their features, strengths and limitations.

Finally, a tool's evaluation is done, showing what points can be better developed to facilitate user interaction and to improve the access to project information.

Keywords: requirements traceability, requirements specification, traceability information mode, documents traceability.

1 INTRODUÇÃO

Os requisitos de um projeto podem ser coletados de múltiplas fontes, como necessidades de mercado, pedidos do cliente, regras comerciais, uma especificação de uma interface externa, um padrão ou regulamentação industrial entre outras. Entre a coleta dessas informações até a criação de uma especificação contendo e detalhando todas as funcionalidades e características a serem atendidas, podem existir várias etapas intermediárias no uso de técnicas de elicitação de requisitos e análise.

Após a criação desse documento chamado de Especificação de Requisitos, o projeto começa a ser desenvolvido. Cada funcionalidade desenvolvida deve estar relacionada a um requisito solicitado na especificação, assim como cada requisito solicitado deve estar relacionado a uma funcionalidade desenvolvida. Várias etapas podem surgir para cada funcionalidade no decorrer do projeto, como criação de casos de uso, definições de arquitetura, diagramas UML e casos de teste.

Durante a vida do projeto, os requisitos podem ir se transformando, sendo acrescentados, removidos ou modificados. De forma semelhante, uma implementação, caso de teste ou diagrama UML pode sofrer alterações e porventura não cumprir mais o requisito inicialmente planejado. Ao acompanhamento do cumprimento dos requisitos necessários, assim como o controle das modificações nos requisitos e suas contrapartes na implementação dá-se o nome de Gerência de Requisitos. Segundo o *Software Engineering Institute*, a "Gerência de Requisitos deve gerenciar os requisitos do produto de um projeto e componentes do produto e identificar inconsistências entre os requisitos e os planos do projeto e resultados do trabalho" (SEI, 2000).

Uma das tarefas do processo de Gerência de Requisitos é avaliar que todos os requisitos foram atendidos, possuindo uma contraparte na implementação do produto final. Outro ponto importante é a avaliação o impacto de alterações nos requisitos e na implementação, analisando se a alteração é isolada ou deve ser refletida nas demais etapas relacionadas. Caso uma implementação, na etapa final do desenvolvimento de uma funcionalidade, deva ser alterada, é fundamental a capacidade de rastrear quais casos de teste, casos de uso e requisitos serão afetados e propagar as modificações. Bem como, em uma alteração de requisito, regulamentação ou padrão, deve ser possível rastrear quais casos de uso, implementações e casos de teste devem ser revistos, com o objetivo de manter todos os relacionamentos consistentes e todas as necessidades do projeto satisfeitas. A capacidade de encontrar todas as etapas relacionadas a partir de qualquer outra etapa (requisito, caso de uso, caso de teste, entre outras) chama-se Rastreabilidade Bidirecional de Requisitos (ou *Bidirectional Requirements Traceability*).

Este trabalho realizará a apresentação de uma ferramenta desenvolvida para auxiliar a manutenção da rastreabilidade de um projeto e de seus documentos. Os principais objetivos traçados durante a concepção da ferramenta foram:

- permitir a visualização em diferentes níveis hierárquicos de organização, para uma melhor abstração dos dados apresentados;
- permitir uma navegação prática e dinâmica entre os diferentes elementos do projeto (documentos e seus segmentos, chamados de cláusulas);
- geração de relatórios e documentos com identificadores únicos para cada cláusula;
- realização de verificações a fim de garantir uma maior consistência nas informações.

Todas essas funcionalidades foram devidamente atendidas com algum nível de satisfação. O resultado alcançado em alguns dos aspectos desejados ultrapassou as expectativas, como a facilidade para navegar entre cláusulas relacionadas e o estabelecimento de um modelo para a criação das cláusulas, o que fomenta o projeto com informações utilizadas como base para testes. Durante o desenvolvimento, no entanto, chegou-se a conclusão de que, apesar da especificação inicial ser completa o suficiente para complementar o processo de rastreabilidade, ainda existiam várias possibilidades de melhorias na interação, o que proporcionaria uma experiência muito mais amigável ao usuário.

Para uma melhor compreensão dos conceitos utilizados e implementados na ferramenta, primeiro irá ser feita uma revisão sobre aspectos da rastreabilidade, sendo apresentados problemas atuais na prática da manutenção e a apresentação. A seguir, será descrita a ferramenta proposta, mostrando as soluções adotadas e apresentando características visuais da mesma. Então, será descrita a implementação da ferramenta, principais classes e forma de persistência utilizados. Por último, as conclusões sobre o trabalho.

2 RASTREABILIDADE DE REQUISITOS

A Rastreabilidade de Requisitos, como definida pelo IEC 61508, é “essencialmente uma análise de impactos para verificar (1) que as decisões tomadas nos estágios iniciais do projeto estão adequadamente implementadas nos estágios procedentes (rastreabilidade para frente), e (2) que decisões tomadas nos estágios finais do projeto são realmente necessárias e obrigadas pelas decisões iniciais (rastreabilidade para trás)” (IEC, 2000).

Segundo o definido pelo IEC 61508, a rastreabilidade para frente (*forward traceability*) tem a função de “verificar que um requisito está adequadamente tratado nos estágios posteriores do projeto” (IEC, 2000). Isso garante que o produto está evoluindo na direção correta, ou seja, que está sendo desenvolvido o produto certo. Se um certo padrão que deve ser atendido não é rastreável até um ou mais requisitos, então a especificação não está correta, podendo levar ao não atendimento de um padrão e à rejeição de mercado. Se um certo requisito não puder ser rastreável até uma implementação, então o desenvolvimento do projeto não está completo. A Figura 2.1 ilustra um exemplo de rastreabilidade para frente, conseguindo rastrear a origem de um requisitos aos requisitos e esses à suas implementação.

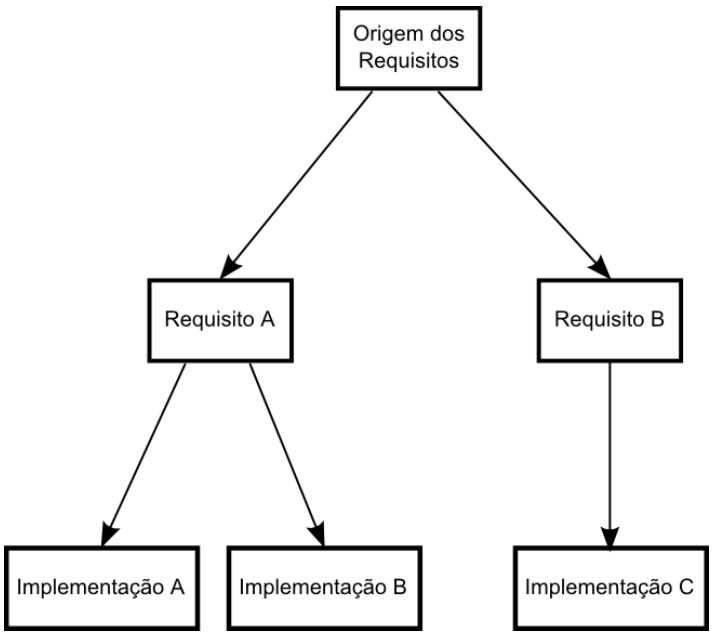

Figura 2.1: Rastreabilidade para frente

Também na definição do IEC 61508, a rastreabilidade para trás (*backward traceability*) é a função de "verificar que todas decisões de implementação (em um contexto amplo, não apenas implementação em código) está claramente justificada por um requisito" (IEC, 2000). A rastreabilidade garante que os produto sendo desenvolvido está devidamente alinhado aos requisitos iniciais ou modificados pela evolução do projeto, ou seja, que está sendo desenvolvido o produto corretamente. A falta de justificativa para uma implementação pode significar que uma justificativa realmente é necessária e deve ser acrescentada. Nesse caso, a rastreabilidade permitiu que uma inconsistência fosse localizada e também irá auxiliar na análise de impacto quando esse novo requisito for adicionado. A outra hipótese para a falta de uma rastreabilidade para trás incompleta é a de que a implementação não deveria existir. Implementações não relacionadas a requisitos apenas adicionam complexidade ao projeto sem que realmente supra uma necessidade do sistema. A ocorrência desse tipo de implementação é chamado de *gold plating* e surge a partir da implementação de uma funcionalidade por iniciativa do desenvolvedor, sem que essa funcionalidade tenha passado por todas etapas de desenvolvimento e não deveria fazer parte do produto. É uma implementação de alto risco, pois, por não ter passado pelas etapas necessárias para o desenvolvimento e não ser uma funcionalidade "oficial" do projeto, provavelmente sofrerá com problemas relacionados à não comunicação adequada. Dessa forma, a implementação poderá não ser testado adequadamente pela equipe de testes, poderá não estar na documentação final e assim por diante (Westfall, L, 2006). A Figura 2.2 demonstra um exemplo de rastreabilidade para trás, mapeando todas as implementações a um requisito e esses à sua origem.

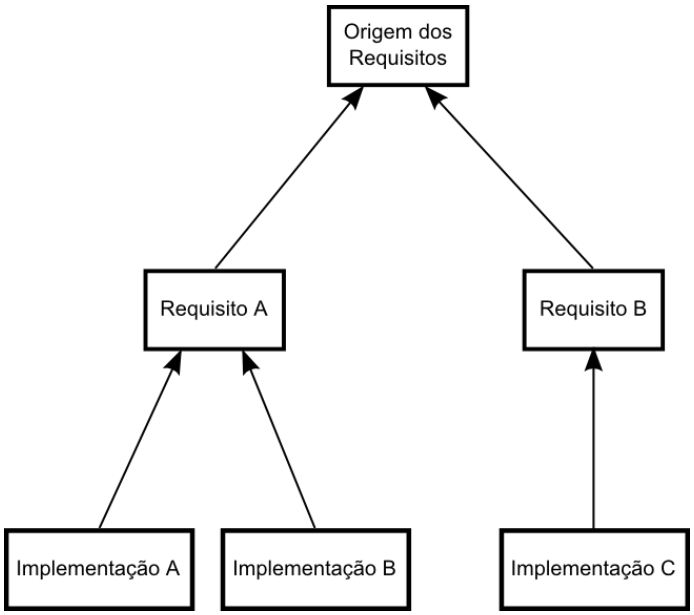

Figura 2.2: Rastreabilidade para trás

2.1 Modelo de Informações de Rastreabilidade

Uma das chaves para uma eficiente execução da rastreabilidade de requisitos do projeto é o planejamento prévio de como ela irá ser realizada. Uma das práticas básicas mais comuns é definir os tipos de artefatos utilizados, assim como as ligações possíveis entre eles. O conjunto dessas informações é chamado de Modelo de Informações de

Rastreabilidade (*Traceability Information Model*, ou TIM*)*. A esse modelo cabe a responsabilidade de guiar toda a posterior execução da implementação da rastreabilidade, orientando quais operações podem ou não ser realizadas, bem como servir de base para as verificações possíveis a fim de garantir uma maior consistência do processo. Outra vantagem conferida com o uso de um modelo é a possibilidade de automação de análises, testes e verificações.

Em Mäder, P. et al. (2009) é explorada a forma de definição e representação do TIM baseado no modelo UML Por ser um modelo largamente difundido entre desenvolvedores, a adaptação facilita a sua adoção, tornando a leitura e compreensão dos diagramas algo natural. Algumas características, como definição de tipos de artefatos e de ligações, definições de cardinalidade e definições de dependência são possíveis de serem representadas no diagrama UML e podem ser extremamente úteis nas verificações e na manutenção da consistência da rastreabilidade de um projeto.

A definição dos tipos de artefatos e ligações permitidas (Figura 2.3) é a base de criação de um TIM e uma das definições mais importantes da rastreabilidade de um projeto. Um projeto em que faltam tipos intermediários de artefatos provavelmente terá uma granularidade muito baixa. Caso isso ocorra, a rastreabilidade terá lacunas ou inconsistências na apresentação de definições, o que possivelmente recorrerá em falta de informações que seriam úteis para compreensão do problema. Demais problemas relacionados à definição de um TIM serão expostos adiante.

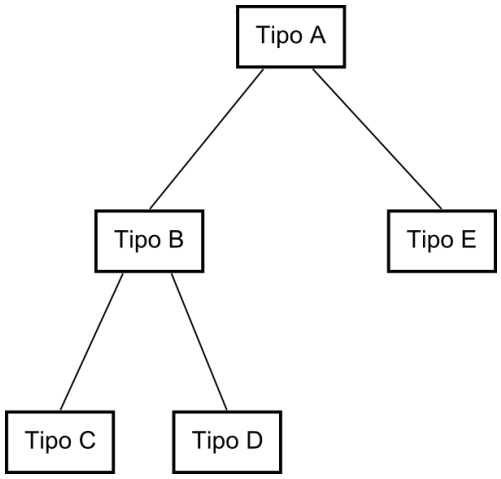

Figura 2.3: Definição de tipos e ligações

Outra representação aproveitada do UML é a de cardinalidade (Figura 2.4). Através dela, é possível determinar qual o número mínimo e máximo de instâncias de outro tipo necessários. Até que essa condição seja satisfeita, o cumprimento das especificações do projeto é considerado incompleto. A principal verificação possível acerca dessa definição é a completude da rastreabilidade para frente e para trás, verificando se algum artefato deveria possuir um ou mais artefatos filho ou pai, conforme definido em seu tipo. A outra análise, de forma semelhante à anterior, verifica se o número de artefatos pai ou filho ultrapassou o definido pelo tipo. Essa verificação serve para uma reavaliação sobre a granularidade esperada para o projeto e poderá levar a mudanças no TIM. Perceba como no exemplo da Figura 2.4, além da relação “1..*” (significando um ou mais), existem relações como “*” e “1”. Nesse caso, na relação entre o tipo B e D, o tipo B pode não possuir nenhuma ligação descendente com tipo D. Por sua vez, o tipo D

sempre possuirá apenas uma ligação ascendente e qualquer quantidade de ligações desse tipo excedendo essa quantidade caracterizará uma inconsistência.

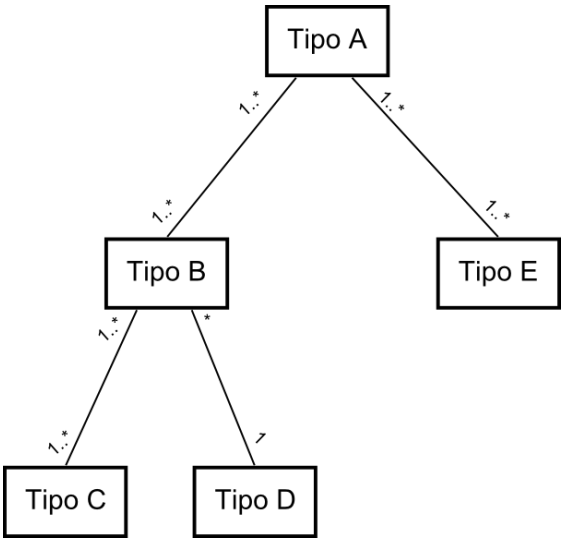

Figura 2.4: Definição de cardinalidade

A última característica possível de ser representada em UML e aproveitada nesse trabalho é a de dependência entre artefatos (Figura 2.5). Criando uma relação de dependência entre dois tipos de artefatos, estabelecemos um tipo dependente e outro independente. Essa definição possibilita a criação de um efeito de propagação de mudanças nos artefatos, de forma que, ao modificar o artefato independente da relação, faz com que o artefato dependente seja sinalizado como suspeito/desatualizado. Uma alteração no lado dependente da relação, não causa alterações no lado independente. Uma abordagem semelhante foi realizada em Leffingwell, D. et al. (2002). Dessa forma, melhora-se o cuidado na manutenção da rastreabilidade dos requisitos, assim como o planejamento do projeto, caso algum aspecto da implementação deva ser alterado. A dependência no padrão UML é representada por uma seta partindo do artefato dependente em direção ao dependente. No caso de dependência dupla, as duas pontas são setas. No UML, assim como em Mäder, P. et al. (2009), utiliza-se, além da seta, a linha tracejada. Na ferramenta proposta, no entanto, será utilizada a linha contínua, pois essa é uma característica comum entre as ligações e o uso do tracejado tornaria a representação das ligações menos evidentes. Também será utilizada a seta partindo do artefato independente em direção ao dependente. Tais mudanças também foram observadas em Leffingwell, D. et al. (2002) e é citado que grande parte das ferramentas de gerência de requisitos adota essas modificações.

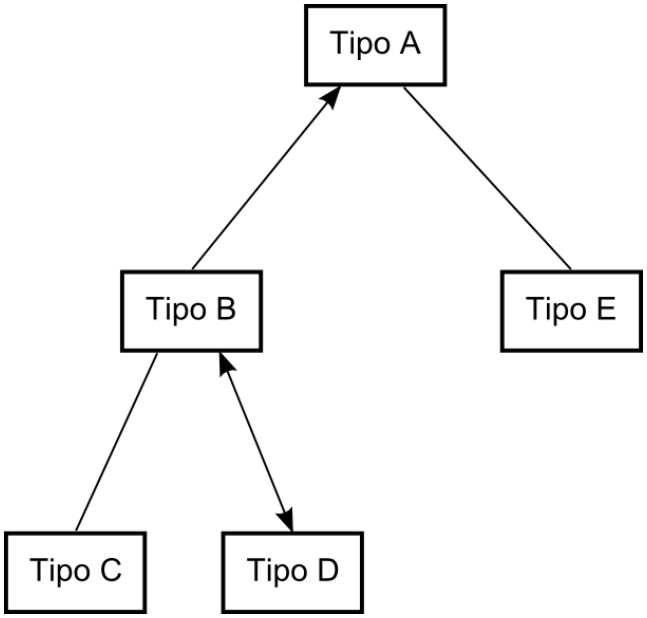

Figura 2.5: Definição de dependência

Uma forma mais complexa de relacionamento e propagação de alterações é apresentado Soonsongtanee, S. et al. (2010). Nesse artigo, explora-se a possibilidade da criação de uma máquina de estados para representar o resultado das mudanças nos artefatos envolvidos no relacionamento. A máquina de estados, pela definição, é customizável. No exemplo dado, utilizando os tipos Requisitos Funcionais e Requisitos do Usuário, são introduzidos estados como “rastreado em”, “suspeito”, “conflito”. O estado “rastreado em” é o estado normal e representa apenas que existe um relacionamento entre os artefatos. O estado “suspeito” é alcançado quando existe uma modificação nos Requisitos de Usuário, indicando que é necessário revisar o artefato Requisito Funcional. De forma semelhante, o estado “conflito” é gerado quando existe uma modificação no estado independente Requisito Funcional. O relacionamento voltaria ao normal após a alteração ou atualização do artefato afetado. Devido à dificuldade de implementação de uma máquina de estados customizável para representação dos diferentes estados dos relacionamentos, essa técnica foi adotada na forma explicada anteriormente, em relacionamentos de dependência. Assim, apenas os estados “rastreado em” e “suspeito” foram representados de maneira fixa, sem possibilidade de customização. No entanto, esse mecanismo seria uma possível melhoria para a ferramenta proposta.

2.2 Problemas atuais da rastreabilidade dos requisitos

Em Mäder, P. et al (2013), os autores analisam os problemas mais comuns presentes em dez documentos com informações da rastreabilidade de projetos entregues à US Food and Drug Administration. Foram identificados nove problemas recorrentes nas submissões realizadas, os quais comprometem a qualidade e consistência da rastreabilidade e de sua verificação/avaliação. Os erros apresentados são erros comuns, frequentemente cometidos e foram separados em três áreas:

- **problemas na definição** do método de rastreabilidade;
- **problemas na criação das ligações** entre artefatos;

- **problemas na apresentação** do documento entregue para análise.

2.2.1 Problemas de definição

São apresentados 3 problemas de definição:

- Falta de uma definição que oriente e regule o método de rastreamento;
- Granularidade muito alta, muito baixa ou híbrida;
- Existência de caminhos redundantes.

A **falta de uma definição** para regulamentar o método de rastreamento pode ter graves consequências na qualidade da rastreabilidade do projeto. É uma etapa do processo que deveria ser cuidadosamente planejada, pois todas etapas posteriores serão baseadas nessas definições iniciais. O documento contendo essas definições (o TIM), tem como função definir os passos necessários para uma decomposição de um problema até chegar às causas, soluções, implementações e testes utilizados. Por exemplo, caso o *hazard* esteja ligado diretamente aos requisitos do sistema, sem uma decomposição intermediária nas falhas que causariam o hazard, a análise do relacionamento entre o requisito e o *hazard* fica muito difícil.

Existem também problemas relacionados à **granularidade das ligações**. Um rastreamento com granularidade muito alta, muito baixa ou inconsistente prejudica a análise da resolução dos riscos e falhas.

A granularidade muito alta ocorre quando existem muitas ligações direcionadas a um único artefato no nível superior, sem que essas ligações contribuam de forma individual para compreensão do problema. De fato, tantas ligações individuais podem distrair a pessoa que analisa, fazendo com que informações realmente importantes não recebam tanto destaque. O exemplo dado para esse caso é o de um requisito que possuia ligação com cada um dos procedimentos de teste, cuja solução seria agrupar os procedimentos em único caso de teste (test case).

A granularidade muito baixa, por outro lado, também possui problemas relacionados. Quando existem artefatos no nível mais baixo (portanto, de um problema ou solução específica) ligados a um outro artefato de ordem mais alta extremamente genérico e abstrato. A solução para o caso seria substituir o artefato genérico por artefatos menores, mantendo-os no mesmo nível, ou criar um novo nível no TIM para maior especificação do problema genérico.

Outro problema relacionado à granularidade é a junção dos dois problemas anteriores, resultando em um documento com granularidade inconsistente, em momentos muito alta, em outros momentos muito baixa. A inconsistência na definição da granularidade torna o processo de quem revisa a rastreabilidade do projeto muito complicada.

O último problema relacionado à definição do método de rastreabilidade são os **caminhos redundantes**. Um caminho é considerado redundante quando existe mais de uma forma de sair de um artefato e chegar a outro. Essa característica deve ser evitada, pois são pontos de potenciais inconsistências. Qualquer modificação em artefatos com caminhos redundantes deve ser refletida em todas direções possíveis e, caso isso não ocorra, haverá discrepância nas informações apresentadas e a interpretação dos dados

poderá depender de qual caminho o usuário irá seguir. Existem casos, no entanto, em que os caminhos redundantes são necessários. O exemplo dado em Mäder, P. et al (2013) é o de que alguns requisitos podem estar representados primeiro em diagramas UML e então em código fonte, enquanto outros diretamente representados por um código fonte. Essas situações devem ser evitadas e utilizadas com consciência de que, no caso de uma modificação no artefato, ela deve ser refletida nos demais caminhos.

A solução utilizada para minizar a ocorrência dos problemas de definição é, ao início de cada projeto, forçar o usuário a criar um TIM, definindo assim tipos possíveis para as cláusulas, assim como os caminhos permitidos para cada tipo. Na definição do tipo, é possível estabelecer um mínimo e um máximo de ligações, auxiliando no controle da granularidade. Esses limites não restringirão a criação das ligações, porém servirão como base para uma verificação, alertando o usuário caso o número de artefatos ligados esteja fora dos limites. De forma semelhante, serão analisados os caminhos redundantes, tanto na criação do TIM quanto, no caso de usuário definir o TIM com redundâncias, nas ligações entre cláusulas.

2.2.2 Problemas nas ligações

Problemas nas ligações entre artefatos são comumente originados por uma definição de rastreabilidade mal feita ou do processo utilizado para manutenção da mesma. São eles:

- Artefatos sem um identificador único e representativo;
- Existência de ligações redundantes;
- Falta de ligações importantes.

O primeiro problema de ligação apresentado é o problema da **falta de identificadores únicos e representativos** para os artefatos. Um exemplo apontado é o de requisitos e teste compartilharem de um mesmo prefixo. Apesar da numeração diferente, o prefixo compartilhado torna a análise confusa e leva a conclusões erradas sobre os artefatos. O uso de identificadores que dependem do posicionamento, por exemplo, em uma tabela ou em uma seção, também é desaconselhado, pois qualquer alteração na tabela ou na seção levaria a uma nova numeração, podendo causar quebras nas ligações.

A **existência de ligações redundantes** também foi um problema encontrado em quase todos os documentos analisados pelos autores de Mäder, P. et al (2013). Além da simples duplicação de ligações idênticas na matriz de rastreabilidade, existe um caso complexo de duplicação, chamado de “redundância multinível”. Ela ocorre quando ligações semelhantes são realizadas em níveis de granularidades diferentes. Por exemplo, no caso de um artefato do tipo “classe” ligado a um artefato de nível superior (“requisito de sistema”, por exemplo). Esse artefato “classe” possui artefatos do tipo “método” ligados a ele. Porém, cada método individualmente já é ligado ao artefato “requisitos de sistema”.

Um dos objetivos da rastreabilidade de requisistos é avaliar se todos os riscos do projeto foram devidamente solucionados ou mitigados. A **falta de ligações importantes** é uma falha grave, pois assim não é criada a evidência de que todos os riscos, falhas e requisitos foram analisados, decompostos e solucionados. Ela pode se apresentar de

duas formas: artefatos que deveriam ter ligações com níveis mais baixos e não a possuem (janelas) e artefatos que deveriam ter origem em um artefato de nível superior e não a tem (órfãos).

A solução adotada para os problemas de identificadores únicos e representativos foi fornecer um prefixo configurável por documento, assim como por tipo de cláusula. Dessa forma, ao analisar o código de uma cláusula, já se sabe qual a origem e função da cláusula. A exibição desses prefixos será configurada para exibir apenas o prefixo de documento, apenas o prefixo de tipo ou ambos. A numeração será sequencial e gerada no momento da consolidação do projeto. Em uma segunda consolidação, será dada a opção de manter a numeração atual e adicionar as demais cláusulas com um índice especial, ou apagar toda numeração anterior e começar um novo processo de consolidação. Como o processo será automatizado, não existe a possibilidade de perda de referências na troca dos índices. Poderá ser disponibilizado no log de consolidação o relatório com a troca dos índices para consulta posterior.

Para o problema da redundância de informação, existem as soluções apresentadas para os problemas de definição. Uma vez que a redundância multinível é causada pela existência de caminhos redundantes e de granularidade inconsistente, as verificações realizadas devem evitar esse tipo de problema ou, no mínimo, alertar o usuário do potencial risco. A existência de ligações exatamente iguais duplicadas não poderá ocorrer, pois não são permitadas criações de links duplicados.

Para evitar o problema da falta de ligações importantes, a proposta é basear-se no definido pelo TIM do projeto. Na criação, uma cláusula sempre deverá ter ligação com uma outra cláusula de nível superior, exceto se o tipo da cláusula for de maior ordem possível, podendo ela assim ser uma cláusula “raiz”. No caso de alteração de um link ou remoção de uma cláusula, criando assim, possivelmente, cláusulas órfãs, uma nova ligação deve ser adicionada manualmente. Também haverá verificações por janelas e cláusulas órfãs com base no TIM, verificando se existe alguma cláusula sem ligação com cláusulas superiores e verificando que toda cláusula tenha uma ligação com uma cláusula do nível mais inferior.

2.2.3 Problemas de apresentação

Mesmo que todos os problemas referentes a criação e manutenção da rastreabilidade dos requisitos do projeto tenham sejam corrigidos, se os resultados tiverem uma apresentação confusa e de difícil leitura, pouco poderá ser utilizado do trabalho feito. As informações devem estar acessíveis de forma fácil e prática para que os benefícios da rastreabilidade sejam aproveitos. Como problemas de apresentação, citam-se:

- Falta de um documento informando o modelo utilizado;
- Apresentação das ligações de forma confusa;
- Rastreabilidade executada no fim do projeto.

Mesmo que a estratégia de rastreabilidade esteja bem definida, se o **documento informando o modelo utilizado não estiver incluso na documentação**, o revisor precisará despender de um tempo considerável para entender o relacionamento entre os diversos tipos presentes no documento, assim como a nomenclatura dos identificadores.

Se o TIM estiver presente, o revisor pode rapidamente interar-se do contexto do projeto e investir o tempo na análise dos requisitos.

A forma como os caminhos e *links* são representados também influenciam na legibilidade e praticidade da análise dos requisitos, sendo um grave problema se forem apresentadas **as ligações de forma confusa**. Tabelas com dezenas de colunas e páginas podem até conter todas as informações necessárias e estarem tecnicamente corretas, porém a dificuldade na verificação e relacionamento entre artefatos a tornam uma escolha não muito adequada. A forma de tabela pode ser mantida, porém deve-se dispor de mais formas de visualização, em geral parciais e mais focadas, de forma a auxiliar o entendimento e acesso a informação.

Um dos maiores problemas é o fato de deixar a análise da **rastreabilidade dos requisitos para a parte final do projeto**, considerando esse processo apenas com o fim de demonstrar que o projeto atende aos requisitos solicitados ou que mitiga as falhas e riscos desejados. Essa medida é considerada contraprodutiva, pois o desenvolvedor tem todos os custos da implementação da rastreabilidade de requisitos, no entanto não beneficia-se das vantagens do processo, pois o projeto está no final da vida do projeto. Além disso, caso o revisor perceba que o documento foi criado apenas por necessidade e não como parte do processo, pode começar a ter dúvidas sobre a qualidade do produto final.

Como solução para a apresentação dos resultados, a solução adotada é de possibilitar a criação de gráficos personalizados da rastreabilidade do projeto e de seus artefatos. Gráficos individuais gerados a partir das cláusulas de maior ordem até uma outra ordem desejada ou então gráficos a partir das cláusulas de menor ordem até a cláusula raiz. O gráfico do TIM do projeto, assim como descrição das abreviações utilizadas. A ferramenta também fornece todo suporte para alteração dos requisitos durante a vida do projeto, possibilitando que o processo seja contínuo, não apenas para fins de demonstração do atendimento de requisitos.

3 FERRAMENTA PROPOSTA

Com base nos conceitos, práticas e recomendações, foi concebida uma ferramenta com objetivo de auxiliar na criação, organização, navegação e visualização dos documentos de projeto. Será possível a criação de um modelo de rastreabilidade e seu devido uso para a manutenção da consistência do projeto.

3.1 Nomenclatura

Cláusula é um segmento de um documento. Representa um techo de texto com características e idéias comuns, podendo conter apenas uma linha ou vários parágrafos. Ela possui as seguintes informações internas:

- histórico de alterações (contendo a data, o responsável pela modificação e a modificação)
- texto da cláusula
- ligações para outros documentos e cláusulas.

Documento é formado por um conjunto de cláusulas. Em geral, cada documento é composto por cláusulas interligadas por um determinado objetivo.

Projeto é um conjunto de documentos relacionados.

3.2 Níveis de Visualização

As informações poderão ser visualizadas segundo três níveis diferentes e hierárquicos: nível de projeto (nível mais abstrato), nível de documento e nível de cláusula (nível mais detalhado).

3.2.1 Nível de Projeto

Na visualização de nível de projeto é possível identificar todos os documentos do projeto e os relacionamentos existentes entre eles. O relacionamento entre documentos é determinado pelas ligações das cláusulas que pertencem a eles. A partir dessa visualização, pode-se acrescentar e remover documentos, bem como alterar a ordem de documentos e cláusulas. A Figura 3.1 ilustra o conceito inicial do nível de projeto.

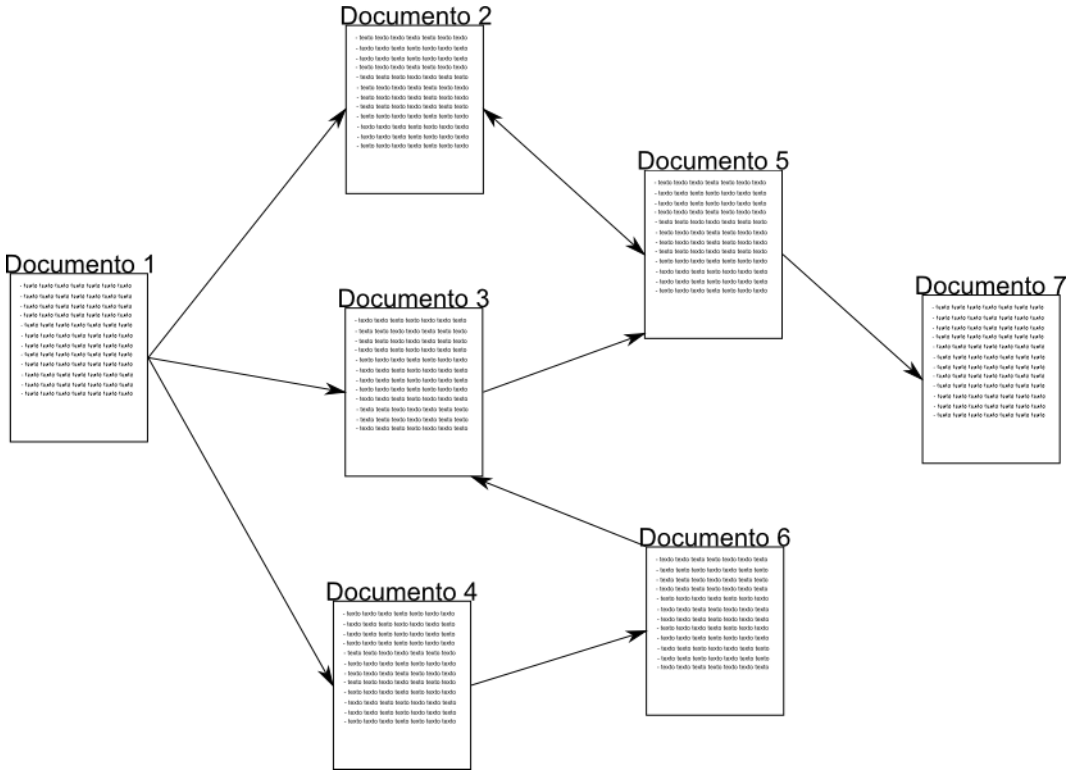

Figura 3.1: Ilustração conceitual da visualização de projeto

3.2.1.1 Criação de um projeto e definição do TIM

Ao iniciar o programa pela primeira vez, é necessário criar um novo projeto ou abrir um projeto existente. Uma janela de configuração no formato *wizard* será acionada solicitando o nome do projeto e o local de armazenamento. Após confirmar esses dados, virá a janela de edição do TIM do projeto.

A tela de TIM estará inicialmente vazia, exceto pelo texto "<adicionar tipo>", que deve ser clicado para iniciar a etapa de seleção de tipo (Figura 3.2). Uma nova janela aparecerá e o usuário terá opção de criar um novo tipo ou adicionar um tipo já existente. No caso de criar um novo tipo, o usuário deverá informar qual o nome desse novo tipo, o prefixo que deseja utilizar para as cláusulas do tipo e uma descrição para ser utilizada na documentação do TIM no relatório de consolidação. Caso escolha um tipo já existente, o usuário escolherá entre os tipos já criados e todas as características do tipo escolhido serão utilizadas. Isso inclui que todos os possíveis filhos do tipo escolhido serão adicionados à arvore de descrição. Caso seja adicionada uma nova possibilidade de tipo filho, a descrição será atualizada em todos os pontos de ocorrência do tipo pai. Em ambos os casos, as características relacionadas à ligação criada (envolvendo o tipo pai e o tipo filho) deverão ser informadas abaixo. No sentido do tipo pai para o tipo filho, deverá ser informado o número mínimo e máximo de filhos daquele tipo que o pai pode ter, além de informar se o filho é dependente do pai. De forma semelhante, no sentido do tipo filho para o tipo pai deverá ser informado o número mínimo e máximo de pais daquele tipo que o filho pode possuir, bem como a dependência do pai em relação ao filho. Esses mínimos e máximos não limitam a criação de filhos ou indicação de pais, no entanto são essenciais para a realização dos testes de granularidade onde essa inconsistência será informada. A característica de dependência servirá para o espalhamento de modificações, quando uma cláusula independente sofre uma

modificação e causa uma mudança de estado na cláusula dependente, que deverá ser revisada para confirmar a sua consistência.

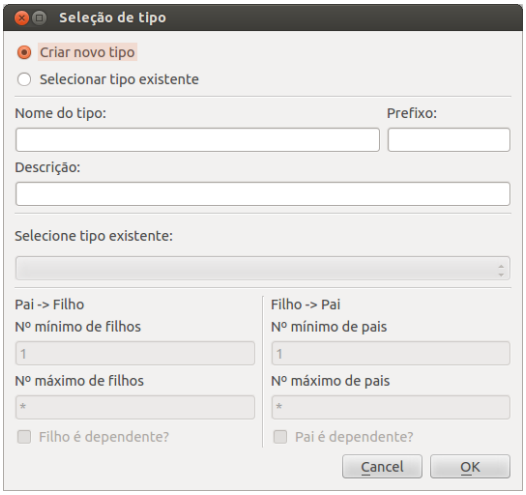

Figura 3.2: Tela de seleção de tipo

Após adicionar todos os tipos necessários, a tela estará apresentando de forma hierárquica todos os tipos. Na Figura 3.3, temos o resultado da criação de um TIM que possui os tipos Tipo 1, Tipo 2, Tipo 3, Tipo 4 e Tipo 5. Importante ressaltar como o Tipo 3 é descrito como um possível filho de Tipo 2 em suas duas aparições na definição do TIM, mesmo que essa informação só tenha sido dada em uma das árvores.

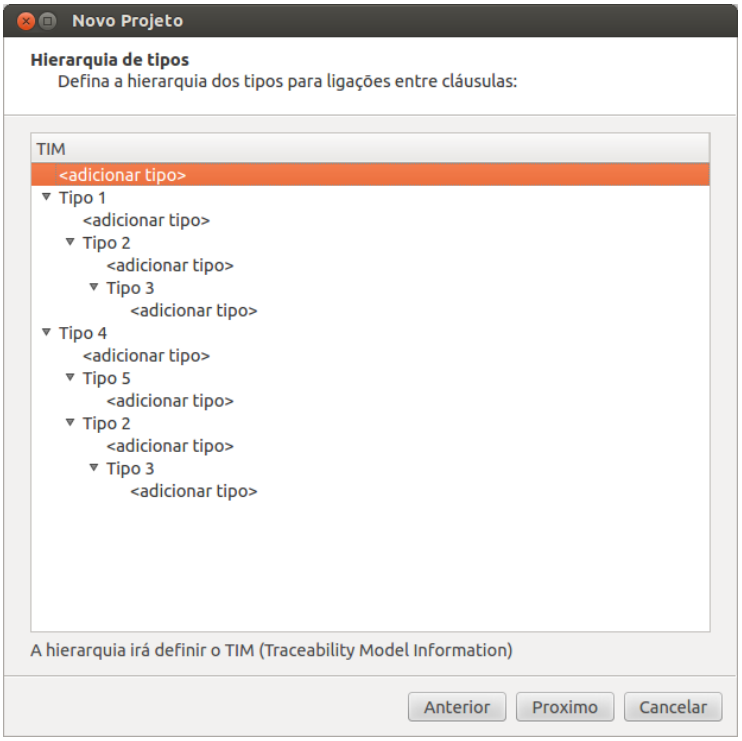

Figura 3.3: Tela de edição do TIM após definição de tipos

Após a confirmação do tipo, a próxima tela (Figura 3.4) exibe as informações de projeto inseridas, incluindo um diagrama do TIM, como suas definições de cardinalidade (números na base de cada ligação) e de dependência (setas direcionadas ao tipo dependente). Esse gráfico pode ser visualizado a qualquer momento do projeto no menu da ferramenta.

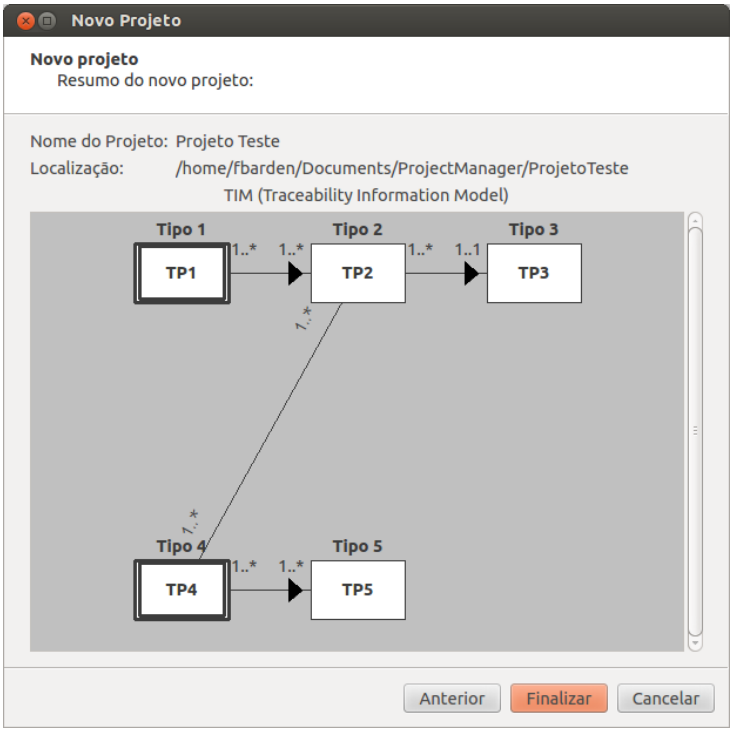

Figura 3.4: Tela de confirmação de novo projeto

Após a confirmação, será apresentada a tela da visualização de projeto com um único item com a opção de criação de um novo documento.

3.2.1.2 Visualização e edição em nível de projeto

Como definido na especificação, a visualização de projeto na ferramenta contém um diagrama representando os documentos e as ligações entre eles . As setas indicam a direção de cláusula pai para cláusula filha. No momento, o diagrama serve apenas para visualização, não sendo possível interagir com ele, porém, como foi implementado utilizando o próprio *framework* de desenvolvimento gráfico Qt, existe o suporte pronto a interatividade com o usuário.

Ao lado do diagrama, existe a lista dos documentos. Expandindo os itens de documento, é possível visualizar e selecionar as cláusulas pertencentes a ele. A edição de documentos e cláusulas pode ser feita no nível de projeto, nos botões abaixo da lista. Através deles é possível alterar a ordem de documentos e cláusulas, bem como excluí-los. A exclusão de um documento recorre em exclusão de todas as cláusulas contidas nele.

Tanto diagrama de documentos quanto lista de documentos e cláusulas presentes na visualização de nível de projeto podem ser vistos na Figura 3.5.

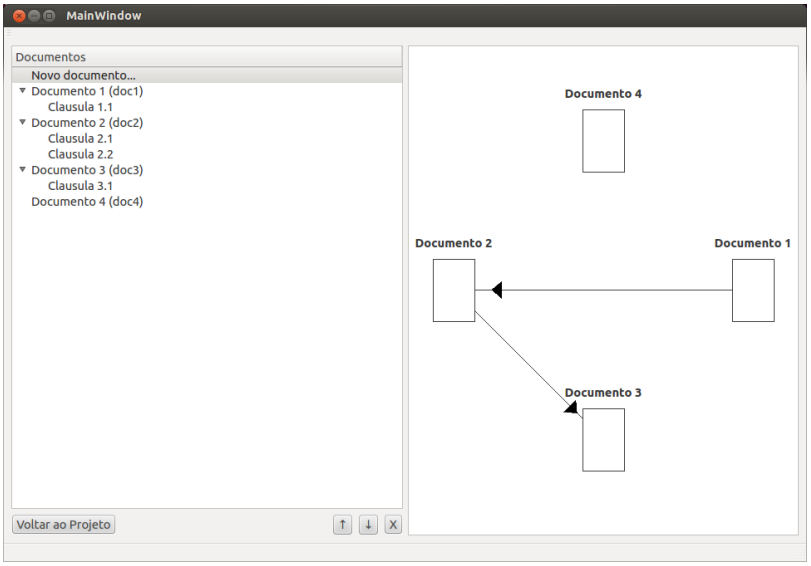

Figura 3.5: Tela da visualização de projetos

3.2.2 Nível de documento

Na visualização de nível de documento é possível inspecionar um documento do projeto. Nessa visualização os documentos são apresentados como uma lista de cláusulas, ao lado das quais são exibidos os documentos relacionados: à esquerda das cláusulas estão indicados os documentos ascendentes (aqueles cujas cláusulas originaram a cláusula que está sendo visualizada) e à direita estão listados os documentos descendentes (aqueles cujas cláusulas foram derivadas da cláusula que está sendo visualizada). A Figura 3.6 ilustra o conceito inicial do nível de documento.

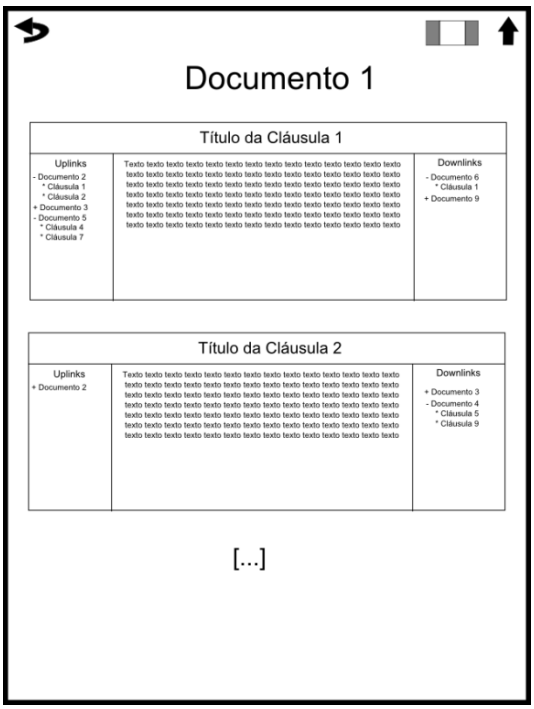

Figura 3.6: Ilustração conceitual da visualização de documento

3.2.2.1 Criação de um documento

Após clicar no item “Novo documento...” da lista de documentos da visualização de projeto, irá aparecer uma janela solicitando o título do documento, seu prefixo e o nome do documento. Essa última informação será utilizada para a criação do arquivo físico onde serão armazenados os dados do documento e de todas as cláusulas dele.

Terminada o processo de criação de um novo documento, a tela de visualização de documento irá aparecer. Ela não possuirá nenhuma cláusula, apenas um texto “Adicionar nova cláusula...” que deve ser selecionado para iniciar o preenchimento do documento com cláusulas.

3.2.2.2 Edição e visualização de nível de documento

A cláusula será apresentada na visualização de documento no formato de tabelas, contendo o título da cláusula no topo, o seu texto no meio e nos lados as ligações (Figura 3.7).

É possível acessar a cláusula do documento selecionando o título da cláusula. Ela então será carregada, passando assim para a visualização de cláusula. Essa é a única forma de alterar o texto ou qualquer outra propriedade de uma cláusula, não sendo possível realizar essas alterações na visualização de documentos.

Nas colunas de *links*, estão exibidos ao lado das cláusulas somente os documentos associados (documentos onde estão as cláusulas relacionadas). Isso foi feito para evitar problemas de formatação devido a muitas cláusulas ligadas. As cláusulas associadas podem ser visualizadas passando o mouse por cima do nome do documento ligado. Ao clicar no *link* do documento, ele será carregado na visualização, substituindo o documento atual. Para visualizar a cláusula associada diretamente, deve-se abrir a visualização de cláusula e clicar no *link* desejado, para então carregá-la, não sendo possível até o momento realizar essa operação diretamente na visualização de documento.

Outra funcionalidade presente é a possibilidade de retirar as colunas de links, acionando o botão “Links” abaixo da visualização. Essa funcionalidade foi desenvolvida no intuito de facilitar a leitura do documento, caso o interesse não seja a navegação entre documentos. Junto dessa funcionalidade, estão outros dois botões: um responsável por abrir uma janela para alteração da ordem e remoção de cláusulas, e outro responsável por exibir e tornar possível a edição das propriedades do documento, como título e prefixo. Ao alterar o prefixo de um documento participante da consolidação, o usuário será informado que essa alteração irá afetar os índices das cláusulas já consolidadas.

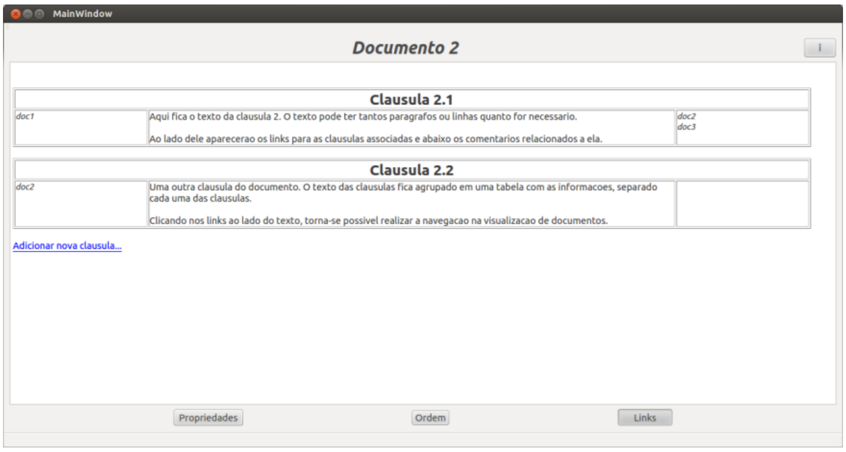

Figura 3.7: Tela de visualização de documento

3.2.3 Nível de Cláusula

Na visualização de nível de cláusula é apresentado o texto da cláusula, assim como suas ligações. Nessa visualização é possível realizar alterações no texto (inclusive de formatação) e nas ligações com as outras cláusulas. A Figura 3.8 ilustra o conceito inicial do nível de cláusula.

As alterações que forem realizadas serão registradas e adicionadas ao histórico de alterações da cláusula. Esse histórico pode ser consultado nessa mesma visualização.

Os documentos relacionados (códigos fontes, normas, entre outras) podem ser importados no projeto e ligados às cláusulas.

Além do texto principal, há um espaço para comentários e observações, que não aparecerão na versão final do documento, porém servirão para notas-guia para compreensão do texto final, assim como qualquer outra informação relevante. Dessa forma, é melhorada a reastreabilidade da cláusula, assim como sua retomada de contexto, se necessária. A inclusão desse campo foi influenciada por Gotel, O. C. Z., et al. (1994), onde é dito que um dos problemas da rastreabilidade é a dificuldade de retomada do contexto original da criação da especificação de requisitos.

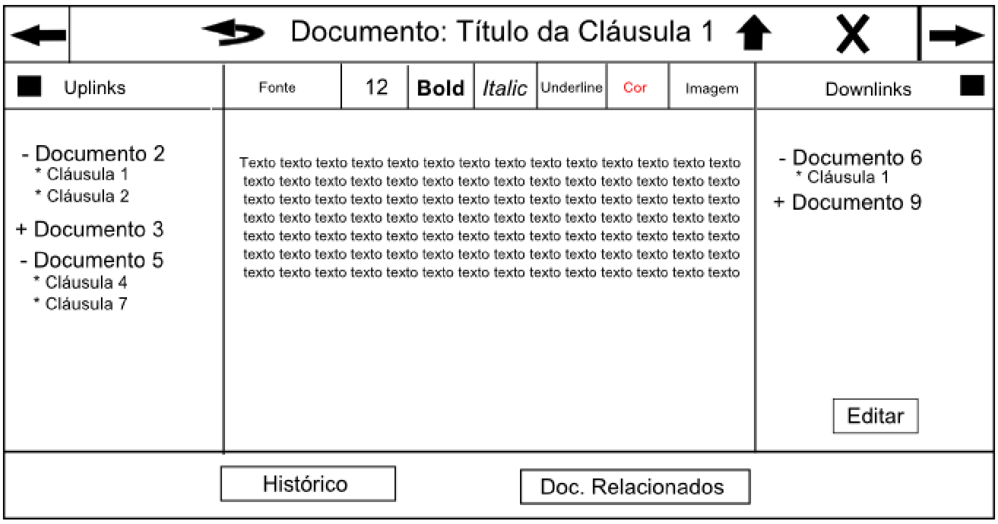

Figura 3.8: Ilustração conceitual da visualização de cláusula

3.2.3.1 Criação de uma cláusula

Existem três formas de iniciar o processo de criação de uma nova cláusula:

1) através do documento, clicando no texto “Adicionar nova cláusula...” sempre presente no fim do documento;
2) a partir de outra cláusula, no botão abaixo da lista de *links* filhos;
3) através do menu da ferramenta.

Ao criar uma nova cláusula a partir de um documento, força-se que a escolha do documento ao qual a nova cláusula pertencerá seja o documento atual. De forma semelhante, no caso de criar uma cláusula a partir de outra, a clásula deverá ser filha da cláusula atual, condicionando os tipos disponíveis na criação aos tipos filho possíveis para a cláusula. A única forma irrestrita de criar uma nova cláusula é a partir do menu da ferramenta, pois sua ação não está vinculada a nenhum documento ou cláusula.

Na janela de criação de uma nova cláusula (Figura 3.9), existem os campos de título e as caixas de seleção de documento, tipo e cláusula pai. Existem algumas coerências observadas na exibição da janela, como a restrição das cláusulas pai exibidas, que depende do tipo escolhido pelo usuário. Outra restrição é a possibilidade de criar uma cláusula sem definir um pai, opção somente possível para cláusulas raízes e cuja seleção torna-se explícita pela escolha do item “Sem Cláusula Pai”.

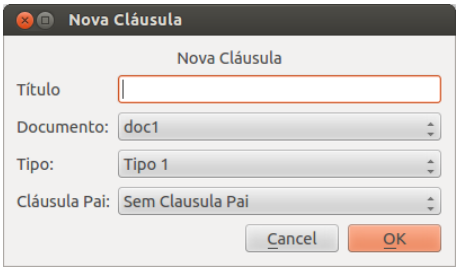

Figura 3.9: Janela de criação de nova cláusula

3.2.3.2 Edição de cláusula

A tela principal da visualização de cláusulas é a tela de edição (Figura 3.10), onde são exibidas as informações mais essenciais da cláusula, como título, documento a qual pertence, tipo a qual pertence, cláusulas pai (*uplinks*), cláusulas filho (*downlink*), texto e comentários.

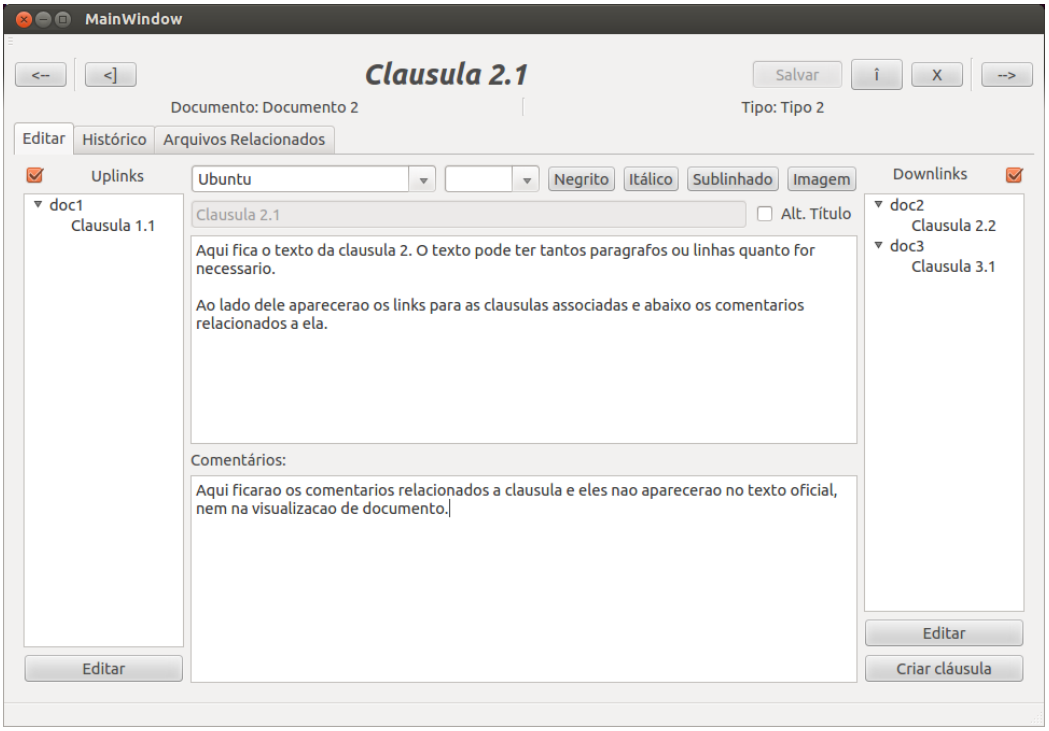

Figura 3.10: Tela de edição de cláusula

No espaço de título, o identificador consolidado irá aparecer entre parenteses caso a cláusula já tenha sido consolidada. É possível alterar o título marcando a caixa de "Alt. Título" e alterando o campo de texto ao lado.

Nos espaços para ligações (*uplinks* e *downlinks*), as cláusulas relacionadas são exibidas de forma muito semelhante à realizada na visualização de projeto, em forma de lista hierárquica. Ao selecionar uma das cláusulas associadas, ela é carregada e substitui a cláusula original na visualização, funcionalidade essencial para navegação no projeto. É possível editar as ligações, escolha na qual uma nova janela abre com a lista dos *links* e opções de adicionar ou remover cláusulas. As possibilidades de cláusulas seguem a coerência com o tipo da cláusula editada. É possível também criar uma nova cláusula filha, conforme descrito no item anterior. Os *checkbox* ao lado do texto *Uplinks* e *Downlinks* permitem esconder as colunas de ligações, permitindo uma visualização mais ampla dos textos da cláusula.

Na área de texto, existe suporte a edição e formatação do texto, bem como inserção de imagens. Essas funcionalidades ainda não estão implementadas, no entanto os textos são guardados no formato HTML, o que possibilita essa manipulação de formatos e seu posterior armazenamento. O espaço para comentários não tem prevista a facilidade de formatação de texto, porém seu texto também é guardado em HTML, o que torna possível a expansão da funcionalidade.

3.2.3.3 Histórico

A funcionalidade de histórico não foi implementada até o momento. A seção de histórico, no entanto, foi planejada para guardar todas as modificações feitas na cláusula, de forma a poder ter alguma forma de rastreabilidade temporal, principalmente em relação ao texto principal e cláusulas ligadas. Para executar a tarefa nas mudanças

de texto, seria utilizada a biblioteca *google-diff-match-apply*, que possui a habilidade de, além de verificar diferenças entre arquivos, aplicar essas diferenças em forma de *patch*, restaurando assim um estado anterior. Demais mudanças, como inclusão e exclusão de *link* ou arquivos relacionados seriam feitas na própria ferramenta, armazenando no *log* a informação sempre que uma mudança nesses campos ocorresse.

3.2.3.4 Arquivos relacionados

Selecionando a aba "Arquivos Relacionados", abre-se outra janela (Figura 3.11), contendo a lista de documentos que possuem alguma associação a essa cláusula. Existem duas opções nesse modo: importar um arquivo externo para o projeto e adicionar um arquivo à cláusula, como arquivo relacionado.

A opção "Importar" abre a janela de importação de arquivos para dentro do projeto, onde existe um campo para descrição do arquivo e outro para o local onde está o arquivo. É possível utilizar a opção de procura por navegação para localizar o arquivo. Confirmada a importação, o arquivo irá ser copiado para uma pasta dentro do projeto onde ficam todos os arquivos importados. Após esse procedimento, o arquivo estará disponível para ser adicionado às cláusulas.

A opção "Adicionar" abre uma janela onde serão exibidos todos os arquivos importados para o projeto com sua respectiva descrição. O usuário deve então selecionar o arquivo desejado e tem a opção de inserir uma observação específica daquele arquivo em relação à cláusula em que está sendo adicionado. Ao confirmar, ele aparecerá na lista de arquivos relacionados, bem como sua descrição e observação.

Ao selecionar o arquivo na tela, a ferramenta irá utilizar a aplicação padrão para execução do formato do arquivo para executá-lo. Esse foi o único ponto da ferramenta em que houve necessidade de distinção entre sistemas operacionais diferentes, porém em Linux e Windows a funcionalidade foi devidamente testada.

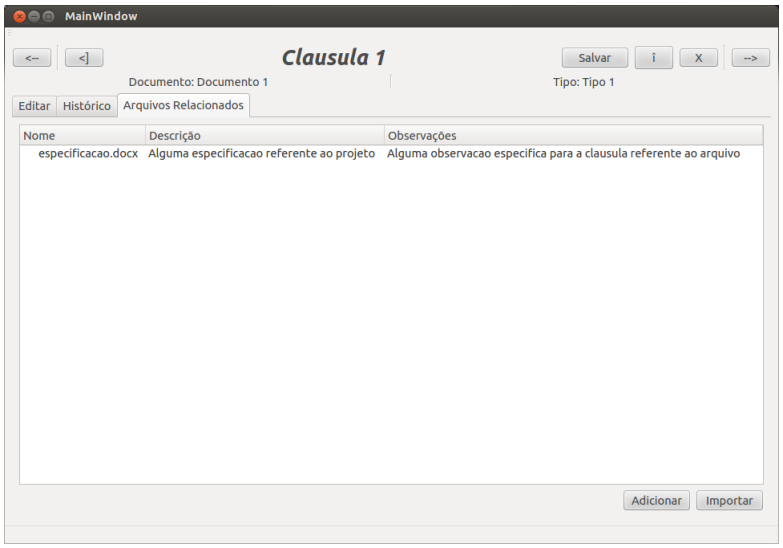

Figura 3.11: Tela de arquivos relacionados

3.2.3.5 Navegação e demais funcionalidades

Além da navegação por links, a visualização possui outras funcionalidades para navegação, como:

- ir para a cláusula anterior ou para a próxima cláusula (na ordem do documento)
- ir para um nível acima (visualização de documento)
- retornar para o *link* anterior (histórico de navageção).

Poderá ser feita também a exclusão da cláusula nesse modo, bem como realizar o salvamento da cláusula após as modificações.

3.3 Consolidação de projeto e verificações

As funções de consolidação tem como objetivo finalizar versões da análise de requisitos do projeto e realizar verificações sobre a rastreabilidade das cláusulas. É esperado que, após a consolidação, a especificação tenha uma maior consistência, tendo suas não-conformidades relatadas.

3.3.1 Consolidação de projeto

O processo de consolidação consiste em realizar diversas verificações nos elementos do projeto e gerar um documento com as cláusulas filtradas por documento e tipo, criando também identificadores únicos para essas cláusulas, baseadas no documento e/ou tipo. Além dos motivos já citados para a existência desse processo e de seus benefícios, uma das influências foi Gotel, O. C. Z., et al. (1994). Cita-se nesse artigo que um dos problemas comuns que dificultam a rastreabilidade é a desatualização da especificação de requisitos. Tornando possível a geração do documento de Especificação de Requisitos juntamente com a ferramenta de rastreabilidade, é possível mantê-lo atualizado e com todos os requisitos rastreáveis.

Ao selecionar a ação de “Consolidação de projeto” o usuário será perguntado se deseja iniciar uma nova consolidação, reconfigurando todas as opções possíveis de consolidação, ou se gostaria de apenas atualizar a consolidação anterior.

Caso decida iniciar uma nova consolidação, o usuário irá escolher quais documentos devem ser incluídos na consolidação. Após a escolha dos documentos, deve-se, da mesma forma, selecionar os tipos que farão parte da consolidação. Realizadas essas duas primeiras escolhas, o usuário deverá decidir os prefixos utilizados na geração de identificadores consolidados para as cláusulas (Figura 3.12). É possível utilizar os prefixos de documento (que auxilia na identificação da origem da cláusula), de tipo (que auxilia na fácil compreensão da função da cláusula), ambos ou nenhum (o que resultará apenas na numeração identificando a cláusula). Também deverá ser escolhida a ordem em que as cláusulas serão dispostas. As duas opções possíveis são:

1) por tipo, de forma a exibir os tipos de ordem mais alta primeiro;
2) como a ordenação atual dos documentos.

Caso a opção de ordenação por tipo seja escolhida, poderá ser habilitada a função de unificar documentos em um único, fazendo com que o documento gerado não possua separação de documentos.

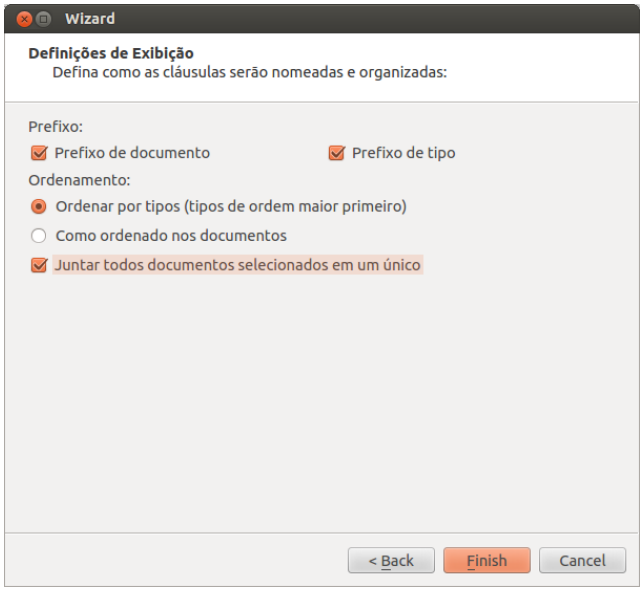

Figura 3.12: Seleção de opções de consolidação

No caso do usuário somente solicitar a atualização, todos os identificadores já consolidados serão mantidos e os demais irão apenas integrar-se. Essa possibilidade é importante para que não se perca a referência entre as cláusulas de diferentes consolidações. Caso decida-se manter os índices originais, será exibido, ao final da consolidação, um *log* com as modificações do projeto em relação à consolidação original.

Após o processo de consolidação, é gerado um documento no formato PDF com as cláusulas ordenadas e com os índices consolidados (Figura 3.13).

Ao consolidar os requerimentos do projeto novamente, o usuário terá a opção de apagar todos os índices e reconsolidar o projeto novamente ou então poderá manter os índices originais e mesclar as novas cláusulas com os índices posteriores, com a possibilidade de utilizar uma marcação para identificar as cláusulas adicionadas.

São inseridos ao final do documento, como anexos, relatórios da especificação do TIM, das cláusulas inseridas na última consolidação (no caso de uma atualização), nas mudanças dos identificadores consolidados (no caso de uma nova consolidação) e também de um relatório dos erros encontrados, como problemas de granularidade, cláusulas suspeitas, cláusulas incompletas e cláusulas órfãs.

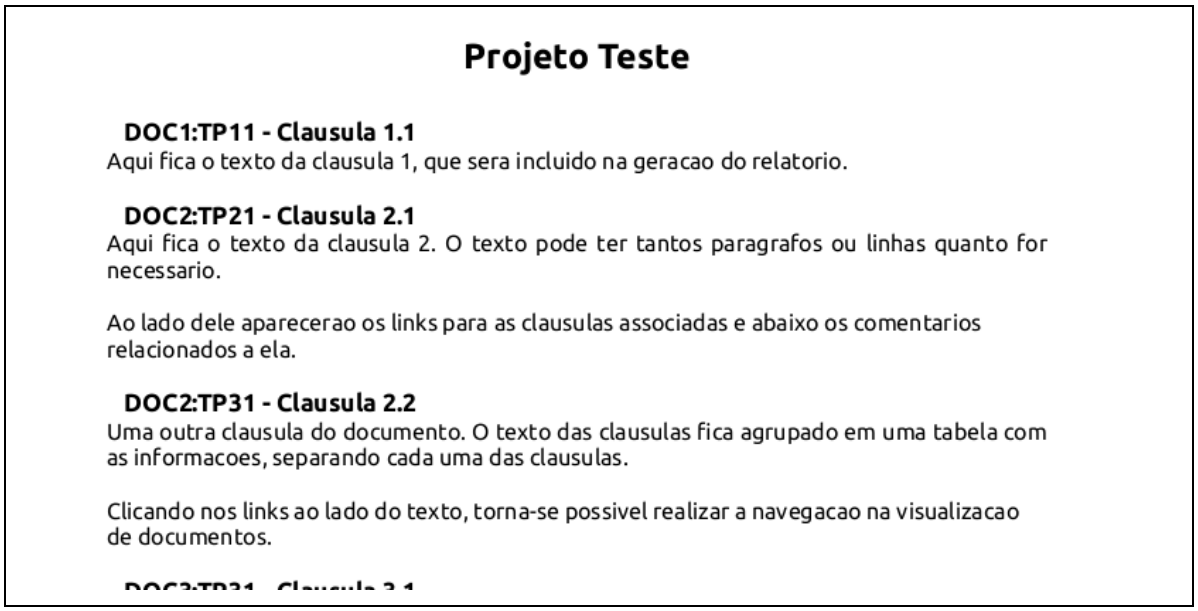

Figura 3.13: Documento gerado ao final da consolidação

3.3.2 Verificações

Existem certas verificações possíveis de serem feitas de forma automática, possibilitando que algumas características da especificação sejam mantidas consistentes. São elas:

1) Verificação de cláusulas órfãs: são cláusulas sem uma ligação com uma cláusula pai e que não são cláusulas-raíz. Essa verificação é feita a cada operação de adição ou remoção de *link* em uma cláusula, sendo possível acusar o problema no momento da alteração da ligação entre duas cláusulas.
2) Verificação de cláusulas incompletas: cláusulas que não possuem cláusula filha e que, segundo o TIM, deveriam possuir. É executada de forma semelhante a verificação de cláusulas órfãs, na operação entre ligações de cláusulas.
3) Verificação por cláusulas redundante: indica se existe mais de um caminho possível entre um tipo de cláusula até outro, potencialmente tornando a análise inconsistente. Esse tipo de verificação é somente feita por demanda, pois exige que os diferentes caminhos possíveis sejam percorridos. Uma versão mais simplificada do teste é executá-lo nas definições de TIM, pois este possui um conjunto consideravelmente menor de elementos e caminhos.
4) Dependência e cláusula suspeita: aconselha o usuário a revisar uma cláusula dependente, cuja cláusula da qual depende foi alterada. Essa verificação é feita a cada salvamento de cláusula. Ao executar essa operação, é emitido um sinal para todas as cláusulas relacionadas, indicando que a houve alterações. Assim as cláusulas dependentes podem entrar em estado de "suspeita" (Figura 3.14), indicando que deve ser revisada e confirmada sua consistência após as alterações.
5) Verificação de granularidade: verifica se as ligações da cláusula estão dentro do limite de cardinalidade estabelecido no TIM para aquele tipo. É semelhante em conceito às verificações de cláusulas órfãs e incompletas, no

entanto essa verificação analisa também os limites superiores. Esse teste foi concebido para instigar uma análise da granularidade dos tipos do projeto, podendo, através dos seus resultados, sugerir mudanças necessárias na definição do TIM ou uma possível separação de uma cláusula em várias.

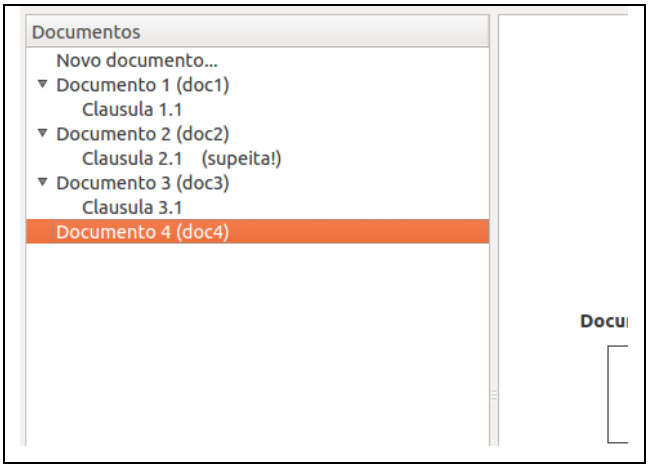

Figura 3.14: indicação de cláusula suspeita para revisão

3.3.3 Geração de diagramas de caminho

Outra funcionalidade presente para auxiliar a análise da rastreabilidade é a geração de diagramas de caminho das cláusulas. É possível escolher uma cláusula do projeto e gerar um diagrama contendo todas as cláusulas ligadas direta ou indiretamente a ela, bem como as ligações e caminhos que levam a cláusula em questão (Figura 3.15). O diagrama ainda contém informações de dependência, indicados pela seta.

Existem várias possibilidades de uso para esse tipo de gráfico que não estão implementadas atualmente, mas para as quais existe suporte, como inserir gráficos semelhantes no relatório de consolidação, ou então utilizar como uma visualização para navegar entre cláusulas, criando-se uma quarta forma de visualização: a visualização de caminhos.

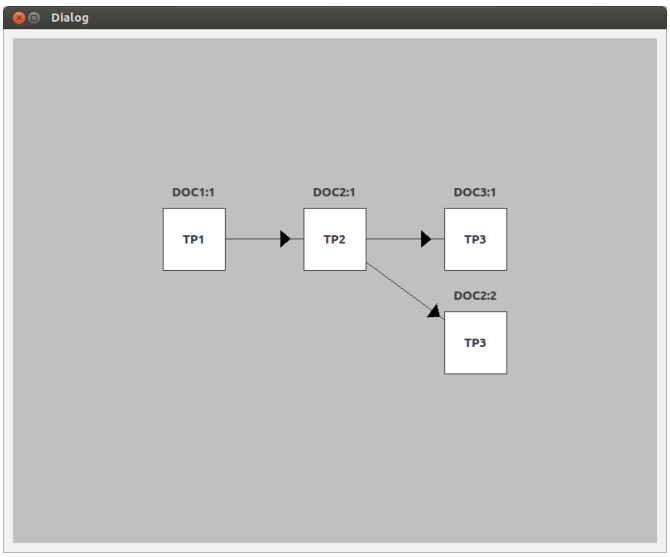

Figura 3.15: Diagrama de caminhos de uma cláusula

4 IMPLEMENTAÇÃO

Tendo os conceitos necessário e uma especificação definida, foi iniciada a implementação da ferramenta proposta.

4.1 Ambiente de Desenvolvimento

A ferramenta foi desenvolvida utilizando a linguagem de programação Python e o framework Qt, através da API PyQt4.

A base para a criação da interface gráfica foi o framework Qt (http://qt.digia.com/), que é um framework multiplataforma desenvolvido pela empresa norueguesa Trolltech, e é muito utilizado para criação de interfaces gráficas. Para auxiliar na implementação da interface gráfica, foi utilizado o Qt Designer (Figura 4.1), aplicativo fornecido pelos mantenedores do projeto Qt. Nesse aplicativo, é possível manipular o layout da interface diretamente, através de *drag-and-drop* dos elementos gráficos disponíveis, e visualizar o resultado imediatamente. É considerada uma boa prática utilizar essa etapa do desenvolvimento para organizar e renomear os objetos gráficos utilizados para futura referência no código Python (os nomes podem ser visualizados no campo superior direito da Figura 4.1). O resultado do layout é um arquivo XML com extensão ".ui", com a disposição e propriedades dos campos de textos, campos de edição, botões, entre outros.

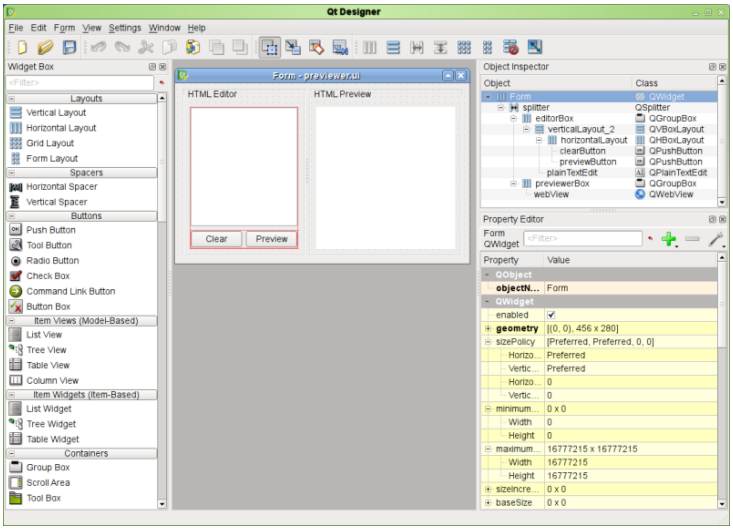

Figura 4.1: Qt Designer

Para realizar a interligação entre o layout desenvolvido no Qt Designer e o código *back end* desenvolvido em Python, existe a API PyQt4, desenvolvida pela Riverbank Computing (http://www.riverbankcomputing.com/software/pyqt/download). Uma das

ferramentas disponíveis no pacote é o *pyuic4*, que realiza a conversão do arquivo UI, gerado pelo Qt Designer, para uma implementação em Python (Figura 4.2). Esse código possui uma classe com responsabilidade de configurar o *widget* definido para base do *layout*(o *frame* mais externo, por exemplo). Entre os métodos da classe, o principal é o método "*setupUi*", que recebe como parâmetro o objeto base do *layout* e dispõe os demais elementos dentro desse objeto. No "*setupUi*" também são realizadas algumas das ligações dos sinais de evento com as respectivas funções ou métodos que devem ser acionados.

```
    class Ui_projectViewWidget(object):
        def setupUi(self, projectViewWidget):

projectViewWidget.setObjectName(_fromUtf8("projectViewWidget"))
            projectViewWidget.resize(400, 300)
            (...)
```

Figura 4.2: Código de layout gerado pelo pelo *pyuic4*

Foi criada uma camada intermediária de código, localizada entre o código de layout gerado automaticamente e o código *back end* desenvolvido. Nesse nível, foram feitas as personalizações necessárias do código do layout. Essa separação foi feita para evitar que alterações no layout realizadas no Qt Designer e, consequentemente, novas gerações de código fizessem com que as customizações fossem perdidas. A arquitetura utilizada para tal separação foi baseada em herança e composição. Uma nova classe foi criada para cada *widget* e a classe pai foi utilizada como classe base do *layout*. No método de inicialização foi adicionado um objeto da classe gerada automaticamente pelo *pyuic4* e executado o método *setupUi*, passando como parâmetro o objeto dessa nova classe criada. Assim, ao final dessa etapa, o objeto já possui todo seu *layout* e sinais de evento configurados, pronto para receber customizações que não puderam ser feitas no Qt Designer. A API PyQt4 é utilizada extensamente para essa tarefa, pois permite acesso aos atributos e métodos dos elementos do Qt4 através de classes que representam os componentes gráficos do framework.

Na camada *back end*, foram implementados os conceitos de projeto, documento e cláusula. O formato de arquivo utilizado para persistência dos dados foi o XML. Para a leitura a *parsing* do XML, foi utilizada a biblioteca ElementTree, já disponível nativamente no Python. Com ela, é possível ler os atributos e texto de cada *tag* XML.

Para a desvincular o aplicativo da necessidade de instalar o interpretador Python e o framework Qt no computador do usuário final, foram utilizadas as ferramentas "freeze" e "py2exe", que compila o código para plataformas Linux e Windows, respectivamente.

Para o controle de versão, histórico e backup do projeto, foi utilizado o sistema de versionamento GIT. O serviço de GIT utilizado foi o GitHub (https://github.com/). O projeto pode ser visualizado em https://github.com/fbarden/ProjectManager.

4.2 Classes base

Como base da ferramenta, foram implementadas as classes de cláusula (*Clause*), documento (*document*), projeto (*project*), além da classe de ligação (*link*), responsável por interligar as cláusulas e gerar relacionamentos. As classes utilizadas como base foram responsáveis por implementar os conceitos mais básicos da ferramenta, formando uma camada de *back end*. Às classes já citadas, soma-se a classe de TIM, representando o Modelo de Informações de Rastreabilidade, assim a classe que representa a definição dos tipos possíveis no TIM.

4.2.1 Classe Project

A classe *Project* é a classe que representa o projeto que está sendo desenvolvido. Ela contém todas as informações que não fariam sentido em uma cláusula ou documento individual, ou que devem ser gerais o suficiente para manter a consistência e padrão durante o desenvolvimento.

Entre os atributos pertencentes a classe, pode-se destacar:

- nome do projeto;
- o conjunto dos documentos do projeto;
- a definição do TIM utilizado para a criação de cláusulas;
- as configurações utilizadas na última consolidação de projeto, para o caso de uma consolidação de atualização;
- local de armazenamento do projeto;
- nome e descrição dos arquivos importados no projeto.

Os métodos presentes em *Project*, além de possibilitarem a manipulação dos atributos da classe, funcionam como uma interface para algumas solicitações mais comuns, porém complexas. Como exemplo desse caso, pode ser citada a solicitação de todas as cláusulas do projeto ou a requisição de um dicionário (estutura de Python que funciona como uma tabela *hash*, associando uma variável a uma chave qualquer) que converte o nome do documento em uma lista de cláusulas que a ele pertencem. Essas tarefas podem ser feitas utilizando a lista de todos os documentos, no entanto é um procedimento bastante comum, justificando a sua implementação um nível acima.

4.2.2 Classe Document

A classe *Document* representa o conceito do documento onde estão dispostas as cláusulas. A sua principal funcionalidade é agrupar cláusulas, tornando o projeto mais organizado. Assim as cláusulas podem ser organizadas conforme suas características comuns conforme o desejo do usuário.

Cada documento possui uma lista de cláusulas ordenadas pertencentes a ele. Possui também um nome (que será o nome do documento no armazenamento das informações), um prefixo (que poderá ser utilizado na consolidação para melhor identificação da origem da cláusula) e um título (que será utilizado para os momentos em que uma exibição mais descritiva é possível).

Os métodos da classe *Document* são utilizados na manipulação de seus próprios atributos. O método de aquisição de lista de cláusulas pertencentes ao documento e de

acesso ao objeto da cláusula através do documento são largamente utilizados em todo o código.

4.2.3 Classe Clause

A classe mais complexa da base da ferramenta, a classe *Clause* representa o conceito de cláusula. Nela estão contidas as informações de texto do documento, bem como as informações necessárias para realizar a ligação com outras cláusulas.

Como atributos da classe, estão definidas as seguintes propriedades:

- identificador utilizado para referência interna, gerado na criação da cláusula e nunca mais alterado;
- identificador consolidado pelo projeto, não relacionado ao identificador interno;
- histórico de alterações do identificador consolidado para acompanhamento e conversão no caso de troca;
- título da cláusula;
- texto oficial da cláusula, que irá ser utilizado na consolidação e geração de relatórios;
- comentários e anotações que serão utilizados para análises e retomadas de contexto feitas pelo usuário;
- definição do tipo da cláusula;
- lista de arquivos externos e importados pelo projeto que estão relacionados à cláusula (junto de possíveis observações daquele arquivo em relação à cláusula);
- lista de ligações com cláusulas filhas, para realização da rastreabilidade para frente;
- lista de ligações com cláusulas pais, para realização da rastreabilidade para trás;
- lista de marcações de erros e avisos relacionados à cláusula, como marcações de cláusula suspeita, órfã, incompleta, granularidade alta, granularidade baixa e em caminho redundante.

Os métodos da classe *Clause* servem para, além de manipular os seus atributos, interagir com as cláusulas com que possui relacionamento. Um exemplo disso é o método *emitChange*, que sinaliza para as cláusulas relacionadas que houve uma mudança no seu conteúdo e, portanto, elas devem analisar sua dependência (através do método *evaluateSuspect*) a fim de criar ou não uma marcação de "suspeita" devido a mudança. Outra característica presente nos métodos da classe *Clause* é a tentativa de facilitar a manipulação de *links*. Considerando que a definição de *link* está em uma classe separada, foi feita a tentativa de simplificar o acesso às cláusulas conectadas. Sendo assim, na manipulação de cláusulas, acessa-se diretamente a cláusula relacionada, sem necessidade de passar pelo *link* intermediador.

4.2.4 Classe Link

A classe *Link* é a classe reponsável por interligar as cláusulas e realizar a manutenção dessas ligações. A definição dessa classe foi a forma de isolar a classe *Clause* da classe *Project*. Assim, essas duas classes fariam a comunicação somente por meio da classe *Link*. Outro motivo para a existência da classe *Link* é a preparação para possíveis atributos e definições que possam vir a ser designadas como próprias da ligação individuais entre as cláusulas.

A classe *Link* possui como atributos o identificador das cláusulas interligadas e, após um processo de inicialização, a referência aos objetos instanciados das cláusulas. Foi necessário manter esses dois atributos separados, pois, ao carregar um projeto salvo em arquivo, nem todas as cláusulas foram criadas ainda, sendo criadas conforme são encontradas as informações no arquivo. Dessa forma, é necessário primeiro criar todas as cláusulas para então procurar referências para gerar as ligações.

Os métodos da classe operam na intermediação entre as cláusulas ligadas. Através do *link*, são adicionadas as cláusulas da ligação e são solicitadas informações, como identificadores de cláusula e documento.

Devido à evolução da ferramenta, a comunicação entre a cláusula e o projeto foi forçado a ocorrer sem a classe *Link*. De forma semelhante, as característica entre as ligações está armazenada dentro da classe *Type*. Qualquer característica de ligação diferente do estabelecido no TIM poderia implicar em um documento sem um padrão bem definido ou em cláusulas inconsistentes. A única vantagem atual para a separação entre ligações e cláusulas é uma facilidade na hora de representá-los graficamente, podendo-se claramente separar as etapas de desenho conforme os objetos requisitados, no entanto essa facilidade é subjugada pelas vantagens da simplificação do código. Analisando, portanto, o comportamento da ferramenta e a designação de responsabilidades dos objetos, percebe-se que a definição de *Link* pode não ser mais necessária, sendo razoável considerar a remoção dessa classe e a distribuição de suas funcionalidades pela classe *Clause*.

4.2.5 Classe TIM

A classe *TIM* é responsável por organizar as informações do modelo utilizado na construção da rastreabilidade do projeto. Além ter a função de agrupar e servir de ponto de acesso para todos os tipos definidos no modelo, armazena informações sobre quais tipos são raízes da estrutura, ou seja, quais tipos são do nível mais alto e podem não possuir um tipo pai anterior.

Os métodos da classe servem para manipulação das raízes do TIM e acesso às definições dos tipos instanciados da classe *Type*.

4.2.6 Classe Type

Na classe *Type* são definidos os tipos contidos na especificação do TIM. Cada tipo é encarregado de guardar as informações comum a todas cláusulas do mesmo tipo, como quais são os possíveis tipos para cláusulas filhas ou pais, número máximo e mínimo de cláusulas necessárias de cada tipo e informações relacionadas à dependência entre cláusulas interligadas.

As informações armazenadas na classe *Type* são:

- nome do tipo, para para identificação e exibição;

- prefixo do tipo, para auxiliar na criação de um identificador consolidado para as cláusulas;
- descrição da função/características do tipo, para utilização na geração das informações do modelo de rastreabilidade no relatório de consolidação;
- lista de tipos filhos permitidos, contendo, além do nome dos tipos possíveis, características relacionadas a ligação entre os tipos, como cardinalidades mínimias e máximas e dependência;
- Lista de tipos pais permitidos, contendo no mesmo formato e com as mesmas informações da lista de filhos permitidos;

Os métodos da classe *Type* servem para acesso dos atributos acima, bem como métodos para facilitar a análise de conformidades da cláusula e abstrair a estrutura de *Type*, principalmente relacionada às ligações possíveis e suas características. Entre essas funcionalidades estão o método *isDependentOf*, que analisa se um tipo é dependente de outro tipo, abstraindo a forma como a dependência entre tipos é implementada, e o método *checkCardinality*, que verifica se determinado número passado como argumento está dentro da cardinalidade destinada para uma ligação com determinado tipo pai ou filho.

4.3 Modelo de persistência

O método utilizado para a persistência das informações salvas foi o armazenamento em arquivos XML. Dessa forma, foi necessário estabelecer uma estrutura hierárquica a fim de organizar onde seriam colocadas cada uma das informações, bem como quais informações deveriam ser salvas e quais poderiam ser geradas na hora do carregamento.

No padrão XML, existem as *tags*, os atributos e os conteúdos (ou textos) (Figura 4.3). *Tags* são os nomes dos campos do documento. Relacionando um XML a uma estrutura em árvore, as *tags* seriam equivalente aos nodos da árvore. Os atributos são os valores atribuídos a cada uma das *tags*, de forma guardar informações relevantes na representação de suas características. Cada *tag* pode possuir tantos atributos quanto necessário. Os conteúdos são os valores existentes entre a abertura e o fechamento da *tag*. Cada *tag* pode possuir somente um único conteúdo.

```
<tag_principal>
  <tag_1 atributo="valor do atributo" atributo_2="valor do outro atributo">
    Conteúdo da tag 1
  </tag_1>
  <tag_2 atributo="valor do atributo" atributo_2="valor do outro atributo">
    Conteúdo da tag 2
  </tag_2>
  ...
<tag_principal>
```

Figura 4.3: Componentes de um XML

Foi decido armazenar os dados em dois tipos de arquivos: o arquivo de projeto e os demais arquivos de documento. Essa escolha foi feita baseado no fato de que o projeto possui várias características que devem ser representadas e são comuns a todo projeto, como o TIM, os documentos, os arquivos importados, local de armazenamento dos

arquivos entre outros, necessitando, portanto de um arquivo separado para essas definições. Os arquivos de documento possuem as definições de documento e ordem das cláusulas e, em uma subtag, cada uma das definições de cláusulas. Decidiu-se agrupar todas as demais definições de documento, cláusula e *links* em um único arquivo devido a diversos fatores:

- evitar um sistema de arquivos muito complexo e com alta granularidade;
- à necessidade de cada cláusula estar vinculada a algum documento, não existindo cláusulas sem essa característica;
- à necessidade de cada ligação estar vinculada a, no mínimo, uma das cláusulas;
- tanto documentos quanto *links* não possuem uma quantidade de informações suficientes para justificar a criação de um arquivo para cada;
- devido ao identificador da cláusula estar intrinsecamente ligado ao documento e, se retirado do contexto, não representa uma informação completa;

Mesmo no caso futuro de alguma representação/implementação de cláusula desviculada a algum documento, ela poderia ser associada a um documento abstrato, que representaria todas as cláusulas desvinculadas.

4.3.1 Arquivo de projeto

O arquivo de projeto, como já dito, é o arquivo que guarda todas as características comuns de todos os documentos e cláusulas do projeto, bem como as definições sob as quais o projeto será construído. Na Figura 4.4, é mostrada a hierarquia de *tags* em que o arquivo de projetos é organizada.

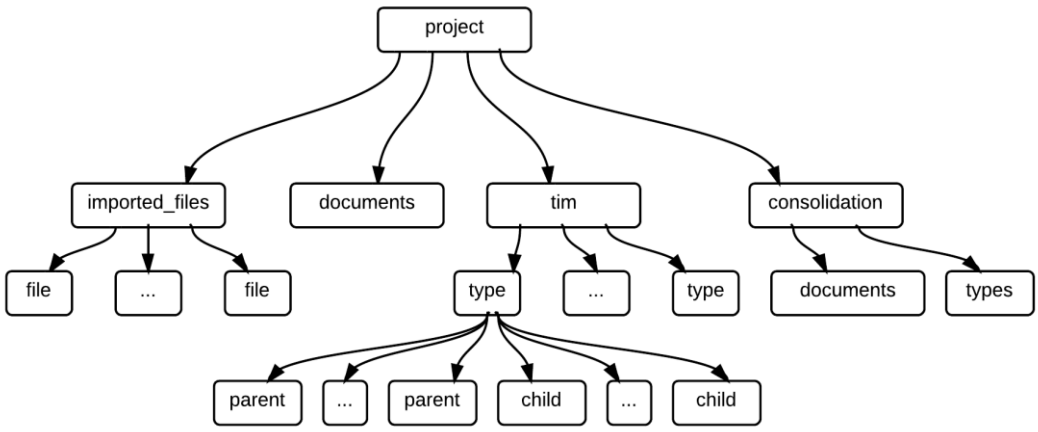

Figura 4.4: Hierarquia de *tags* XML do arquivo de projeto

A *tag* principal foi chamada de *project*. Ela contém como atributo somente o nome do projeto através do atributo *name*. Ela serve de raíz para todas as demais *tags* do arquivo. Dentro dela estão as *tags*: *imported_files*, *documents*, *tim* e *consolidation*.

Em *imported_files*, existem tantas *subtags* nomeadas *file* quanto necessárias. Essas *subtags* possuem um atributo chamado *name*, onde é indicado o nome do arquivo que foi importado para o projeto. O texto da *tag* possui a descrição do conteúdo importado.

A *tag documents* possui como conteúdo a lista ordenada de documentos que pertencem ao projeto, cada um deles separado por ponto-e-vírgula. O nome utilizado na informação é o nome do arquivo do documento, retirando-se a extensão ".xml".

De forma semelhante à *tag imported_files*, o nodo *tim* serve para agrupar diversas *subtags* do tipo *type*. Cada uma delas armazena as definições do tipo, possuindo como atributos os campos *name* e *prefix*. No caso do tipo descrito na *tag* ser do tipo raíz, ou seja, poder ser instanciado sem um tipo pai, o nodo ainda receberá um atributo *root* com valor igual a *yes*. Dentro das *tags type* ainda existe mais um nível contendo *subtags* do tipo *parent* e *child*¸ que definem as possíveis ligações com outros tipos e suas características. Ambos tipos de nodo contém os atributos *name* (que aponta o nome do tipo relacionado), *dependent* (indicando se o tipo descrito é dependente desse tipo pai ou filho), *minCard* e *maxCard* (estabelecendo os limites de cardinalidade do tipo em relação ao tipo associado).

Por último, temos o tipo *consolidation*. Nesse nodo são descritos as configurações utilizadas na última consolidação e são guardadas para a necessidade de atualização da mesma. Os atributos descritos nessa *tag* são *docPrefix e typePrefix*, que especificam as configurações de prefixo sob as quais a última consolidação foi feita, e *unifyDocuments*, guardando a configuração de unificação de documentos, na qual todas cláusulas contidas no documento são misturadas para geração da ordem. As *subtags* contidas em *consolidation* são *documents* e *types*. Elas listam os documentos e os tipos, respectivamente, utilizados como filtro para seleção das cláusulas a serem consolidadas.

Na figura 4.5, temos um exemplo de XML de arquivo de projeto.

```
<project name="Nome do Projeto">
  <documents>doc1;doc2;doc3;doc4</documents>
  <imported_files>
    <file name="filename1.ext">Descrição do arquivo importado 1</file>
    <file name="filename2.ext">Descrição do arquivo importado 2</file>
    <file name="filename3.ext">Descrição do arquivo importado 3</file>
  </imported_files>
  <tim>
    <type name="Tipo 1" prefix="TPU" root="yes">
      <child dependent="yes" maxCard="*" minCard="1" name="Tipo 2" />
    </type>
    <type name="Tipo 2" prefix="TPD">
      <parent dependent="no" maxCard="1" minCard="1" name="Tipo 1" />
      <child dependent="yes" maxCard="*" minCard="1" name="Tipo 3" />
    </type>
    <type name="Tipo 3" prefix="TPT">
      <parent dependent="no" maxCard="*" minCard="1" name="Tipo 2" />
    </type>
  </tim>
  <consolidation docPrefix="no" typePrefix="yes" unifyDocuments="no">
    <documents>doc1;doc3</documents>
    <types>Tipo 1; Tipo 2</types>
</project>
```

Figura 4.5: Exemplo de XML do arquivo de projeto

4.3.2 Arquivo de documento

Tendo poucas características próprias de documento para representar, a principal função do arquivo de documento é agrupar as cláusulas que pertencem a ele. A *tag* principal (*document)* possui como atributos o prefixo do documento, o seu nome e o título do documento. Possui como filhos os nodos *clauses* e *order*. *Clauses* é o *tag* utilizada para agrupar todas as cláusulas descritas no documento, enquanto *order* informa em seu conteúdo qual ordem elas devem ser exibidas, como uma lista de identificadores separada por ponto-e-vírgula. A Figura 4.6 exibe a hierarquia de *tags* do arquivo de documento.

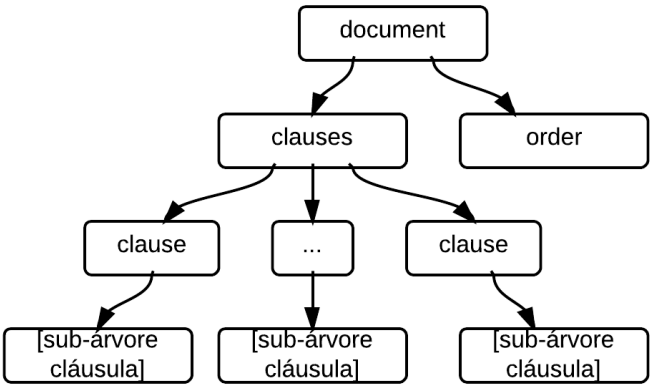

Figura 4.6: Organização hierárquica do arquivo de documento

Dentro de cada *tag* do tipo *clause* existem todas as informações necessárias para descrever uma cláusula e recuperar seu estado no ponto em que foi salva. No nodo *title* é armazeado o título da cláusula. Nas *tags text* e *comments* são armazenados em formato HTML o texto oficial e os comentários associados à cláusula, respectivamente. No nodo *suspects* é armazenada a lista de cláusulas modificadas que provocaram o estado de suspeita para revisão na cláusula. Essa informação está presente apenas para fins de manutenção da documentação da cláusula, pois após a atualização da cláusula (realizando uma operação de salvar), a lista será apagada e o estado de suspeita será removido. A Figura 4.7 exibe um detalhamento da hierarquica de *tags* da sub-árvore de cláusula do arquivo de documento.

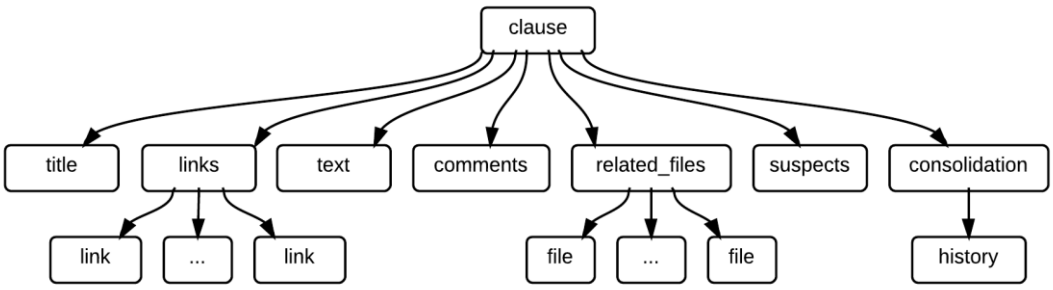

Figura 4.7: Detalhamento da estutura da sub-árvore cláusula

O nodo *links* indica as cláusulas filhas da cláusula descrita. Foi escolhido armazenar somente um dos sentidos das ligações, pois o outro sentido pode ser recuperado durante o carregamento das cláusulas. Considerando a falta de características individuais para cada sentido da ligação (elas são armazenadas no TIM do projeto), não há necessidade de armazenar as duas no arquivo de documento.

A *tag related_files* armazena os arquivos relacionados à cláusula. Seu único atributo é *filename* onde é escrito o nome do arquivo (com sua extensão). Não há necessidade de escrever o diretório, pois todos os arquivos importados encontram-se no mesmo diretório. O seu conteúdo possui a observação do arquivo realizada em relação à cláusula.

A *tag consolidation* é onde estão as informações de consolidação específicas à cláusula. Até o momento, o identificador consolidado é conservado no formato de atributo e o histórico de identificadores é salvo em formato de lista separada por ponto-e-vírgula no conteúdo da *tag*. Os primeiros identificados são os identificadores mais antigos, enquanto os últimos são os mais recentes.

Na figura 4.8, temos um exemplo de XML de arquivo de documento.

```
<document name="doc1" prefix="DOC" title="Documento 1">
  <clauses>
    <clause id="1" type="Tipo 1">
      <supects>clausula_modificada_1;clausula_modificada_2</suspects>
      <title>Título da cláusula 1</title>
      <text>Texto da cláusula no formato HTML</text>
      <comments>Texto de comentário, também no formato HTML</comments>
      <links>
        <link clause="2" document="doc1"/>
        <link clause="4" document="doc3" />
        [...]
      </links>
      <related_files>
        <file filename="arquivo.ext">Observação de arquivo relacionado à cláusula</file>
      </related_files>
      <consolidation id="DOC:TPU1">
        <history>DOC1;TPU1</history>
      </consolidation>
    </clause>
    <clause id="2" type="Tipo 2">
      [...]
    </clause>
  </clause>
  <order>1;2;3</order>
  </clauses>
</document>
```

Figura 4.8: Exemplo de XML do arquivo de documento

5 CONCLUSÃO

Apesar de não ter sido investida uma quantidade de tempo suficiente para pesquisa de ferramentas semelhantes disponíveis, é razoável considerar que a ferramenta proposta possui recursos interessantes para auxiliar a manutenção da rastreabilidade dos documentos de um projeto. A capacidade de criação de relatórios consolidados a partir de certos documentos e tipos de cláusulas permite ao usuário unificar em um único processo de rastreabilidade as etapas antes da especificação de requisitos e depois da especificação de requisitos. Ou seja, existe a possibilidade de explorar desde a coleta de requisitos através de normas, padrões e entrevistas com cliente, por exemplo, até as implementações e testes utilizados para validação dos requisitos, passando pela própria criação da especificação de requisitos, gerada na própria ferramenta.

Outro fator importante, considerado na criação da ferramenta, foi sua flexibilidade. A criação do TIM é feita de forma livre, definindo-se os tipos necessários e, portanto, pode-se criar um projeto compatível com quase qualquer processo desejado. Além disso, não existe forma fixa na criação da cláusula, podendo-se utilizar qualquer de descrição, sendo adaptável ao projeto.

Claramente, as vantagens existentes trazem juntamente algumas desvantagens. A própria flexibilidade acaba limitando a quantidade de testes e análises possíveis de serem feitas. Além disso, a forma livre de representação da cláusula cria uma margem a criações inconsistentes de cláusulas, sem todas as informações necessárias para o tipo.

Percebe-se, portanto, que existem diversos pontos a melhorar na ferramenta, dentre as possibilidades podem ser citadas:

- uma maior flexibilidade na apresentação das cláusulas, por exemplo, no relatório de consolidação, permitindo ao usuário configurar o relatório para ser exibido exatamente como ele gostaria,
- a exploração dos diversos pontos de interatividade, como os diagramas existentes na ferramenta, que atualmente servem apenas para exibição e possuem suporte para interações, como selecionar a cláusula desejada ou editar o componentes escolhido;
- possibilidade de reorganização de nodos dos gráficos através de *drag-and-drop*, pois algumas vezes os gráficos não estão dispostos de forma clara;
- inclusão de diagramas na criação de relatórios de consolidação;
- criação de *templates* para tipos de cláusula, o que poderia mitigar o problema da falta de forma das cláusulas e forçaria o usuário a informar todos os elementos necessários na criação de uma cláusula.

Para concluir, é possível dizer que a ferramenta obriga o usuário, se não a estabelecer, ao menos a considerar várias definições acerca do planejamento e execução da manutenção da rastreabilidade do projeto. Dessa forma, grande parte dos problemas apresentados nesse trabalho estarão mitigados ou eliminados. O maior problemas de todos, no entanto, não é facilmente corrigido por uma ferramenta: a falta de consciência da necessidade e dos benefiêncios de um processo de rastreabilidade bem executado.

REFERÊNCIAS

MÄDER, P. et al. "Strategic Traceability for Safety-Critical Projects," ***Software, IEEE*** , vol.30, no.3, pp.58,66, Maio-Junho 2013

WESTFALL, L. “Bidirectional Requirements Traceability”, **The Westfall Team**, Allentown, 2006, Disponível em: <http://www.compaid.com/caiinternet/ezine/westfall-bidirectional.pdf>. Acesso em: jul. 2013

SOONSONGTANEE, S.; LIMPIYAKORN, Y., "Enhancement of requirements traceability with state diagrams," **Computer Engineering and Technology (ICCET)**, vol.2, no., pp.V2-248,V2-252, 16-18 Abril 2010

MÄDER, P.; GOTEL, O.; PHILIPPOW, I., "Getting back to basics: Promoting the use of a traceability information model in practice," **Traceability in Emerging Forms of Software Engineering**, vol., no., pp.21,25, 18-18 May 2009

GOTEL, O. C Z; FINKELSTEIN, A. C W, "An analysis of the requirements traceability problem," **Requirements Engineering, 1994., Proceedings of the First International Conference on**, vol., no., pp.94,101, 18-22 Apr 1994

INTERNATIONAL ELECTROTECHNICAL COMMISION, “IEC 61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems”. **IEC**, 2000

SOFTWARE ENGINEERING INSTITUTE , “CMMI for Systems Engineering/Software Engineering”, Version 1.02, CMU/SEI-2000-TR-018, **Software Engineering Institute**, Carnegie Mellon University, 2000.

LEFFINGWELL, D., & WIDRIG, D., The role of requirements traceability in system development. **The Rational Edge, 2**, 2002.