TraceRG: Ferramenta Visual para Geração de Regras de Rastreamento entre Artefatos de Software

Rafael Ferreira da Silva[1], Maria Lencastre[1], Gilberto Cysneiros Filho[2]

[1] Escola Politécnica de Pernambuco, Universidade de Pernambuco, Recife, Brasil
{rfs, mlpm}@ecomp.poli.br
[2] Universidade Federal Rural de Pernambuco, Recife, Brasil
g.cysneiros@gmail.com

Resumo. A rastreabilidade tem sido reconhecida como uma atividade importante no processo de desenvolvimento de *software*. Relações de rastreabilidade podem garantir e melhorar a qualidade do *software*, e ajudar em diversas tarefas, tais como a análise de impacto, manutenção, verificação e validação. Apesar dos vários benefícios na prática da rastreabilidade, o custo para criar essas relações de rastreabilidade é alto. Para resolver este problema várias abordagens têm sido propostas para capturar as relações de rastreabilidade automaticamente. Algumas delas são abordagens baseadas em regra. Estas abordagens são eficazes, porém o processo de criação das regras requer em grande parte um especialista. Este artigo apresenta parte de um projeto de pesquisa em andamento para prover a rastreabilidade entre artefatos de *software* criados durante o ciclo de vida do desenvolvimento de *software*. Em particular, o artigo descreve a ferramenta TraceRG, que propõe resolver o problema de criar regras, com uma abordagem visual.

Palavras-chave: Rastreamento de s*oftware*, Regras de rastreabilidade, Xquery, Ferramenta.

1 Introdução

Rastreabilidade de *software* é a habilidade de relacionar artefatos criados durante o ciclo de vida do desenvolvimento de um sistema de *software*. Mais especificamente, do ponto de vista de requisitos, rastreabilidade foi definido como "a habilidade de descrever e seguir o ciclo de vida de um requisito, em ambos os sentidos, para frente e para trás (i.e. de sua origem, através de seu desenvolvimento e especificação, para a sua implantação e utilização posterior, e por períodos de refinamento contínuo e iteração em qualquer destas fases) [1]. A importância da rastreabilidade no processo de desenvolvimento de *software* tem sido defendido por vários padrões de gestão de qualidade e processos de melhoria, como CMMI [2] e SPICE [3].

Relações de rastreabilidade podem ajudar a garantir a qualidade do *software* e dão suporte a várias atividades do processo de desenvolvimento de *software*, tais como: manutenção e evolução, reuso, análise de impacto, verificação e validação. Apesar da

importância e do grande progresso na área de rastreamento desde o surgimento do chamado *traceability problem*, no início dos anos 90 [4], a prática de rastreamento ainda é um grande desafio. Um estudo recente feito pelo grupo do *Center of Excellence for Software Traceability* (CoEST) levanta um conjunto de desafios que foram documentados no "*Grand Challenge Document*" [5]. Exemplos desses desafios são: a necessidade de se definir a semântica dos relacionamentos de rastreamento; a falta de métodos e ferramentas que deem suporte à criação, visualização, manutenção e evolução dos relacionamentos de rastreamento.

A evidência da importância de ferramentas de suporte ao rastreamento também pode ser observada através do grande número de ferramentas comerciais, disponíveis no mercado, que fornecem suporte à captura de informação de rastreamento em várias atividades do ciclo de vida de desenvolvimento de *software*. Alguns exemplos são ferramentas voltadas para o gerenciamento de requisitos, gerenciamento de mudanças e de configuração, além de ferramentas de gerenciamento de projetos.

Contudo, a maioria dessas ferramentas requer alguma intervenção do usuário para conseguir criar as relações de rastreabilidade. Além disso, nesses tipos de ferramentas, o usuário tem que selecionar os elementos fonte e destino a serem relacionados. Porém, algumas ferramentas já fornecem algum mecanismo para auxiliar a definição das relações de rastreamento. Por exemplo, as ferramentas Rational DOORS [6] e CaliberRM [7] podem importar automaticamente requisitos de documentos em Microsoft Word baseando-se nos estilos do texto, enquanto a ferramenta Rational RequisitePro [8] pode importar os requisitos baseando-se em palavras chave. Contudo, uma vez que os requisitos tenham sido importados, as relações têm que ser identificadas manualmente.

Assim, a maioria das ferramentas comerciais refletem a ausência de solução para os grandes desafios da área. Por exemplo, elas não dão suporte à definição de diferentes tipos de relações de rastreamento, ou seja, não incluem apoio à definição da semântica das relações. Outra questão é a falta de automação, um problema sério no desenvolvimento de sistemas complexos, onde, em geral, a quantidade de artefatos é grande; há uma necessidade de se estabelecer relações de rastreamento entre esses artefatos, normalmente criados por ferramentas que não são interoperáveis e cujos artefatos evoluem independentemente.

Este projeto dá continuidade ao trabalho apresentado por Cysneiros [9] que apresenta uma abordagem para suportar rastreamento entre artefatos de *software* criados durante o desenvolvimento de sistemas multi-agentes, usando uma abordagem orientada a regras. As regras são criadas manualmente e isso requer um grande esforço e tempo, além de conhecimento específico da linguagem XQuery [10], uma linguagem padronizada pela W3C para consulta de documentos XML. Além disso, a abordagem de Cysneiros suporta apenas o desenvolvimento de sistema multi-agentes criados quando utilizando o *framework* i*, a metodologia Prometheus e a linguagem JACK.

A nossa proposta visa fornecer suporte a linguagens de modelagem e de programação diversas, bastando ao usuário configurar propriamente como se dará a entrada dos dados. Tais entradas devem variar de acordo com as saídas geradas por essas linguagens e ferramentas utilizadas, por isso a necessidade de uma configuração prévia, para

que a ferramenta consiga entender os novos formatos. Isso possibilita que a ferramenta trabalhe de forma genérica, conseguindo englobar tanto as limitações do trabalho proposto por Cysneiros, como abre possibilidade para outros tipos de abordagens.

Este artigo descreve uma ferramenta proposta para facilitar a geração das regras de rastreabilidade no *framework* proposto em [9]. O artigo se encontra organizado da seguinte forma: a seção 2 apresenta o referencial teórico sobre o trabalho de [9], base deste artigo. A seção 3 apresenta a proposta da ferramenta. A seção 4 apresenta trabalhos relacionados e, por fim, a seção 5 apresenta conclusões finais e trabalhos futuros.

2 Referencial Teórico

O trabalho de Cysneiros [9] foca na geração de relações de rastreabilidade entre modelos heterogêneos criados durante o desenvolvimento de *software*, e na identificação de inconsistências e incompletudes. Para gerar essas relações são criadas regras que definem a forma com que os artefatos irão se relacionar. A abordagem baseada em regras foi adotada, uma vez que as regras podem automatizar e auxiliar na tomada de decisão: se a condição de uma regra é satisfeita, então uma ação é executada. As regras podem ser usadas para a geração de relações de rastreabilidade e também para identificação de elementos ausentes, ou seja, aqueles que não possuem relação alguma de acordo com a regra.

A abordagem baseada em regras é utilizada em vários momentos do processo de desenvolvimento de software, desde a geração de modelos e rastreamento, como em [9], até para modelar o processo de desenvolvimento de software que encapsulam o desenvolvimento de atividades, tentando manter a consistência e possibilitando a automação [18]. Uma das desvantagens das abordagens orientadas a regras é a necessidade de se gastar um certo tempo, a priori, para entender o que precisa ser rastreável, e definir como esses relacionamentos podem ser identificados.

As regras utilizadas para expressar as relações em [9] são expressas em uma versão estendida da linguagem XQuery, e devem ser criadas por um especialista que defina os tipos de relacionamentos. A fim de apoiar a heterogeneidade dos modelos e ferramentas utilizados durante o ciclo de vida de desenvolvimento de *software*, o trabalho assume que os modelos são representados em XML. Esta escolha se deve a várias razões: (a) o XML apoia o intercâmbio de dados entre as ferramentas e aplicativos heterogêneos, (b) existe um grande número de aplicações que utilizam XML para representar informações internamente ou como um formato de exportação padrão, e (c) para permitir a utilização de XQuery como uma maneira padrão de exprimir regras de rastreabilidade.

A figura 1 apresenta uma visão geral deste *framework*, onde inicialmente os modelos são gerados em seus formatos nativos utilizando ferramentas proprietárias. Caso necessário, estes modelos são então traduzidos para o formato XML utilizando um tradutor de modelos

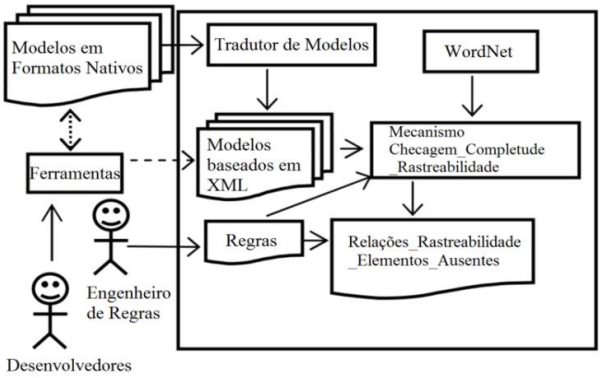

Fig. 1. *Framework* de rastreabilidade

Os modelos baseados em XML e as regras são usados como entradas para o Mecanismo Checagem_Completude_Rastreabilidade, componente utilizado para gerar as relações de rastreabilidade entre os modelos e identificar elementos ausentes.

Este componente também usa o WordNet para suportar a identificação de sinônimos entre os nomes dos elementos nos modelos. O WordNet é um componente importante porque em geral convenções de nome podem variar entre modelos de níveis mais altos (ex.: requisitos de negócio) e modelos de níveis mais baixos (por exemplo, código). O WordNet, apesar de possuir uma base de sinônimos, pode encontrar problemas de ambiguidade, problemas esses intrínsecos da linguagem natural. Uma das possíveis soluções seria criar uma tabela de sinônimos, a parte do WordNet, com as prováveis palavras que poderiam ocasionar problemas de ambiguidade. Essas palavras seriam definidas de acordo com o contexto ao qual é inserido o projeto, reduzindo assim o número de palavras a serem inseridas na tabela.

As relações de rastreabilidade e as informações sobre incompletudes e inconsistências são representadas em um documento XML (Relações_Rastreabilidade_Elementos_Ausentes). O uso de um documento separado para representar as relações de rastreamento e informações sobre incompletudes e inconsistências é importante para preservar os modelos originais, para permitir o uso desses modelos por outras aplicações e ferramentas, e para permitir que as relações geradas sejam usadas para suportar a identificação de outras relações de rastreamento que dependem da existência das relações previamente identificadas. Como mostrado na figura 1, o documento "Relações_Rastreabilidade_Elementos_Ausentes" é utilizado como entrada para o "MecanismoChecagem_Completude_Rastreabilidade" para dar suporte à geração das relações de rastreabilidade.

Foram identificados nove diferentes tipos de relações de rastreabilidade entre vários elementos nos modelos utilizados na abordagem de Cysneiros. Os tipos de relações de rastreabilidade são *overlaps*, *contributes to*, *uses*, *creates*, *achieves*, *depends on*, *composed of*, *send* e *receive*. Essa classificação é baseada nos tipos propostos na literatura como *overlaps*, *contributes*, e *depends* [11]. Além de alguns tipos relaciona-

dos a sistemas baseados em agentes como *uses, creates, achieves, composed of, sends* e *receives*. Esses novos tipos foram identificados de conceitos do i*, Prometheus e JACK. Como exemplo de relação temos o *overlaps,* onde neste tipo de relação um elemento e1 *overlaps* um elemento e2, se e1 e e2 referenciarem elementos com aspectos comuns. A tabela 1 mostra exemplos de relações entre elementos do Prometheus e i* *Strategic Rationale* (SR). Porém, nada impede que possamos ter novos tipos para outros contextos.

Tabela 1. Relações entre elementos Prometheus e i* SR

*i** Prometheus	SR Goal	SR Resource		SR Task
Goal	Overlaps	---		Overlaps
Role	Achieves	Uses	Creates	Achieves
Agent	Achieves	Uses	Creates	Achieves
Capability	Contributed by	Uses	Creates	Achieves
Plan	Achieves	Uses	Creates	Achieves
Action	---	---		Overlaps
Data	Used by	Overlaps		Used by
Scenario	Composed of	Uses	Creates	Composed of

Pode-se definir, por exemplo, uma regra em que a condição é que, se uma *task* da Strategic Rationale (SR) de um ator que representa um sistema multi-agente, e um *goal* em Prometheus são semelhantes, então uma relação de rastreabilidade *overlaps* pode ser criado entre a *task* SR no i* e o *goal* em Prometheus. A figura 2 mostra um exemplo de como o componente "Mecanismo Checagem_Completude_Rastreabilidade" processa a regra entre o *goal* "Arrumar Entrega" no Prometheus e a *task* "Organizar entrega" em i*.

A regra primeiro verifica se as palavras utilizadas para dar o nome para o *goal* em Prometheus são sinônimos para as palavra utilizadas para dar o nome para a *task* em SR. Em "Arrumar entrega" e "Organizar entrega", "Arrumar" é sinônimo de "Organizar" de acordo com o dicionário WordNet, e "entrega" e "entrega" são a mesma palavra. Em seguida, a regra verifica se os *sub-goals* do *goal* "Arrumar entrega" e as *sub-tasks* da *task* "Organizar entrega" são semelhantes. A semelhança é calculada com base nos nomes dados aos elementos e um *threshold*. Neste caso particular, o *threshold* é de 50%, isto significa que se mais de cinqüenta por cento dos nomes dados aos *sub-goals* do *goal* "Arrumar entrega" em Prometheus coincidirem com (ou seja, sinônimos) os nomes dados as *sub-tasks* da *task* "Organizar entrega" em i*, então eles são semelhantes. "Adquirir opções de entrega" é sinônimo de "Obter opções de entrega" e "Calcular tempo estimado de entrega" é sinônimo de "Computar tempo estimado de entrega". O percentual de *sub-goals* do *goal* "Arrumar entrega" que é semelhante ao das *sub-tasks* da *task* "Organizar entrega" é de 66,7%, ou seja, maior do que o *threshold* de 50% definido. Portanto, uma relação de rastreabilidade *overlaps* é criada entre o *goal* "Arrumar entrega" do Prometheus e a *task* "Organizar tare-

fa" do SR (figura 2). Adquirir e obter, calcular e computar são sinônimos no dicionário WordNet.

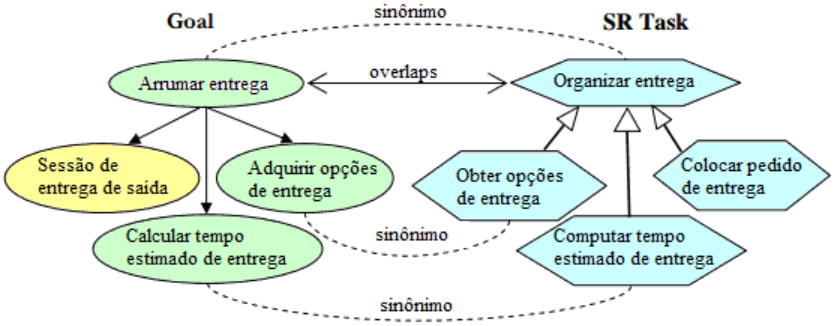

Fig. 2. Exemplo do uso da regra

O *sub-goal* "Sessão de entrega de saída" do *goal* "Arrumar entrega" não coincide com nenhuma *sub-task* da *SR task* "Organizar entrega" em i*. Esta informação é representada como um elemento ausente.

Foi proposto pelo *framework* o uso de um *template* padrão para criação dessas regras, que será visto na figura 3 em forma de exemplo, com uma regra identificando a relação de rastreabilidade que existe entre atores no i* e agentes no Prometheus.

A primeira parte consiste na identificação da regra, e contém um identificador único (id), a priorização da regra (*priority*), o tipo da regra (*type*), o tipo do elemento fonte a ser rastreado (elemTypeA), o tipo do elemento alvo a ser rastreado (elemTypeB), e uma breve descrição da regra (*description*).

A segunda parte contém código XQuery válido, e é composto por outras subpartes. A primeira diz respeito a declaração dos *namespaces*, variáveis, e sequências de elementos utilizados pela regra. A declaração de *namespaces* serve para utilizar funções implementadas em Java, que estende as funcionalidades de XQuery, ou seja, é possível inserir funções que não são comtempladas por padrão no *framework*.

Já a declaração de variáveis definem uma sequência de elementos que são utilizados pela regra, e os nomes dos documentos fonte e alvo que serão comparados pela regra, fazendo com que a ferramenta consiga identificá-los. Já a segunda parte efetua a iteração para comparar os elementos dos dois tipos inseridos na regra, que utiliza a expressão *FLWOR (FOR, LET, WHERE, ORDER BY, RETURN)* de XQuery.

Como ponto principal temos a condição, que define como esta regra deve ser satisfeita, e consequentemente gerando as relações de rastreabilidade, e/ou elementos ausentes. No exemplo, é verificado se os elementos são sinônimos, um indicador para efetuar a relação de rastreabilidade, porém outras condições poderiam ser utilizadas.

```
<Rule id="Rule49" priority="1" type="overlaps" elementTypeA="Istar Actor"
    elementTypeB="Prometheus Agent"
    description="Rule identify traceability relations between Actors in i* and Agents in Prometheus">
    <XQuery>
      <![CDATA[
        declare namespace f = "java:retratos.XQueryFunctions";
        declare namespace syn = "java:retratos.XQuerySynonymsFunctions";
        declare namespace sim = "java:retratos.XQuerySimilarityFunctions";
        declare namespace cc = "java:retratos.XQueryCompletenessCheckingFunctions";
        declare namespace pdt = "java:retratos.XQueryPDTFunctions";
        declare namespace taom = "java:retratos.XQueryTAOMFunctions";
        declare namespace xmi="http://www.omg.org/XMI";

        let $istarDoc := doc(taom:getTAOMFileName())
        let $systemActors := $istarDoc//TroposClasses[@xsi:type=
                              'it.itc.sra.taom4e.model.core.informalcore.formalcore:FActor'
                              and @isSystem='true']
        let $prometheusDoc := doc(pdt:getPDTFileName())
        let $prometheusAgents := $prometheusDoc//object[@type='Agent']
        for $systemActor in $systemActors, $prometheusAgent in $prometheusAgents
           return
             if ( f:clr() and
                  syn:isSynonyms($systemActor/@name,
                          $prometheusAgent/base/field[@name='name']/text()) )
             then
               <TraceabilityRelation type="overlaps"  ruleID="rule49a"
                                     degreeOfCompleteness="{cc:getDegreeOfCompleteness()}">
                  <Element doc="{taom:getTAOMFileName()}"
                           name="{$systemActor/@name}"
                           type="Actor" id="{$systemActor/@xmi:id}">
                  </Element>
                  <Element doc="{pdt:getPDTFileName()}" type="Agent"
                           name="{$prometheusAgent/base/field[@name='name']/text()}"
                           id="{$prometheusAgent/@id}">
                  </Element>
               </TraceabilityRelation>
             else
               ""
      ]]>
    </XQuery>
</Rule>
```

Fig. 3. *Template* de regra em exemplo

3 Proposta

A abordagem de Cysneiros [9] é baseada no uso de regras para identificar relacionamento de rastreamento, que são codificadas usando uma extensão da linguagem XQuery. Porém, a criação dessas regras não é uma atividade simples; ela necessita conhecimento da linguagem XQuery e o uso de algumas funções específicas fornecidas pela abordagem. Para tornar a abordagem mais flexível, é importante criar um editor para abordagem que suporte a criação de regras usando um ambiente gráfico.

A ferramenta proposta visa auxiliar na geração de regras de rastreamento, vistas no trabalho de Cysneiros, buscando facilitar as atividades vivenciadas por analistas, tanto na criação das regras, assim como na manutenção das mesmas. Com a ferramenta objetiva-se: a não necessidade de conhecimento especializado para criação das regras, contribuir na redução do esforço na criação da rastreabilidade entre artefatos de *sof-*

tware, aumentar o reuso das regras, possibilitar a interoperabilidade entre ferramentas do gênero, através de tipos variados de entradas e saídas, permitir o relacionamento entre os mais distintos tipos de documentos, tendo em vista que será possível ao usuário configurá-los na ferramenta, e estimular a adoção da prática do uso de regras de rastreamento.

A proposta inclui a criação de uma ferramenta web que auxilie na criação de novos projetos, onde será possível definir diversas regras; visando também reduzir o tempo e esforço gastos para geração das regras. Inicialmente serão trabalhados como estudos de caso regras com o *framework* i*, a metodologia Prometheus, e o código JACK, os mesmos utilizados e suportados pela proposta de Cysneiros, com o intuito de verificar a efetividade da abordagem com a ferramenta com relação a abordagem manual, e em um outro momento, validar em situações reais junto a profissionais da área.

Porém, a ferramenta não se limitará a trabalhar com esses três escopos vistos no trabalho de Cysneiros, sendo possível ao usuário inserir na ferramenta o suporte a outros tipos de artefatos, que serão configurados pelo próprio usuário na ferramenta. Ou seja, a ferramenta foi projetada de forma a generalizar os tipos de entradas passados pelo usuário. Isso provê ao usuário a criação do rastreamento entre os mais diversos elementos, dependendo da forma de mapeamento que é feito na ferramenta, isto é, possibilita o uso da ferramenta pelas mais variadas abordagens, e a se enquadrar em diferentes contextos de acordo com a regra montada.

Apesar da generalidade, alguns problemas ainda podem ocorrer, como, por exemplo, em uma situação em que um desenvolvedor expressa algo como objetivo, e outro como tarefa. Esta, e outros tipos de situação, devem ser previstas na montagem das regras, que devem abordar as possibilidades encontradas dentro do contexto específico.

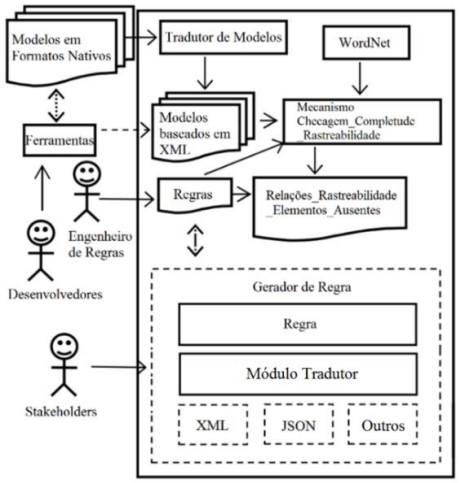

Fig. 4. *Framework* de rastreabilidade com Gerador de Regras

A proposta aborda o módulo do gerador de regras através da ferramenta TraceRG, proposta deste artigo, para o *framework* proposto por Cysneiros. A figura 4 apresenta o mesmo esquema da figura 1, complementado pela etapa do gerador de regras.

É possível visualizar que os *stakeholders* estão em contato direto com o gerador de regras, objetivando assim que qualquer um dos envolvidos com o projeto consiga manipular as regras, mas em geral, analistas e gerentes de projeto que farão uso da ferramenta. Sua grande finalidade é gerar uma regra, e para isso, é preciso definir em qual tipo de saída a regra será definida. Inicialmente foi definido que a ferramenta possuirá como padrão as saídas em XML, utilizada como padrão pela proposta de Cysneiros, e JSON, uma alternativa ao XML. Porém, com o módulo tradutor isso pode ser personalizado, dando a possibilidade de ser implementada a definição de novas saídas, seguindo as assinaturas do módulo, sem afetar o restante da ferramenta.

3.1 Características da ferramenta

A proposta pode ser caracterizada como uma ferramenta *web*, cujos usuários podem ser analistas que desejem criar regras, mas não necessariamente possuem conhecimento especializado para tal. A ferramenta possui a arquitetura MVC (*Model-view-controller*), que separa a camada da informação da interação do usuário. O modelo (*model*) consiste nos dados da aplicação, regras de negócios, lógica e funções. Uma visão (*view*) pode ser qualquer saída de representação dos dados, como uma tabela ou um diagrama. É possível ter várias visões do mesmo dado, como um gráfico de barras para gerenciamento e uma visão tabular para contadores.

O controlador (*controller*) faz a mediação da entrada, convertendo-a em comandos para o modelo ou visão. As ideias centrais por trás do MVC são a reusabilidade de código e separação de conceitos. A linguagem de programação utilizada foi o C#, da plataforma .NET, em sua versão 4.5, por se tratar de uma linguagem que segue o paradigma de orientação a objetos, bastante difundida, de fácil manutenção e indicada para a plataforma web com ASP.NET.

O banco de dados utilizado é o SQLServer Express, por ser um banco gratuito, ter alta integração com a plataforma .NET, e de fácil manipulação, possuindo grande aceitação entre os que trabalham com desenvolvimento de *software*.

Além disso, a ferramenta está sendo desenvolvida em inglês, por ser um padrão de língua adotado nas ferramentas do mundo A continuidade deste capítulo apresenta a ferramenta, incluindo as suas principais funcionalidades.

3.2 Funcionalidades

A partir da definição do método de geração das regras de rastreamento, foi feito o levantamento dos requisitos. As principais funcionalidades da ferramenta podem ser divididas em dois grupos:

- Funcionalidades básicas – inerentes a ferramentas da área, a citar: (a) cadastro de usuário; (b) cadastro de projeto; (c) convidar colaboradores; (d) login de usuário.
- Funcionalidades de regras: (a) criação de regras; (b) busca e seleção de regras; (c) associação de regra ao projeto; (d) importar/exportar regras geradas.

A figura 5 apresenta as principais funcionalidades através do diagrama de caso de uso.

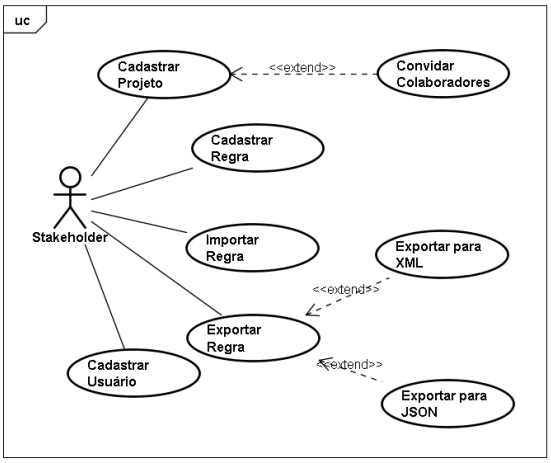

Fig. 5. Diagrama de caso de uso

3.3 Geração das Regras

Como principal módulo da ferramenta temos a geração de regras, proveniente da funcionalidade de cadastrar regra. Para um melhor entendimento, é apresentado o macro processo da geração das regras e o detalhamento das atividades.

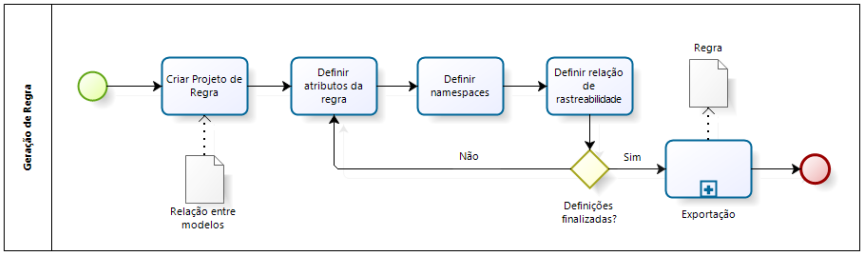

Fig. 6. Macro processo para a Geração das Regras de Rastreabilidade

O projeto de regra pode ser visto como o esqueleto da regra. É esse projeto que poderá ser salvo e editado quantas vezes for necessário, conseguindo gerar inclusive, regras diferentes de acordo com as modificações. A partir daí pode-se começar a montar a regra efetivamente, associando as características da regra ao projeto, quais elementos essa regra está associando, o id da regra, uma descrição e o tipo de relacionamento que esta regra cria. Entende-se como a assinatura da regra. As regras podem utilizar funções que não necessariamente estão no escopo inicial da ferramenta. Para isso se faz valer o uso de *namespaces*, onde é possível utilizar um *namespace* customizado pelo usuário, e as suas funções estarão disponíveis para uso na regra. Já as relações de rastreabilidade definem quais as condições que devem ser satisfeitas para gerar uma relação de rastreabilidade.

No exemplo da figura 3, uma das condições é que os elementos fossem sinônimos, sendo um dos indicadores para demonstrar a relação de rastreabilidade. Caso tais condições não sejam satisfeitas, o elemento ausente será indicado pela ferramenta, e não existirá a relação. Ao término das etapas anteriores será possível gerar arquivos de texto com código tanto em XML, como em JSON, para uso em outras aplicações, facilitando com isso a integração entre diversas ferramentas.

A figura 7 exibe a tela principal da ferramenta, que demonstra como será efetuada a criação dessas regras visualmente, a priori, e se baseia em um exemplo semelhante ao da figura 3.

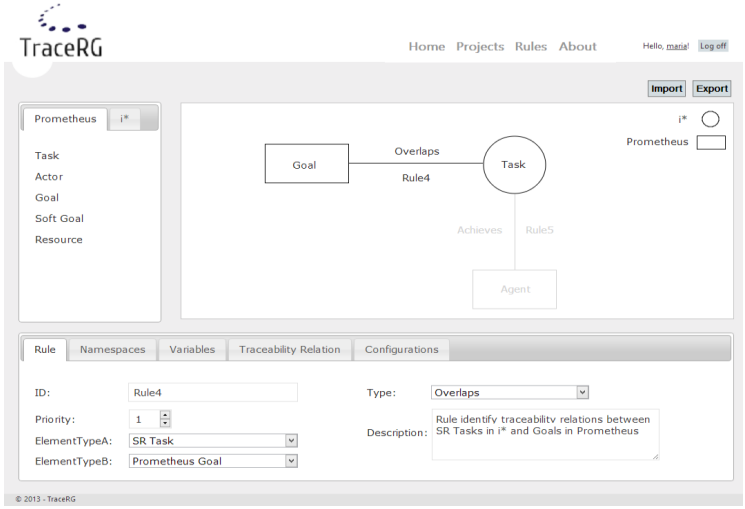

Fig. 7. Propriedades da regra na ferramenta

No lado esquerdo da ferramenta conseguimos visualizar os artefatos que estão disponíveis para relacionar e gerar as regras. No exemplo temos duas abas, Prometheus e i*, que podem ser acessadas durante todo o desenvolvimento da regra. Com isso, será possível arrastar qualquer dos itens das abas para a área de trabalho visual (centro).

No exemplo, foram utilizados o Goal do Prometheus e Task do i*. Assim como na figura 3, temos o tipo de relação Overlaps, e o ID Rule4. Tais informações são inseridas no setor inferior da ferramenta, assim como outras características, porém essas duas por serem as pricipais propriedades (uma é o identificador da regra, outra o tipo da relação), são exibidas também na área de trabalho, facilitando a identificação das mesmas. Os elementos também possuem formas únicas para os itens trabalhados, em questão o Prometheus (retangular) e o i* (circular), como indicado no setor direito da ferramenta, para diferenciar qual o tipo do elemento que está sendo visualizado.

Existe também neste mesmo projeto outra relação definida, porém sem foco para edição. A outra relação diz que, a Task em i*, possui uma relação Achieves com o Agent do Prometheus, e esta regra possui identificador "Rule5". Assim, mesmo sem estar editando a regra, é possível visualizar de uma forma mais ampla como os elementos se relacionam, e as regras que já existem.

Após a aba de regra, temos a aba *namespaces*, permitindo ao usuário inserir seus *namespaces* e consequentemente a usufruir de suas próprias funções dentro das regras. Já a aba de variáveis permite ao usuário definir como a ferramenta irá identificar os artefatos de acordo com a entrada passada. Por exemplo, em um arquivo xml como entrada, o usuário precisa indicar qual atributo identifica o artefato (tipo, nome, etc). Na figura a seguir apresentamos como as regras de rastreabilidade são colocadas na ferramenta

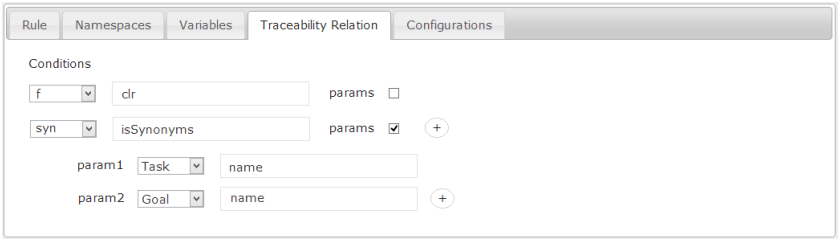

Fig. 8. Relação de rastreabilidade na ferramenta

As regras de rastreabilidade são baseadas em condições, que precisam ser satisfeitas para gerar uma relação. Tais condições são geradas em cimas de funções com retorno booleano, que por sua vez são provenientes de *namespaces* definidos previamente na ferramenta (aba *namespaces*). Com isso, podemos definir quais funções queremos utilizar para efetuar as condições, e se elas possuem parâmetros. Na figura, temos a função clr, que não possui parâmetros, invocada através do *namespace* f. Já na segunda condição, temos a função *isSynonyms*, invocada através do *namespace* syn, porém que necessita de dois parâmetros, que dirá se os dois são sinônimos. Para o exemplo, foi escolhido que, como parâmetros temos o nome da *Task* e do *Goal*, através de seus respectivos atributos *name*, para verificarmos se eles são sinônimos ou não, que é um indicador utilizado nesta regra para garantir a relação entre os elementos.

4 Trabalhos Relacionados

Rastreabilidade de *software* tem sido o foco de pesquisas há muitos anos, e várias abordagens e técnicas têm sido propostas para apoiar os seus diferentes aspectos [11]. Estas abordagens e técnicas podem ser classificadas em quatro grupos principais: (a) o estudo e definição de diferentes tipos de relações de rastreabilidade, (b) o apoio à geração de relações de rastreabilidade, (c) o desenvolvimento de arquiteturas, ferramentas e ambientes para representar e manter relações de rastreabilidade, e (d) o estudo de como usar as relações de rastreabilidade em várias atividades de desenvolvimento.

A geração de relações de rastreabilidade pode ser classificada como manual, semiautomática, e automática [11]. Abordagens semiautomáticas estão preocupadas com a geração de relações de rastreabilidade com base em um conjunto de regras anteriormente estabelecidas [12], ou quando as relações de rastreabilidade são geradas como um subproduto do processo de desenvolvimento de software (abordagens orientadas por processos) [13]. Além do trabalho de Cysneiros [9], base para o nosso trabalho, um trabalho similar foi proposto em [14]. Na proposta, trabalha-se com MDD (*Model Driven Development*), em cima de rastreamento na transformação de modelos, com regras de transformações, para conseguir efetuar a transformação no caminho inverso à transformação original. Outro trabalho é o XTraQue [19], voltado para a rastreabilidade dentro de linhas de produto de *software*, e que também utiliza XQuery e XML.

5 Conclusão

Está sendo desenvolvida uma ferramenta para apoiar a criação de regras de rastreamento, motivada pela inexistência de ferramentas que apoiem a proposta do *framework* de rastreabilidade [9]. Entre os objetivos da ferramenta tem-se:

- A não necessidade de conhecimento especializado para criação das regras;
- Diminuição do esforço para criação das regras;
- Estimular o reuso de informações entre as regras;
- Facilitar a busca por regras criadas através da ferramenta;
- Possibilitar a integração com outras ferramentas;
- Estimular a adoção da prática do uso de regras de rastreamento.

Como recomendações e sugestões de trabalhos para evoluir a presente pesquisa pode se destacar:

- Finalização de versão da ferramenta;
- Coleta de dados com testes em ambientes de desenvolvimento;
- Definição de métricas para garantir a eficácia da ferramenta;
- Efetuar melhorias, proveniente de testes, na ferramenta.

Referências

1. O. Gotel, A. Finkelstein, “An Analysis of the Requirements Traceability Problem,” in Proceedings of the First IEEE International Conference on Requirements Engineering (ICRE’94), IEEE Computer Society Press, Los Alamitos, 1994, pp. 94-101.
2. Software Engineering Institute, “CMMI for Development, Version 1.2 – Improving Process for Better Products”, Carnigie Mellon University, Pittsburgh, 2006.
3. H. van Loon, “Process Assessment and ISO/IEC 15504 – A Reference Book. Springer, Berlin, Heidelberg, New York, 2004.
4. Gotel, O. & Finkelstein, A. (1994). An analysis of the requirments traceability problem. Proceedings of the 1st International Conference on Requirements Engineering. 94-101.
5. Cleland-Huang, J., Dekhtyar, A., & Hayes, J. (2007). Problem statements and grand challenges. (COET-GCT-06-01-0.9). Center of Excellence for Traceability.
6. IBM Rational. (2013). Rational DOORS. Acesso em Dezembro, 2013, de http://www-03.ibm.com/software/products/ratidoor/
7. Borland. (2013). CaliberRM. Acesso em Dezembro, 2013, de http://www.borland.com/products/caliber/
8. IBM Rational. (2013). Rational RequisitePro. Acesso em Dezembro, 2013, de http://www-03.ibm.com/software/products/reqpro/
9. Cysneiros, G. (2011) Doctoral thesis, City University London, Software Traceability for MultiAgent Systems Implemented Using BDI Architecture;
10. XQuery. (2013). Query. Acesso em Dezembro, 2013, de http://www.w3.org/standards/xml/query
11. Spanoudakis, G., Zisman, A., "Software Traceability: A Roadmap," in S. K. Chang, ed., Handbook of Software Engineering and Knowledge Engineering, August, 2005.
12. Egyed, A., "A Scenario-Driven Approach to Trace Dependency Analysis," IEEE Trans. on SoftwareEngineering, vol. 29, 2003.
13. Pohl, K., "Process-Centered Requirements Engineering," Wiley/Research Studies Press, New York, 1996.
14. Shah, S., Anastasakis, K., e Bordbar, B., “Using Traceability for Reverse Instance Transformations with SiTra,” in Design and Architectures for Signal and Image Processing (DASIP). Special Session on Formal Models, Transformations and Architectures for Reliable Embedded System Design., Bruxelles, Belgium, 2008.
15. Spanoudakis, G., Zisman, A., Pérez-Miñana, E., Krause, P. (2004). Rule-based generation of requirements traceability relations. Journal of Systems and Software, 72(2), 105-127.
16. Sherba, S. A., "Towards Automating Traceability: An Incremental and Scalable Approach," PhD Thesis, Department of Computer Science, University of Colorado, 2005;
17. Castro-Herrera, C. & Cleland-Huang, J. (2007). Towards a unified process for automated traceability. Proceedings of the 4th ACM International Workshop on Traceability in Emerging Forms of Software Engineering, Lexington, KY, USA.
18. Barghouti, S., Kaiser, E. Scalling up rule-based software development environments. International Journal on Software Engineering & Knowledge Engineering, 1992.
19. Jirapanthong, W. & Zisman, A. (2009). XTraQue: Traceability for product line systems. Software and Systems Modeling, 8(1), 117-144.