Uma Ferramenta para Construção de Catálogos de Padrões de Requisitos com Comportamento

Taciana N. Kudo[1,2], Renato F. Bulcão-Neto[1], and Auri M. R. Vincenzi[2]

[1] Instituto de Informática, Universidade Federal de Goiás, Goiânia-GO, Brazil
{taciana,rbulcao}@ufg.br
[2] Departamento de Computação, UFSCAR, São Carlos-SP, Brazil
auri@dc.ufscar.br

Resumo Um padrão de requisitos de software (PRS) reúne comportamentos e serviços de aplicativos com características semelhantes. Apesar dos benefícios obtidos com a adoção de PRS em projetos de software, há uma carência de pesquisas sobre PRS em outras fases do desenvolvimento, além da Engenharia de Requisitos. Com base em descobertas na literatura, o uso prático de PRS pode ser melhor experimentado pelo uso conjunto de metodologias de desenvolvimento bem estabelecidas, ferramentas de software orientadas a PRS e catálogos de PRS de modo sistematizado. Nesse sentido, o metamodelo *Software Pattern MetaModel* (SoPaMM) permite relacionar um PRS com outros tipos de padrão de software e incorpora comportamentos sob a influência da metodologia ágil *Behavior-Driven Development* (BDD). Neste artigo, propõe-se a ferramenta *Terminal Model Editor* (TMEd) para apoiar a construção de modelos terminais para domínios específicos, usando o metamodelo SoPaMM como modelo de referência. Um exemplo de uso da TMEd é apresentado com a elaboração de um catálogo de padrões de requisitos legais para a certificação de Sistemas de Registro Eletrônico de Saúde (S-RES) no Brasil. Espera-se que os esforços com a ferramenta TMEd e o catálogo de padrões de requisitos para S-RES baseados no metamodelo SoPaMM ajudem a comunidade a melhor compreender o impacto geral do uso de PRS no ciclo de vida de software, não limitando-se à Engenharia de Requisitos.

Keywords: Padrão de requisito de software · BDD · Ferramenta · Catálogo

1 Introdução

À medida que uma empresa de desenvolvimento de software produz especificações de requisitos, é natural que uma porção desses requisitos seja específica de cada software, enquanto que uma parcela significativa dos requisitos se repita ao longo do tempo [22]. Ou seja, nem todos os requisitos que definem um software lhe são específicos, e reusar o conhecimento adquirido em projetos anteriores é uma estratégia adequada para melhorar a qualidade dos requisitos e a eficiência do processo de Engenharia de Requisitos [16].

Uma das abordagens voltadas ao reúso de requisitos é a de Padrão de Requisitos de Software (PRS), uma abstração que agrega comportamentos e serviços comuns a vários sistemas e que pode ser reutilizada em software similar [22]. A literatura reporta propostas de PRS para diversos domínios de aplicação, como sistemas embarcados [6], de gerenciamento de conteúdo [17] e de computação em nuvem [2]. Das experiências existentes do uso de PRS em projetos de software, identifica-se uma série de benefícios que afetam as atividades de coleta e especificação de requisitos, como a economia de tempo e a melhoria na qualidade dos requisitos quanto à completude, uniformidade, consistência e clareza [20,21,22].

Entretanto, estudos secundários [8,9] destacam a carência de relatos dessas vantagens em outras fases do ciclo de vida de software (CVS), vantagens reportadas em 76 diferentes estudos sobre PRS na Engenharia de Requisitos. Esses mesmos estudos secundários reportam apenas oito pesquisas sobre PRS em *design* de software, uma em construção, uma em teste, e nenhuma em manutenção, apesar da intrínseca relação entre requisitos e artefatos produzidos nessas fases.

Nesse contexto, o metamodelo *Software Pattern MetaModel* (SoPaMM) [10] é proposto de modo a relacionar PRS com padrões voltados às demais fases do CVS. Para cobrir a lacuna de uso de PRS na fase de teste [8,9], o metamodelo SoPaMM permite relacionar PRS com padrões de teste de software (PTS), uma abstração de práticas de teste repetitivas, semelhantes e de alto valor [12].

Contudo, uma abordagem de metamodelo em si não é suficiente. Evidências encontradas em [7] apontam que, para promover os estados da arte e da prática de PRS, as abordagens de PRS devem combinar:

(a) metodologias de desenvolvimento estabelecidas;
(b) ferramentas de software; e
(c) catálogos (ou conjuntos) de PRS de modo padronizado.

Em resposta ao item (a), o metamodelo SoPaMM relaciona PRS a PTS, representa um PRS com comportamentos como descritos na metodologia *Behavior-Driven Development* (BDD) [4] e possibilita transformar esse PRS em uma especificação executável capaz de gerar testes automatizados [10].

Em relação ao item (b), propõe-se neste artigo a ferramenta *Terminal Model Editor* (TMEd) para apoiar a construção de modelos terminais (instâncias) do metamodelo SoPaMM para domínios específicos, segundo o padrão *Meta Object Facility* (MOF) [13].

Quanto ao item (c), a ferramenta TMEd é utilizada para a elaboração de um catálogo de PRS baseados em comportamento com base em requisitos legais definidos e utilizados pela Sociedade Brasileira de Informática em Saúde (SBIS) no processo de certificação de Sistemas de Registro Eletrônico de Saúde (S-RES) [18]. A ferramenta TMEd é, até o presente momento, a única a permitir a criação de catálogos de PRS baseados em comportamento.

Este artigo está assim organizado: a Seção 2 descreve e analisa trabalhos relacionados; a Seção 3 apresenta fundamentos teóricos da pesquisa; a Seção 4 detalha o desenvolvimento e um exemplo de uso da ferramenta TMEd; e a Seção 5 traz nossas considerações finais e propostas de trabalhos futuros.

2 Trabalhos Relacionados

Esta seção descreve ferramentas de gerenciamento de catálogos de PRS, que organizam PRS em catálogos e permitem a criação, edição, exclusão e relacionamento de PRS, assim como a ferramenta TMEd proposta.

PABRE-Man [14] é uma ferramenta *desktop* que se conecta a uma base de catálogos de PRS [15] que seguem o modelo de referência PABRE definido pelos seus autores [5]. As principais funcionalidades da PABRE-Man são o gerenciamento e a busca por PRS e a exportação e impressão de um catálogo de padrões PABRE. Embora a ferramenta suporte PRS de domínios de aplicação variados, os PRS são relativos apenas a requisitos não funcionais.

Já a ferramenta proposta em [1] é específica para o domínio de Sistemas de Informação e os PRS manipulados representam tanto requisitos funcionais quanto não funcionais. Três módulos principais da ferramenta fornecem apoio à manutenção de usuário, cliente e projeto de software, especificação e gestão de PRS e instanciação de PRS. O módulo de gestão de PRS dessa ferramenta é o que mais se assemelha à proposta da ferramenta TMEd, enquanto que o módulo de instanciação de PRS apoia a elaboração de documentos de especificação de requisitos a partir do reúso de PRS.

A ferramenta *Requirement Pattern Editor* (*RP Editor*) [3] gerencia catálogos de PRS específicos de segurança. Assim como a TMEd, sua implementação é baseada na plataforma *Eclipse Modeling Framework* (EMF), mas apresenta suporte para edição gráfica de PRS via *Graphical Editing Framework* (GEF) e *Graphical Modeling Framework* (GMF). A ferramenta *RP Editor* permite exibir, criar, modificar e excluir PRS de segurança. Complementar a essa ferramenta, existe a *Instantiated Requirement Pattern Editor* (*InstRP Editor*) [3], que instancia os PRS de segurança para a produção de documentos de especificação de requisitos.

Comparativamente, a ferramenta TMEd possui menos funcionalidades que as apresentadas, como o fato de não apoiar a elaboração de instâncias de PRS [1,3] ou gerenciar perfis de usuários [1]. Por outro lado, os PRS gerenciados pela TMEd são independentes de domínio de aplicação e são relativos a requisitos funcionais e não funcionais. Mais importante ainda, a ferramenta TMEd é a única, até o presente momento, a permitir a criação de catálogos de PRS com comportamento, combinando padrões de requisitos e de testes.

3 Fundamentação Teórica

Esta seção discorre sobre dois pilares da proposta da ferramenta TMEd: a metodologia ágil BDD e o metamodelo SoPaMM, desenvolvido pelos autores.

3.1 *Behavior-Driven Development* (BDD)

BDD é uma abordagem de desenvolvimento ágil que promove a colaboração e o entendimento comum entre equipes técnica e de negócios em relação ao comportamento esperado do software a ser desenvolvido [4]. A comunicação entre essas

equipes ocorre por meio da linguagem *Gherkin*, que descreve cenários usando a sintaxe *Given-When-Then*, em que: *Given* define as condições para executar o cenário; *When* descreve os passos do cenário; e *Then* especifica os comportamentos esperados com a execução do cenário.

Os comportamentos derivam-se dos objetivos de negócio decompostos em *features*, cada qual associada a uma ou mais histórias de usuário (AS <interessado>, I CAN <o quê?>, SO THAT <por quê?>). Cada história de usuário pode ter um ou mais cenários que descrevem comportamentos esperados da aplicação. A seguir, apresenta-se uma descrição resumida de uma *feature* de autenticação de usuário por *login* e senha, na forma de história de usuário, com um comportamento de sucesso esperado descrito como cenário em *Gherkin*. Detalhes adicionais sobre a abordagem BDD podem ser obtidos em [4,19].

```
Feature: autenticação de usuário com nome de usuário e senha
  AS usuário do sistema
  I CAN autenticar com nome de usuário e senha
  SO THAT eu possa ter acesso ao sistema

  Scenario: usuário autenticado com sucesso
    GIVEN que estou executando a função de login
    WHEN eu digito o nome de usuário e a senha corretos
    THEN eu tenho acesso ao sistema
```

3.2 *Software Pattern Metamodel* (SoPaMM)

O metamodelo SoPaMM permite a reutilização de padrões de requisitos de software com o benefício adicional destes apresentarem comportamentos baseados em conceitos e práticas da abordagem BDD. O objetivo principal com o uso do SoPaMM é construir especificações de requisitos e de testes de melhor qualidade, em um tempo menor, além da geração de testes automatizados [10].

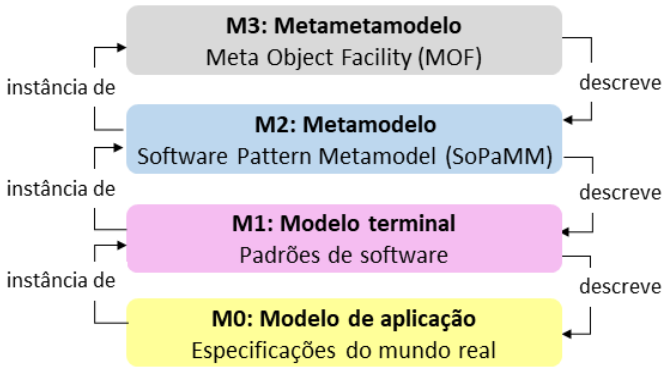

Figura 1. Relacionamentos entre MOF, SoPaMM e modelos terminais e de aplicação.

A construção do metamodelo SoPaMM segue a arquitetura de metamodelagem do *Meta Object Facility* (MOF) [13], tal como mostra a Figura 1. Nessa arquitetura, os modelos são refinados progressivamente e organizados em três categorias [11]: metametamodelos, metamodelos e modelos terminais. Na camada M3, estão os conceitos do MOF, que atuam como um metametamodelo. Na camada M2, encontra-se o metamodelo SoPaMM, que tem como modelo de referência o metametamodelo MOF. Na camada M1, encontram-se instâncias do metamodelo SoPaMM, ou seja, modelos terminais que representam padrões de requisitos e de testes, por exemplo. Por fim, na camada M0, está o modelo de aplicação, que descreve as especificações de aplicações do mundo real, e segue como modelo de referência o modelo terminal da camada M1.

No contexto desta pesquisa, o foco é a construção de modelos terminais a partir da estrutura do metamodelo SoPaMM. Os modelos terminais reúnem padrões de requisitos e de testes conforme o esquema do SoPaMM, cujos principais conceitos aparecem em destaque na Figura 2: *Catalogue*, elemento que reúne todas as definições do modelo terminal; *SoftwarePatternBag*, que agrupa padrões de software; *SoftwarePattern* que pode ser especializado em *RequirementPattern* e *TestPattern*, representados por PRS e PTS, respectivamente; e os relacionamentos[3] entre diferentes *SoftwarePatternBag* ou *SoftwarePattern*.

A classe *FunctionalRequirementPattern*, especializada da classe *RequirementPattern*, corresponde aos padrões de requisitos funcionais e são detalhados conforme os conceitos do BDD. Um padrão de requisito funcional é constituído por elementos da classe *Feature*, cujos comportamentos são descritos por elementos da classe *Scenario* que, por sua vez, são apoiados por dados de teste na classe *Example*. Estes e outros conceitos do metamodelo SoPaMM [10] apoiam a definição da ferramenta TMEd para a construção de instâncias desse metamodelo.

4 *Terminal Model Editor* (TMEd)

Esta seção descreve a ferramenta TMEd sob a ótica de seus requisitos fundamentais, detalhes de implementação, operação de principais funcionalidades e um exemplo de uso. Em linhas gerais,

- os padrões de software (ou modelos terminais) construídos a partir da TMEd seguem o esquema do metamodelo SoPaMM (vide níveis M1 e M2 na Figura 1, respectivamente);
- a saída da ferramenta TMEd é generalizada como padrões de software porque o metamodelo SoPaMM permite representar outros tipos de padrões de software, que não unicamente PRS; e
- a ferramenta apoia os papéis de analistas de requisitos e de testes, pois a versão atual do SoPaMM permite especificar PRS com comportamentos e ainda relacioná-los a PTS e elementos de interface com usuário para fins de teste de aceitação.

[3] Um exemplo de tipo de relacionamento é o *Refers to*, que pode permitir que um padrão de software refira-se a outro que contém informações adicionais sobre um tópico específico [22].

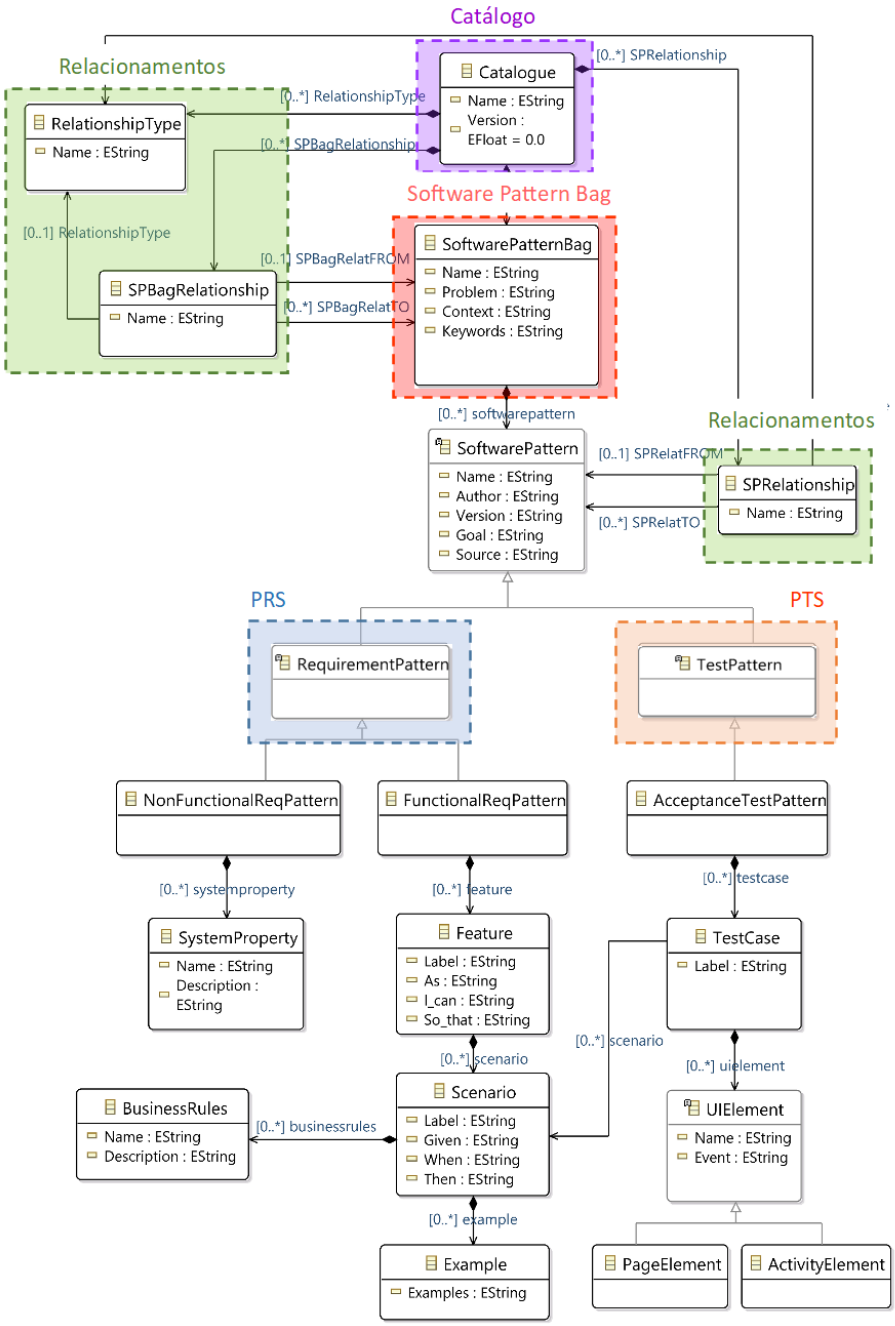

Figura 2. Visão geral do metamodelo SoPaMM, destacando principais conceitos [10].

4.1 Requisitos

Em geral, ferramentas que fornecem apoio à construção de PRS utilizam algum modelo ou *template* de referência. Neste caso, o requisito essencial da TMEd é implementar o esquema do metamodelo SoPaMM, para que o usuário possa explorar a construção de modelos terminais que incluem PRS e PTS.

Derivados desse requisito-base e influenciados por experiências relatadas em [5], seguem os principais requisitos da ferramenta TMEd:

1. gerenciar metadados de catálogos de padrões de software;
2. gerenciar tipos de relacionamentos;
3. gerenciar *bags* de padrões em um catálogo;
4. definir relacionamentos entre *bags* de padrões;
5. definir relacionamentos entre padrões de software;
6. gerenciar padrões de requisitos não-funcionais de uma *bag* (e suas *system properties*);
7. gerenciar padrões de requisitos funcionais de uma *bag* (composto de *features*, *scenarios* e *examples*);
8. gerenciar padrões de testes de aceitação de uma *bag* (composto de casos de teste e elementos de interface com usuário); e
9. validar a estrutura do catálogo com a gramática do metamodelo SoPaMM.

4.2 Desenvolvimento

A ferramenta TMEd foi desenvolvida utilizando a plataforma *Eclipse* e o *plug-in Eclipse Modeling Framework*[4] (EMF), cujo arcabouço consiste de três elementos:

- *EMF Ecore*, que descreve e oferece suporte para modelos, incluindo notificação de alterações, persistência com serialização XMI e uma API para manipular objetos EMF;
- *EMF.Edit*, que fornece classes para criar editores para modelos EMF e permite que os modelos sejam exibidos usando visualizadores de área de trabalho padrão;
- *EMF.Codegen*, que inclui uma GUI na qual as opções de geração dos editores são especificadas e os geradores são chamados.

O metamodelo SoPaMM foi representado em formatos nativos do EMF: o *ecore* e o *genmodel*. A Figura 3 ilustra, à sua esquerda, o arquivo *soPaMM.ecore* que armazena os atributos e relacionamentos do modelo. No lado direito dessa figura, encontra-se o detalhamento do *soPaMM.ecore* com as classes, os atributos e os relacionamentos entre as classes do metamodelo. A Figura 4, por sua vez, apresenta, à sua esquerda, o arquivo *soPaMM.genmodel* que contém as configurações para geração do código do editor de modelos terminais TMEd. No lado direito dessa figura, na aba *Properties*, estão as propriedades do catálogo como, por exemplo, os indicadores para as classes *Editor Plug-in Class* e *Editor Plug-in ID*, utilizadas na criação do editor.

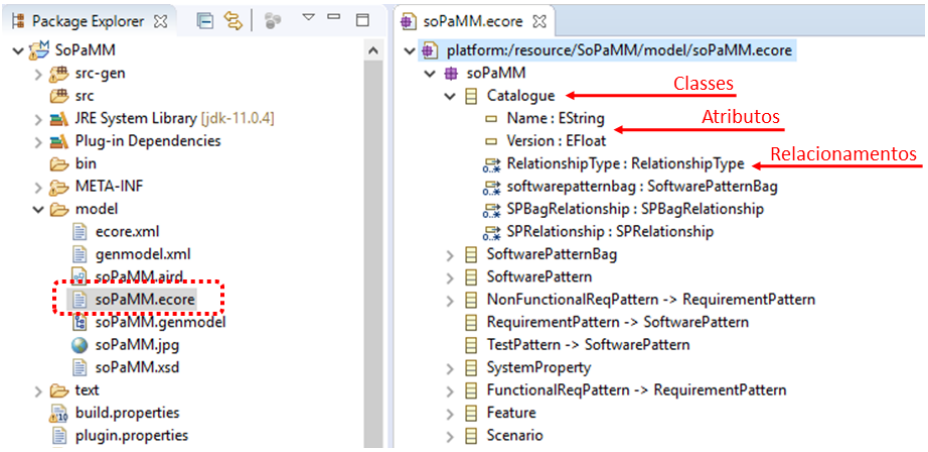

Figura 3. Arquivo de descrição *ecore* do SoPaMM.

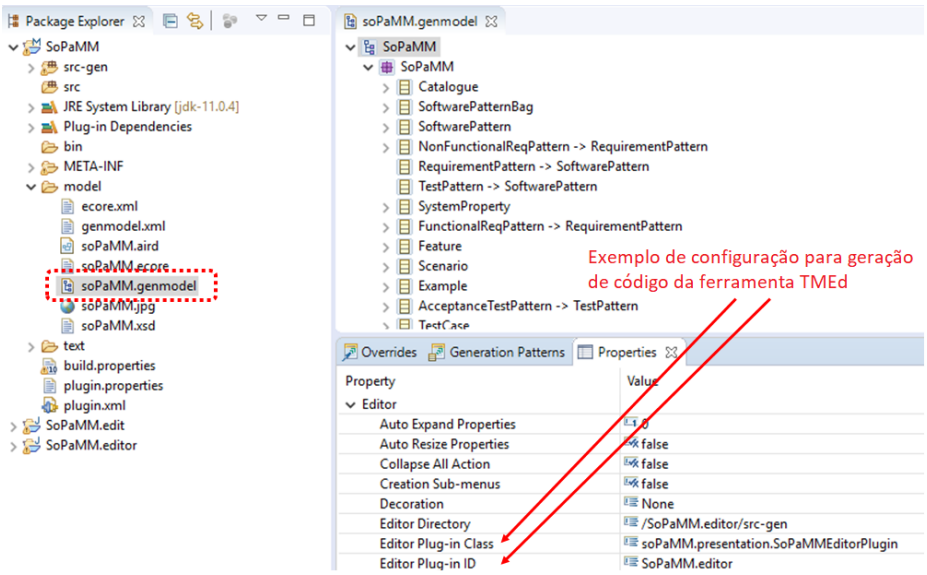

Figura 4. Arquivo de descrição *genmodel* do SoPaMM.

Com os arquivos de descrição *soPaMM.ecore* e *soPaMM.genmodel* construídos, o próximo passo foi a geração dos arquivos da TMEd com o *EMF.Edit*. Neste momento, foram criados os arquivos correspondentes ao *Edit* e ao *Editor*.

[4] Disponível online em https://www.eclipse.org/modeling/emf/.

4.3 Criação de Modelos Terminais

Para utilizar a TMEd, é necessário executar o arquivo *.genmodel* como uma nova aplicação no EMF. Ao realizar essa ação, é aberta uma nova instância do Eclipse com a TMEd e, a partir daí, é possível gerenciar modelos terminais.

Na Figura 5 é apresentada a interface de criação de modelos terminais baseados no SoPaMM. Uma interface do tipo *wizard* permite criar um novo modelo do tipo *SoPaMM Model* na pasta *Example EMF Model Creation Wizard*.

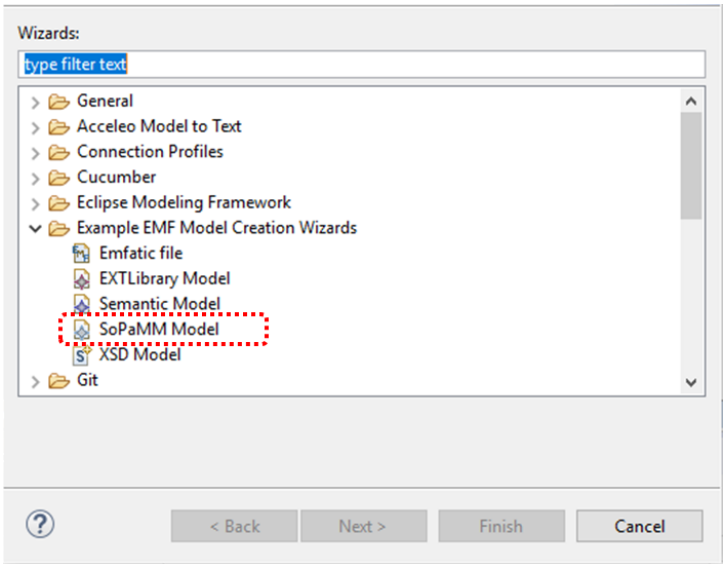

Figura 5. Interface de criação de modelo terminal baseado no SoPaMM.

Em conformidade com a gramática do metamodelo SoPaMM e, atendendo aos requisitos da TMEd, *Catalogue* é a primeira classe gerada em cada novo modelo terminal (requisito 1). A partir desta, é possível então incluir as demais entidades e atributos do modelo terminal.

A interface do TMEd para manipulação de um catálogo é mostrada na Figura 6 e, como exemplo, é apresentado o menu de inclusão de tipos de relacionamentos (*RelationshipType*), *bags* de padrões de software (*SoftwarePatternBag*), relacionamentos entre *bags* de padrões de software (*SoftwarePatternBag*) e relacionamentos entre padrões de software (*SPRelationship*), que correspondem aos requisitos 2 a 5, respectivamente.

Uma vez criada uma *bag* de padrões de software (*SoftwarePatternBag*), pode-se criar as especializações de padrão de software permitidas pelo SoPaMM, neste caso, padrões de requisitos funcionais e não funcionais e padrões de teste de aceitação (requisitos 6 a 8). Todo catálogo de padrões construído na TMEd é exportado para um arquivo XML, segundo o esquema do SoPaMM (requisito 9).

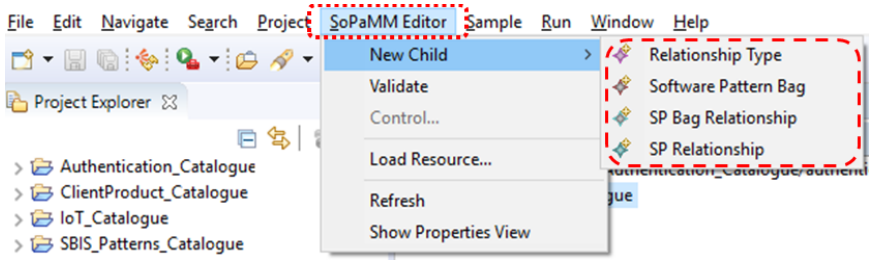

Figura 6. Interface de manipulação de catálogo de PRS.

4.4 Exemplo de Uso

A ferramenta TMEd foi utilizada na construção de um catálogo de PRS baseados em comportamento com base em requisitos legais utilizados na certificação de Sistemas de Registro Eletrônico de Saúde (S-RES) e elaborados pela Sociedade Brasileira de Informática em Saúde (SBIS) [18].

Para a definição desses requisitos, além de realizar uma extensa revisão de experiências e projetos similares de S-RES, a SBIS também utilizou normas ISO e padrões nacionais e internacionais. Com esses requisitos, a SBIS visa a garantir o alinhamento com tendências, bem como a adesão com a legislação nacional.

Os requisitos definidos pela SBIS são divididos em dois Níveis de Garantia de Segurança (NGS). O primeiro nível de certificação (NGS1) determina requisitos obrigatórios para a troca de informação em saúde suplementar, enquanto que o segundo nível (NGS2) permite a substituição de registros de saúde em papel por seus equivalentes eletrônicos. Seguindo o método sistemático de Withall [22], o catálogo elaborado com a TMEd contempla todos os requisitos do NGS1 [18].

A Figura 7 exibe a interface da ferramenta TMEd com as definições do catálogo de padrões para S-RES. No lado esquerdo, tem-se o arquivo *sbisSoPaMM*, que corresponde ao próprio catálogo, e a sua representação equivalente no formato XML. Já à direita, vê-se o conteúdo do catálogo da SBIS, organizado como uma árvore de elementos estruturados segundo a gramática do metamodelo SoPaMM. Por exemplo, o elemento *Catalogue* (A) encontra-se em um nível hierárquico acima das definições de todos os padrões do catálogo.

Considere o conteúdo da *bag* de autenticação *SPB_AuthenticationLoginPassword* (B), composta pelos padrões de requisito funcional e de teste de aceitação *FRP_AuthenticationLoginPassword* (C) e *ATP_AuthenticationLoginPassword* (D), respectivamente. O padrão de requisito descreve a funcionalidade de autenticação com login e senha em S-RES, enquanto que o padrão de teste lista os eventos para a realização dos testes de aceitação dessa funcionalidade. Observe que o padrão de requisito em questão é descrito com os conceitos do BDD, com *feature* (E), *scenario* (F) e *example* (G).

Três cenários estão especificados no padrão de requisito *FRP_AuthenticationLoginPassword*: o de sucesso, *Scenario_SuccessfulLogin* (F), e os de erro, *Scenario_LoginUnsuccessfull_InvalidUserName* e *Scenario_LoginUnsuccessfull_Invalid-*

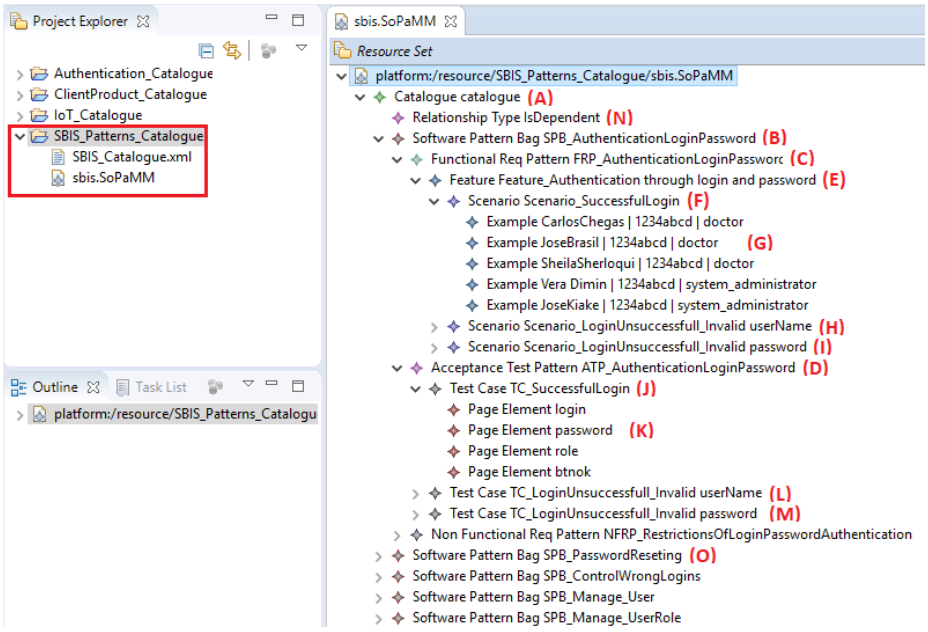

Figura 7. Catálogo de padrões para S-RES na ferramenta TMEd.

Password (H e I, respectivamente). Observe que os elementos do tipo *example* correspondem aos dados utilizados como entrada para os casos de testes, dados estes fornecidos pelo próprio manual de certificação da SBIS.

Já no padrão de teste de aceitação *ATP_AuthenticationLoginPassword* (D) da Figura 7, para cada cenário do padrão de requisito *FRP_AuthenticationLoginPassword* (E) existe um caso de teste (J, L e M) associado com elementos de interface *PageElement* (K). Esses elementos de interface podem ser utilizados para apoiar a automatização de testes de uma aplicação Web ou *mobile*, funcionalidade esta que não é do escopo da ferramenta TMEd.

Além das *bags* de padrões e dos padrões de software em si, a gramática do metamodelo SoPaMM permite a especificação de relacionamentos. No lado direito da Figura 7, tem-se a definição de um relacionamento de dependência, i.e., *RelationshipType IsDependent* (N). Com essa definição, estabeleceu-se uma relação de dependência entre duas *bags* de padrões, no caso, a *bag SPB_PasswordResetting* (O) só pode ser utilizada a partir da *bag SPB_AuthenticationLoginPassword* (B).

Por fim, os autores estão cientes de que a construção de catálogos de PRS com comportamento não é uma tarefa trivial. Para que sejam uma realidade no cotidiano de empresas, os catálogos devem ser elaborados com a ajuda de especialistas de domínio. Neste exemplo de uso da TMEd, o próprio manual de certificação da SBIS serve como compilação dos requisitos legais na visão de especialistas em S-RES, o que serviu de apoio à construção do catálogo realizado por um pesquisador com 15 anos de experiência em Engenharia de Requisitos.

5 Considerações Finais

Apesar das experiências positivas com o uso de PRS na Engenharia de Requisitos e da inerente associação entre requisitos e artefatos produzidos ao longo do ciclo de vida de software, um número reduzido de pesquisas tem investigado a adoção de PRS nas fases de *design*, construção, testes e manutenção.

Voltada para atender às lacunas descritas em [7,8,9], a ferramenta TMEd proposta neste artigo é um dos elementos de nossa abordagem de PRS, ao apoiar a elaboração de catálogos de padrões de requisitos integrados com padrões de teste por meio de especificações de comportamentos segundo as práticas do BDD.

A ferramenta TMEd é resultado de um esforço em andamento, não estando ainda madura para uso em produção, já que lhe falta, por exemplo, uma funcionalidade de gerenciamento de perfis de usuários. Com isso, os autores acreditam que o suporte oferecido pela TMEd seja corroborado por uma avaliação qualitativa junto a profissionais instruídos sobre a gramática do metamodelo SoPaMM.

Espera-se que, em breve, a ferramenta TMEd seja utilizada para produzir catálogos de PRS para outros domínios, além daquele desenvolvido para certificação de S-RES. À medida que novos catálogos forem produzidos, estes deverão ser disponibilizados publicamente, por exemplo, na forma como opera a iniciativa Diaspora[5], para maior disseminação da proposta de PRS com comportamento.

Os catálogos produzidos pela TMEd servirão de insumo para uma ferramenta em desenvolvimento, chamada *behavior-DRivEn Application Model generator* (DREAM). Essa ferramenta apoia a construção de modelos de aplicação usando os catálogos de PRS da TMEd como referência, assim como aquelas descritas em [1,3]. Sendo assim, a ideia é aproximar-se da indústria para investigar, em projetos reais, os benefícios do uso da abordagem de PRS com comportamento proposta. Um exemplo de parceria em andamento é a de membros da SBIS para avaliar o quanto o catálogo de padrões desenvolvido pode auxiliar no processo de certificação de S-RES.

Por fim, espera-se que todos esses esforços – o metamodelo SoPaMM, as ferramentas TMEd e DREAM e os catálogos de PRS com comportamento – ajudem a comunidade a melhor compreender o impacto do uso de PRS além da fase de Engenharia de Requisitos

Agradecimentos

Este estudo foi parcialmente financiado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Referências

1. Barcelos, L.V., Penteado, R.: Especificação de requisitos no domínio de sistemas de informação com o uso de padrões. In: Proceedings of XIX Ibero-American Conference on Software Engineering, CIbSE 2016, Quito, Ecuador, April 27-29, 2016. pp. 338–351 (2016)

[5] Disponível online em `https://wiki.diasporafoundation.org/`.

2. Beckers, K., Côté, I., Goeke, L.: A catalog of security requirements patterns for the domain of cloud computing systems. In: Proceedings of the ACM Symposium on Applied Computing. pp. 337–342. Gyeongju, Republic of Korea (2014)
3. Beckers, K., Heisel, M., Côté, I., Goeke, L., Güler, S.: Structured pattern-based security requirements elicitation for clouds. In: 2013 International Conference on Availability, Reliability and Security. pp. 465–474 (Sep 2013). https://doi.org/10.1109/ARES.2013.61
4. Chelimsky, D., Astels, D., Helmkamp, B., North, D., Dennis, Z., Hellesoy, A.: The RSpec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends. Pragmatic Bookshelf, Raleigh, NC, 1st edn. (2010)
5. Franch, X., Palomares, C., Quer, C., Renault, S., De Lazzer, F.: A metamodel for software requirement patterns. In: Wieringa, R., Persson, A. (eds.) Requirements Engineering: Foundation for Software Quality. pp. 85–90. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
6. Konrad, S., Cheng, B.H.C.: Requirements patterns for embedded systems. In: Proceedings IEEE Joint International Conference on Requirements Engineering. pp. 127–136 (2002)
7. Kudo, T.N., Bulcão-Neto, R.F., Vincenzi, A.M.R.: Requirement patterns: A tertiary study and a research agenda. IET Software **14**(1), 18–26 (2020)
8. Kudo, T.N., Bulcão Neto, R.F., Macedo, A.A., Vincenzi, A.M.R.: Padrão de requisitos no ciclo de vida de software: Um mapeamento sistemático. In: Proceedings of the Conference: CIbSE2019 - Congresso Ibero Americano em Engenharia de Software. pp. 420–433. Havana, Cuba (2019)
9. Kudo, T.N., Bulcão Neto, R.F., Macedo, A.A., Vincenzi, A.M.R.: A revisited systematic literature mapping on the support of requirement patterns for the software development life cycle. Journal of Software Engineering Research and Development **7**, 9:1–9:11 (dec 2019)
10. Kudo, T.N., Bulcão Neto, R.F., Vincenzi, A.M.R.: A conceptual metamodel to bridging requirement patterns to test patterns. In: Proceedings of the XXXIII Brazilian Symposium on Software Engineering. pp. 155–160. ACM, New York, NY, USA (2019). https://doi.org/10.1145/3350768.3351300
11. Kurtev, I., Bézivin, J., Jouault, F., Valduriez, P.: Model-based DSL frameworks. In: Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications. pp. 602–616. OOPSLA '06, ACM, New York, NY, USA (2006). https://doi.org/10.1145/1176617.1176632
12. Meszaros, G.: XUnit Test Patterns: Refactoring Test Code. Prentice Hall PTR, Upper Saddle River, NJ, USA (2006)
13. OMG: Meta Object Facility (MOF) Specification, version 1.4. Object Management Group, Inc. (2002)
14. Palomares, C., Quer, C., Franch, X.: Pabre-Man: Management of a requirement patterns catalogue. In: 2011 IEEE 19th International Requirements Engineering Conference. pp. 341–342 (Aug 2011). https://doi.org/10.1109/RE.2011.6051666
15. Palomares, C., Quer, C., Franch, X., Guerlain, C., Renault, S.: A catalogue of non-technical requirement patterns. In: 2012 Second IEEE International Workshop on Requirements Patterns (RePa). pp. 1–6 (Sep 2012). https://doi.org/10.1109/RePa.2012.6359969
16. Palomares, C., Franch, X., Quer, C.: Requirements reuse and patterns: A survey. In: Proceedings of the 20th International Working Conference on Requirements Engineering: Foundation for Software Quality - Volume 8396. pp. 301–308. REFSQ 2014, Springer-Verlag New York, Inc., New York, NY, USA (2014)

17. Palomares, C., Quer, C., Franch, X., Renault, S., Guerlain, C.: A catalogue of functional software requirement patterns for the domain of content management systems. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Coimbra, Portugal, March 18-22, 2013. pp. 1260–1265 (2013)
18. da Silva, M.L., Junior, L.A.V.: Manual de certificação para sistemas de registro eletrônico em Saúde. Sociedade Brasileira de Informática em Saúde, 4.3 edn. (mar 2019)
19. Smart, J.: BDD in Action: Behavior-Driven Development for the Whole Software Lifecycle. Manning Publications (2014), https://books.google.com.br/books?id=2BGxngEACAAJ
20. Supakkul, S., Hill, T., Chung, L., Tun, T.T., do Prado Leite, J.C.S.: An NFR pattern approach to dealing with NFRs. In: 2010 18th IEEE International Requirements Engineering Conference. pp. 179–188 (2010)
21. Wiegers, K.E., Beatty, J.: Software Requirements 3. Microsoft Press, Redmond, WA, USA (2013)
22. Withall, S.: Software Requirement Patterns. Best practices, Microsoft Press, Redmond, Washington (2007)