Aplicação de Lógica Descritiva para Documentação e Validação de Requisitos em SPL

Fernando Antônio A. Nóbrega [1], **Vaston Gonçalves**[1], **Luanna L. Lobato**[1,2]

[1] Departamento de Ciência da Computação - Universidade Federal de Goiás (UFG)
Catalão, GO – Brasil

[2]Centro de Informática - Universidade Federal de Pernambuco (UFPE)
Recife, PE – Brasil

{fernandoasevedo,vaston}@gmail.com, lll@cin.ufpe.br

Abstract. *The approach of the Product Line Software suffers from high initial cost of the project, in essence caused by the complexities that exist in its phase of requirements engineering due to the characteristics of this development process. Aiming thus this work to present a method of requirements elicitation logically formal guided by the initial decrease of these costs is financial or time, using methods of Knowledge Engineering in Description Logics for this purpose, is presenting an example of using Ontology formalization of requirements, emphasizing the use of automated testers to validate requirements in order to automate and streamline the requirements elicitation phase.*

Resumo. *A abordagem de Linha de Produto de Software sofre com alto custo inicial de projeto, em sua essência ocasionado pela complexidade existente em sua fase de Engenharia de Requisitos devido às características deste processo de desenvolvimento. Objetivando assim este trabalho à apresentar um método de elicitação de requisitos de maneira lógica formal pautado na diminuição destes custos iniciais seja financeiro ou tempo, utilizando-se de métodos de Engenharia de Conhecimento em Lógica Descritiva para este propósito, é apresentando um exemplo da utilização de formalização de requisitos em Ontologias, enfatizando a utilização de provadores automáticos para validação de requisitos afim de automatizar e otimizar a fase de elicitação de requisitos.*

1. Introdução

A engenharia de *software* depara-se com problemas observados ainda na fase de documentação de requisitos, fase esta que possui o objetivo de identificar de forma concreta as funcionalidades e regras que o sistema deve atender. Documentações incoerentes, imprecisas e/ou ambíguas são problemas que levam em sua maioria à atrasos no desenvolvimento de sistemas, impulsionando a uma baixa da qualidade do produto.

A abordagem de Linha de Produto de *Software* (*Software Product Line*, SPL) diferencia-se do padrão de Desenvolvimento de Sistema Individual (*Single System Development*, SSD), pelo emprego da prática de reutilização de seus artefatos e por tratar diferentes necessidades no projeto. Tal metodologia é capaz de gerar produtos com algumas funcionalidades diferentes dentro do domínio ao qual a SPL é desenvolvida.

As características da abordagem SPL geram um maior custo inicial de projeto, ocasionado pela necessidade de tratar as variações existentes dentro de um domínio

de mercado [Neiva 2009] e [Clements 2001]. Conceito este que objetiva este trabalho, pautando-se na possibilidade de representar os requisitos existentes e forma em que são documentados e validados de maneira formal utilizando-se de uma linguagem lógica e matemática.

A documentação de requisitos em SPL, seja a representação textual ou através de diagramas UML, em suas necessidades de especificar diferentes produtos busca auxílio no modelo ortogonal proposto em [Pohl 2005] que, de forma visual ou através de links, busca representar toda a especificidade bem como as relações entre os requisitos em SPL. Criando, assim, uma rede de relação entres os *assets*.

Porém, a tarefa de verificar a coerência e validar os requisitos é deixada de lado nesta forma de especificação, ficando assim, a cargo da equipe de desenvolvimento. Devido ao aumento da complexidade dos requisitos em SPL, quando comparada com o SSD, a verificação e validação acabam por exigir um custo maior e se tornam mais suscetíveis à erros [Neiva 2009].

Assim, este artigo tem como foco utilizar-se da área de representação de conhecimento juntamente com o processo de documentação de requisitos em SPL. Utilizando como linguagem de representação DL (Lógica Descritiva, do inglês *Description Logic*) mais especificamente Ontologias, este trabalho visa apresentar aplicação de engenharia de conhecimento e representação formal à fase de elicitação de requisitos em SPL buscando uma metodologia para validação automática da documentação, como meta amenizar os custos iniciais de projeto.

2. Lógica Descritiva e Ontologias

Lógica Descritiva é um termo usado para referenciar um conjunto de linguagens lógicas para fins da representação do conhecimento utilizadas em Inteligência Artificial (IA). Tais linguagens se originam de formalismos de representação de conhecimento da década de 70, como redes semânticas e quadros (*frames*) [Baader et al. 2003].

Uma forma de utilização dos conceitos de DL dá-se no emprego de Ontologias, que constitui-se de uma descrição formal de conceitos em um determinado domínio (classes ou conceitos), propriedades de cada conceito descrevendo suas características e/ou atributos (regras ou propriedades) e restrições [Natalya and Deborah 2009].

Ontologias podem ser expressas em diversas linguagens e padrões, adotaremos a linguagem OWL (*Web Ontology Language*) que fornece o formalismo necessário para DL em uma sintaxe XML de fácil integração com aplicações web [1] , capacitando agentes de *softwares* a trocarem informações e inferirem novos conhecimentos da Base de Conhecimento (*Knowledge Base*, KB).

Estes agentes de *software* normalmente provadores automáticos de DL, possuem a tarefa de verificação se determina fórmula é consequência lógica de uma TBox. Esta designação dos provadores dá-se pelas três formas em que são realizadas consultas à uma KB em DL:

1. Insatisfabilidade: Verifica se determinado conceito não satisfaz as designações da KB;

[1] Outras razões para utilização desta linguagem serão apresentadas na seção 6

2. Equivalência: Verifica se conceito X é equivalente à outro(s) na mesma KB;
3. Disjunção: Verifica se a união de dois ou mais conceitos da KB não é consequência lógica da mesma;

O conhecimento desses métodos de consulta à KB são importantes para o processo de Classificação de Ontologia [2] . A informação inserida em uma KB seja por engenharia de conhecimento ou agente de *software*, armazena dados que sendo interpretados geram conhecimentos explícitos, porém tais informações em geral possuem conhecimentos implícitos muitas vezes descobertos pela aplicação de provadores em DL, auxiliando a verificação e validação dos dados.

3. Características de Documentação para Requisitos em SPL

Uma Linha de Produtos de *Software* é um conjunto de sistemas de software que compartilham um conjunto de *features* (características) comuns e gerenciáveis, que satisfazem as necessidades de um segmento de mercado particular [Clements 2001].

SPL distingue-se de SSD, por ser capaz de gerar produtos com funcionalidades diferentes dentro do domínio ao qual é proposta, atingindo uma maior variedade de necessidades de seus usuários. Para este propósito torna-se necessário mecanismos de documentação de requisitos que lidem com estas variações, deixando mais complexo o processo de desenvolvimento.

A documentação de requisitos em SPL deve tratar com as seguintes características:

- Variação;
- Obrigatoriedade ou Comunalidade;
- Cardinalidade;
- Dependências;
- Exclusões;
- Reuso;

Variação é o ponto chave dentro de uma SPL, a elicitação e documentação de requisitos devem ser claras quanto à este elemento da linha. A distinção do que é Variável(*Variation*, V) e do que é Pontos Variáveis (*Variation Point*, VP) devem ser objetivas e concisas.Denomina-se VP um requisito no sistema alvo de variação, ou seja, um requisito que pode portar-se de maneiras distintas e/ou mais abrangente para atender um determinado domínio da linha, sendo estas manerias seus respectivos Vs (suas variações).

Obrigatoriedade ou Comunalidade tratam da necessidade de se instanciar determinado requisito nos produtos da linha, são requisitos comuns à variação da linha. Em uma SPL existem requisitos comuns e opcionais para seu correspondente segmento de mercado, sendo os requisitos necessários denominados mandatórios.

Os *assets* de ER devem distinguir tais distinções entre requisitos, ou seja, representar quais são os componentes mandatórios e quais os opcionais. Permitindo assim, o gerenciamento dos distintos produtos finais que atendam subdomínios em particular.

Cardinalidade, determinados componentes (VP) em SPL podem conter uma quantidade específica de variações (Vs) que podem ser instanciadas em conjunto. Exemplificando por um sistema de segurança, onde se deve identificar os usuários esta pode ser

[2]Geração de novos conhecimentos implícitos inseridos na KB, seja por intervenção humana ou computacional

feita por impressão digital, senha de autenticação, cartão magnético dentre outros, assim o sistema pode ser composto por um ou mais destes métodos.

Dependência assim como na abordagem SSD em SPL pode haver relação de dependência entre os requisitos. Trata-se quando determinado componente a ser instanciado no sistema obriga a inclusão de algum(s) outro(s) componente(s), ou seja, são componentes que necessitam de outras características do sistema para ser implementado.

Exclusões de forma análoga à dependência as exclusões tornam-se necessárias devido a existência de requisitos conflitantes, onde determinado V opcional do sistema conflita-se com outro requisito e assim, sendo instanciados de forma disjunta no produto final.

Reuso é alvo desta abordagem. SPL enfatiza o processo de desenvolvimento com maturação de artefatos, onde componentes implementados, testados e documentados anteriormente venham a ser utilizados em produtos posteriores, caracterizando também a importância de correta documentação.

Todas estas características são abordadas por elementos gráficos do modelo Ortogonal mostrado na figura 1 [Pohl 2005].

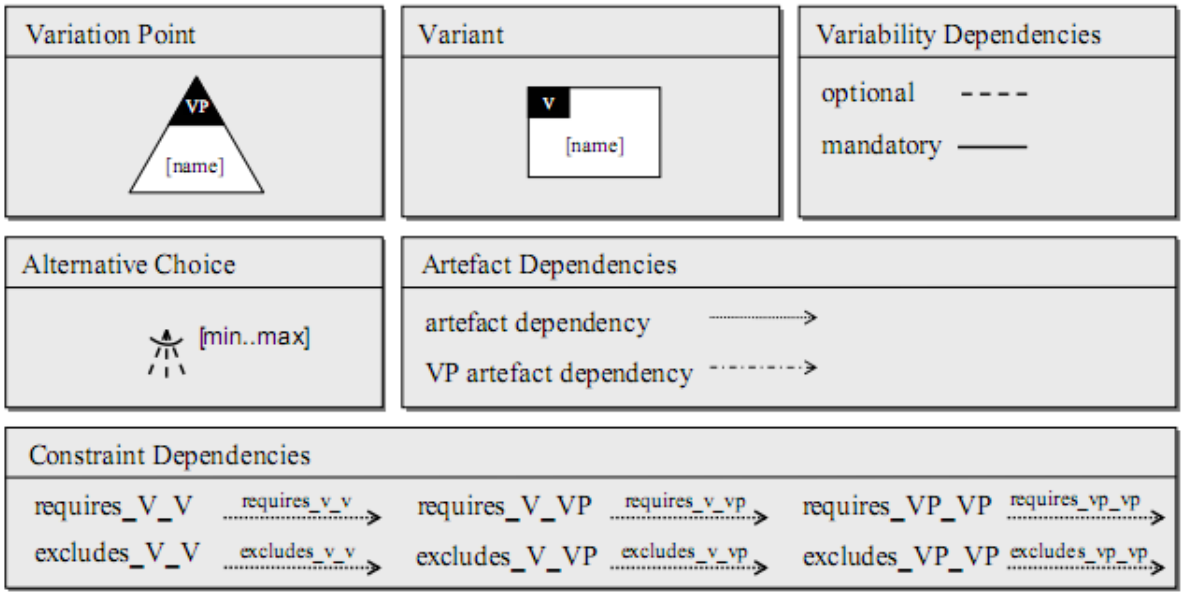

Figura 1. Modelo Ortogonal.

É visto que requisitos em SPL englobam um conjunto de características, não simplificadas à Variação do produto, há uma criação de rede de relacionamento entre os requisitos demarcando inclusão, exclusão e numeração além da necessidade de distinção entre requisitos opcionais e mandatórios. É esta complexidade de documentação um dos pontos que acarretam um maior custo inicial de projeto [Neiva 2009].

4. Mapeamento de Requisitos SPL para DL

Nesta seção é apresentando um exemplo de utilização de modelos formais de documentação de requisitos SPL em Ontologias, a fim de expor a sua aplicabilidade e benefícios

de utilização. Seguindo esta proposta de validação da documentação, é abordado uma situação contendo erros de especificação simulando uma situação real de incoerência nos dados coletados na fase de levantamento de requisitos.

Este exemplo trata um módulo de cadastro de usuários de determinado sistema, é especificado que o usuário deve ser caracterizado por um nome do tipo *String* e uma senha do tipo *String*. Adotando para este sistema dois tipos distintos de usuários: usuário administrador (Admin) e usuário cliente (Client).

Para atingir este domínio é especificado que os tipos de usuários é um VP do sistema, neste caso podendo ser composto pelo usuário Client e opcionalmente pelo usuário Admin. Assim temos o ambiente composto pelo VP User (objetivo do sistema é cadastrar usuário) e seus respectivos Vs Client e Admin (formas distintas de se cadastrar um usuário). O mapeamento em DL foi descrito com as seguintes classes em Ontologia:

- DataType representando os tipos dos dados do sistema;
- Boolean subclasse de DataType representa o tipo de dado lógico;
- Integer subclasse de DataType representa o tipo numero inteiro;
- String subclasse de DataType representa uma cadeia de caracteres não numéricos;
- User define um usuário do sistema;
- Client subclasse de User, define o V cliente do VP User;
- Admin subclasse de User, define o V administrador do VP User;

A nomenclatura aqui é utilizada no idioma inglês devido a padronização de escrita de modelos em Ontologias, também é padrão a presença da classe *Thing* presentes nas figuras 2 e 3, usada para descrever a classe mais genérica da representação [Horridge et al. 2004].

Para este ambiente foram também definidas duas propriedades, sendo elas utilizadas para representar as características de um usuário possuir nome e senha:

- hasName especificando o tipo do dado do nome;
- hasPassword especificando o tipo do dado utilizado na senha;

Representamos um User com as propriedades acima como sendo do tipo String, ou seja, a senha e o nome serão compostos por uma cadeia de caracteres. User trata-se da classe mais genérica (o VP do sistema) que possuis os filhos (Vs) Client e Admin, estes devem respeitar a definição dos tipos dos dados.

Inserimos um erro na especificação, onde Admin terá senha composta por um número inteiro. Isso foge a especificação de User que deve possuir senha formada por caracteres, formalizando assim um erro de coerência na documentação. Ou seja, há um erro de especificação onde User possui senhas do tipo String e Admin que é um tipo de User possui senhas do tipo Integer.

É apresentado na figura 2 a hierarquia de VP e Vs antes da aplicação do raciocinador automático. Apresentando Client e Admin como subclasses de User caracterizando a relação de Vs e VP nesta ordem, o mesmo ocorre com DataType e suas descendentes. Este grafo respeita as especificações documentadas manualmente.

Após elaboração desses requisitos formalmente em ontologia, foi aplicado um provador automático de teorema para análise das expressões registradas. Utilizando-se do *software* Protégé foi possível encontrar, o erro de especificação inserido na documentação, apresentado na figura 3.

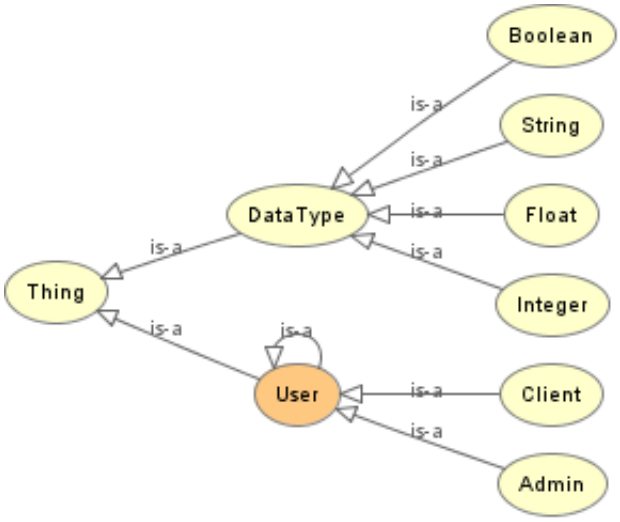

Figura 2. Hierarquia da documentação do sistema.

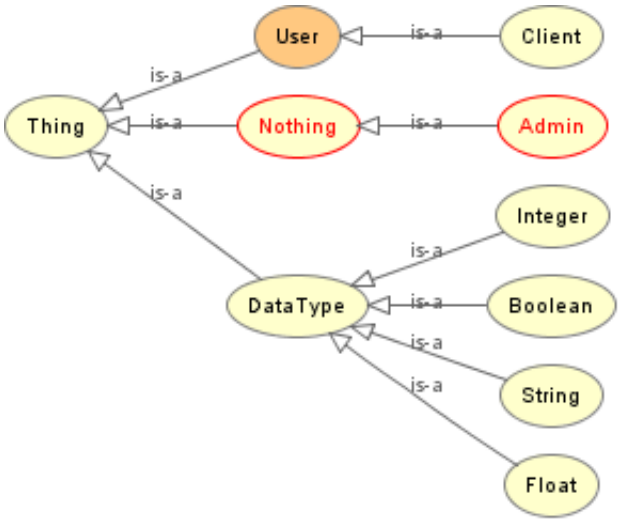

Figura 3. Resultado do provador de teorema.

È apresentado com o grafo gerado pelo raciocinador uma situação onde, o requisito Admin é designado como subclasse de Nothing, padrão para especificação de erros de coerência [Natalya and Deborah 2009]. Em outras palavras, a documentação contém erros de coerência (citados em parágrafos anteriores) tornando incapaz o tratamento de Admin com sendo uma variante de User. Assim a forma que foi elicitado os requisitos apresenta erros.

5. Trabalhos Relacionados

Nesta seção é apresentando alguns trabalhos semelhantes ao abordado neste artigo, apresentando ao leitor que a área abordada neste trabalho trata-se de fonte de interesse de estudos científicos.

É apresentando em [Nardi and de Almeida Falbo 2006] o desenvolvimento de uma ontologia denominada ODE (Ambiente de Desenvolvimento de *Software* Baseado

em Ontologias, do inglês *Ontology-based Software Development Environment*). Pautado em estabelecer uma conceituação básica para requisitos, servindo como base para a construção de ferramentas de apoio ao processo de Engenharia de Requisitos (ER).

O trabalho supracitado apresenta uma Ontologia afim de formalização do conhecimento sobre requisitos, seguindo métodos de engenharia de conhecimento semelhantes ao utilizados neste trabalho. Sua distinção para com este, dá-se na geração de uma ontologia de propósito geral para requisitos, não tratando as singularidades da abordagem SPL.

Outra distinção do ODE, dá-se no seu desenvolvimento como sendo parte de uma Ontologia maior denominada ADS (Ambiente de Desenvolvimento de *Software*) que visa incluir todo o processo de desenvolvimento de sistemas computacionais. Este trabalho a princípio aborda apenas a fase de ER, podendo futuramente estender-se para todo o domínio da engenharia de *software*.

Outro processo de ER também voltado à SPL porém não formal (matemático ou lógico), trata-se do RIPLE-RE (*Requirements Engineering in Process for Product Line Engineering*, no português Engenharia de Requisito do Processos para Engenharia de Linha de Produto) desenvolvido com apoio do grupo RiSE [3] [Neiva 2009].

O RIPLE-RE aborda um processo sistemático para ER. Sua abordagem dá-se no emprego de três fases distintas sendo estas: Modelagem de Escopo, Modelagem de Requisitos e Definição de Casos de Usos; ressaltando os riscos da ER em SPL e propondo maneiras para gerenciamento de variabi-lidade, apontando esta última característica com ponto chave para os altos custos de ER em SPL.

Outra utilização de especificação formal de requisitos, é vista na utilização da linguagem Z (pronuncia-se Zed) sendo esta caracterizada pela utilização de lógica de primeira ordem, com sintaxe semelhante à linguagem Prolog. Trata-se de uma linguagem de especificação formal de propósito geral [Moura 2001].

6. Trabalhos Futuros

A linguagem OWL citada na seção 2, utilizada para descrição de ontologia possui sintaxe XML, também encontrada em documentos de requisitos SPL, objetivando diminuição de tempo de migração de documentação, propõem-se a criação futura de *software* para tradução de arquivos XML de requisitos SPL para o padrão OWL.

Em trabalho já realizado verificou-se a possibilidade de mapeamento de conteúdo textual para Ontologia de forma automática para posterior verificação de coerência [da Silva et al. 2007]. Viabilizando possível geração de ferramenta computacional para tradução automatizadas de documentação textual de requisitos para ontologias, auxiliando o manuseio de formalismos DL por usuários sem o conhecimento prévio nesta área.

Buscando atender maior abrangência quanto à pesquisadores e desenvolvedores da engenharia de *software* em SPL para com este método formal de documentação, propõem-se o desenvolvimento de mecanismos de explicação automática de prova [4] (mais sofistica-

[3] *Reuse in Software Engineering*

[4] explicação automática de prova, trata-se de área de estudo destinada à pesquisa de tradução dos re-

dos que apenas a geração de um grafo) viabilizando o melhor entendimento dos resultados de provadores de teorema por indivíduos não habituados à estes conceitos.

7. Conclusões

Com a representação formal de requisitos torna-se viável a aplicações de provadores automáticos de teoremas, os quais, por sua vez verificam se as relações entre os requisitos e a coerência na representação dos mesmos está condizente. O processo de ER ponde conter erros de elicitação e/ou alterações exigidas pelos *stakeholders*, consequentemente alterando os requisitos, algumas destas mudanças podem tornar os novos requisitos inconsistentes ou inviáveis quando confrontados com os requisitos de versão anterior do sistema.

Referências

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. (2003). *The Description Logic Handbook*. Cambridge University Press.

Clements, P.; Northrop, L. (2001). *Software product lines: practices and patterns*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition.

da Silva, G. M. H., Rademaker, A., de Vasconcelos, D. R., Amaral, F. N., Bazilio, C., Costa, V., and Haeusler, E. H. (2007). Dealing with the formal analysis of information security policies through ontologies : A case study. In Meyer, T. and Nayak, A. C., editors, *Third Australasian Ontology Workshop (AOW 2007)*, volume 85 of *CRPIT*, pages 55–60, Gold Coast, Australia. ACS.

Horridge, M., Knublauch, H., Rector, A., Stevens, R., and Wroe, C. (2004). *A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODETools*. University Of Manchester, 1 edition.

Moura, A. (2001). *Especificações em Z: uma introdução*. Unicamp.

Nardi, J. C. and de Almeida Falbo, R. (2006). *Uma Ontologia de Requisitos de Software*. PhD thesis, Universidade Federal de Espírito Santo.

Natalya, F. and Deborah, L. M. (2009). *Ontology Development 101: A Guide to Creating Your First Ontology*. Stanford University, Stanford, CA, 94305.

Neiva, D. F. S. (2009). *RiPLE-RE:ARequirements EngineeringProcessforSoftware ProductLines*. PhD thesis, Universidade Federal de Pernambuco.

Pohl, K.; Böckle, G. v. d. L. F. (2005). *Software Product Line Engineering*. Springer.

sultados oriundos dos provadores de teorema para outra linguagem, em geral com maior facilidade de entendimento tais como: linguagem natural, grafos, etc,