Uma abordagem para gerenciamento de Linhas de Produtos de Software baseada em serviços

Karen Dias Rabelo Pacini

Karen Dias Rabelo Pacini

Uma abordagem para gerenciamento de Linhas de Produtos de Software baseada em serviços

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestra em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientadora: Profa. Dra. Rosana Teresinha Vaccare Braga

USP – São Carlos
Junho de 2016

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

Pacini, Karen Dias Rabelo
P118a Uma abordagem para gerenciamento de Linhas de Produtos de Software baseada em serviços / Karen Dias Rabelo Pacini; orientadora Rosana Teresinha Vaccare Braga. - São Carlos - SP, 2016.
123 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2016.

1. Reúso de Software. 2. Linhas de Produtos de Software. 3. LPS. 4. Gerenciamento de LPS. 5. Web Service. I. Braga, Rosana Teresinha Vaccare, orient. II. Título.

Karen Dias Rabelo Pacini

A service-based approach for managing Software Product Lines

Master dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP, in partial fulfillment of the requirements for the degree of the Master Program in Computer Science and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Profa. Dra. Rosana Teresinha Vaccare Braga

USP – São Carlos
June 2016

Este trabalho é dedicado primeiramente a Deus,
a minha família, aos meus professores, aos meus amigos
e a todos os pesquisadores,
em especial, aos pesquisadores do ICMC-USP e da FACOM-UFMS.

AGRADECIMENTOS

Em primeiro lugar gostaria de agradecer à Deus que me deu o dom e a oportunidade de estar aqui desenvolvendo este trabalho, Ele que guia os meus passos e segura a minha mão, sempre me ajudando a levantar e lutar não importa quais sejam as dificuldades.

Em segundo lugar gostaria de agradecer à minha família. Em especial gostaria de agradecer ao meu esposo Alexandre, por todo amor, carinho, dedicação e paciência, por cozinhar o jantar e trazer almoço nos tempos mais corridos deste trabalho, sempre dedicado e compreensivo. Gostaria também de agradecer à minha irmã Stefany, pelo incentivo, ajuda e paciência, sempre com empolgação e bom humor ela participa dos meus desafios, e agradeço até mesmo pelas distrações que ela insiste em me trazer. Agradeço imensamente aos meus pais Carlos e Catarina, que sempre me incentivaram e lutaram pra que eu pudesse ter uma boa educação e sempre apoiaram o meu crescimento, tanto pessoal quanto profissional, eles que me ensinaram todos os valores que aprendi e estiveram presentes em cada conquista e dificuldade, oferecendo seu amor, seus conselhos e seu apoio. Gostaria ainda de agradecer aos meus filhotes, Annie e Lucky, pela companhia e por alegrarem sempre o meu dia.

Gostaria de agradecer imensamente também à Rosana, minha orientadora, por ter tanta paciência e por ser tão parceira. Obrigada pelo seu tempo e dedicação em todas as reuniões, escritas de artigos, discussão de ideias, apresentações, aulas, e nem lembro o que mais. Muito mais que orientadora, ela se tornou uma grande amiga, sempre me escutando e aconselhando tanto na vida acadêmica quanto na vida pessoal.

Em especial agradeço também a um grande amigo Shay que me compreendeu, me apoiou e lutou por mim no momento em que eu mais precisei. E também a todos os meus amigos que compreenderam a minha falta em churrascos e confraternizações por conta deste trabalho, e que me apoiaram de muitas formas.

Agradeço a todos os professores do ICMC e da FACOM-UFMS que participaram da minha formação.

Agradeço a CAPES por financiar este trabalho e valorizar a pesquisa.

“Sonhos determinam o que você quer,
Ação determina o que você conquista.”
(Aldo Novak)

RESUMO

PACINI, K.D.R.. **Uma abordagem para gerenciamento de Linhas de Produtos de Software baseada em serviços**. 2016. 123 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Linhas de Produtos de Software (LPS) abstraem a semelhança entre produtos e envolvem o conceito de reúso de software para desenvolver software em larga escala com mais rapidez e qualidade. O reúso dos artefatos em uma LPS é planejado e executado desde sua concepção, sendo assim possível armazenar esses artefatos em um repositório (núcleo de ativos reusáveis) para utilização posterior. Porém, ao se construir um repositório para uma certa LPS, este atende apenas às especificações desta arquitetura e não promove o reúso desses ativos em outras LPS, causando uma certa perda no potencial de reusabilidade dos ativos produzidos e armazenados. Além disso, a definição e os dados da execução do processo de desenvolvimento dessas LPS também não são armazenados prevendo o reúso, o que gera retrabalho para definir e instanciar um processo de software sempre que uma nova LPS é criada. Neste contexto, este trabalho tem por objetivo apresentar uma abordagem baseada em um conjunto de serviços para promover o amplo reúso de ativos produzidos e de elementos de processo utilizados, além de facilitar a criação e o desenvolvimento de linha de produtos de software, permitindo o gerenciamento do núcleo de ativos reusáveis. Por possuir uma interface baseada em serviços, esta abordagem poderá ser incorporada também a outros ambientes. Para validar a abordagem, verificar vantagens e sugerir melhorias, foi conduzida uma prova de conceito com sua aplicação. Os resultados dessa prova comprovaram a viabilidade da abordagem em cumprir os objetivos propostos.

Palavras-chave: Reúso de Software, Linhas de Produtos de Software, LPS, Gerenciamento de LPS, Web Service.

ABSTRACT

PACINI, K.D.R.. **A service-based approach for managing Software Product Lines**. 2016. 123 f. Master dissertation (Master student Program in Computer Science and Computational Mathematics) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Software Product Lines (SPL) extract the similarity among products and use the software reuse concept to produce large-scale software with increased quality and reduction in the development time. Reuse in SPL is planned and executed since the beginning for each artifact resulting from the development process, and then these artifacts are stored in a repository (reusable core assets) for later use. However, when a repository is built for a specific SPL, it supports only the specifications for that architecture and do not promote the reuse of these assets among distinct SPL, which inhibits their reuse potential. In addition, the definition and execution data of the development process applied are not stored properly for reuse, which causes a re-work to define and instantiate a software process when creating a new SPL. In this context, this work aims at present a service-based approach to promote the wide reuse of the produced assets and process elements, also to facilitate the creation and the management of SPL and to allow the management of the reusable core assets. Once this approach has a service-based interface, it can be embedded into other environments. In order to validate this approach, point out advantages and suggest improvements, a concept proof was performed applying it. The results of this proof ensured the approach viability to fulfill the proposed goals.

Key-words: Software Reuse, Software Product Lines, SPL, Managing SPL, Service-based Development.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE CÓDIGOS-FONTE

LISTA DE TABELAS

SUMÁRIO

CAPÍTULO

1

INTRODUÇÃO

A indústria de software vem se adaptando ao grande aumento de demanda decorrente de uma constante evolução das tecnologias. O conceito de reúso de software ganha muito espaço nessa nova forma de se fabricar software, na qual o tempo de desenvolvimento é reduzido enquanto a qualidade é melhorada (CLEMENTS; NORTHROP, 2002). Ao longo do tempo, várias abordagens surgiram para tentar alcançar este objetivo: paradigma de orientação a objetos, desenvolvimento baseado em componentes, arquitetura orientada a serviços, entre outros.

As linhas de produtos de software (LPS) surgiram neste contexto, para apoiar o reúso realizando a construção de sistemas sob medida, específicos para as necessidades de determinados clientes ou grupos de clientes. Em LPS, o reúso sistemático de cada artefato resultante do processo de desenvolvimento é planejado e executado. Segundo Ezran, Morisio e Tully (2002) esses artefatos encapsulam o conhecimento e são muito valiosos às organizações, pois são uma coleção de produtos de software relacionados que podem ser reusados de uma aplicação para outra. Neste documento é apresentado um trabalho de mestrado relacionado ao reúso de software, especificamente na área de linha de produtos de software. As seções seguintes apresentam a motivação e o objetivo deste trabalho.

1.1 Motivação

A melhoria de qualidade e produtividade de desenvolvimento é um dos grandes desafios da Engenharia de Software e, nesse contexto, técnicas de reúso figuram como uma das soluções já consolidadas (SOMMERVILLE *et al.*, 2008). Por meio do reúso, facilita-se o reaproveitamento de artefatos de qualidade produzidos previamente, tais como documentos de requisitos, modelos de análise e projeto, código-fonte, casos de teste, documentação, entre outros. Entre as técnicas de reúso, linhas de produtos de software (LPS) destacam-se por permitir benefícios como: ganho de produtividade em larga escala; melhoria do tempo de produção; manutenção da presença no mercado; resposta a um rápido crescimento na demanda; aumento de qualidade; aumento da

satisfação do cliente; alcance dos objetivos de reúso; habilitação da customização em massa; e redução de custos. Para alcançar esses benefícios, uma vez que a linha de produtos foi estabelecida, existe uma economia direta a cada produto construído (CLEMENTS; NORTHROP, 2002).

Grande parte das abordagens existentes para desenvolvimento de LPS, tais como *Product Line Pratice* (PLP) (CLEMENTS; NORTHROP, 2002) e *Product Line UML-based Software Engineering* (PLUS) (GOMAA, 2006), focam em processos para apoiar a engenharia de domínio e/ou engenharia de aplicação, sem levar em conta ferramentas computacionais de apoio ao processo. Desta forma, o uso de ferramentas é independente do processo e está mais associado às fases de definição dos Modelos de *Features*[1] e mapeamento para os artefatos que as implementam. Exemplos dessas ferramentas incluem Pure::Variants (BEUCHE, 2012), Gears (FLORES; KRUEGER; CLEMENTS, 2012), e GenArch (CIRILO; KULESZA; LUCENA, 2008), entre outras.

Foi realizado um mapeamento sistemático para a verificação do estado da arte com respeito às ferramentas que oferecem algum tipo de apoio ao gerenciamento de LPS. Neste mapeamento, ficou evidenciado que essa é uma área crescente e atual de pesquisa, porém não existem muitas soluções disponíveis que oferecem tal apoio, e as poucas mais completas são soluções privadas. Ficou claro também que grande parte das soluções não promove o reúso dos artefatos entre outros projetos ou até mesmo entre outras linhas de produtos. Além disso, muitas soluções possuem limitações referentes à linguagem de programação e a complexidade de utilização que dificulta sua adoção. Este mapeamento é apresentado em detalhes em um capítulo deste trabalho.

A partir do estado da arte observado, motivou-se a criação de uma abordagem para o gerenciamento de LPS que seja flexível e cuja utilização seja independente de linguagem de desenvolvimento. Essa abordagem também poderia facilitar e difundir o reúso de forma a não compartilhar somente os artefatos produzidos pela LPS, mas também estender o reúso no nível de processos e de elementos de processo. Esse compartilhamento não necessariamente deve permanecer somente dentro do contexto de uma linha de produtos específica, mas entre diversas aplicações em diferentes contextos.

Essa abordagem deve permitir a diminuição dos custos associados tanto à engenharia de domínio quanto à engenharia de aplicação e promover a reusabilidade e o compartilhamento de informação em toda a comunidade científica. Na engenharia de domínio, a abordagem deve facilitar o reúso de artefatos em níveis de abstração que vão além do código fonte (modelos, por exemplo), além do reúso dos elementos de processo, e maior facilidade de manutenção dos artefatos durante a evolução da LPS. Na engenharia de aplicação, um diferencial da abordagem se refere ao processo de geração de produtos e ao gerenciamento e compartilhamento desses produtos.

[1] Será utilizada a palavra em inglês por esta não possuir tradução correspondente na língua portuguesa.

1.2 Objetivo

Dada a motivação, este trabalho tem por objetivo apresentar uma abordagem baseada em serviços para apoiar o gerenciamento de LPS e potencializar o reúso. A utilização de serviços objetiva tornar a abordagem flexível, sendo facilmente customizada; acessível, disponível na *web;* e interoperável, sendo que sua utilização é independente de linguagem de desenvolvimento. A potencialização do reúso ocorre por meio da criação de uma abordagem para representar os ativos reusáveis utilizando uma especificação comum e estendendo o reúso aos elementos de processo e a níveis de requisitos e *features* individuais.

Os resultados obtidos com a aplicação da abordagem por meio da prova de conceito, demonstraram a viabilidade da utilização dos serviços desenvolvidos para cumprir o que está sendo esperado nesses objetivos estabelecidos, a partir de uma abordagem que oferece a engenheiros de domínio e de aplicação o apoio à reutilização tanto de processos quanto de artefatos de diversas fases além do gerenciamento de todo o processo de desenvolvimento e de manutenção tanto da LPS quanto dos sistemas derivados.

1.3 Organização

Este trabalho está organizado da seguinte forma: o Capítulo 2 apresenta uma fundamentação teórica dos principais conceitos necessários que servem de base para este trabalho; no Capítulo 3 é apresentada uma revisão bibliográfica que inclui um mapeamento sistemático e trabalhos relacionados; o Capítulo 4 apresenta uma abordagem de mapeamento de elementos de processo para uma especificação padrão (RAS); o Capítulo 5 apresenta os serviços que foram construídos para promover o gerenciamento das LPSs; o Capítulo 6 apresenta a modelagem de casos de uso de uma ferramenta que pode ser implementada utilizando os serviços oferecidos; e finalmente o Capítulo 7 apresenta as conclusões deste trabalho e as sugestões para trabalhos futuros.

CAPÍTULO

2

FUNDAMENTAÇÃO TEÓRICA

Alguns conceitos importantes se fazem necessários para a compreensão deste trabalho. Assim, neste capítulo são apresentados os conceitos de reúso de software; linhas de produtos de software, com uma breve introdução de sua origem, objetivos, benefícios e uma visão geral de algumas abordagens de desenvolvimento; especificação de ativos reusáveis; abordagens dirigidas por modelos; linhas de produtos dirigidas por modelos; e a apresentação do conceito de *web services* com uma breve introdução do descritor WSDL e do protocolo SOAP.

2.1 Reúso de Software

A engenharia de software baseada em reúso é uma abordagem de desenvolvimento que tenta maximizar o reaproveitamento de esforços anteriores com base em um software já existente. O movimento para o desenvolvimento baseado em reúso foi uma resposta às demandas por menores custos de produção e manutenção de software, entregas mais rápidas de sistemas e aumento da qualidade de software. As unidades de software reusadas podem ser de tamanhos e níveis de abstração diferentes. Um sistema de aplicação pode ser reusado por completo, incorporando-o a outros sistemas sem modificá-lo, ou pode-se apenas reusar componentes de uma aplicação, ou classes de objetos, funções (bibliotecas), modelos, testes, entre outros.

O reúso pode trazer vários benefícios, como: redução de custo; redução de risco; aceleração do desenvolvimento; maior confiabilidade; conformidade com padrões; entre outros. Porém, além dos benefícios, há também problemas como: custo associado ao entendimento da viabilidade de um componente; incompatibilidade com algumas ferramentas CASE; custo de criação e manutenção de uma biblioteca; procura, compreensão e adaptação de componentes (SOMMERVILLE *et al.*, 2008).

Técnicas desenvolvidas para apoiar o reúso de software exploram o fato de que sistemas em um mesmo domínio de aplicação são similares, havendo padrões para componentes reusáveis

que facilitam e potencializam o reúso. Algumas abordagens que apoiam o reúso de software são: Padrões de Projeto (do inglês *Design Patterns*), Desenvolvimento Baseado em Componentes, Frameworks de Aplicação, Sistemas Orientados a Serviços, Linhas de Produtos de Software, Desenvolvimento de Software Orientado a Aspectos, etc.

Existe uma grande variedade de técnicas de reúso de software, no entanto, a decisão de reusar é mais gerencial do que técnica, o que pode desmotivar o reúso, uma vez que os riscos associados ao reúso podem ser desconhecidos em relação ao desenvolvimento de um novo software, mesmo nos casos em que a segunda opção é mais cara (SOMMERVILLE *et al.*, 2008).

2.2 Linha de Produtos de Software

As LPS são definidas por Clements e Northrop (2002), como um conjunto de sistemas intensivos que compartilham um núcleo comum, tendo características que satisfazem às necessidades específicas de um determinado segmento de mercado ou missão. São desenvolvidas a partir de um conjunto comum de ativos principais de uma forma pré-estabelecida.

O sucesso de uma linha de produtos se deve ao fato do compartilhamento de semelhanças entre os produtos poder ser explorado para atingir uma economia na produção. Uma linha de produtos oferece economia de escopo, o que significa uma vantagem econômica sobre o fato de vários produtos serem muito similares por terem sido projetados dessa forma.

Os produtos gerados por uma linha de produtos são chamados famílias de produtos. A construção de um novo produto de uma família de produtos acontece por meio da geração e montagem de componentes ao invés de simples criação, sendo a integração mais predominante que a programação. Para cada linha de produto existe um plano ou guia pré-definido que especifica exatamente o processo de construção de um novo produto.

Artefatos são quaisquer produtos resultantes do processo de desenvolvimento de software, como documento de requisitos, modelos, arquivos de código fonte, scripts, casos de teste, entre outros. Todos os artefatos do processo de desenvolvimento são armazenados no chamado núcleo de ativos reusáveis (também conhecido como plataforma ou engenharia de domínio) e formam a base para o desenvolvimento das linhas de produtos. Além dos artefatos já citados, o núcleo de ativos reusáveis geralmente inclui: arquitetura, componentes reusáveis, modelos de domínio, documentação e especificação, modelos de performance, cronogramas, orçamentos, planos de testes, planos de trabalho, descrição de processos, entre outros.

Existem vários tipos de ativos e cada um deles tem suas especificidades, o que pode incluir artefatos e contextos. Existem três dimensões chaves que descrevem os ativos reusáveis: granularidade, variabilidade e articulação (GROUP, 2005).

A granularidade descreve o quão particular é o problema ou solução que o ativo resolve ou oferece, quanto maior a granularidade, mais complexo é o ativo.

A variabilidade e a visibilidade são propriedades fundamentais de um ativo. Ativos chamados de 'caixa-preta' são invariáveis e seu funcionamento interno não é visível, como no caso de ativos que são componentes binários. Por outro lado, os ativos chamados de 'caixa-branca' são criados com a expectativa de serem editados e alterados, como no caso dos artefatos de desenvolvimento (requisitos, modelos, arquivos, etc.). Quando um ativo possui características ou funcionalidades que podem ser adicionadas, alteradas e/ou removidas, estas são chamadas variantes. A variabilidade de um ativo é definida pelo conjunto de variantes que ele possui e sua aplicação é definida por um conjunto de regras. A verificação e a validação na aplicação dessas regras são realizadas pela Gerência de Variabilidades.

A articulação descreve o grau de completude dos artefatos em fornecer a solução. Os ativos em que artefatos especificam uma solução mas não a fornecem, possuem um baixo grau de articulação, enquanto que os ativos que possuem artefatos que especificam e implementam uma solução juntamente com os documentos de apoio (requisitos, casos de uso, testes, etc.) têm um maior grau de articulação. O conjunto de ativos e o plano de como eles serão utilizados para a construção de novos produtos requerem visão organizacional, investimento, planejamento, direção, e ainda um pensamento estratégico que vai além de um único produto.

Apesar das vantagens como melhorias no tempo e qualidade, entre outros, as linhas de produtos de software possuem algumas dificuldades. Ao iniciar um projeto de LPS, um alto investimento se faz necessário, incluindo altos custos durante o processo de desenvolvimento e manutenção da linha de produtos. Uma vez que a linha de produtos é estabelecida, a produtividade da organização acelera rapidamente e os benefícios superam os custos. No entanto, uma organização que pretende instituir uma linha de produtos sem ter consciência deste fato, tende a abandonar o projeto muito antes de poder observar os resultados. Outra grande dificuldade é a escolha da abordagem mais indicada para desenvolver a linha de produtos, pois uma escolha errada poderia resultar em grandes prejuízos.

2.3 Abordagens para Desenvolvimento de LPS

Dentre as várias abordagens existentes para o desenvolvimento de LPS, aqui são citadas apenas duas de maior interesse neste projeto: *Product Line Pratice* (PLP) proposta por Clements e Northrop (2002) e *Product Line UML-based Software Engineering* (PLUS) proposta por Gomaa (2006).

2.3.1 PLP

Na abordagem PLP existem três atividades essenciais para a produção de uma linha de produtos: desenvolvimento do núcleo de artefatos, desenvolvimento do produto e gerenciamento, como mostra a Figura 1. O desenvolvimento do núcleo de artefatos também pode ser chamado de Engenharia de Domínio, enquanto que o desenvolvimento do produto também pode ser chamado

de Engenharia de Aplicação. Em sua essência, primeiramente desenvolve-se o núcleo de artefatos e a partir dele desenvolve-se o produto, ambos sob o apoio técnico e organizacional da gerência. Porém, esses processos podem ser realizados em ambas as ordens: um novo produto é construído a partir de um núcleo de artefatos ou o núcleo de artefatos pode ser construído por meio de produtos já existentes.

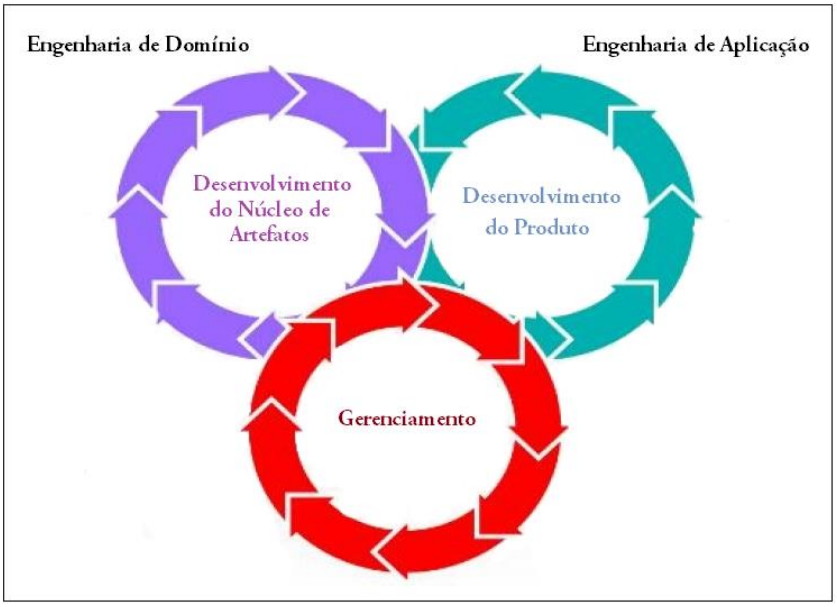


Figura 1 – Atividades PLP - Adaptada de Clements e Northrop (2002).

A Figura 1 ilustra essas três atividades, explicitando sua interatividade e interligação por flechas rotativas que se sobrepõem. A engenharia de domínio envolve a revisão e a inclusão de novos artefatos, de acordo com a demanda da engenharia de aplicação, podendo essas mudanças gerar novos produtos. Durante todo o desenvolvimento dessas atividades, o gerenciamento sempre se faz presente organizando todo o processo.

2.3.2 PLUS

O processo utilizado pela abordagem PLUS é evolucionário e possui duas atividades principais: a Engenharia da Linha de Produto (ou Engenharia de Domínio: modelagem de várias versões da LPS) e a Engenharia de Aplicação (configuração do sistema alvo que resulta em novos produtos). A Figura 2 ilustra esse processo.

Para cada uma das atividades principais, tanto Engenharia de Domínio quanto de Aplicação, existe um processo evolucionário correspondente definido (*Evolutionary Software Product Line Engineering Process* - ESPLEP).

Conforme pode ser observado na Figura 3[1], no contexto de Engenharia de Domínio, existem cinco fases: Modelagem de Requisitos, Modelagem da Análise, Modelagem do Projeto,

[1] Modelos e diagramas UML foram criados utilizando a ferramenta Astah (VISION, 2016)

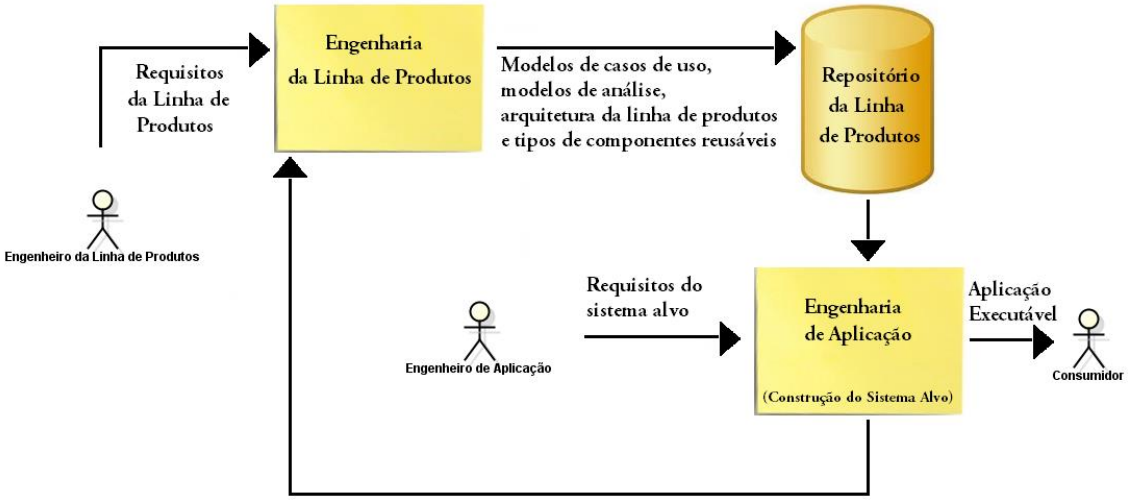


Figura 2 – Processo de aplicação da abordagem PLUS - Adaptada de Gomaa (2006).

Implementação Incremental de Componentes e Teste de Software. Na fase de Modelagem de Requisitos é definido o escopo da LPS e são realizadas as modelagens de casos de uso e criado o Modelo de *Features*. A fase de Modelagem de Análise compreende as modelagens estática, dinâmica e de máquina de estados finitos, a construção de objetos e a análise de dependência de *features* e/ou classes. A Modelagem de Projeto envolve a definição da arquitetura da LPS, que é baseada em padrões.

Nos diagramas são usados estereótipos para identificar os diferentes tipos de casos de uso, de classes ou de objetos: «kernel» (corresponde a uma invariante), «optional» (corresponde a uma *feature* opcional), «alternative» (corresponde a uma *feature* alternativa) ou «variant» (corresponde a uma variação).

2.4 Reusable Assets Specifications (RAS)

Conforme já discutido ao longo deste documento, há uma demanda crescente para facilitar o reúso de software. No entanto, o reúso envolve altos custos em termos de criação, busca, entendimento e utilização dos ativos encontrados em um contexto específico. Assim, para tornar o reúso mais factível, faz-se necessário criar padrões para organizá-los e empacotá-los.

Nesse contexto, a *Object Management Group* (OMG) propôs uma especificação para ativos reusáveis denominada *Reusable Assets Specifications* (RAS) (GROUP, 2005). Trata-se de um grupo de padrões de gerenciamento de objetos para empacotar artefatos digitais com o objetivo de aumentar a reusabilidade. A RAS fornece um conjunto de diretrizes e recomendações sobre a estrutura, o conteúdo e a descrição dos ativos reusáveis, tentando reduzir os conflitos associados às transações de reúso por meio de um empacotamento padrão consistente. Essas especificações são dependentes das especificações de *Unified Modeling Language* (UML) e *Extensible Markup Language* (XML).

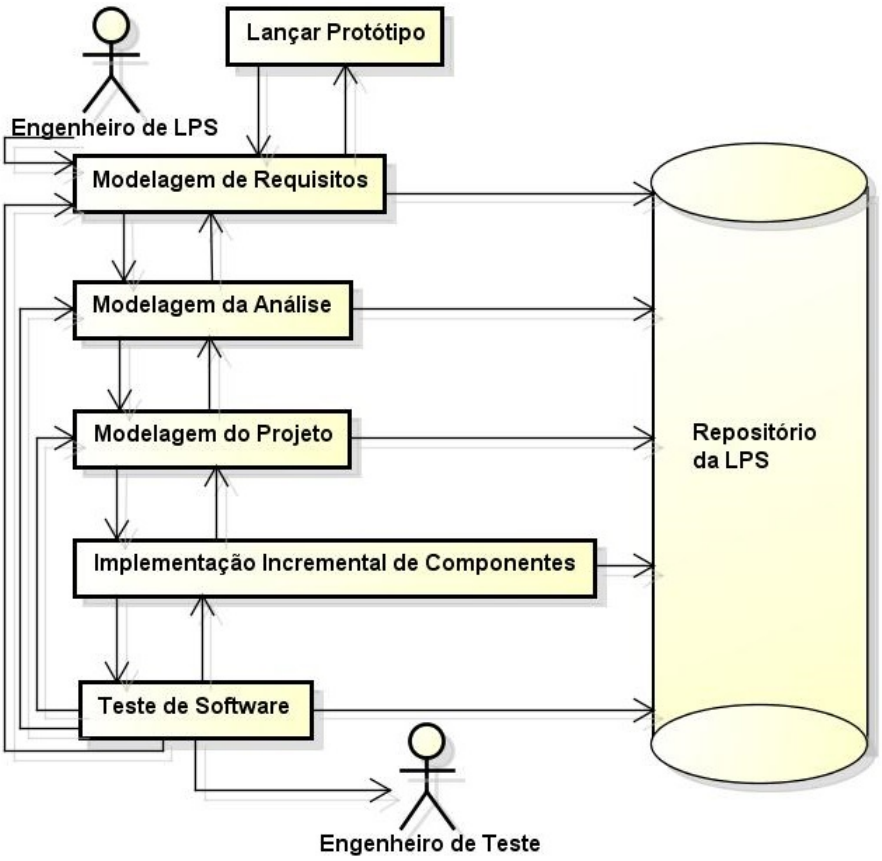


Figura 3 – Processo Evolucionário de Linha de Produto de Software - Adaptada de Gomaa (2006).

Dentro do núcleo de ativos reusáveis, o empacotamento é feito por meio de componentes. A estrutura do empacotamento, ou seções principais do *Core* da RAS, divide-se em: 1 - *Classification*: contém descritores, *tags* e valores, além do contexto (domínio, desenvolvimento, teste, implantação); 2 - *Usage*: contém instruções de uso, atividades e preenchimento dos pontos de variação; 3 - *Solution*: contém um ou mais artefatos (visão geral, requisitos, modelos, códigos, testes, documentos, entre outros); 4 - *RelatedAsset*: contém relacionamentos entre ativos; e 5 – *Profile*: define o perfil utilizado para descrever o ativo, como mostra a Figura 4 (a estrutura completa do *Core* da RAS pode ser observada na Figura 40 do Anexo A).

O pacote resultante possui uma extensão '.RAS' e um documento descritor em XML chamado de manifesto. A estrutura do manifesto é descrita e reconhecida por meio de um documento XML Schema. Para ser considerado um ativo reusável, o pacote deve possuir pelo menos um manifesto e pelo menos um artefato relacionado. Os artefatos podem ser empacotados no arquivo .RAS, ou podem ser apenas referenciados, o que diminui o acoplamento e aumenta o reúso.

A Figura 5 mostra um exemplo onde os artefatos são empacotados no arquivo .RAS, que é tratado como um arquivo compactado comum (.ZIP). O arquivo manifest.rmd é o arquivo manifesto do pacote e sua localização é considerada como diretório raiz, assim, tudo o que é referenciado no arquivo manifesto possui sua localização neste contexto, estando dentro da raiz

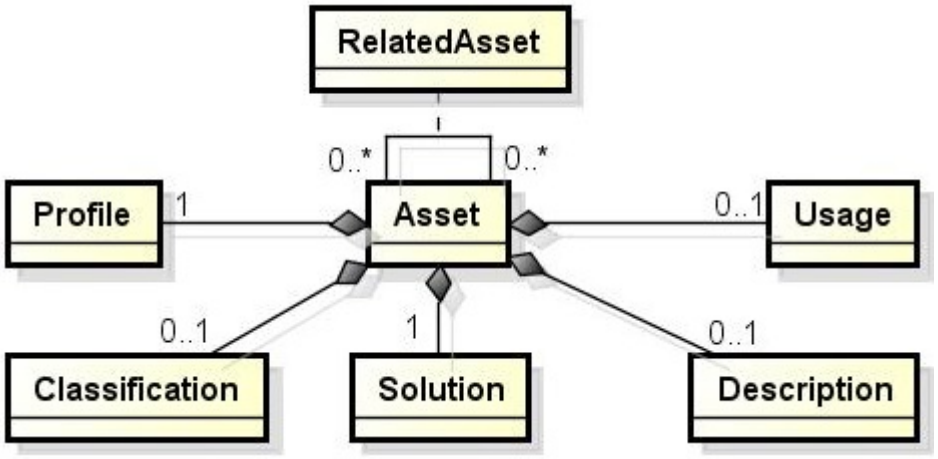


Figura 4 – Estrutura do empacotamento (*Core RAS*) - Adaptada de Group (2005).

ou em algum de seus subdiretórios.

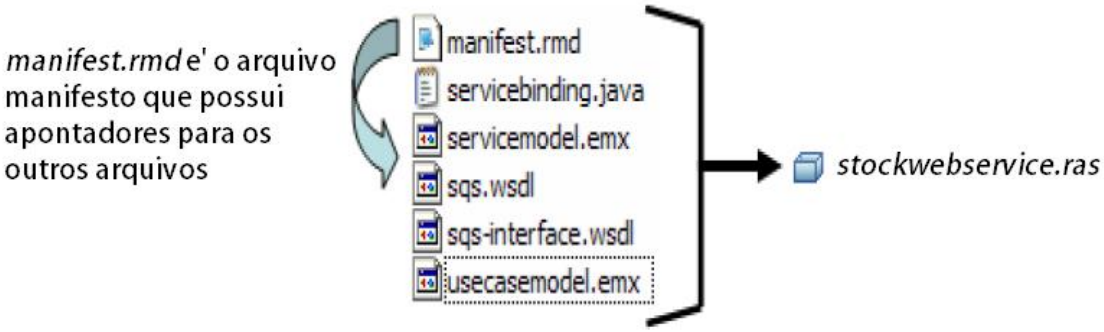


Figura 5 – Exemplo de artefatos empacotados - Adaptada de Group (2005).

Trabalhar em equipe geralmente inclui o uso de algum tipo de controle de versões. Nesse contexto, uma abordagem para definir os ativos é incluir o arquivo manifesto no controle de versão e referenciar os artefatos em suas localizações originais. A Figura 6 mostra um exemplo dessa abordagem.

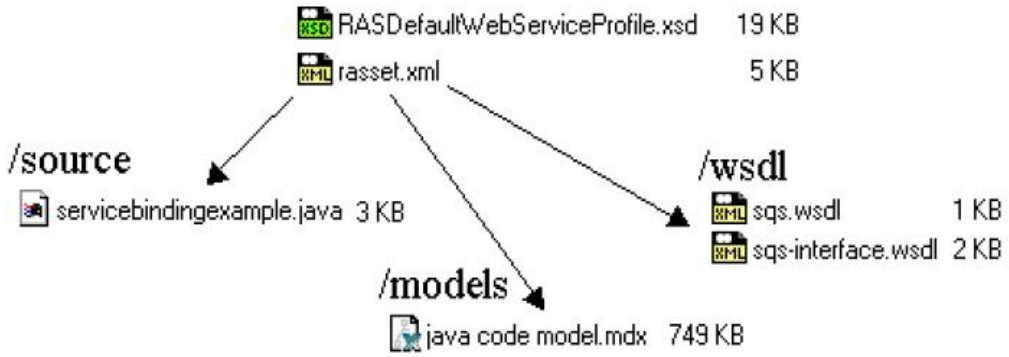


Figura 6 – Exemplo de artefatos referenciados - Adaptada de Group (2005).

Os ativos especificados pela RAS são descritos em duas categorias principais: *Core* RAS e perfis. O *Core* RAS é um meta modelo que representa os elementos fundamentais das especificações dos ativos. Os perfis, por sua vez, descrevem extensões para esses elementos fundamentais, porém, não alteram suas definições ou semântica. A Figura 7 ilustra o *Core* RAS e os perfis com seu relacionamento padrão.

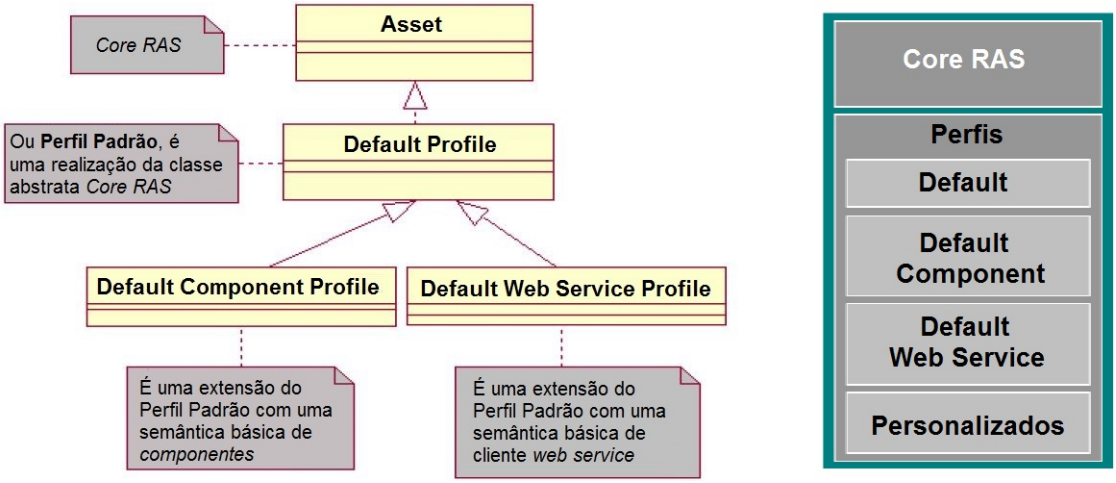


Figura 7 – Meta-modelo do *Core* RAS, perfis padrões e seus relacionamentos - Adaptada de Group (2005).

A informação de especificação do *Core* RAS pode ser adaptada para as circunstâncias especiais. Um perfil RAS é uma extensão formal de uma meta-informação estrutural. Essa é uma maneira de adicionar ou aumentar informações para a especificação da base. Um perfil RAS pode ser criado para introduzir semântica e restrições mais rígidas, mas sem alterar as constantes do *Core*.

2.5 *Web Services*

Existem diversas tecnologias que permitem a implementação de serviços, como por exemplo, Remote Procedure Calls (RPC) (BIRRELL; NELSON, 1984), Java Remote Method Invocation (RMI) (ORACLE, 2016), Common Object Request Broker Architecture (CORBA)(GROUP, 2012a), Distributed Component Object Model (DCOM) (MICROSOFT, 2007) e *Web Services*. Dentre essas tecnologias, os *Web Services* são os que mais têm se destacado nos últimos anos, uma vez que são independentes de linguagem de programação e favorecem a interoperabilidade entre sistemas heterogêneos.

Segundo Booth Hugo Haas (2004), um *Web Service* é um tipo específico de serviço identificado por um *Uniform Resource Identifier* (URI) que possui uma interface descrita em uma linguagem processável por um computador chamada *Web Services Description Language* (WSDL) (CHINNICI JEAN-JACQUES MOREAU, 2007).

WSDL é uma linguagem utilizada para compor o documento *Web Service Description* (WSD), ou documento de descrição de *Web Service*, que formaliza a descrição do serviço.

Ela fornece elementos que permitem especificar as interfaces do *Web Service* de uma maneira computável, utilizando uma formatação XML (CURBERA *et al.*, 2002). A WSDL compreende todos os detalhes técnicos sobre o serviço, incluindo o endereço de rede onde pode ser invocado, as operações disponíveis, os tipos de dados a serem trocados e seu formato de serialização, a estrutura das mensagens a serem transmitidas e recebidas e os protocolos de troca de mensagens entre aplicações clientes e provedores (BOOTH HUGO HAAS, 2004). Em outras palavras, o documento WSDL fornece todas as informações que uma aplicação cliente necessita para interagir com o *Web Service*.

Web Services podem ser definidos como sistemas de software projetados para suportar interações de forma interoperável entre máquinas em uma rede (BOOTH HUGO HAAS, 2004). Essas interações são definidas pela troca de mensagens tipicamente transportadas via requisições *Hypertext Transfer Protocol* (HTTP) e serializadas utilizando XML. Para a transmissão dessas mensagens, existem duas estruturas de empacotamento que são mais utilizadas: *Simple Object Access Protocol* (SOAP) e *Representational State Transfer* (REST).

O protocolo SOAP (MITRA, 2007) é um conjunto de convenções definidas pela *World Wide Web Consortium* (W3C) para a troca de mensagens transmitidas através do protocolo HTTP. A estrutura de uma mensagem SOAP é definida por um envelope (*Envelope*) que contém dois elementos filhos chamados: cabeçalho (*Header*) e corpo (*Body*). O envelope serve como um pacote para transmissão das mensagens. O cabeçalho é um elemento opcional que pode conter informações de configuração, endereçamento e *Quality of Service* (QoS), enquanto o corpo é um elemento obrigatório que contém a própria mensagem (PAUTASSO; ZIMMERMANN; LEYMANN, 2008).

A troca de mensagens através do protocolo SOAP exige que ambas as aplicações, servidora e consumidora, possam interpretar essas mensagens. Assim, mecanismos de interpretação deste protocolo sempre devem ser desenvolvidos em ambas as máquinas.

Os *Web Services* que utilizam a arquitetura REST são chamados RESTful. Essa arquitetura apoia sistemas hipermídia distribuídos e enfatiza a generalização das interfaces, a escalabilidade da interação entre os componentes e a instalação independente dos mesmos (FIELDING; TAYLOR, 2002).

Embora as duas tecnologias, SOAP e REST, sejam amplamente difundidas e utilizadas na implementação de *web services*, a abordagem apresentada neste trabalho selecionou o SOAP para a implementação dos serviços.

2.6 Considerações Finais

A fundamentação teórica apresentada neste capítulo permite uma melhor compreensão deste trabalho. Os conceitos de reúso e de LPS são fundamentais uma vez que a abordagem foi

criada para apoiar o gerenciamento das fases de desenvolvimento de uma LPS. As abordagens de desenvolvimento apresentadas, especialmente a PLUS, serviram como base para o desenvolvimento desta abordagem. A especificação RAS apresentada foi usada para alavancar o potencial do reúso dos ativos e dos elementos de processo, pois facilitam seu armazenamento, busca e compartilhamento, além de torná-los mais compatíveis com outros sistemas e até mesmo outras ferramentas. Os conceitos de *Web Services* foram essenciais para a construção e a exposição dos serviços oferecidos por esta abordagem para a comunidade científica.

O próximo Capítulo apresenta a Revisão Bibliográfica que contém um mapeamento sistemático realizado para investigar quais ferramentas oferecem suporte ao gerenciamento das fases do desenvolvimento de LPS, além de outros trabalhos relacionados.

CAPÍTULO

3

REVISÃO BIBLIOGRÁFICA

Desenvolvedores de Linhas de Produtos de Software (LPS) possuem um apoio computacional restrito e geralmente muito específico para uma área do desenvolvimento. Como já discutido anteriormente, esta é uma das razões que dificultam a ampla adoção de LPS no mercado, pois além do desafio de se realizar uma boa engenharia de domínio embasado em um conhecimento teórico e prático do assunto, a falta de apoio computacional, a pouca documentação disponível e a complexidade e/ou a indisponibilidade das ferramentas existentes acabam desmotivando a adoção e o uso da LPS.

Nesse contexto, é proposto um mapeamento sistemático sobre ferramentas de apoio ao gerenciamento de linhas de produtos, com o objetivo de identificar e reunir soluções existentes na literatura, evidenciando sua completude e complexidade, para auxiliar desenvolvedores e destacar áreas de pesquisa e trabalhos futuros. Este mapeamento foi publicado parcialmente em uma conferência (PACINI; BRAGA, 2015b), no entanto, a seção seguinte apresenta protocolo e a análise completa dos resultados.

3.1 Mapeamento Sistemático

O mapeamento sistemático assemelha-se à revisão sistemática, porém, ao invés de focar na profunda análise e comparação de estudos primários, preocupa-se mais em levantar o estado da arte em um assunto específico. O mapeamento sistemático então realiza uma ampla revisão dos estudos primários, identificando evidências disponíveis, observando lacunas no conjunto dos estudos primários para posterior foco de revisões sistemáticas e apontando áreas onde são necessários mais trabalhos e pesquisas (KITCHENHAM, 2004). Assim, o mapeamento sistemático fornece uma visão geral de uma área de pesquisa específica, podendo ser observados a quantidade de estudos selecionados, o tipo, os resultados disponíveis, a frequências de publicações ao longo do tempo, entre outros (PETERSEN *et al.*, 2008).

A condução deste mapeamento sistemático é baseada no processo proposto por Kitchenham (2004) que contém três fases principais: 1 – Planejamento: nesta fase são definidos os objetivos da pesquisa e o protocolo do mapeamento sistemático é elaborado; 2 – Condução: durante esta fase, estudos primários são identificados, selecionados e avaliados de acordo com os critérios de inclusão e exclusão previamente estabelecidos; 3 – Extração e análise dos dados: nesta fase, o relatório final é organizado e apresentado. As próximas seções apresentam essas fases em detalhes.

3.1.1 Planejamento

Nesta fase é estabelecido o protocolo do mapeamento. Para isso são especificados: objetivo do mapeamento, questões de pesquisa, abrangência e especificidade, critério de seleção das fontes, definição dos estudos e critérios de seleção.

3.1.1.1 Objetivos da pesquisa

Este mapeamento sistemático tem por objetivo investigar o estado da arte com respeito ao apoio computacional ao gerenciamento de linhas de produtos de software, com respeito a fases do desenvolvimento, manutenção e evolução da LPS, a fim de identificar a quantidade e a qualidade das ferramentas disponíveis atualmente, considerando sua completude e complexidade evidenciando seus benefícios e limitações.

3.1.1.2 Questões de pesquisa

A partir do objetivo descrito, pretende-se encontrar, possivelmente, todos os estudos primários para entender e reunir evidências sobre as ferramentas existentes para apoiar o gerenciamento de linhas de produtos de software. Para isso, a seguinte questão de pesquisa (RQ) foi estabelecida:

> **RQ1:** Quais são as soluções existentes na literatura que apresentam um apoio computacional ao gerenciamento de linhas de produtos de software?
>
> a) A solução apresentada utiliza algum padrão?
>
> b) Qual é a tecnologia utilizada no gerenciamento das linhas de produtos de software que a solução apresenta?
>
> c) Quais fases da linha de produtos de software são cobertas pela solução?

Uma questão de pesquisa bem formulada é geralmente composta e analisada de acordo com diferentes pontos de vista que definem variedade e especificidade. Assim, foi definido o PICO (População, Intervenção, Comparação e Resultados) para este mapeamento sistemático. População identifica o grupo de população a ser observado na intervenção; Intervenção se refere

ao que será observado no contexto do mapeamento sistemático; Comparação define o que será comparado no contexto da avaliação sistemática e, finalmente, as conclusões são expressas nos Resultados. O PICO deste mapeamento sistemático é apresentado como segue:

- **População:** Estudos primários que apresentem ferramentas, métodos, técnicas, abordagens, *frameworks* e/ou metodologias que apoiem o gerenciamento das fases de desenvolvimento de uma linha de produtos de software.
- **Intervenção:** Quais são as fases do processo de desenvolvimento cobertas pela solução proposta nos estudos primários, quais os benefícios e limitações dessa solução, o tipo da solução, se existem padrões empregados e em quais recursos da engenharia de software a solução é embasada.
- **Comparação:** Neste caso não se aplica, pois este mapeamento tem o objetivo de somente reunir informações de soluções existentes e extrair suas características sem realizar um profundo estudo comparativo.
- **Resultados:** Uma visão geral das soluções propostas na literatura que oferecem um apoio computacional ao gerenciamento do desenvolvimento e manutenção de linhas de produtos de software, com suas características, benefícios, limitações e especificidades.

3.1.1.3 Critérios de Inclusão e Exclusão

Para refinar os resultados da busca dos estudos primários no contexto deste mapeamento sistemático, foram definidos dois critérios de inclusão:

- **IC1:** Estudos que apresentam uma ferramenta, abordagem, técnica, processo, método ou qualquer outra solução prática existente com recursos da engenharia de software para gerenciar uma ou mais fases do ciclo de desenvolvimento e manutenção de linhas de produtos de software.
- **IC2:** Estudos que apresentam uma proposta de solução, mesmo que ainda em projeto, para gerenciar uma ou mais fases do ciclo de desenvolvimento e manutenção de linhas de produtos de software.

Diante dos resultados retornados pelas buscas e dos critérios de inclusão estabelecidos, para descartar estudos irrelevantes ao contexto deste mapeamento sistemático foram definidos quatro critérios de exclusão:

- **EC1:** Estudos que não apresentam uma solução prática, consolidada ou não, para o gerenciamento de uma ou mais fases do ciclo de desenvolvimento e manutenção de linhas de produtos de software.

- **EC2:** Estudos que são versões resumidas de trabalhos mais completos.
- **EC3:** Estudos incompletos, indisponíveis e/ou duplicados.
- **EC4:** Estudos que descrevem eventos, são índices ou programação.

3.1.1.4 Seleção das Fontes de Busca

As fontes foram escolhidas considerando a sua relevância na área de engenharia de software e um parecer de especialista no caso de conferências, livros e *workshops* não indexados nas máquinas de busca. Para as máquinas de busca, foram consideradas aquelas que possuem um conteúdo atualizado, a disponibilidade dos estudos, o mecanismo de busca, a qualidade dos resultados e versatilidade para exportar. Assim foram selecionadas as seguintes fontes: *IEEE Xplore*, *ACM Digital Library*, *Science Direct*, *Scopus* e *Web of Science*.

3.1.1.5 Construção da String de Busca

A partir de um grupo de estudos selecionados pelo especialista, denominado na revisão ou mapeamento sistemático como grupo de controle, juntamente com o objetivo deste mapeamento sistemático, a *string* de busca foi definida conforme o Quadro 1.

Quadro 1 – Termos, palavras-chave e sinônimos

Termo	Palavra-chave	Sinônimos
A	Product Line	"SPL", "Product Lines", "Product Family", "Product Families", "Product-Line", "Product-Lines", "Product-Family", "Product-Families"
B	Tool	"tools", "tool-supported", "support", "supported", "supporting"
C	Management	"manage", "managing", "storage", "repository"
D	Software	-

O Quadro 1 apresenta os termos da *string* de busca com base em quatro palavras-chave e seus sinônimos. O termo A garante uma busca por resultados relacionados a linhas de produtos enquanto o termo D restringe que essas linhas de produtos sejam de software. O termo C refere-se ao gerenciamento que deve ser abordado nos estudos resultantes enquanto o termo B garante que a solução deve ser uma ferramenta ou oferecer algum tipo de apoio ao gerenciamento de linhas de produtos. Desta forma, **a *String* resultante é: (A) AND (B) AND (C) AND (D)**.

3.1.2 Condução do Mapeamento Sistemático

A condução do mapeamento sistemático logo após a definição do protocolo foi dividida em duas fases de seleção. Na primeira fase, a pesquisa foi executada nas máquinas de busca de acordo com a *string* definida considerando publicações entre 2000 e 2013. A busca foi realizada utilizando a sintaxe específica de cada base de dados e considerando apenas título, palavras-chave

e *abstract*. Após concluídas as buscas foram retornados 1.046 estudos que foram sumarizados por base de dados na Tabela 4.

Tabela 1 – Sumarização dos resultados por base de pesquisa

Base de Dados	**Resultado**
ACM Digital Library	**63**
IEEE Xplore	**261**
Science Direct	**35**
Scopus	**492**
Web of Science	**195**
Total de estudos	**1.046**

Essa primeira fase descrita é chamada de identificação dos estudos. A identificação dos estudos define o grupo de estudos que servirá de base para a segunda fase da condução do mapeamento sistemático, chamada de seleção dos estudos. Na segunda fase, logo após definido o grupo de estudos identificados na primeira fase, são aplicados sobre o grupo os critérios de inclusão e exclusão definidos no protocolo do mapeamento sistemático. Ao final da segunda fase, onde acontece o processo de seleção, obtém-se um grupo refinado de acordo com o contexto do mapeamento sistemático para extração dos dados.

A fase de seleção deste mapeamento sistemático foi dividida em três partes. Na primeira seleção foram aplicados os critérios de inclusão e exclusão sobre o grupo de estudos identificados considerando apenas título e *abstract*, onde foram incluídos 95 estudos, excluídos 951, dos quais 585 foram excluídos pelos critérios de exclusão e 366 foram marcados como duplicados. Na segunda seleção foi considerado o texto completo dos estudos, assim, 41 estudos foram incluídos e 54 foram excluídos. A terceira seleção contou com a ajuda de um especialista para validar os estudos selecionados, assim, 9 estudos que haviam sido excluídos foram incluídos novamente por consenso. A sumarização desta seleção é apresentada na Tabela 2.

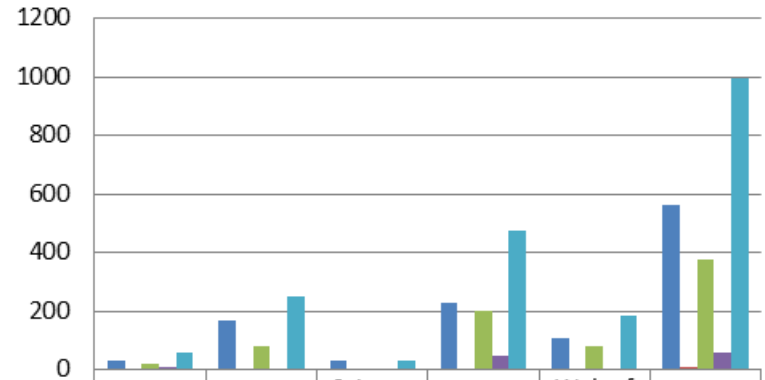
Tabela 2 – Sumarização dos resultados por etapas de seleção

Seleção	**Descrição**	**Estudos Incluídos**	**Estudos Excluídos**
1	Seleção por título e abstract	95	951
2	Seleção por leitura completa	41	54
3	Seleção validada por especialista	50	45
Total Geral		50	996

A maior parte dos trabalhos (561) foi excluída de acordo com o critério de exclusão **EC1**, ou seja, não oferecem soluções práticas para o gerenciamento das fases de desenvolvimento e manutenção de linhas de produtos de software. Em segundo lugar, o critério de exclusão **EC3**, que se refere a estudos indisponíveis, incompletos ou duplicados, foi responsável pela exclusão de 374 estudos. Já os critérios **EC2** (versões resumidas) e **EC4** (índices, etc.) foram responsáveis

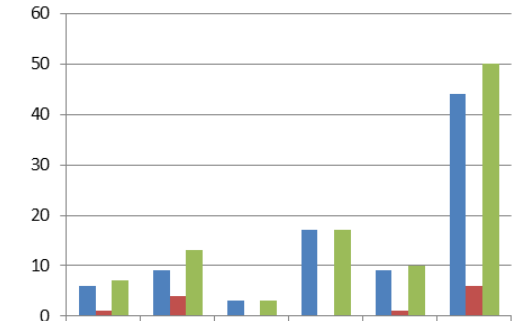
pela exclusão de somente 6 e 55 estudos respectivamente. A sumarização dos estudos excluídos separados por base de dados pode ser observada na Figura 8.

Figura 8 – Sumarização dos estudos excluídos por base de dados e critério de exclusão



	ACM	IEEE	Science Direct	Scopus	Web of Science	Total
EC1	31	168	29	225	108	561
EC2	0	2	0	4	0	6
EC3	19	77	3	198	77	374
EC4	7	0	0	48	0	55
Total	57	247	32	475	185	996

Os estudos incluídos, em sua grande maioria (44), apresentam soluções reais já existentes, o que corresponde ao critério de inclusão **IC1**, enquanto uma minoria (6) apresenta apenas a ideia de uma solução a ser implementada ainda, o que corresponde ao critério de inclusão **IC2**. A sumarização dos estudos incluídos separados por base de dados pode ser observada na Figura 9.



	ACM	IEEE	Science Direct	Scopus	Web of Science	Total
IC1	6	9	3	17	9	44
IC2	1	4	0	0	1	6
Total	7	13	3	17	10	50

Figura 9 – Sumarização dos estudos incluídos por base de dados e critério de inclusão

Após a seleção dos estudos a serem incluídos no mapeamento, já é possível prosseguir para a próxima fase do processo de mapeamento sistemático, onde se realiza a extração dos dados.

3.1.3 Extração dos Dados

A fase de extração dos dados tem por objetivo sumarizar os dados dos estudos selecionados para uma posterior análise. Para a extração dos dados deste mapeamento, foi utilizado o formulário apresentado na Figura 10.

Formulário de Extração

Título: _____
Autor(es): _____
Ano: _____
Publicação: _____
Tipo de solução (Abordagem, ferramenta, framework, ...): _____
Descrição resumida da solução: _____
Fases cobertas pela solução
( ) Engenharia de Domínio
( ) Engenharia de Aplicação
( ) Ambas
A solução é específica?
( ) Sim
( ) Não
Qual especificidade? _____
Recursos da Engenharia de Software utilizados na solução: _____
Padrões utilizados: _____
Quais os benefícios da solução? _____
Quais as limitações? _____
A solução possui validação?
( ) Sim
( ) Não
Apoio a qual parte do gerenciamento? _____

Figura 10 – Formulário para extração de dados

Este formulário tem o objetivo de ajudar a responder as questões de pesquisa deste mapeamento sistemático. Além de recolher as informações básicas sobre o estudo, como o título, autores, ano e o tipo de publicação (periódico, congresso, etc.), o formulário também recolhe informações específicas desta pesquisa. Dentre as informações específicas estão o tipo de solução apresentada pelo estudo, a descrição resumida da solução, a especificidade, as fases cobertas, os recursos de engenharia de software que a solução utiliza, a utilização de padrões, benefícios e limitações, validação e partes gerenciadas.

O tipo da solução refere-se ao que os autores da abordagem declararam, podendo possuir mais de um tipo, como por exemplo ser uma abordagem, ou ferramenta, ou ambos, ou qualquer outro recurso da engenharia de software que apresente uma solução prática para o gerenciamento das fases do desenvolvimento e manutenção de linhas de produtos de software. A descrição

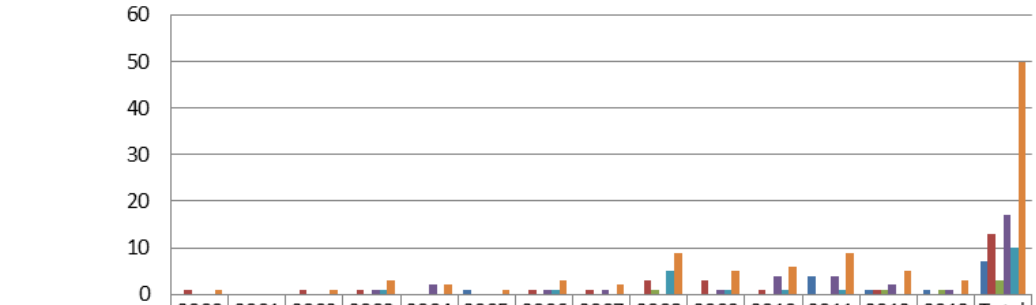
da solução, benefícios e limitações, são resumos dessas informações que foram extraídas do texto. A especificidade refere-se à granularidade da solução quanto ao gerenciamento, como por exemplo, o gerenciamento somente de variabilidades. As fases cobertas explicitam se a solução apoia a engenharia de domínio, a engenharia de aplicação ou ambas. Os recursos de engenharia de software utilizados na solução referem-se aos recursos utilizados predominantes, como uma linguagem específica de domínio, ou o uso de *Unified Modeling Language* (UML) (GROUP, 2011), *eXtensible Markup Language*, etc. A utilização de padrões explicita se a solução informa a utilização de padrões, como o *Reusable Asset Specification* (RAS) (GROUP, 2005), *Common Variability Language* (CVL) (GROUP, 2012b), etc. A questão de validação tem o objetivo de verificar a confiabilidade da ferramenta oferecida, se ela já passou por testes, experimentos e/ou estudos de caso. A parte gerenciada refere-se ao tipo do gerenciamento fornecido, por exemplo, se apoia o gerenciamento do desenvolvimento, ou manutenção de uma linha de produtos de software.

3.1.4 Análise Geral dos Resultados

Após uma extração cuidadosa dos dados, são reunidas informações suficientes para a realização de uma análise de resultados. A análise dos estudos selecionados foi dividida de acordo com as características descritas no formulário de extração que foram observadas.

3.1.4.1 Frequência de Publicação

Foi observado que a frequência de publicações de soluções para o gerenciamento de linhas de produtos de software subiu significativamente a partir de 2008 e teve seu ápice em 2011, depois manteve-se estabilizada. Na Figura 11 esta tendência pode ser observada graficamente, indicando que esta é uma área de interesse atual de pesquisa.



	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
ACM	0	0	0	0	0	1	0	0	0	0	0	4	1	1	7
IEEE	1	0	1	1	0	0	1	1	3	3	1	0	1	0	13
ScienceDirect	0	0	0	0	0	0	0	0	1	0	0	0	1	1	3
Scopus	0	0	0	1	2	0	1	1	0	1	4	4	2	1	17
WebOfScience	0	0	0	1	0	0	1	0	5	1	1	1	0	0	10
Total	1	0	1	3	2	1	3	2	9	5	6	9	5	3	50

Figura 11 – Gráfico de estudos divididos em Ano de Publicação x Base

3.1.4.2 Tipo de Solução

O tipo da solução extraído dos estudos, considera somente como os autores classificam a própria solução em sua publicação. Na Figura 12 pode ser observado que a maior parte das soluções se considera do tipo Ferramenta e/ou Abordagem. Este já era um resultado esperado, uma vez que grande parte das soluções encontradas são práticas e não teóricas, muitas vezes oferecendo inclusive uma interface gráfica.

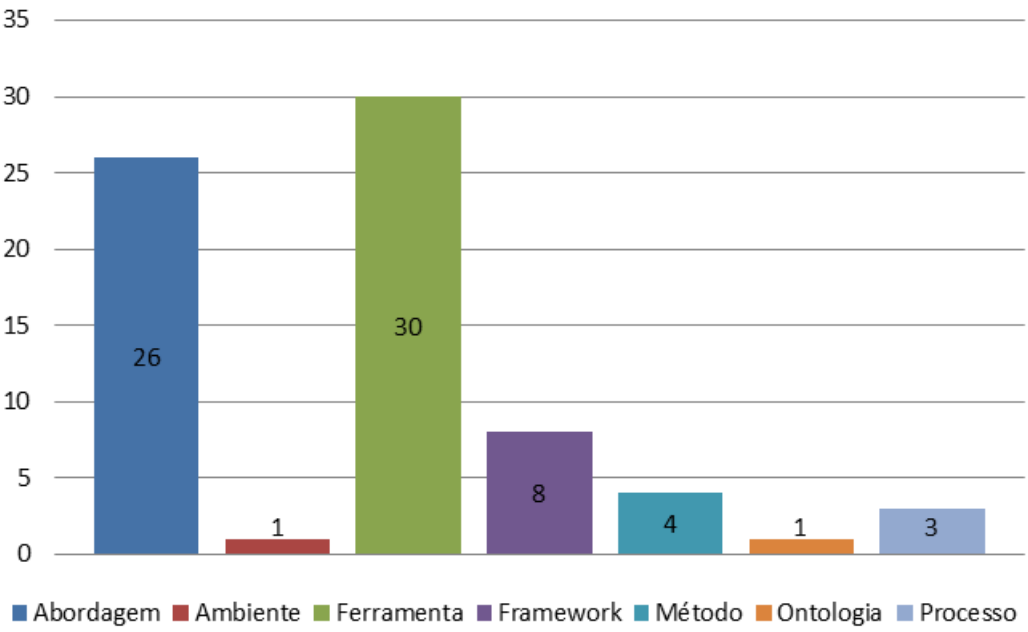


Figura 12 – Gráfico de estudos divididos em Tipo de Solução

3.1.4.3 Especificidade da Solução

Grande parte das soluções encontradas são específicas para alguma fase do desenvolvimento da linha de produtos, como pode ser observado na Figura 13. Dentre as especificidades, destaca-se o gerenciamento de variabilidades. De fato, a maioria das soluções disponíveis tanto comercialmente quanto de forma livre são voltadas a apoiar o gerenciamento de variabilidades, oferecendo processos, modelos, métodos, abordagens, ferramentas, entre outros. Como exemplo disso pode-se citar o KobrA (MORENO-RIVERA; NAVARRO; CUESTA, 2011) e o COVAMOF (SINNEMA; DEELSTRA; HOEKSTRA, 2006).

3.1.4.4 Utilização de Padrões

Nas soluções observadas foi evidenciada a falta do uso de padrões em seu desenvolvimento, como pode ser observado na Figura 14. A falta da utilização de padrões pode dificultar a flexibilidade da solução, além de muitas vezes dificultar sua adoção e sua integração com outras soluções existentes. Apesar de gerar um esforço adicional, a utilização de padrões ao se desenvolver uma solução é fortemente recomendada por seu custo-benefício.

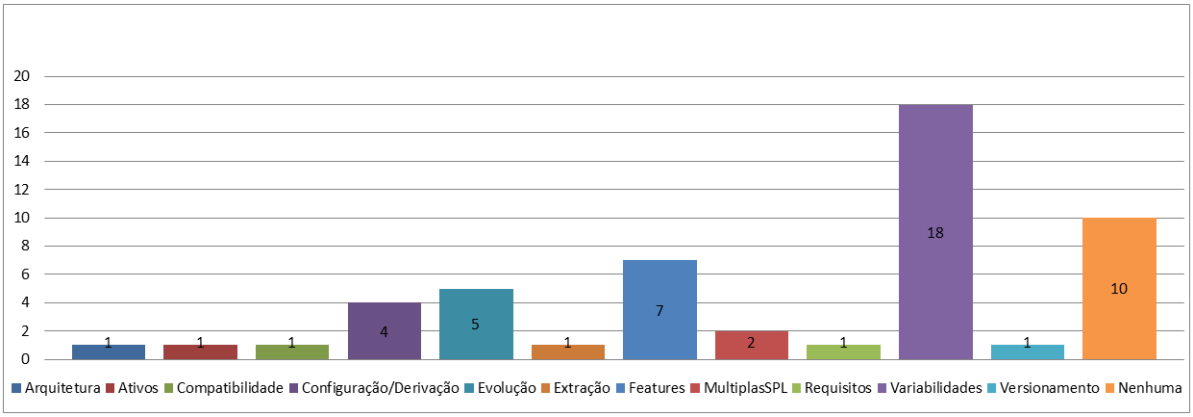


Figura 13 – Gráfico de estudos divididos em Especificidade da Solução

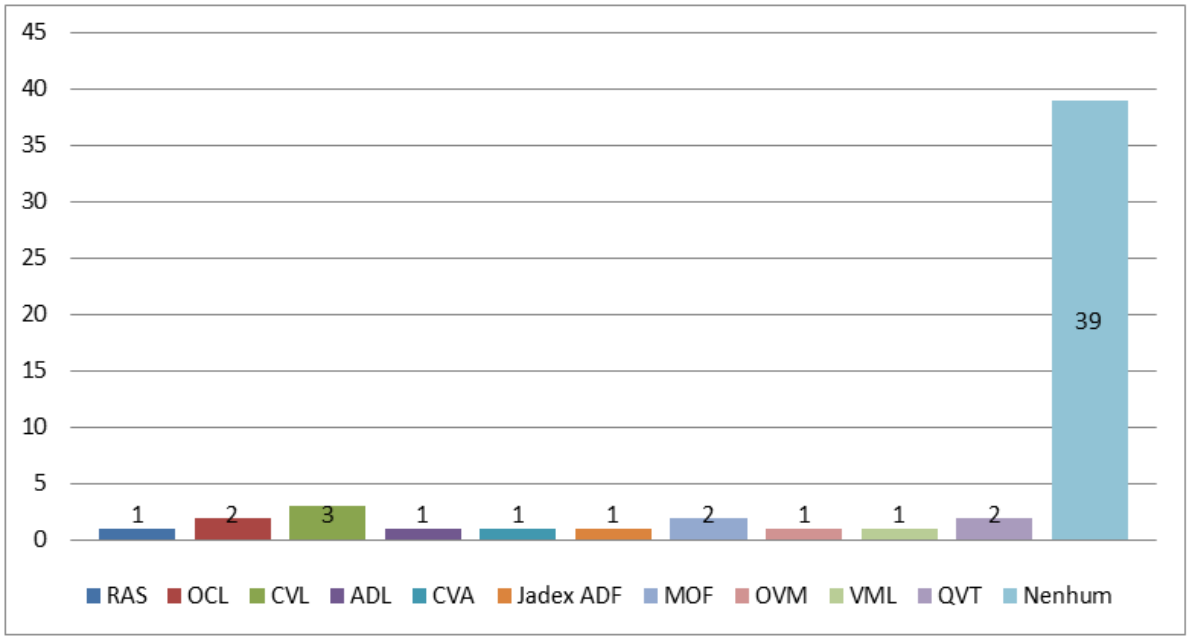


Figura 14 – Gráfico de estudos divididos em utilização de Padrões.

3.1.4.5 Cobertura das Fases

Dos 50 trabalhos selecionados, somente 17 oferecem suporte a ambas as fases de desenvolvimento (engenharia de domínio e engenharia de aplicação), do restante, cinco apoiam somente a engenharia de aplicação e 28 somente a engenharia de domínio, isso fica evidenciado na Figura 16.

As informações extraídas desses 17 estudos mais relevantes que apoiam ambas as fases de desenvolvimento são apresentadas e detalhadas na Tabela 3 e são avaliados mais especificamente na segunda fase da análise.

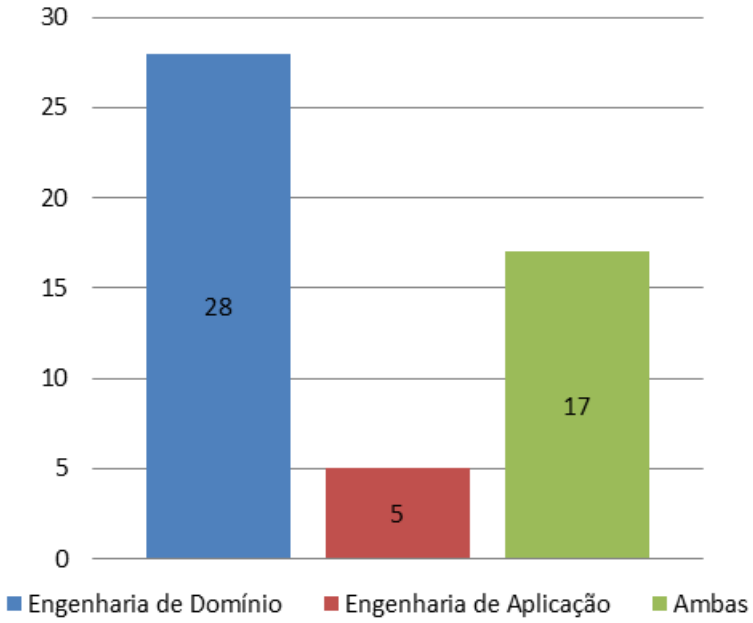


Figura 15 – Gráfico de Estudos x Cobertura das Fases

3.1.5 Análise dos Estudos Selecionados

Dessas 17 soluções apresentadas pelos estudos, 14 oferecem apoio ao desenvolvimento da LPS enquanto duas oferecem suporte apenas à manutenção e evolução de linhas de produtos já existentes. Das 14 soluções que apoiam o desenvolvimento, uma apoia somente este processo; cinco apoiam o desenvolvimento com manutenção sendo que dentre eles uma apoia também a derivação de LPS e uma apoia também a evolução; oito apoiam tanto o desenvolvimento, quanto manutenção quanto oferece uma visualização da LPS com suas variabilidades; e somente uma oferece apoio ao desenvolvimento, manutenção, evolução e oferece uma visualização. Essas informações ficam mais claras no gráfico da Figura 16.

O gráfico da Figura 16 mostra que apenas duas soluções oferecem suporte desde o desenvolvimento até a manutenção e evolução das linhas de produtos, sendo que somente uma delas oferece também uma visão geral das variabilidades e ativos produzidos.

Desta forma, fica evidente a carência de soluções que apoiam todas as fases de desenvolvimento e evolução de LPS, o que caracteriza uma lacuna a ser investigada.

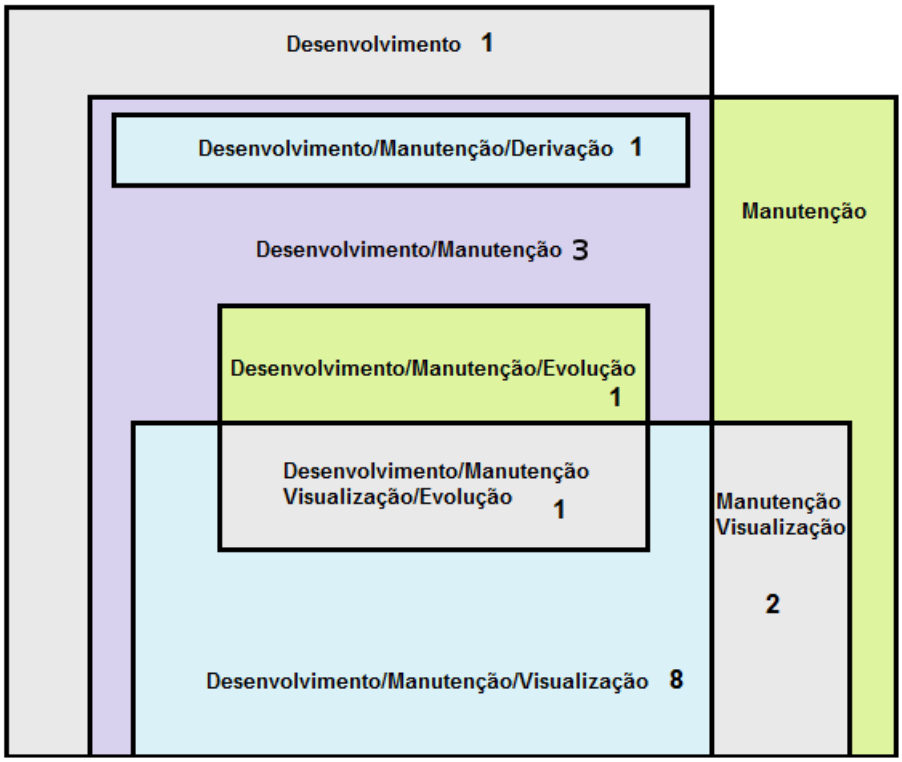


Figura 16 – Gráfico de porcentagem de Estudos x Cobertura das Fases

Tabela 3 – Análise de dados dos estudos mais relevantes

Título	**Ano**	**Base de Busca**	**Tipo da Solução**	**Gerência de Variabilidades**	**Outras Especificidades**	**Utiliza Padrões**	**Possui Validação**	**Apoio ao Desenvolvimento**	**Apoio à Manutenção**	**Apoio à Evolução**
An approach to variability management in service-oriented product lines (KHOSHNEVIS, 2012)	2012	Scopus	Ferramenta	•				•	•	
Automated software product line engineering and product derivation (GOMAA; SHIN, 2007)	2007	IEEE	Ferramenta				•	•	•	

Continua na próxima página

Tabela 3 – *Continuação*

Título	Ano	Base de Busca	Tipo da Solução	Gerência de Variabilidades	Outras Especificidades	Utiliza Padrões	Possui Validação	Apoio ao Desenvolvimento	Apoio à Manutenção	Apoio à Evolução
Automating software product line development: A repository-based approach (FILHO *et al.*, 2010)	2010	IEEE	Ferramenta			•	•	•	•	
Automating the product derivation process of multi-agent systems product lines (CIRILO *et al.*, 2012)	2012	ACM	Abordagem	•	•	•	•	•	•	
Evolving KobrA to support SPL for WebGIS development (MORENO-RIVERA; NAVARRO; CUESTA, 2011)	2011	IEEE	Ferramenta	•		•	•	•	•	
Feature model to product architectures: Applying MDE to Software Product Lines (PEROVICH; ROSSEL; BASTARRICA, 2009)	2009	Scopus	Processo				•	•	•	•
FLiP: Managing Software Product Line Extraction and Reaction with Aspects (ALVES *et al.*, 2008)	2008	Scopus	Ferramenta		•				•	
Holmes: a system to support software product lines (SUCCI *et al.*, 2000)	2000	Scopus	Ferramenta					•	•	

Continua na próxima página

Tabela 3 – *Continuação*

Título	Ano	Base de Busca	Tipo da Solução	Gerência de Variabilidades	Outras Especificidades	Utiliza Padrões	Possui Validação	Apoio ao Desenvolvimento	Apoio à Manutenção	Apoio à Evolução
Involving Non-Technicians in Product Derivation and Requirements Engineering: A Tool Suite for Product Line Engineering (RABISER *et al.*, 2007)	2007	Web Of Science	Ferramenta	•	•			•	•	
Modeling and Building Software Product Lines with Pure::Variants (BEUCHE, 2008)	2008	IEEE	Ferramenta				•	•	•	
New methods in software product line development (KRUEGER, 2006)	2006	Web Of Science	Método					•	•	
Support for complex product line populations (ELSHARKAWY; KRöHER; SCHMID, 2011a)	2011	Scopus	Ferramenta		•			•	•	
Supporting heterogeneous compositional multi software product lines (ELSHARKAWY; KRöHER; SCHMID, 2011b)	2011	Scopus	Ferramenta				•	•	•	
The BigLever Software Gears Unified Software Product Line Engineering Framework (KRUEGER, 2008)	2008	Web Of Science	Framework				•	•	•	
The COVAMOF derivation process (SINNEMA; DEELSTRA; HOEKSTRA, 2006)	2006	IEEE	Framework	•	•		•		•	

Continua na próxima página

Tabela 3 – *Continuação*

Título	Ano	Base de Busca	Tipo da Solução	Gerência de Variabilidades	Outras Especificidades	Utiliza Padrões	Possui Validação	Apoio ao Desenvolvimento	Apoio à Manutenção	Apoio à Evolução
Toward an Architecture-Based Method for Selecting Composer Components to Make Software Product Line (TANHAEI; MOAVEN; HABIBI, 2010)	2010	Web Of Science	Framework		•		•	•		
UML support for designing software product lines: The package merge mechanism (LAGUNA; MARQUéS, 2010)	2010	IEEE	Ferramenta			•	•	•	•	•

3.1.6 Discussão

O resultado deste mapeamento sistemático possibilitou identificar as soluções existentes na literatura para apoiar computacionalmente o gerenciamento de linhas de produtos de software. Com os resultados obtidos, foi possível identificar os benefícios e as limitações que mais se destacam nas soluções analisadas como um todo.

Dentre os benefícios observados nas soluções encontradas, destaca-se que grande parte das soluções objetiva oferecer guias para resolução de problemas ou parte de problemas, além de fluxos de execução para se atingir uma meta. Esses fluxos são apresentados em forma de processo ou até mesmo *wizards*. Isso possibilita que usuários tanto iniciantes quanto experientes atuem em determinada área do gerenciamento, como o desenvolvimento ou evolução da LPS, com a segurança de que o processo foi seguido corretamente e seu resultado é consistente, uma vez que o gerenciamento oferecido é explícito e organizado. Além disso, parte das soluções ((ACHER *et al.*, 2011), (SINNEMA; DEELSTRA; HOEKSTRA, 2006), (RABISER *et al.*, 2007), (DHUNGANA *et al.*, 2010), (DEHLINGER; LUTZ, 2008), (BOTTERWECK *et*

al., 2008), (SELLIER; MANNION; MANSELL, 2008)) ainda fornece ao usuário um auxílio na tomada de decisões e no entendimento do processo. Como auxílio na tomada de decisões, algumas soluções ((GOMAA; SHIN, 2007), (MUTHIG; SCHROETER, 2013), (SINNEMA; DEELSTRA; HOEKSTRA, 2006), (AJILA; KABA, 2008), (MORENO-RIVERA; NAVARRO; CUESTA, 2011), (JUNIOR; GIMENES; MALDONADO, 2010)) ainda oferecem ao usuário visões gerais e específicas do estado da LPS, evidenciando os objetos escolhidos para serem observados. A manutenção da consistência também é uma preocupação que grande parte das soluções encontradas ((CAVALCANTI *et al.*, 2011), (ROMERO *et al.*, 2013), (GOMAA; SHIN, 2007), (O'BRIEN *et al.*, 2002), (MENDONCA; BRANCO; COWAN, 2009), (SCHUBANZ *et al.*, 2012), (KHOSHNEVIS, 2012), (GÓMEZ; RAMOS, 2011), (CIRILO *et al.*, 2012), (SINNEMA; DEELSTRA; HOEKSTRA, 2006), (ACHER *et al.*, 2010), (PEROVICH; ROSSEL; BASTARRICA, 2009), (THAO, 2012)) aborda, algumas estabelecendo restrições, verificações e validações e outras ainda disponibilizando a rastreabilidade dos ativos, funcionalidade que auxilia também na evolução da LPS.

Além desses benefícios citados, também vale destacar a grande redução de custos e de esforço, pois muitas soluções ((KHOSHNEVIS, 2012), (GARG *et al.*, 2003), (ZHANG; JARZABEK, 2003), (GOMAA; SHIN, 2007), (GÓMEZ; RAMOS, 2011), (PEROVICH; ROSSEL; BASTARRICA, 2009), (BACHMANN; NORTHROP, 2009), (KRUEGER, 2006), (O'BRIEN *et al.*, 2002)) oferecem várias funcionalidades automatizadas para apoiar o gerenciamento. Algumas soluções ((GOMAA; SHIN, 2007), (ACHER *et al.*, 2011), (ACHER *et al.*, 2013), (ZHENG *et al.*, 2009), (KRUEGER, 2006)), oferecem ainda recursos que reduzem significativamente a complexidade de tarefas que seriam trabalhosas para se realizar manualmente.

A preocupação com a promoção do reúso também é abordada por algumas soluções ((CIRILO *et al.*, 2012), (ACHER *et al.*, 2011), (ROUILLÉ *et al.*, 2012), (BARREIROS; MOREIRA, 2009), (NGUYEN; COLMAN; HAN, 2011), (FILHO *et al.*, 2010), (MENDONCA; BRANCO; COWAN, 2009), (ROMERO *et al.*, 2013), (KRUEGER, 2006), (DHUNGANA *et al.*, 2010), (LAGUNA; MARQUéS, 2010), (BEUCHE; PAPAJEWSKI; SCHRÖDER-PREIKSCHAT, 2004)). Estas soluções oferecem muitas vezes opções de importação e exportação de configurações, compatibilidade com outras ferramentas e compartilhamento de ativos entre linhas de produtos, isso promove a interoperabilidade e potencializa o reúso dentro da própria solução e de forma externa também.

Contudo, apesar dos vários benefícios observados, foi possível identificar também várias limitações que muitas vezes desmotivam a utilização da solução. Dentre as principais limitações, pode-se destacar a falta de soluções completas para gerenciar linhas de produtos. Uma grande parte das soluções analisadas ((AJILA; KABA, 2008), (GARG *et al.*, 2003), (RABISER *et al.*, 2007), (ACHER *et al.*, 2013), (ALVES *et al.*, 2008), (PEROVICH; ROSSEL; BASTARRICA, 2009), (THAO, 2012), (SELLIER; MANNION; MANSELL, 2008), (ZHENG *et al.*, 2009), (SCHUBANZ *et al.*, 2012), (HOLL *et al.*, 2011), (MENDONCA; BRANCO;

COWAN, 2009), (SARABURA; BOWDEN, 2008), (EL-SHARKAWY; KRöHER; SCHMID, 2011a), (SINNEMA; DEELSTRA; HOEKSTRA, 2006), (TANHAEI; MOAVEN; HABIBI, 2010)) preocupa-se em resolver somente um problema, ou seja, auxiliam o gerenciamento em somente uma área bem específica. Além disso, diversas soluções ((AJILA; KABA, 2008) , (KHOSHNEVIS, 2012), (ACHER *et al.*, 2010), (ROMERO *et al.*, 2013), (EL-SHARKAWY; KRöHER; SCHMID, 2011a), (TANHAEI; MOAVEN; HABIBI, 2010)) possuem uma limitação bem acentuada em relação a sua abrangência em abordar uma área, podendo ser tão específica como gerenciar somente requisitos, por exemplo, e além disso não oferecem possibilidade de extensão ou parametrização para generalizar seu uso.

Os autores de um grupo de soluções ((MORENO-RIVERA; NAVARRO; CUESTA, 2011), (SUCCI *et al.*, 2000), (ROUILLÉ *et al.*, 2012), (CIRILO *et al.*, 2012), (ROMERO *et al.*, 2013), (SINNEMA; DEELSTRA; HOEKSTRA, 2006), (BEUCHE; PAPAJEWSKI; SCHRÖDER-PREIKSCHAT, 2004)) declaram que seu uso pode ser muitas vezes mais complexo do que o desejado, visto que para utilizar a solução precisa-se primeiro estudar e dominar bem suas especificações, o que muitas vezes dificulta sua adoção. Além da complexidade, um dos obstáculos também a ser considerado, é que boa parte das soluções analisadas ((SINNEMA; DEELSTRA; HOEKSTRA, 2006), (MUTHIG; SCHROETER, 2013), (MORENO-RIVERA; NAVARRO; CUESTA, 2011), (ROUILLÉ *et al.*, 2012), (SUCCI *et al.*, 2000), (PEROVICH; ROSSEL; BASTARRICA, 2009), (JUNIOR *et al.*, 2005), (CIRILO *et al.*, 2012), (BACHMANN; NORTHROP, 2009), (BOTTERWECK *et al.*, 2008), (SINNEMA; DEELSTRA; HOEKSTRA, 2006)) necessita de uma grande intervenção de um especialista, que entenda dos modelos e especificações e realize muitos procedimentos manuais.

Uma das limitações que também é importante ao se considerar a motivação para o uso é a falta de uma interface gráfica ou uma interface muito complexa ou pobre, como ocorre em algumas soluções ((MORENO-RIVERA; NAVARRO; CUESTA, 2011), (ACHER *et al.*, 2011), (ZHANG; JARZABEK, 2003), (BACHMANN; NORTHROP, 2009)). Uma solução ((MUTHIG; SCHROETER, 2013)) não apresenta também suporte ao controle de versões, o que dificulta sua utilização para realizar a manutenção e a evolução em LPS.

Um dos grandes obstáculos, principalmente na academia, é o fato de várias soluções serem proprietárias ((KRUEGER, 2006), (BEUCHE, 2012), (KRUEGER, 2008)). Infelizmente são essas que oferecem um conjunto mais completo de soluções em gerenciamento de linhas de produtos além de suporte ao usuário e validação. Algumas soluções disponíveis de forma livre ((BARREIROS; MOREIRA, 2009), (SELLIER; MANNION; MANSELL, 2008), (BOTTERWECK *et al.*, 2008), (HOLL *et al.*, 2011)) que foram analisadas ainda nem foram validadas ou implementadas, ou até mesmo não oferecem uma abordagem prática.

3.2 Trabalhos Relacionados

O conceito de LPS vem ganhando destaque nos últimos anos, como mostrou a Figura 11. Apesar de sua dificuldade de adoção, existem grandes investimentos e pesquisas nesta área. Por isso, na literatura, existem vários estudos que visam catalogar e/ou classificar os recursos disponíveis, como mapeamentos e revisões sistemáticas. Cada pesquisa possui seus próprios requisitos de busca que definem seu foco e granularidade, assim como o mapeamento sistemático apresentado neste capítulo.

Um exemplo de pesquisa semelhante ao mapeamento realizado neste trabalho é a revisão sistemática conduzida por Lisboa *et al.* (2010). A revisão apresentada busca classificar ferramentas de apoio à fase de Engenharia de Domínio das LPS. Os resultados foram avaliados de acordo com o tipo de suporte oferecido, a completude e a qualidade da ferramenta, e se ela atende ao esperado. A revisão também segue o processo proposto por Kitchenham (2004) para conduzir uma revisão sistemática e selecionou 19 ferramentas para avaliar de acordo com a funcionalidade das ferramentas, documentação, interface, entre outros. Os resultados oferecem ao usuário uma diretriz a respeito das ferramentas existentes e suas funcionalidades, no contexto somente da Engenharia de Domínio.

Por outro lado, este mapeamento buscou identificar as abordagens que oferecem suporte para ambas as fases de desenvolvimento da LPS, ou seja, Engenharia de Domínio e Engenharia de Aplicação. O objetivo era identificar soluções que se assemelham à proposta neste trabalho. Dentre os resultados, somente duas soluções foram consideradas mais completas, ou seja, oferecem suporte desde a elaboração até a manutenção e evolução da LPS, como mostrou a Figura 16.

A primeira solução foi apresentada por Laguna e Marqués (2010). Eles desenvolveram uma ferramenta incorporada ao MS Visual Studio que permite a modelagem da variabilidade, bem como automatiza transformações requeridas incluindo a geração de código do produto final. A ferramenta propõe manter modelos de projetos dentro dos padrões utilizando o mecanismo de mesclagem de pacotes da UML 2. Nessa abordagem, variabilidades e similaridades da arquitetura da LPS são descritas com a utilização do pacote de diagramas da UML, onde os pacotes são ligados de acordo com seus relacionamentos. Todas as modificações e evolução do modelo de *features* são registradas garantindo sua rastreabilidade. A instanciação de um produto então é realizada a partir de um modelo de configuração de *features* seguido de uma transformação automática para código. Dentre os benefícios da solução apresentada, pode-se ressaltar a facilidade de importação/exportação das *features* em XMI para uso em ferramentas CASE UML; a geração automática de produtos a partir de configuração de *features*; e a compatibilidade com ferramentas CASE.

A segunda solução foi proposta por Perovich, Rossel e Bastarrica (2009). Eles definiram um processo de desenvolvimento de LPS com a utilização de técnicas de MDE que sistematiza

a fase de Engenharia de Domínio para automatizar a fase de Engenharia de Aplicação. Ao sistematizar a construção da arquitetura, eles alcançaram a consistência entre requisitos. A manutenção e evolução da LPS são feitas por meio de transformações aplicadas no modelo de projeto, o que permite a rastreabilidade. Ao final da aplicação do processo baseado em MDE o produto é gerado automaticamente por meio de *Domain Specific Language* (DSL) que pode ser associada a regras próprias de transformações.

No processo proposto, apoia-se uma arquitetura independente do produto e as decisões arquiteturais são relativas apenas a *features* filhas, então modificações possuem apenas impacto local, o que aumenta a consistência em caso de manutenção ou evolução da LPS, mas por outro lado pode ser considerada também uma limitação. Outras limitações listadas também pelos autores são: a solução não possui escalabilidade; a implementação compreende apenas o nível de processo; a arquitetura oferecida não é apoiada por métodos tradicionais, mas as arquiteturas geradas sim, apesar de se tornarem muito custosas dependendo do número de produtos gerados; e finalmente, para utilização do processo é necessário um bom conhecimento em modelos de transformação para desenvolver as complexas regras de transformações associadas a cada *feature*, o que pode dificultar a adoção do processo.

Em comparação a este trabalho, as duas soluções possuem foco na criação e não no gerenciamento da LPS, além de não mostrarem uma preocupação com o armazenamento, o gerenciamento e o reúso dos ativos produzidos durante o planejamento e construção da LPS. A compatibilidade com outras ferramentas favorece o reúso das *features* e subclasses, porém os outros artefatos produzidos não possuem um padrão de armazenamento, o que dificulta sua reutilização em outros projetos. Assim, o uso dessas soluções pode, por exemplo, ser associado a este trabalho, cada um cumprindo seu objetivo, um de construir e outro de gerenciar, uma vez que a geração de produtos está fora do escopo deste trabalho.

Um grande diferencial neste trabalho é oferecer o máximo de padronização dos ativos para alavancar o potencial de reúso tanto de artefatos, quanto de *features*, processos e elementos de processos utilizando a RAS. Ao buscar na literatura exemplos e descrições de como representar e empacotar ativos reusáveis, não foram encontrados resultados. Grande parte dos estudos demonstram como identificar e utilizar esses ativos, mas não como mapeá-los de uma estrutura particular para uma estrutura RAS. Esses estudos geralmente propõem extensões da RAS para se adequar a vários propósitos diferentes, assim, a utilização da especificação nesses casos é bem específica de cada contexto, como por exemplo representar componentes, serviços, diferentes tipos específicos de artefatos, e assim por diante.

Um exemplo de extensão encontrada foi apresentada por Moon *et al.* (2007), que estende o *RAS Default Profile* para armazenar, gerenciar e rastrear variabilidades de LPS. Já o uso da especificação RAS proposto por Elgedawy e Ramaswamy (2009), visa representar *Component Business Maps* (CBMs) para permitir a identificação precoce de ativos reusáveis em um projeto, embora eles não tenham especificado qual perfil RAS foi utilizado e nem se uma extensão foi

criada.

A solução proposta por Zhou *et al.* (2010) não estende a RAS, eles apresentam um método de integração e análise de reúso para ativos legados no contexto de *Service Oriented Architecture* (SOA), onde os ativos extraídos são armazenados no repositório *IBM Rational Asset Manager Repository* (RAM), que geralmente é utilizado para armazenar ativos não estruturados, como .jar, .war e .ear além de outros documentos.

Outros exemplos de repositórios RAS online são o LAVOI criado por Moura (2013) e o OpenCom criado por Hong-min, Zhi-ying e Jing-zhou (2009). Ambos estendem perfis da RAS para melhor adaptação de diversos tipos específicos de ativos, facilitando sua classificação, busca e uso.

Apesar do número de abordagens encontradas na literatura que são relacionadas à RAS, nenhuma delas apresenta um exemplo explícito de como utilizar a RAS e mapear atributos, além de nenhuma delas sugerir o reúso de elementos de processos representando-os em RAS.

3.3 Considerações Finais

Este capítulo teve por objetivo identificar e avaliar soluções existentes na literatura para apoiar o gerenciamento de linhas de produto de software. Para isso, foi realizado um mapeamento sistemático, onde foi definido o protocolo da pesquisa para a condução da pesquisa que seguiu um processo proposto por Kitchenham (2004). A partir dos dados extraídos dos estudos selecionados, foi possível reunir informações suficientes para responder às questões de pesquisa propostas.

Com o resultado da pesquisa realizada foi possível identificar 50 soluções existentes na literatura que oferecem apoio a pelo menos uma fase do gerenciamento de linhas de produtos de software, o que caracteriza uma oferta muito reduzida de soluções que oferecem apoio ao gerenciamento de LPS. Grande parte das soluções existentes não é completa, muitas vezes não oferecendo suporte a todas as fases do desenvolvimento, manutenção e evolução da LPS e não oferecendo também uma utilização de todo o potencial de reúso que artefatos e *features* podem oferecer.

As lacunas apontadas nos resultados deste mapeamento sistemático sugerem perspectivas de trabalhos futuros, onde pode-se destacar:

- *Potencialização do reúso:* Das soluções apresentadas, somente uma (FILHO *et al.*, 2010) permite a utilização de ativos do repositório em outras LPS. Uma abordagem em que os ativos armazenados não fossem presos a uma LPS e pudessem ficar disponíveis para serem utilizados em qualquer contexto de forma aberta iria potencializar o reúso não somente dentro da LPS mas globalmente.

- *Utilização de padrões:* A falta da utilização de padrões ficou evidenciada nesta pesquisa.

Uma solução que oferecesse recursos de forma padronizada, ofereceria mais segurança e organização ao usuário, além de promover a flexibilidade e interoperabilidade com outras ferramentas. Como exemplo da utilização de padrões para as soluções voltadas a linhas de produtos de software pode-se destacar a RAS (GROUP, 2005), a CVL (GROUP, 2012b), entre outros.

- *Utilização de serviços:* A utilização de orientação a serviços também é uma boa opção para promover a interoperabilidade entre ferramentas e facilitar o acesso às funcionalidades da solução. Este é um recurso que não foi encontrado na maior parte das soluções analisadas, com exceção da solução proposta por Khoshnevis (2012).

- *Ferramenta livre e completa de apoio ao gerenciamento:* A criação e disponibilização de uma abordagem livre e completa para o apoio ao gerenciamento de linhas de produtos de software, com validação, documentação e suporte satisfatórios poderia aumentar o interesse da comunidade em adotar o conceito de LPS, tanto para uso quanto para desenvolvimento.

Essas soluções poderiam incentivar a adoção da LPS nos segmentos educacionais e profissionais, além de promover o amplo reúso de ativos mesmo em outros contextos diferentes de LPS.

O próximo capítulo apresenta uma abordagem criada para promover o reúso de elementos de processo de software utilizando a RAS.

CAPÍTULO

4

MAPEAMENTO DE ELEMENTOS DE PROCESSO PARA RAS

Conforme apresentado na Seção 2.4, em 2005 a OMG propôs uma abordagem comum para descrever ativos reusáveis, a RAS (GROUP, 2005). No caso particular de LPS, o uso da RAS na modelagem do repositório contribui para tornar os ativos compatíveis entre diversas aplicações.

Existe na literatura uma variedade de extensões para o perfil padrão (*Default*) da RAS, sendo que a maior parte delas tem o objetivo de melhorar a representação de tipos específicos de ativos reusáveis (MOURA, 2013)(MOON *et al.*, 2007)(HONG-MIN; ZHI-YING; JING-ZHOU, 2009). No entanto, não foram encontrados na literatura estudos que demonstrem como a RAS pode ser utilizada para representar os elementos de um processo, em particular, no domínio de LPS. Essa não é uma tarefa simples, uma vez que existem muitas decisões a serem tomadas, por exemplo, como elementos de processo que são composições de outros elementos devem ser representados e armazenados utilizando a RAS.

Isso motivou a proposição de um mapeamento de elementos de processo para RAS é descrito neste capítulo. Este mapeamento foi publicado em um evento (PACINI; BRAGA, 2015a) e é aqui reproduzido após tradução e pequenas adaptações. Como descrito na motivação, a intenção do mapeamento aqui proposto é estender o reúso a nível de processo, ou seja, facilitar o reúso de fases, atividades, e qualquer outro artefato relacionado ao próprio processo. No caso específico de LPS, é importante considerar abordagens que já foram aplicadas com sucesso na prática e que são bem documentadas, como as abordagens propostas por Gomaa (2006) e Clements e Northrop (2002), para criar uma prova de conceito aplicando este mapeamento. A utilização dessas abordagens é apoiada por suas vastas documentações que incluem cenários de aplicações reais.

Embora diferentes processos de desenvolvimento de LPS contenham muitas variabilidades, eles geralmente dividem uma base comum, no entanto, as abordagens atuais não estimulam

o reúso do processo, o que causa retrabalho toda vez que o processo necessita ser instanciado. Além disso, a experiência ganha ao executar com sucesso um projeto não é considerada quando novos projetos similares surgem, porque eles não são armazenados adequadamente para o reúso.

Por isso, nesse contexto, o mapeamento criado permite a representação e o armazenamento de elementos de processo utilizando RAS, em particular no domínio LPS. Isso visa alavancar o reúso de cada elemento de processo entre muitos projetos de forma independente. Como elementos de processo podem ser compostos de outros elementos de processo, o reúso pode ser feito tanto para elementos singulares quanto para completas árvores hierárquicas que contenham muitos elementos. Para isso foi definida uma estrutura de modelagem específica para representar processos.

4.1 Estrutura de Modelagem de Processos

A estrutura de modelagem de processo utilizado no mapeamento foi inspirada no *Software & Systems Process Engeneering Meta-Model* (SPEM) versão 2.0 (GROUP, 2008). O SPEM foi escolhido pois está sendo amplamente aplicado como linguagem de modelagem de processo de software, como indicado por García-Borgoñon *et al.* (2014). O SPEM foi usado como base para definir uma estrutura própria para representar muitos tipos de processos, como apresentado neste capítulo. É importante notar que embora este capítulo apresente um processo LPS para exemplificar o mapeamento, qualquer processo de software que se encaixe no modelo proposto pode ser utilizado.

A Figura 17 apresenta o modelo criado. Ele contém vários elementos: *process model*, *phase*, *activity*, *artifact*, etc. Essa estrutura visa representar todos os elementos de processo para um processo de desenvolvimento de software, isso é, é capaz de representar processos de diferentes abordagens de desenvolvimento existentes na literatura (no domínio de LPS pode-se citar ESPLEP - ver Seção 2.3.2).

É importante notar que esse modelo pode ser utilizado para representar tanto o modelo do processo (*process model* ou *process template*) quanto a instância do processo, que é derivada da instanciação do modelo do processo em um contexto particular. A instância de um processo faz referência ao modelo, porém possui seus próprios elementos, de acordo com sua execução. Isso é importante porque pode-se querer reusar não somente os modelos, mas também as instâncias que foram bem sucedidas em um contexto particular e que assim podem ser recomendadas quando situações similares ocorrerem. Por exemplo, PLUS é um modelo de processo que pode ser reusado em um projeto de LPS real, resultando em uma instância de processo. Mais tarde, quando um novo projeto se iniciar em um contexto similar, ao invés de reusar o PLUS, é possível reutilizar essa instância, porque já está personalizada para esse contexto.

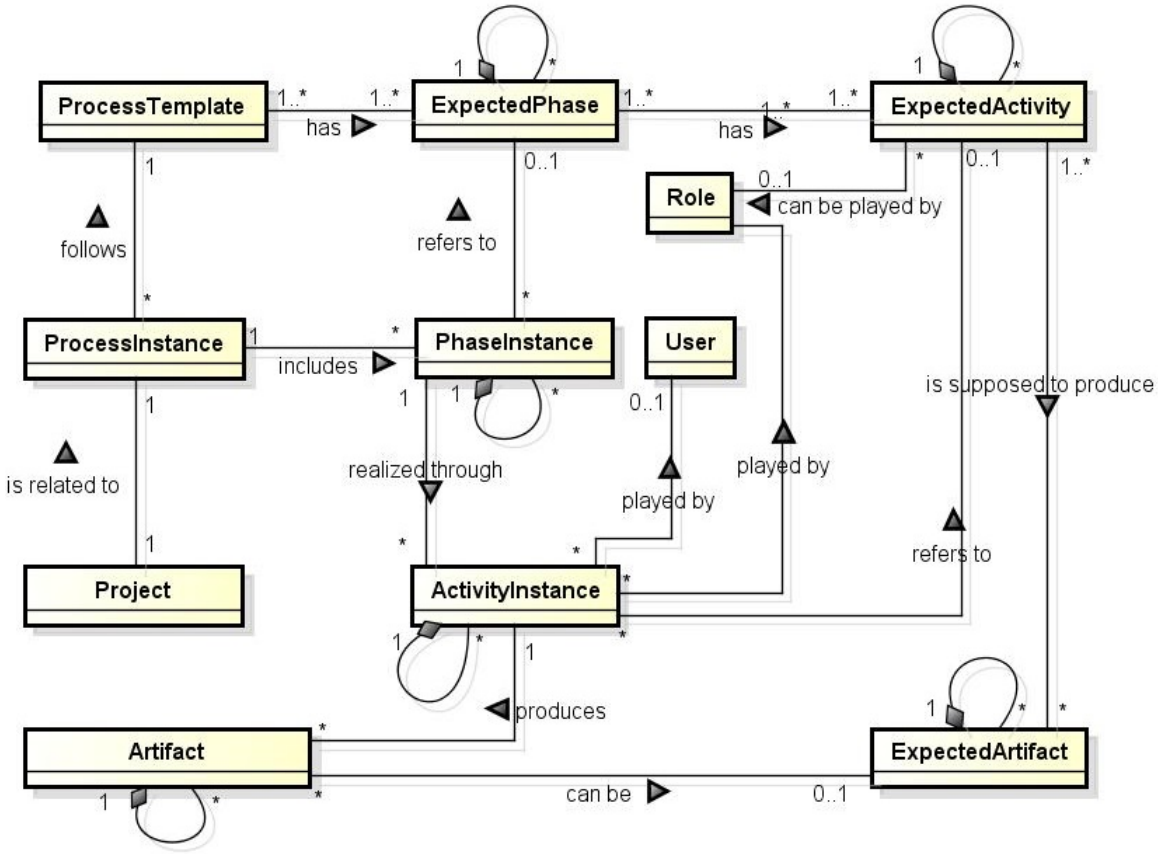


Figura 17 – Meta modelo de processo criado inspirado no SPEM.

4.1.1 Estrutura de Modelagem Aplicada a Modelo e Instância de Processo

Para ilustrar o uso do modelo proposto na Figura 17, a Figura 18 apresenta um exemplo aleatório de uma empresa de software que está desenvolvendo uma LPS para facilitar a criação de aplicações para gerenciamento de hotéis (esse exemplo foi escolhido por ter regras de negócio simples que ajudam a entender a diferença entre modelo de processo e instância de processo). O processo de desenvolvimento foi instanciado a partir do ESPLEP (Figura 3, Seção 2.3.2) e neste contexto o foco é somente na fase de engenharia de domínio. A engenharia de domínio se inicia pela criação do projeto(*Project: Hotel SPL Project*) e associado a ele, a instância do processo (*ProcessInstance: Hotel SPL Process*), que por sua vez está associado ao ESPLEP (*ProcessTemplate*). Na figura, foram utilizados estereótipos para identificar os papeis de cada classe no exemplo e [...] para expressar que outras instanciações análogas podem ser feitas.

Para cada elemento de processo do ESPLEP, é necessário analisar se ele é adequado para o processo *Hotel SPL Process* e, se for, criar sua respectiva instância. Além disso, o processo *Hotel SPL Process* pode ser adaptado a um contexto particular de uma empresa, por exemplo, adicionando novas atividades ou artefatos, ignorando atividades opcionais, etc., enquanto o ESPLEP permitir essas adaptações. Isso é possível porque durante a definição da modelagem do

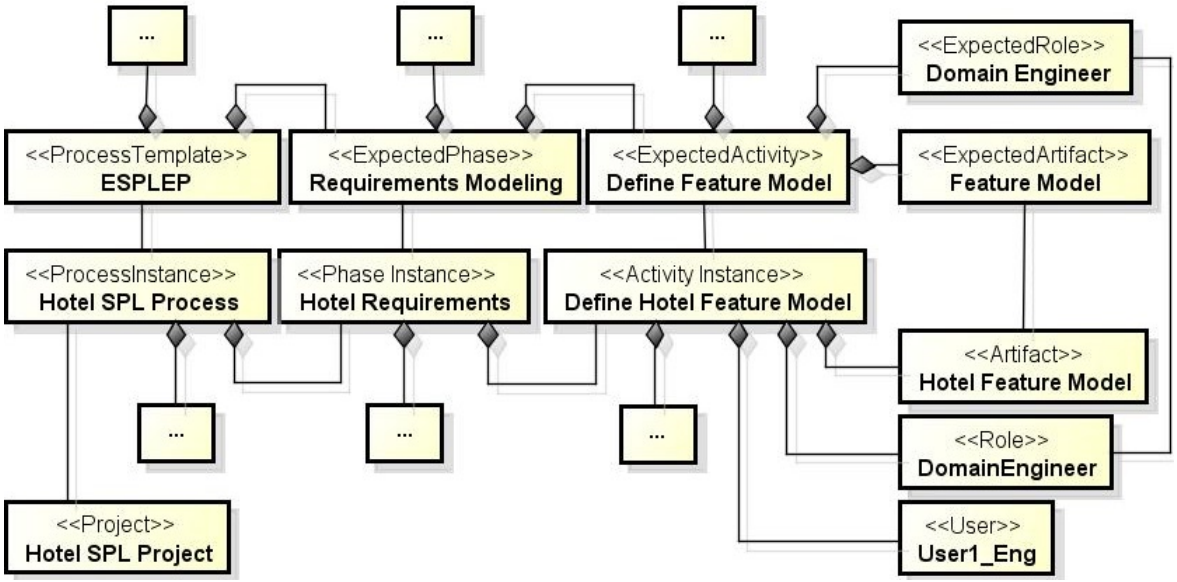


Figura 18 – Exemplo da modelagem do ESPLEP utilizando o meta-modelo criado.

ESPLEP foram definidos quais elementos eram obrigatórios ou opcionais. Elementos obrigatórios necessitam ter um instância na execução do projeto, enquanto elementos opcionais podem ser omitidos.

4.2 Mapeando Elementos de Processo para RAS

A ideia é mapear cada elemento de processo para um ativo (*RAS Asset*) independente, como mostra a Figura 19. A discussão detalhada é apresentada mais tarde na Seção 4.3. Na figura, esse mapeamento é feito considerando o exemplo de um modelo de processo e seus elementos associados. Do lado esquerdo da figura pode ser observado o modelo do processo (*ProcessTemplate*), ao centro a estrutura RAS, e do lado direito exemplos de valores.

Como apresentado na figura, os atributos *name* e *id* são correspondentes em ambas as estruturas (*ProcessTemplate* e *RAS Asset*). O atributo *description* é mapeado para o atributo RAS *Description*, mas no atributo *short-description* do elemento RAS *Asset* seu valor é limitado aos primeiros 50 caracteres. O atributo *type* é mapeado para o elemento RAS *Classification*. Os outros elementos de processo são tratados como ativos (*Assets*) independentes, então eles são criados separadamente em uma estrutura RAS e relacionados por meio do elemento RAS *RelatedAsset*.

Após feito esse mapeamento, toda a informação do processo está armazenada em uma estrutura RAS, no entanto, de acordo com a RAS, alguns elementos obrigatórios ainda precisam ser preenchidos. Por exemplo, o *Asset* tem atributos: *date*, *state*, *version* e *access-rights*, que podem ser preenchidos com valores padrão como mostra a Figura 19. O elemento *Profile* é obrigatório e identifica qual perfil está sendo utilizado para representar o ativo (*Asset*). Neste caso, este mapeamento utiliza o perfil padrão (*DefaultProfile*, versão 2.1) como apresentado na

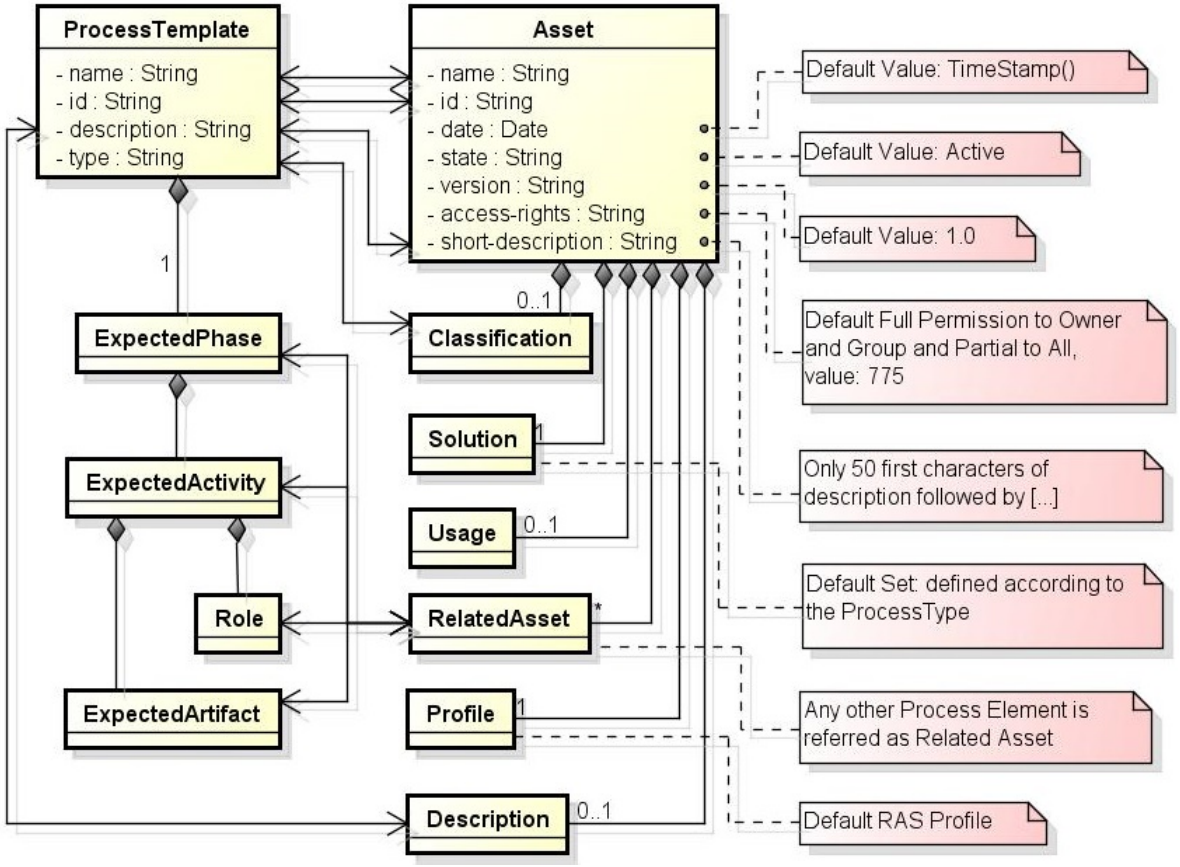


Figura 19 – Exemplo da estrutura do mapeamento do elemento *Process Template* para RAS.

Seção 4.4.1.

Outro elemento obrigatório importante na estrutura RAS é o *Solution*, que deve ser preenchido com as informações e documentos relacionados que são correspondentes ao elemento de processo sendo armazenado. Por exemplo, no modelo de processo a representação poderia ser um documento XML contendo a estrutura completa do processo. Existem outros elementos opcionais que são exemplificados na prova de conceito.

4.3 Aumentando o Potencial de Reúso

A principal motivação para a criação deste mapeamento é alcançar o potencial de reúso dos processos, bem como decrementar tempo, esforço e espaço de armazenamento quando for criar e instanciar processos. A partir do armazenamento independente de cada elemento de processo como ativo reusável, é possível compartilhá-lo entre diferentes processos e instâncias de processos, como ilustrado na Figura 20. Quando um elemento é compartilhado entre processos, toda árvore hierárquica de elementos associados ao elemento raiz também é compartilhada.

Como pode ser observado na Figura 20, o elemento de processo identificado por *ExpAct_3* representa uma atividade esperada (*ExpectedActivity*) do processo, que está sendo referenciada por três diferentes fases esperadas (*ExpectedPhase*) de processo: *ExpPhs_2*, *ExpPhs_3*

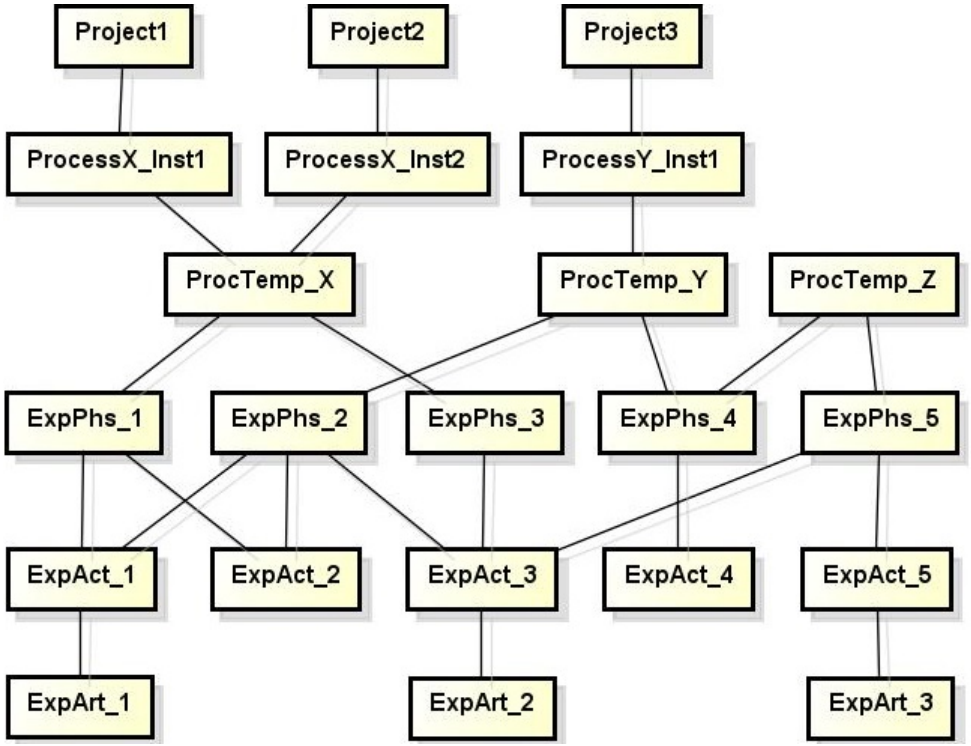


Figura 20 – Exemplo do potencial de reúso que pode ser alcançado.

e *ExpPhs_5*. Essas fases, por sua vez são referenciadas por três processos (*ProcessTemplate*) diferentes: *ProcTemp_X*, *ProcTemp_Y* e *ProcTemp_Z*. O reúso dos elementos de processo como sugerido por este mapeamento permite o compartilhamento desses elementos tanto no modelo de processo quanto na instância do processo, embora o exemplo da Figura 20 apresente o reúso somente dos elementos do modelo. Isso leva a um grande potencial de reúso, além do fato de que qualquer repositório ou ferramenta baseada em RAS podem ser facilmente utilizados.

4.4 Prova de Conceito

Como descrito na seção anterior o mapeamento aqui apresentado permite a representação de elementos de processo de acordo com a RAS, tanto para modelo de processo quanto para sua execução. O ESPLEP é composto por cinco fases esperadas (veja a Seção 2.3.2): modelagem de requisitos, modelagem de análise, modelagem de projeto, implementação incremental de componentes e teste de software. Cada fase é composta por atividades esperadas e seus artefatos esperados.

A RAS é flexível quanto às suas possibilidades de uso, extensão e representação de ativos de acordo com cada necessidade de projeto. O mapeamento criado apresenta somente uma de muitas possíveis formas de usá-la. É recomendado seguir este mapeamento para facilitar a recuperação dos ativos posteriormente, isto é, aplicações cliente que buscarem ativos serão mais facilmente implementadas se tiverem conhecimento de que a estrutura é baseada em RAS. No entanto, é importante observar que existe um conjunto mínimo de informações que são

obrigatórias para um ativo reusável ser considerado em conformidade com a RAS: ele deve indicar o perfil utilizado, possuir ao menos um artefato além das informações básicas do ativo, como mostrado na Figura 19. Além disso, existem outras informações que também podem ser armazenadas, assim, esta prova de conceito pretende mostrar uma possível forma de utilizar RAS para armazenar elementos de processo. Isto é apresentado nas seções seguintes.

4.4.1 RAS Profile

O elemento *Profile* (Figura 21) se refere à estrutura de representação do ativo que está sendo utilizada. O *RAS Core* é abstrato, assim somente perfis podem ser instanciados e todos eles derivam do *Core*. O perfil derivado que está mais próximo do (*Core*), e foi escolhido pra ser utilizado neste mapeamento, é chamado *Default Profile*. Mais especificamente, adotou-se a versão 2.1, uma vez que ela é mais compatível com qualquer outro perfil que estenda este.

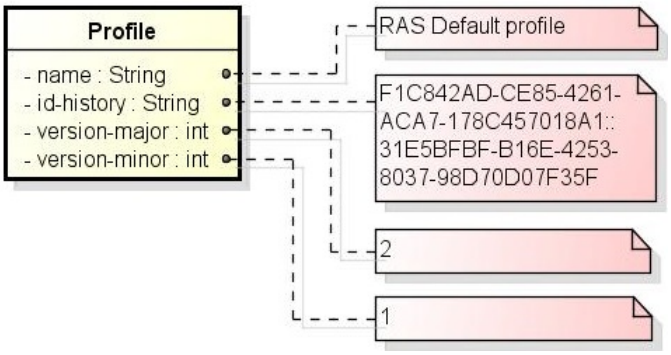


Figura 21 – Exemplo do elemento RAS *Profile*.

4.4.2 RAS Solution

O elemento *Solution* se refere aos artefatos que compõem o ativo reusável. Um ativo pode ser visto como um conjunto de artefatos (ao menos um artefato é obrigatório). Os artefatos são o principal objetivo do reúso, e eles podem ser classificados em muitos tipos, como documento (doc, pdf, txt), código (Java, SQL, PHP, C#), descritores (XML, XSD, HTML), entre outros.

A Figura 22 apresenta uma instância do elemento *Solution*. Considerando a LPS de hotel apresentada na Seção 4.1.1, o *Solution* contém o modelo de *features* (*Feature Model*), definido durante a atividade *Define Hotel Feature Model* na fase *Hotel Requirements*.

O principal artefato (*Artifact*) armazena (de fato é uma referência para onde o objeto real está) o modelo em si, e possui atributos como nome (*name*), identificador (*identifier*), tipo (*type*, como por exemplo .astah), versão (*version*) e os direitos de acesso (*accessRights*) do artefato. Neste exemplo, os direitos de acesso foram definidos seguindo o modelo Unix com valor 775, isto é, o dono do artefato e o grupo ao qual o dono pertence possuem direitos totais sobre o artefato, enquanto outros usuários podem somente ler e executar. A RAS também recomenda o uso de *Universally Unique IDentifiers* (UUIDs) para os identificadores.

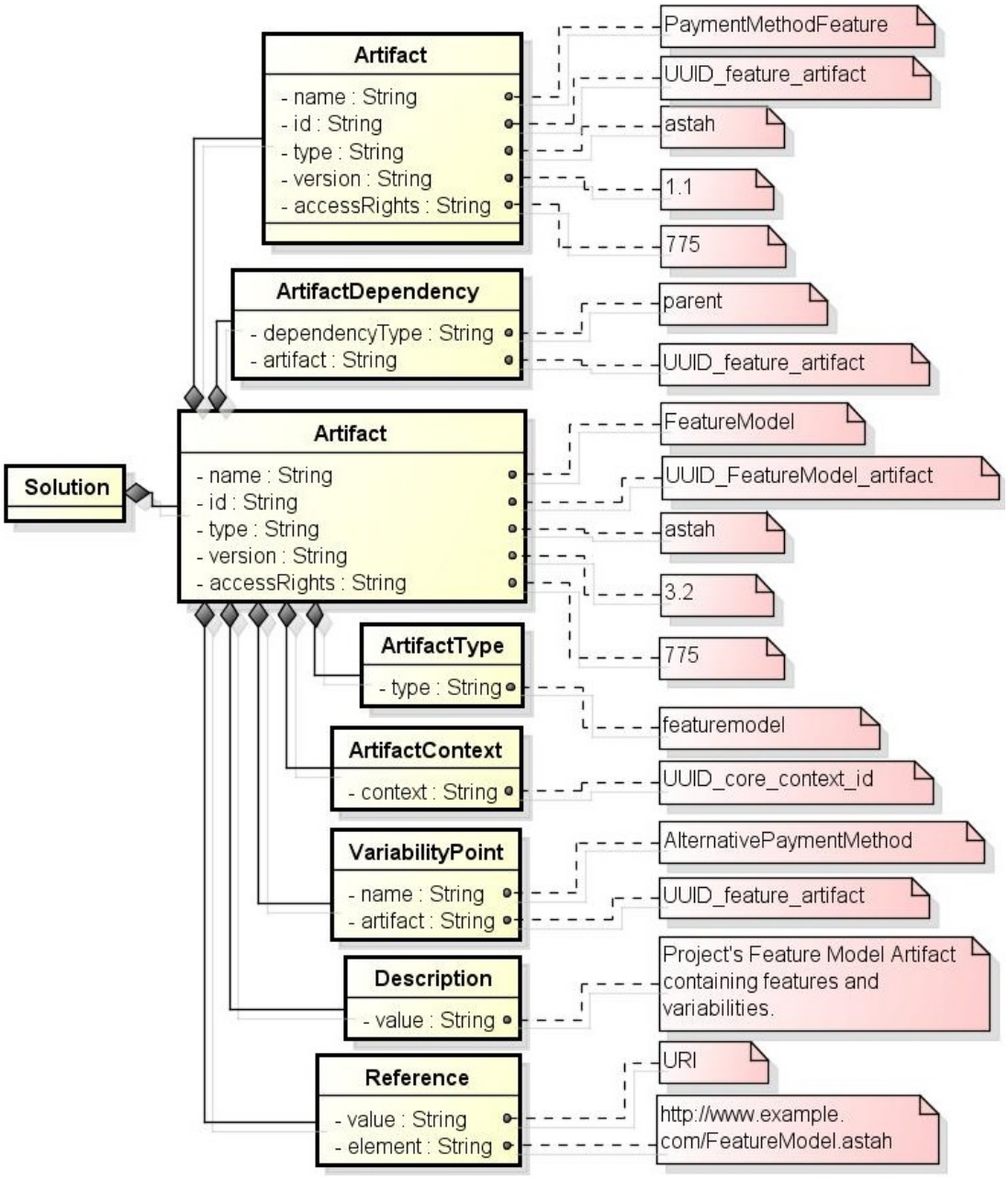


Figura 22 – Exemplo do elemento RAS *Solution*.

O *ArtifactType* representa o tipo lógico do artefato, indicando o que o artefato representa no modelo (no exemplo, é um modelo de *features* da LPS). O *ArtifactContext* representa o contexto no qual o artefato é utilizado. No exemplo, como o modelo de *features* é essencial para o ativo, é classificado como tipo *Core*, como mostrado na figura.

O elemento *VariabilityPoint* descreve as variabilidades do artefato, no exemplo, o artefato possui uma *feature* chamada *PaymentMethod*, que possui várias alternativas. Neste caso, a *feature* que contém a variabilidade é apresentada neste elemento, enquanto as *features* alternativas correspondentes e as regras da variabilidade são definidas no elemento *Usage*, descrito posteriormente. Também, este artefato tem outros artefatos dependentes representando cada *feature* do modelo

de *feature*. Isso permite o reúso do modelo de *feature* em si e dos artefatos correspondentes que o implementam, não somente durante a engenharia de aplicação, mas também na engenharia de domínio de outras LPS do mesmo domínio (por exemplo, *PaymentMethod* poderia ser utilizado em diferentes LPS).

4.4.3 RAS Classification

O elemento *Classification* se refere aos descritores ou classificadores do ativo. Ele pode incluir mais de um descritor ou até mesmo esquemas (*Schema*) para descrever o ativo. Também é possível definir contextos que serão referenciados por artefatos e pelo elemento *Usage* e suas atividades.

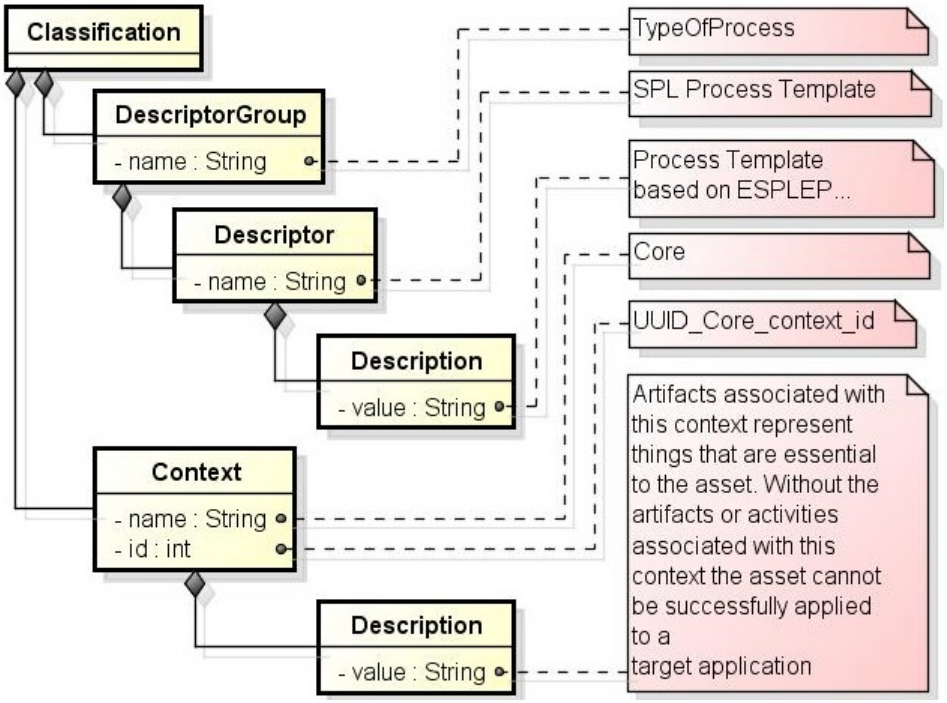


Figura 23 – Exemplo do elemento RAS *Classification*.

A Figura 23 apresenta uma instância do elemento *Classification*. Supondo que seja necessário classificar o elemento de processo *SPL Process Template*, uma sugestão seria classificá-lo como *TypeOfProcess* (ou tipo de processo). Para mapear essa informação na RAS primeiro é necessário definir um grupo de classificadores que representam qual o tipo de classificação que se deseja fazer, neste caso, deseja-se classificar como *TypeOfProcess*, então esse será o nome do elemento RAS *DescriptorGroup*. Tendo isso definido, pode-se definir valores para este tipo, cada qual como uma instância do elemento RAS *Descriptor*, que neste caso é *SPL Process Template*. Um ativo pode ter muitos classificadores, por exemplo esse mesmo ativo poderia ser classificado por *DescriptorGroup*: *ProcessElement* e *Descriptor*: *ProcessTemplate*.

De maneira análoga, todos os outros elementos de processo podem ser classificados utilizando essa estrutura, por exemplo *SPL Process Instance* que também é parte do *DescriptorGroup*:

TypeOfProcess; fase esperada (*ExpectedPhase*), atividade esperada (*ExpectedActivity*) e artefato esperado (*ExpectedArtifact*), que são também parte do *DescriptorGroup*: *ProcessElement*; e assim por diante.

O elemento *Context* é usado para referenciar artefatos e atividades no contexto do ativo. Como mostrado na figura, o contexto *Core* representa que o artefato/atividade relacionado(a) é essencial para o ativo.

4.4.4 RAS Usage

O elemento *Usage* é usado para armazenar informações sobre como utilizar o ativo (manuais do usuário por exemplo), bem como quais atividades devem ser executadas para que o ativo funcione corretamente. Esta informação pode ser relativa ao contexto, a um artefato específico, ou ao ativo como um todo.

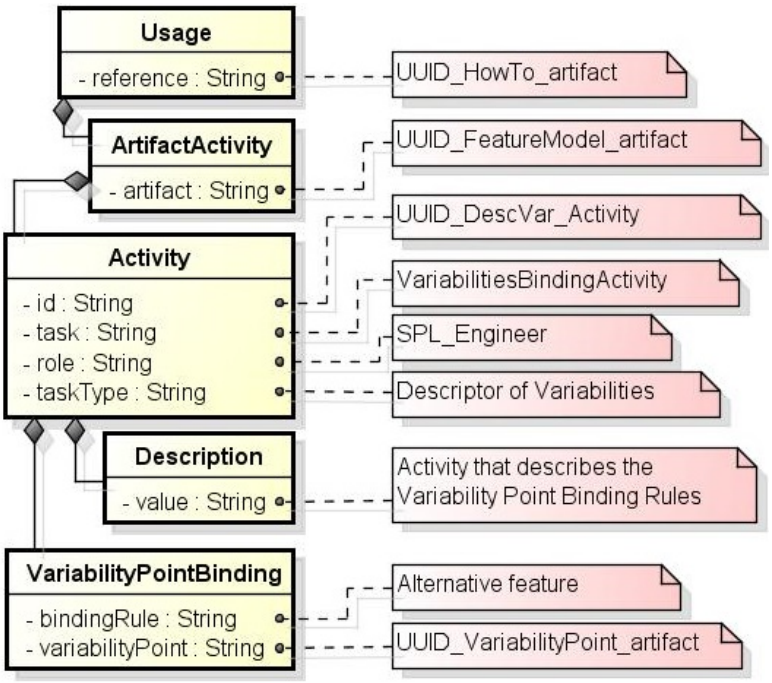


Figura 24 – Exemplo do elemento RAS *Usage*.

A Figura 24 apresenta uma instância do elemento *Usage* na qual o atributo *reference* se refere a um artefato contido em outro ativo, identificado por *UUID_HowTo_artifact*. Esse artefato representa um documento com instruções de como utilizar e interpretar os componentes do elemento *Usage*. Embora este elemento não seja obrigatório, ele pode ser útil para promover o reúso. Esquemas e outros tipos de artefatos também podem ser referenciados.

No exemplo do Hotel SPL, como mencionado anteriormente, existe uma *feature* chamada *PaymentMethod* que representa uma variabilidade. A Figura 24 apresenta as regras para utilizar esta variabilidade a partir de instâncias do elemento *VariabilityPointBinding*. Essas instâncias estão contidas no elemento *Activity*, que por sua vez está contido no elemento *ArtifactActivity*.

Isso significa que a atividade é relevante somente no contexto do artefato referenciado. A atividade (*VariabilitiesBindingActivity*) descreve as regras que devem ser seguidas para aplicar as variabilidades no artefato do modelo de *features*.

No exemplo, o artefato identificado *UUID_feature_artifact* tem uma variabilidade chamada *AlternativePaymentMethods*. De acordo com a regra definida em *VariabilityPointBinding*, os artefatos dependentes (*children*) se referem a *features* alternativas do modelo de *features* como definido no atributo *bindingRule*.

4.4.5 RAS Related Asset

O elemento *RelatedAsset* especifica os relacionamentos entre ativos reusáveis. A partir desses relacionamentos é possível construir uma árvore de dependência com todos os ativos relacionados.

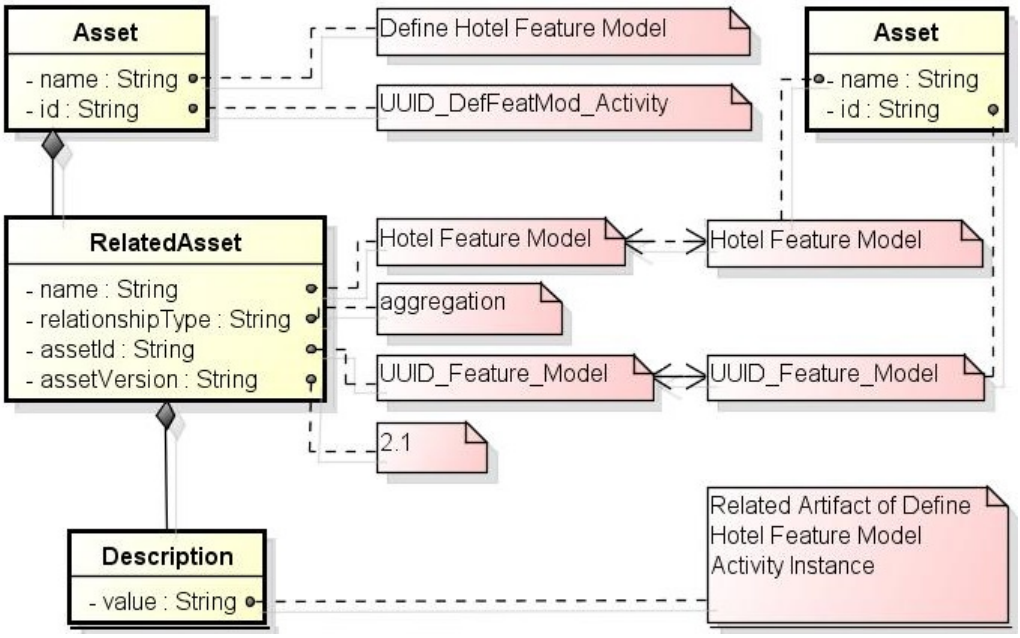


Figura 25 – Exemplo do elemento RAS *Related Asset*.

A Figura 25 apresenta uma instância do *RelatedAsset*. Este elemento possui um atributo *name* que corresponde ao nome do ativo ao qual está se relacionando, assim como o *assetID* e *assetVersion* representam, respectivamente, o ID e a versão do asset relacionado. A RAS define alguns tipos de relacionamentos (por exemplo, *aggregation* como na figura) e permite que outros tipos sejam criados.

Neste exemplo, existe um relacionamento entre a instância de atividade «ActivityInstance» Define Hotel Feature Model e o artefato «Artifact» Hotel Feature Model. Neste cenário, somente a atividade está relacionada ao artefato, não o oposto. Isto significa que somente a atividade possui navegabilidade para o artefato. Se a navegabilidade fosse nas duas direções o artefato deveria possuir um relacionamento com a atividade do tipo *parent*.

A navegabilidade nos relacionamentos é muito importante no contexto do reúso. Por exemplo, quando selecionar um elemento para reusar, todas as dependências desse elemento devem ser carregadas com ele. Neste caso, se for desejado reusar a atividade da Figura 25, o artefato seria carregado com ela. Todavia, se for desejável reusar somente o artefato, isso é possível porque o artefato não possui relacionamentos (dependências). Assim, se um elemento depende de outro e eles sempre devem ser carregados juntos, um relacionamento deve ser definido em ambos.

4.5 Considerações Finais

Este capítulo apresentou um mapeamento criado para representar elementos de processo como ativos reusáveis utilizando a RAS a partir de um meta-modelo de processo criado para a estrutura RAS. O mapeamento não somente representa as principais informações dos elementos de processo nas estruturas obrigatórias da RAS, mas também demonstra a utilização de várias outras estruturas disponíveis no *Default Profile* (Perfil Padrão) da RAS. Essa representação torna possível a criação de um repositório de elementos de processo, o que pode alavancar o potencial de reúso. O reúso de processos e de elementos de processos traz diversos benefícios, como reduzir o tempo e o custo do desenvolvimento enquanto aumenta a qualidade. Além disso, no repositório ainda podem ser construídos mecanismos de recomendação desses processos e elementos de processos de acordo com o tipo do usuário e o contexto da aplicação.

O mapeamento criada, apesar de ter foco no contexto de LPS, pode ser usado em processos de outros contextos, desde que sua estrutura siga o meta-modelo proposto. Para processos com diferentes estruturas, um mapeamento análogo ao apresentado neste capítulo pode ser feito.

Outra vantagem de utilizar este mapeamento é que os elementos de processos também podem ser compartilhados entre diferentes contextos e projetos, tanto para modelos quanto para instâncias de processo. Além disso, este mapeamento serve como exemplo documentado de como utilizar a especificação RAS em um modo prático, uma vez que nenhum exemplo detalhado do seu uso foi encontrado na pesquisa realizada na literatura.

O próximo capítulo apresenta uma solução implementada por meio de serviços para realizar este mapeamento e apoiar o gerenciamento de LPS potencializando o reúso.

CAPÍTULO

5

SERVIÇOS DE APOIO AO GERENCIAMENTO DE LPS

De acordo com a motivação e os objetivos deste trabalho, este capítulo apresenta uma abordagem baseada em serviços para gerenciar Linhas de Produtos de Software e potencializar o reúso. Os objetivos de implementar as funcionalidades para o gerenciamento de LPS e disponibilizá-las em forma de serviços, são promover a interoperabilidade, por serem independente de plataforma, a flexibilidade, por ser customizável ao contexto e ao ambiente de cada aplicação e a disponibilidade, por ficarem disponíveis na *web* potencializando sua utilização e facilitando o reúso nos repositórios.

As funcionalidades aqui apresentadas visam apoiar o gerenciamento e não o desenvolvimento de LPS, ou seja, o usuário é conduzido por todas as fases do processo de desenvolvimento da LPS até a geração dos produtos, enquanto todos os ativos resultantes desse processo são adequadamente armazenados para o reúso sem que o usuário precise se preocupar com isso. Assim, a seção seguinte apresenta os requisitos para o desenvolvimento desses serviços.

5.1 Requisitos

A fim de suprir as funcionalidades requeridas para o gerenciamento de uma LPS, foram definidos alguns requisitos que os serviços devem atender. Os requisitos foram estabelecidos de acordo com os resultados do mapeamento sistemático realizado e o grupo de pesquisa. Primeiramente, os serviços criados devem oferecer suporte à segurança e à privacidade das informações e dos documentos manipulados, assim foi definido o Requisito 1 (RQ1).

Considerando o ciclo de vida da LPS desde sua concepção, deve haver um processo que guie esse ciclo de vida de forma a garantir a qualidade e a correspondência dos requisitos definidos com as funcionalidades implementadas. Assim, o Requisito 2 (RQ2) define o gerenciamento de modelos de processos de desenvolvimento para guiar o ciclo de vida da LPS. Por gerencia-

mento de modelos de processos, quer-se dizer que funcionalidades de criação, atualização, busca, recuperação e exclusão devem ser oferecidas, tanto para o modelo do processo quanto para seus elementos dependentes, como fases, atividades e artefatos esperados.

Após a definição dos modelos de processo disponíveis para o desenvolvimento, a escolha do processo a instanciar depende do tipo de projeto que se deseja criar e se este processo atenderá às necessidades desse projeto. Assim o Requisito 3 (RQ3) corresponde ao gerenciamento dos possíveis tipos de projeto, que considerando o contexto de LPS podem ser uma nova LPS ou uma nova configuração para um sistema alvo derivado da LPS, e o Requisito 4 (RQ4) corresponde ao gerenciamento da instância do processo selecionado e todos seus elementos dependentes.

É na instanciação do processo que são armazenadas as informações sobre a execução desse processo e onde os ativos resultantes são manipulados. Ainda é possível que durante a execução de um processo, algumas fases, atividades e artefatos que não eram esperados de acordo com o modelo, sejam necessários, portanto devem ser suportados, ou também eram esperados mas são opcionais e não foram necessários, sendo assim necessário ignorá-los. Pode-se chamar este de Requisito 5 (RQ5).

A manipulação dos ativos dentro do processo é prevista no Requisito 6 (RQ6). Os ativos devem poder ser criados, atualizados, buscados, recuperados, movidos e removidos de acordo com as necessidades do projeto. O Requisito 7 (RQ7) ainda define que esses ativos devem ser mapeados e representados utilizando a estrutura RAS para potencializar o reúso. O mapeamento dos artefatos é semelhante ao mapeamento apresentado para os elementos de processo no capítulo anterior, assim, tanto os artefatos resultantes do processo de desenvolvimento quanto os próprios elementos de processo, sejam do modelo ou da instância, são mapeados e representados em RAS.

Como o foco da abordagem é a potencialização do reúso, as funcionalidades que alteram, movem ou removem ativos do repositório devem ser especialmente tratadas, pois se o ativo possui dependências, ou seja, outros ativos fazem referência a ele, este não pode remover artefatos compartilhados ou ser completamente removido. Para a remoção real de um ativo do repositório, é necessário que este não possua nenhuma dependência, assim, se alguma dependência existir, esta deve primeiramente ser removida antes do ativo poder ser removido. Caso a dependência não possa ser removida, e ainda sim se desejar remover o ativo, ele pode ser simplesmente excluído do projeto, mas não do repositório, ou seja, somente sua referência no projeto será removida. Este é o Requisito 8 (RQ8).

Como os serviços são oferecidos no contexto de LPS, é desejável que existam serviços de acesso rápido aos principais artefatos deste contexto, como a Arquitetura da LPS, o Modelo de *Features* e o Módulo de Implementação das *Features*. Este é o Requisito 9 (RQ9).

Ainda visando potencializar o reúso, é desejável que seja oferecida uma opção de desmembramento tanto do Modelo de *Features* quanto do Modelo de Requisitos para que cada

feature ou requisito possa ser armazenado de forma independente e diretamente relacionado com outros elementos. Isso significa que o reúso seria estendido a nível de cada *feature* e requisito, tornando possível o compartilhamento individual de cada um. Este é o Requisito 10 (RQ10).

Após definidos os requisitos das funcionalidades que os serviços devem implementar, a próxima seção apresenta os modelos dos dados que foram criados para serem utilizados na construção dos serviços.

5.2 Modelos de Dados

Foram criados dois modelos de dados descritos em arquivos XSD. Esses modelos tem por objetivo descrever os elementos que serão utilizados como parâmetros de entrada e saída na chamada dos serviços. Um desses modelos já foi apresentado na Figura 17 no Capítulo 4, porém essa figura representa apenas o modelo conceitual dessa estrutura, a Figura 26[1] apresenta o modelo XSD detalhado.

A Figura 26 apresenta a descrição dos elementos XML que devem ser transmitidos nas mensagens SOAP ao se consumir um serviço. O modelo apresenta o que corresponderia às classes em um diagrama de classes. Com efeito, a partir deste modelo foi utilizado um gerador de código para gerar as classes em Java na implementação dos serviços, mas isso será discutido com mais detalhe na Seção 5.4. Da mesma forma que foram geradas as classes em Java a partir desses modelos XSD, também é possível que as aplicações clientes gerem essas classes em diversas linguagens, conforme a necessidade de seus projetos.

A estrutura de alguns tipos de dados criados não estão presentes na figura, porém são descritos nos itens abaixo:

- **AccessRights:** define as permissões do ativo. Conforme ilustrado na Figura 22 do Capítulo 4, este conjunto de permissões segue o modelo Unix que estipula permissões de leitura, escrita e execução para o proprietário, para o grupo e para todos, nesta ordem. Assim, esta estrutura possui três atributos denominados *owner* (proprietário), *group* (grupo) e *all* (todos) que podem assumir valores preestabelecidos que são correspondentes às permissões *rwx* (leitura, escrita e execução).

- **ObjectType:** pode assumir valores preestabelecidos que são correspondentes aos tipos de objetos disponíveis no modelo, como por exemplo *ProcessTemplate*, *ActivityInstance*, e assim por diante.

- **ProcessElements:** pode assumir valores preestabelecidos que são correspondentes somente aos elementos que pertencem a um processo, como por exemplo *ExpectedActivity*,

[1] Modelagem criada utilizando a IDE Eclipse Helios (FOUNDATION, 2016a)

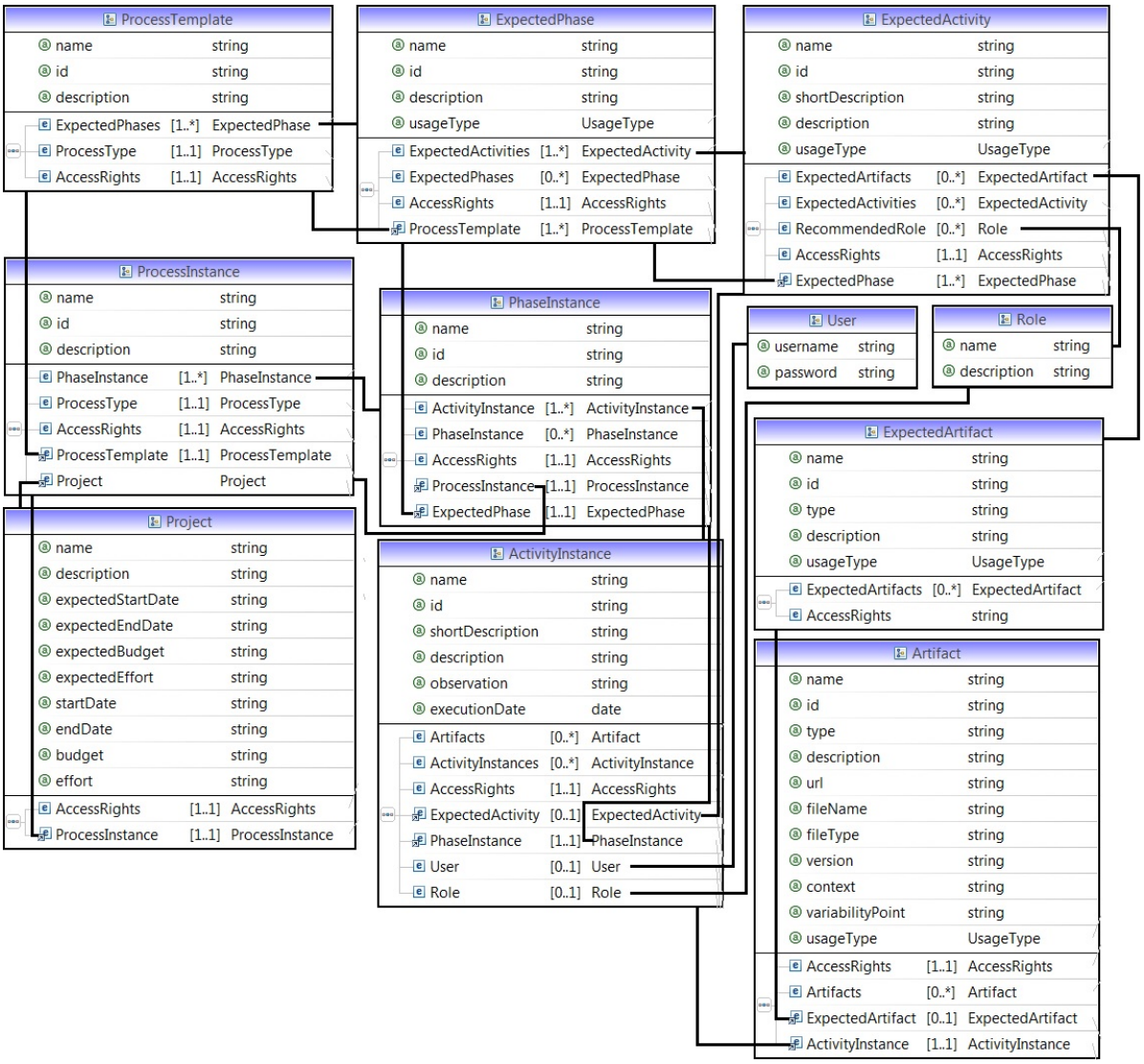


Figura 26 – Arquivo XSD referente à estrutura dos dados utilizados nos serviços.

Artifact, e assim por diante. Elementos como *ProcessTemplate* e *Project*, por exemplo, não estão contidos dentro de um processo, portanto não são valores válidos.

- **ProcessType:** pode assumir um de dois valores preestabelecidos que são correspondentes ao tipo do processo sendo manipulado: *SPL Development* e *Software Development*, correspondentes ao processo de desenvolvimento de uma LPS e de configuração de um sistema alvo, respectivamente.

- **UsageType:** utilizado para expressar o tipo de uso do elemento referenciado. Pode assumir os valores: *Required* (obrigatório) e *Optional* (opcional).

Considerando o RQ10, foi estabelecida uma estrutura para receber as informações do modelo de *features* e do modelo de requisitos de forma a facilitar seu desmembramento para mapeamento individual de *features* e requisitos. A Figura 27 apresenta esta estrutura.

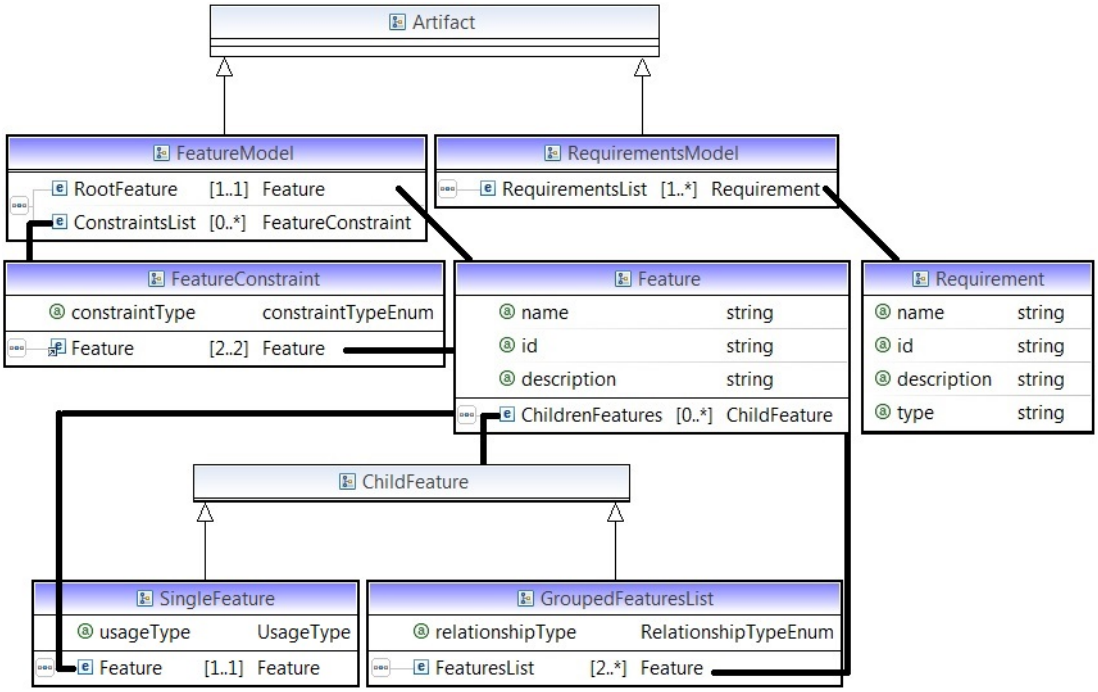


Figura 27 – Estrutura dos dados dos modelos de *features* e de requisitos no arquivo XSD.

Como pode ser observado na figura, o Modelo de *Features* (*FeatureModel*) herda a estrutura do artefato (*Artifact*) e possui dois elementos adicionais: *RootFeature*, que representa a raiz do modelo de *features*; e *ConstraintsList*, que representa uma lista de *constraints* (ou restrições) entre *features* do modelo de *features*. Essas restrições são utilizadas para indicar que uma *feature*, se utilizada, deve obrigatoriamente incluir (*includes*) ou excluir (*excludes*) a utilização de outra, assim cada instância do elemento *FeaturesConstraint* define uma restrição entre duas *features*.

O elemento *Feature*, por sua vez, representa a *feature* em sua individualidade, indicando um nome (*name*), um identificador (*id*) e uma descrição (*description*) opcional. O relacionamento hierárquico das *features* é definido pela lista de *features* filhas (*ChildrenFeatures*). Esta lista é vazia se a *feature* não possui *features* dependentes. Quando existem *features* dependentes, quando forem individuais, podem ser do tipo: obrigatória (*Required*) ou opcional (*Optional*), sendo esse valor atribuído pelo atributo *usageType* descrito acima. Quando existe um agrupamento das *features* dependentes, o atributo *relationshipType* define o tipo deste agrupamento: *OR* (quando uma ou mais *features* podem ser selecionadas), *XOR* (quando somente uma das *features* do grupo pode ser selecionada) ou *Mutex* (quando um número específico de *features* pode ser selecionado).

A estrutura do Modelo de Requisitos (*RequirementsModel*) é bem simples, como pode ser observado também na Figura 27. Este elemento também herda a estrutura do artefato, mas possui como adicional uma lista de requisitos (*RequirementsList*). Esta lista deve possuir ao menos um requisito (*Requirement*), que é definido pelo nome do requisito (*name*), um identificador (*id*), a descrição do requisito (*description*) e a indicação opcional do tipo do requisito (*type*), como por exemplo “Requisito Funcional”, “Requisito de Base”, e assim por diante.

O outro modelo XSD foi definido para descrever estruturas de dados compatíveis com o modelo fornecido pela RAS (*DefaultprofileXML.xsd*) (GROUP, 2006). Essas estruturas foram criadas para interagir com o repositório de ativos reusáveis. Este modelo possui os seguintes elementos:

- **ArtifactList:** esta estrutura serve como contêiner para uma lista de elementos *RAS Artifact*.
- **AssetList:** esta estrutura serve como contêiner para uma lista de elementos *RAS Asset*.
- **IDObjectList:** esta estrutura serve como contêiner para uma lista de elementos *IDReference*.
- **IDReference:** estrutura criada para ser utilizada em buscas nos repositório. Ela possui três atributos: identificador (*ID*), nome (*name*) e tipo (*type*). O tipo refere-se ao tipo do elemento que está sendo buscado.
- **ParamType:** estrutura utilizada para a definição de parâmetros utilizados nas buscas no repositório. Possui três atributos: o tipo do parâmetro (*type*) definido pelo elemento *ParamEnum*, o valor textual do parâmetro, e o critério (*criteria*) de busca definido pelo elemento *CriteriaEnum*.
- **SearchALL:** estrutura definida para modelar buscas com vários parâmetros. No caso específico da *SearchALL*, todos os parâmetros (*ParamType*) passados como condição (*condition*) devem ser verdadeiros. Esta estrutura também oferece suporte a subexpressões *SearchANY* entre as condições, mas deve-se observar que tanto as subexpressões quanto as outras condições devem ser verdadeiras para que o resultado seja retornado.
- **SearchANY:** análoga a *SearchALL*, porém, o resultado é retornado se pelo menos uma das condições for verdadeira.
- **SearchSingle:** possui somente uma condição de busca definida por *ParamType*.
- **Transaction:** estrutura composta somente com a requisição XML utilizada para consumir um serviço.
- **TransactionPack:** estrutura que agrupa um conjunto de transações (*Transaction*). O objetivo da *TransactionPack* é garantir a atomicidade quando várias chamadas de serviços forem necessárias. Assim, se a execução de pelo menos um serviço falhar, todo o pacote de transações deve ser revertido (*rollback*).
- **CriteriaEnum:** conjunto de valores que definem os critérios de busca: igual (*is*), diferente (*is not*), contém (*contains*), não contém (*not contains*), maior que (*greater then*) e menor que (*less then*).

- **ParamEnum:** conjunto de valores que definem os tipos de parâmetros que podem ser buscados: nome (*name*), descritor (*descriptor*), identificador (*id*), contexto (*context*), data (*date*), entre outros.

Foi necessário criar elementos contêiner porque o retorno da execução dos serviços deve ser um único XML (*XMLResponse*) com um único elemento raiz. Além disso, é importante ressaltar que todos os elementos do tipo *enumeration* podem ser facilmente customizados de acordo com o contexto e as necessidades de cada projeto.

Após definida a estrutura dos dados que será utilizada na definição dos serviços, a próxima seção apresenta o documento WSDL criado para descrever esses serviços e a descrição geral de cada tipo de serviço criado.

5.3 Definição dos Serviços

Visando atender aos requisitos estabelecidos na Seção 5.1, foram definidos 50 serviços. Esses serviços foram construídos somente para gerenciar a LPS, assim, necessitam interagir com outros serviços que realizam o acesso ao repositório. Esse repositório deve oferecer serviços para gerenciamento dos ativos (inserção, busca, recuperação, atualização e remoção) e deve ser capaz de importar e exportar esses ativos como pacotes/objetos RAS.

Os serviços foram descritos em um documento WSDL de acordo com as especificações definidas. O Apêndice A reúne a interface de todos os serviços criados divididos por contexto em figuras diferentes.

A Figura 34 (Apêndice A) apresenta todos os serviços que criam novos ativos no repositório. Esses serviços são responsáveis por manipular as entradas recebidas a partir do modelo apresentado e mapeá-las para sua representação em RAS. Esse mapeamento é realizado tanto para os elementos de processo quanto para os artefatos resultantes do processo.

Dentre esses serviços existem dois serviços diferentes para criar o artefato do modelo de *features*, o serviço número 46 (Figura 34 - Apêndice A), que importa um modelo já existente em qualquer formato, e o serviço número 11 (Figura 39 - Apêndice A), que importa um modelo já existente no formato definido pela Figura 27. A diferença entre eles é que o 46 somente empacota o modelo como um único artefato, enquanto o 11 desmembra cada *feature* do modelo com suas variabilidades e restrições como artefatos independentes que podem ser diretamente referenciados por outros elementos. Isto potencializa o reúso a nível de cada *feature* tornando possível, por exemplo, que a implementação dessa *feature* a referencie diretamente, bem como os requisitos relacionados a ela, assim, quando somente aquela funcionalidade for desejada em outros projetos, ela pode ser compartilhada de maneira independente do contexto do modelo como um todo.

Os serviços referenciados pelos números 12 e 13, em sua essência, também recebem

como parâmetro de entrada estruturas correspondentes ao elemento artefato (*Artifact*) definido no modelo da Figura 26. O objetivo desses serviços de acesso direto a elementos específicos do contexto de LPS é facilitar seu reconhecimento no processo de desenvolvimento, adicionando características particulares durante o mapeamento desses elementos para RAS. É importante ressaltar que seria equivalente preencher todas essas informações manualmente e invocar o serviço número 9, porém o trabalho é maior.

Quando durante o processo de desenvolvimento é necessário criar novas fases ou atividades, os serviços referenciados pelos números 5 e 7 podem ser invocados.

A execução de uma atividade durante o processo de desenvolvimento não envolve a chamada de serviços, uma vez que o processo foi recuperado, todos os seus elementos, incluindo fases e atividades são recuperados com ele. Durante a execução dessa atividade, os serviços de gerenciamento de ativos no repositório podem ser invocados, além dos serviços de gerenciamento da própria atividade, como o serviço 7 (Figura 34 - Apêndice A) que criaria novas subatividades, por exemplo, ou o serviço 19 (Figura 33 - Apêndice A) que atualizaria as informações da atividade em contexto.

A Tabela 4 apresenta a associação de todos os serviços envolvidos na implementação de cada requisito especificado na Seção 5.1.

Tabela 4 – Associação de Requisitos com Serviços Implementados

Requisito	**Resumo**	**Nº Serviço(s) correspondente(s)**
RQ1	Segurança	1
RQ2	Gerenciamento de modelo de processos	2, 4, 6, 8, 14, 16, 18, 20, 23, 24, 26, 28, 30
RQ3	Gerenciamento de Projetos	10, 22-23, 34
RQ4	Gerenciamento da instância de processos	3, 5, 7, 15, 17, 19, 23, 25, 27, 29
RQ5	Gerenciar elementos opcionais de processos	5, 7, 38
RQ6	Gerenciar ativos reusáveis	9, 11-13, 21, 23, 31-33, 35-36, 45-49
RQ7	Mapear ativos e elementos de processo para RAS	2-13, 46, 50
RQ8	Verificar dependências	37-40, 42-23
RQ9	Acesso direto aos elementos principais da LPS	11-13, 32-33, 46, 50
RQ10	Desmembrar modelo de *features* e de requisitos	11, 50

Pode ser verificado na tabela quais serviços estão envolvidos e podem ser invocados para cada funcionalidade no gerenciamento da LPS. Como descrito anteriormente, todos os serviços que envolvem a criação e o mapeamento de um novo ativo estão presentes na Figura 34 - Apêndice A. Quando a manipulação de um ativo já existente é necessária, primeiro deve-se localizar este ativo no repositório por meio de um dos serviços apresentados na Figura 36 -

Apêndice A e depois recuperá-lo utilizando o serviço correspondente disponível na Figura 37 - Apêndice A. Depois do ativo já recuperado e em contexto, são utilizados os serviços apresentados nas Figuras 33 - Apêndice A e 35 - Apêndice A, para atualização e remoção, respectivamente.

Também pode ser desejado transferir, ou mover, um ativo ou elemento de processo de uma hierarquia para outra, por exemplo, uma atividade esperada que foi definida dentro de uma fase esperada pode ser transferida para outra fase ou até mesmo se tornar uma subatividade contida em outra atividade. Esse serviço está disponível na Figura 39 - Apêndice A referenciado pelo número 45.

De acordo com o RQ8, ao se desejar remover ou mover um ativo (artefato ou elemento de processo) deve-se primeiro verificar se este não possui dependências. Esta verificação é realizada por meio dos serviços apresentados na Figura 38 - Apêndice A. O número de referência dos serviços pode ser verificado na linha RQ8 da Tabela 4.

Ao observar os serviços apresentados no Apêndice A, todos, com exceção do serviço referenciado pelo número 1, possuem um atributo chamado *tokenID*. Este atributo é passado como parâmetro ao invocar serviços de acesso ao repositório e utilizado para reconhecer e autenticar as permissões do usuário que está invocando o serviço. O valor deste atributo é fornecido pela própria aplicação responsável pelo repositório ao invocar o serviço número 1 (*login* - Figura 39 - Apêndice A). É importante ressaltar que se o repositório tiver outra forma de autenticar o usuário, este *token* pode ser removido ou substituído por outro mecanismo de autenticação.

A próxima seção apresenta alguns detalhes de como a implementação desses serviços foi realizada.

5.4 Implementação

Para a implementação dos serviços foi utilizado um gerador de código Java chamado *apache-cxf-3.0.2* (CXF) (FOUNDATION, 2016b). O CXF pode gerar as classes a partir dos modelos XSD e todas as interfaces dos serviços, tanto do lado cliente quanto do lado servidor, a partir do WSDL. Basta executar o comando:

```
C:\apache-cxf-3.0.2\bin>
wsdl2java -d output/SPLServices -all
http://localhost/airesweb/spl/splfunctions.wsdl
```

Este comando executa o gerador de WSDL para Java (*wsdl2java*) com os seguintes parâmetros: a opção *-d* denomina o diretório de saída *output/SPLService*; a opção *-all* denomina que todas as classes e interfaces devem ser geradas; e o caminho do arquivo WSDL

que será utilizado, no exemplo o WSDL está disponível em um servidor local no endereço *http://localhost/airesweb/spl/splfunctions.wsdl*.

Após a execução deste comando, todos os arquivos solicitados são gerados. No caso desta abordagem, foram geradas as classes correspondentes a cada documento XSD utilizado no arquivo WSDL: o XSD para interação com o repositório, chamado *common*; o XSD que descreve as estruturas referentes aos elementos das LPS, chamado *splfunctionsmodel*; e o XSD fornecido pela RAS (OMS) correspondente à estrutura do *DefaultProfile* chamado *org.omg.spec.ras._-20060101.defaultprofilexml*. A estrutura dos pacotes gerados pode ser observada na Figura 28.

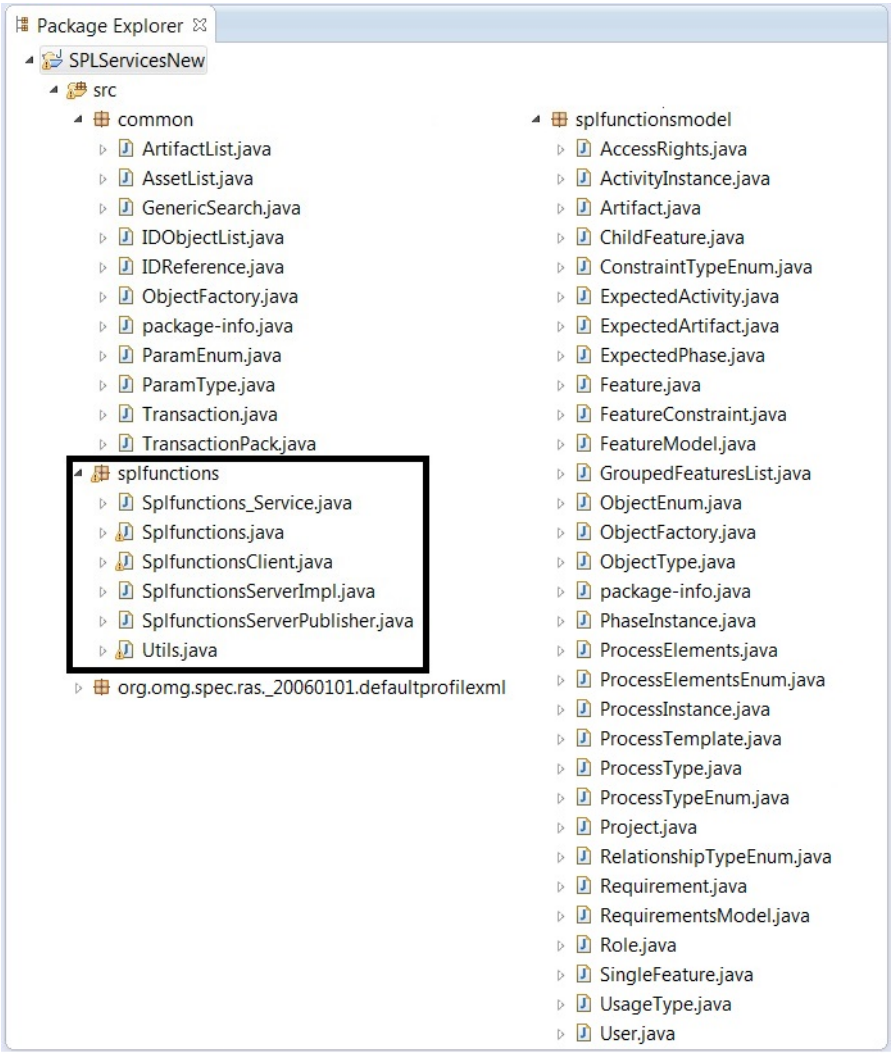


Figura 28 – Estrutura dos pacotes gerados pelo CXF a partir dos arquivos XSD e WSDL.

Na Figura 28 o pacote destacado, denominado *splfunctions*, corresponde às classes geradas com as interfaces dos serviços gerados, incluindo tanto as classes do lado do cliente quanto as do lado do servidor. Um exemplo de interface gerada é apresentado no Código-fonte 1

.

Código-fonte 1: Interface Gerada de Serviço

```
1 @WebResult (name = "activityID", targetNamespace = "", partName = "
    activityID")
2 @WebMethod (action = "http://127.0.0.1/splfunctions/
    newActivityInstance")
3 public String newActivityInstance(
4     @WebParam (partName = "activityInstanceXML", name = "
    ActivityInstance", targetNamespace = "http://splFunctionsModel")
5     ActivityInstance activityInstanceXML,
6     @WebParam (partName = "expectedActivityID", name = "
    expectedActivityID", targetNamespace = "")
7     String expectedActivityID,
8     @WebParam (partName = "phaseInstanceID", name = "phaseInstanceID"
    , targetNamespace = "")
9     String phaseInstanceID,
10    @WebParam (partName = "processInstanceID", name = "
    processInstanceID", targetNamespace = "")
11    String processInstanceID,
12    @WebParam (partName = "activityInstanceParentID", name = "
    activityInstanceParentID", targetNamespace = "")
13    String activityInstanceParentID,
14    @WebParam (partName = "tokenID", name = "tokenID",
    targetNamespace = "")
15    String tokenID
16 );
```

No exemplo do Código-fonte 1 a interface gerada corresponde ao serviço 7 (*newActivityInstance*) de criar uma nova instância de uma atividade. O parâmetro esperado de retorno é chamado *activityID* denominado pela anotação *@WebResult*. A anotação *@WebMethod* denota o endereço do serviço no servidor e as anotações *@WebParam* denotam os parâmetros que são esperados por essa interface.

Depois de todas as classes e interfaces geradas, a implementação das funcionalidades se deu no corpo dos métodos gerados. Além disso, no lado cliente foram implementados alguns *stubs* para testar a invocação dos serviços criados. Os *stubs* são métodos que retornam parâmetros esperados com valores simbólicos e sem nenhum processamento ou lógica de negócio. Depois de realizados os testes das chamadas dos serviços, os *stubs* foram substituídos por uma prova de conceito criada para testar as funcionalidades dos serviços. Esta prova de conceito é apresentada na próxima seção.

O Código-fonte 2 apresenta um exemplo de implementação de um serviço (Serviço número 13 da Figura 34 - Apêndice A) chamado *newArchitecture*, que insere um novo artefato de arquitetura da LPS no repositório. Este serviço recebe como parâmetros o identificador da instância de atividade sendo executada (*activityInstanceID*) onde esse artefato foi gerado; o

identificador da instância do processo sendo executado (*processInstanceID*), o identificador de autenticação e permissões do usuário (*tokenID*) e o objeto que se deseja inserir no repositório, que no caso é um artefato correspondente à arquitetura da LPS (*artifactArchitectureXML*). A saída esperada após a execução do serviço é o identificador do artefato inserido com sucesso no repositório (*id*).

Código-fonte 2: Exemplo de implementação de um serviço.

```
1  public String newArchitecture(Artifact artifactArchitectureXML,String
       activityInstanceID,String processInstanceID,String tokenID) {
2        try {
3          LOG.info("Executing operation newArchitecture");
4
5          //New Asset
6          org.omg.spec.ras._20060101.defaultprofilexml.Asset arch = new
       org.omg.spec.ras._20060101.defaultprofilexml.Asset();
7
8          //Insert DefaultProfile
9          Profile profile = Utils.AssembleRASDefaultProfile();
10         arch.setProfile(profile);
11
12         //Insert Classification
13         FreeFormDescriptor ffd = Utils.AssembleRASFreeFormDescriptor(
       "Architecture", "Architecture Model of SPL.");
14         DescriptorGroup dg = Utils.AssembleRASDescriptorGroup("
       ArtifactType", ffd);
15         Context ctx = Utils.AssembleRASContext("Core", "");
16
17         Classification classification = Utils.
       AssembleRASClassification(dg, ctx);
18         arch.setClassification(classification);
19
20         //Insert Solution/Artifact
21         Solution solution = Utils.AssembleRASSolution(
       artifactArchitectureXML);
22         arch.setSolution(solution);
23
24         //Insert Relations
25         RelatedAsset relAssetProcess = new RelatedAsset();
26         relAssetProcess.setName("ParentProcess");
27         relAssetProcess.setAssetId(processInstanceID);
28         relAssetProcess.setRelationshipType("parent");
29         arch.getRelatedAsset().add(relAssetProcess);
30
31         RelatedAsset relAssetActivity = new RelatedAsset();
32         relAssetActivity.setName("ParentActivity");
33         relAssetActivity.setAssetId(activityInstanceID);
```

```
34          relAssetActivity.setRelationshipType("parent");
35          arch.getRelatedAsset().add(relAssetActivity);
36
37          //Service to Insert the Asset on DB
38          String id = aires.newAsset(arch, tokenID);
39
40          //Insert Relations with Parents on DB
41          aires.appendRelatedAsset(artifactArchitectureXML.getName(),
      id, "", "aggregation", activityInstanceID, tokenID);
42
43          return id;
44        } catch (java.lang.Exception ex) {
45          ex.printStackTrace();
46          throw new RuntimeException(ex);
47        }
48      }
```

É pré-requisito para a utilização desses serviços que o repositório de ativos esteja em perfeito funcionamento e ofereça os serviços necessários para a manipulação dos ativos desse repositório. Assim, com o objetivo de testar a funcionalidade dos serviços implementados foram construídos *stubs* para simularem a resposta do repositório às chamadas dos serviços. A conexão com esse repositório foi denominada *aires* e sua utilização pode ser verificada nas linhas 38 e 41 do Código-fonte 2.

Definida a conexão com o repositório de ativos, a implementação do serviço focou em mapear o artefato de entrada para sua representação em RAS. Para isso, foi criada uma classe chamada *Utils.java* que pode ser verificada na região destacada da estrutura de pacotes na Figura 28. Essa classe oferece métodos auxiliares para facilitar o mapeamento dos objetos do modelo da LPS para a estrutura RAS. O Código-fonte 3 apresenta dois desses métodos criados para construir a estrutura *Solution* da RAS. A chamada desse método ocorre na linha 21 do Código-fonte 2.

Código-fonte 3: Métodos auxiliares para o mapeamento dos objetos do modelo XSD de LPS para o modelo da RAS

```
1 public static Solution AssembleRASSolution(Artifact artifact){
2     Solution sol = new Solution();
3
4     org.omg.spec.ras._20060101.defaultprofilexml.Artifact rasArtifact
      = Utils.AssembleRASArtifact(artifact);
5     sol.getArtifact().add(rasArtifact);
6
7     return sol;
8   }
9
```

```
10 public static org.omg.spec.ras._20060101.defaultprofilexml.Artifact
       AssembleRASArtifact(Artifact art){
11     org.omg.spec.ras._20060101.defaultprofilexml.Artifact response =
       new org.omg.spec.ras._20060101.defaultprofilexml.Artifact();
12
13     //Set Name
14     response.setName(art.getName());
15
16     //Set Object Type
17     response.setType(art.getFileType());
18
19     //Set Version
20     response.setVersion(art.getVersion());
21
22     //Set Access Rights
23     if(art.getAccessRights() != null) {
24       StringBuilder accessRights = new StringBuilder();
25       accessRights.append(art.getAccessRights().getOwner());
26       accessRights.append(art.getAccessRights().getGroup());
27       accessRights.append(art.getAccessRights().getAll());
28       response.setAccessRights(accessRights.toString());
29     }
30
31     //Set Artifact Context
32     if(!(art.getContext()).isEmpty()){
33       ArtifactContext context = new ArtifactContext();
34       context.setContext(art.getContext());
35       response.getArtifactContext().add(context);
36     }
37
38     //Set Variability Point
39     if((art.getVariabilityPoint()) != null){
40       VariabilityPoint vp = new VariabilityPoint();
41       vp.setName(art.getVariabilityPoint());
42     }
43
44     //Set Description
45     Description desc = new Description();
46     desc.setValue(art.getDescription());
47     response.setDescription(desc);
48
49     //Set Artifact Type
50     response.setType(art.getType());
51
52     //Set Reference
53     Reference ref = new Reference();
54     ReferenceKind refk = new ReferenceKind();
```

```
55      refk.setName("URI");
56      ref.setReferenceKind(refk);
57      ref.setValue(art.getUrl());
58      response.setReference(ref);
59
60      return response;
61    }
```

As linhas de comentários presentes nos dois códigos denotam as atividades sendo executadas em cada trecho e demonstram passo a passo como ocorre o mapeamento desses objetos LPS para RAS.

A próxima seção apresenta uma breve prova de conceito realizada para testar as funcionalidades implementadas pelos serviços desenvolvidos.

5.5 Prova de Conceito para a Utilização dos Serviços

Esta prova de conceito realizada para avaliar o funcionamento esperado dos serviços implementados foi constituída em algumas etapas. A Figura 29 mostra o modelo de objetos utilizados para construí-la.

Na primeira etapa, foi realizada a definição de um modelo simplificado de processo para LPS baseado no ESPLEP (SimpESPLEP) para guiar o desenvolvimento, conforme mostra a Figura 29. Este modelo de processo possui somente uma fase esperada (ModelagemRequisitos), uma atividade esperada (DefinirModFeatures) e um artefato esperado (ModeloFeatures).

Após o processo definido, o próximo passo foi a criação de um projeto LPS (ProjetoLPS) que gerou a instância (ProcessoLPS) do modelo de processo definido com seus respectivos elementos: a fase ModReqLPS e a atividade DefModFeatureLPS. Em seguida, foi executada a atividade de definição do modelo de *features* (CriarModFeatures) sendo realizada a inclusão de um artefato correspondente modelo de *features* (ModeloFeatureLPS). Este artefato foi modelado conforme a estrutura proposta na Figura 27.

Como o repositório é simulado, foram utilizados mecanismos do padrão de projeto *Singleton* para armazenar e acessar os dados na memória em tempo de execução. A instância correspondente ao *web service* do repositório simulado foi chamada *aires*, como mostra as linhas 38 e 41 do Código-fonte 2.

5.5.1 Processo LPS

Conforme as etapas descritas acima, antes da criação do projeto foi definido um modelo simples processo LPS baseado no ESPLEP com somente uma fase esperada correspondente a “Modelagem de Requisitos”, uma atividade esperada correspondente a “Definir Modelo de

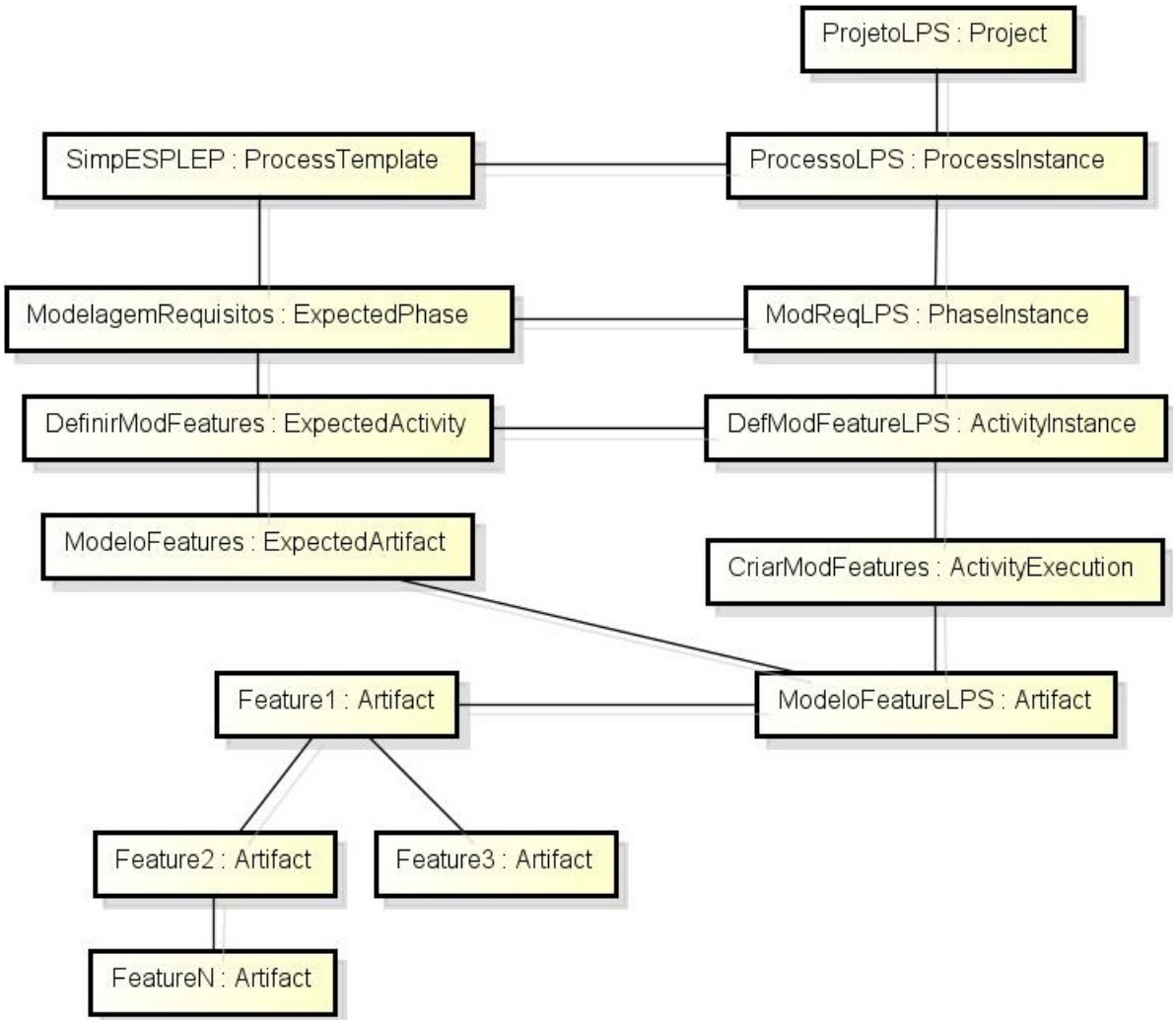


Figura 29 – Estrutura dos objetos criados para a prova de conceito.

Features" e um artefato esperado correspondente ao "Modelo de Features". Para isso foi utilizado o serviço referenciado pelo número 2 na Figura 34 - Apêndice A (*newProcessTemplate*). Para verificar se o processo foi corretamente inserido, o identificador *id*, retornado pela execução do serviço foi utilizado como parâmetro para a invocação do serviço número 24 da Figura 37 - Apêndice A (*retrieveProcessTemplate*).

5.5.2 Projeto LPS

Após verificado que os dados do processo foram armazenados corretamente, o serviço número 10 da Figura 34 - Apêndice A (*newProject*) foi chamado para criar um novo projeto LPS (ProjetoLPS - Figura 29). Ao invocar o serviço para criar um projeto, o identificador do modelo de processo inserido é passado como parâmetro e, durante a criação do projeto, o serviço número 3 da Figura 34 - Apêndice A (*newProcessTemplate*) é invocado para criar uma instância de processo baseada no modelo de processo referido.

A partir dos identificadores retornados pela invocação dos serviços 10 e 3, foram invoca-

dos os serviços 34 e 25 da Figura 37 - Apêndice A (*retrieveProject* e *retrieveProcessInstance*) para verificar se os dados do projeto e da instância do processo, respectivamente, foram corretamente inseridos.

5.5.3 Modelo de Features

Após verificar que o Projeto e a Instância do Processo foram inseridos corretamente, o próximo passo foi buscar a atividade para executá-la. Como o objeto completo correspondente à instância do processo já foi retornado pela invocação do serviço 25, a busca pelas fases e atividades é feita internamente por meio de código de execução local. Os dados da execução da atividade são armazenados no objeto CriarModFeatures.

O objeto correspondente ao modelo de *features* é então inserido invocando o serviço de número 11 da Figura 39 - Apêndice A (*newFeatureModel*). Ao invocar este serviço, os parâmetros referentes aos identificadores da atividade e da instância do processo são utilizados para atualizar esses objetos no repositório, adicionando os relacionamentos pertinentes à inclusão desse novo ativo criado. Essas atualizações ocorrem por meio da invocação interna dos serviços de número 15 e 19 da Figura 34 - Apêndice A (*updateProcessInstance* e *updateActivityInstance*).

A partir do identificador retornado pela invocação do serviço 11, o modelo inserido pode ser recuperado utilizando o serviço de número 31 da Figura 37 - Apêndice A (*retrieveProcessTemplate*) passando o identificador como parâmetro, ou utilizando o serviço de número 32 da mesma figura (*retrieveProcessTemplate*) passando como parâmetro o identificador do projeto, o identificador da instância do processo ou o identificador do próprio artefato, somente um dos parâmetros já é o suficiente para realizar a recuperação do modelo de *features*.

Com o objeto retornado foi possível verificar que as *features* foram desmembradas corretamente e armazenadas individualmente como artefatos dependentes, conforme o esperado.

5.5.4 Avaliação da utilização dos serviços

Apesar de somente esta prova de conceito ter sido ilustrada neste capítulo, uma prova para cada funcionalidade desenvolvida foi elaborada e executada. A partir da execução dessas provas foi possível verificar que todos os serviços testados responderam de forma satisfatória ao esperado. A variedade dos serviços disponibilizados foi suficiente para atender a todos os requisitos estabelecidos e garantir a funcionalidade esperada para apoiar o gerenciamento da LPS e dos sistemas derivados, além de apoiar o reúso por meio dos mapeamentos realizados para a representação dos ativos em conformidade com a especificação RAS.

5.6 Considerações Finais

Este capítulo apresentou uma abordagem baseada em serviços para apoiar o gerenciamento de LPS e potencializar o reúso dos ativos reusáveis. Os serviços foram implementados utilizando a linguagem Java a partir do gerador de código Java CXF. O CXF gerou tanto as classes correspondentes aos modelos XSD utilizados quanto as interfaces dos serviços descritos no arquivo WSDL. O protocolo de transmissão utilizado foi o SOAP. A implementação dos serviços foi feita de forma a mapear os ativos em conformidade com a especificação RAS e o acesso ao repositório foi testado de forma simulada. Foi ainda realizado uma breve prova de conceito para verificar a funcionalidade dos serviços implementados com resultados satisfatórios.

A partir dos resultados obtidos, pôde ser verificada a viabilidade da abordagem em cumprir os objetivos propostos por este trabalho, ou seja, oferecer suporte ao gerenciamento de LPS; motivar o reúso por meio da representação de ativos em RAS; e a extensão do reúso a nível de elementos de processos, *features* e requisitos individuais.

O próximo capítulo apresenta a modelagem de casos de uso e a proposta de uma ferramenta *web* para consumir os serviços apresentados neste capítulo. Esta ferramenta deve implementar uma interface gráfica e um conjunto de funcionalidades implementadas a partir da utilização desses serviços desenvolvidos.

CAPÍTULO

6

PROPOSTA DE FERRAMENTA

Conforme apresentado e discutido no capítulo anterior, neste trabalho foram definidas as funcionalidades de gerenciamento de LPS por meio de *web services*, para facilitar a compatibilidade e a integração dessas funcionalidades com outras ferramentas e ambientes. Neste capítulo é apresentada a modelagem de casos de uso e a proposta de desenvolvimento de uma ferramenta *web* que utiliza os serviços criados para gerenciar a criação e o desenvolvimento de uma LPS. A ferramenta não foi desenvolvida neste trabalho para deixar aberta ao usuário a opção de customizar a interface gráfica e implementar somente as funcionalidades relevantes ao contexto de seus projetos, ou até mesmo, somente consumir esses serviços em soluções já em uso em seu próprio ambiente de desenvolvimento.

6.1 Proposta de Desenvolvimento

O objetivo da ferramenta proposta é gerenciar os recursos reusáveis do ciclo de desenvolvimento de uma LPS, como os artefatos produzidos e os elementos de processo de forma a potencializar o reúso. Não é o foco da ferramenta gerar produtos, por exemplo na Engenharia de Aplicação, e nem modelar e gerenciar variabilidades do modelo de *features*, embora no capítulo anterior tenha sido apresentada uma sugestão de modelo e representação de variabilidades em *features*.

Esta ferramenta então possui foco no processo de desenvolvimento e no armazenamento adequado desses ativos reusáveis para que sejam facilmente encontrados e compartilhados entre grupos ou em repositórios *onlines* abertos. Assim, é pré-condição para sua construção e utilização que exista um repositório capaz de receber e interpretar arquivos no formato ".RAS" por meio de serviços. A modelagem estrutural desse repositório é irrelevante, desde que este possua serviços para importar e exportar ativos neste formato.

Além do repositório, também é necessário um ambiente adequado de desenvolvimento

para as fases do processo, sendo a ferramenta proposta útil somente para guiar o usuário por essas fases e empacotar esses ativos de forma a potencializar o reúso.

A Figura 30 apresenta uma sugestão de interface gráfica para esta ferramenta proposta. Conforme apresentado na figura, o usuário teria acesso rápido ao seu perfil, seus grupos e seus projetos, além de poder visualizar as últimas atividades relevantes a seus projetos e grupos, e eventuais mensagens.

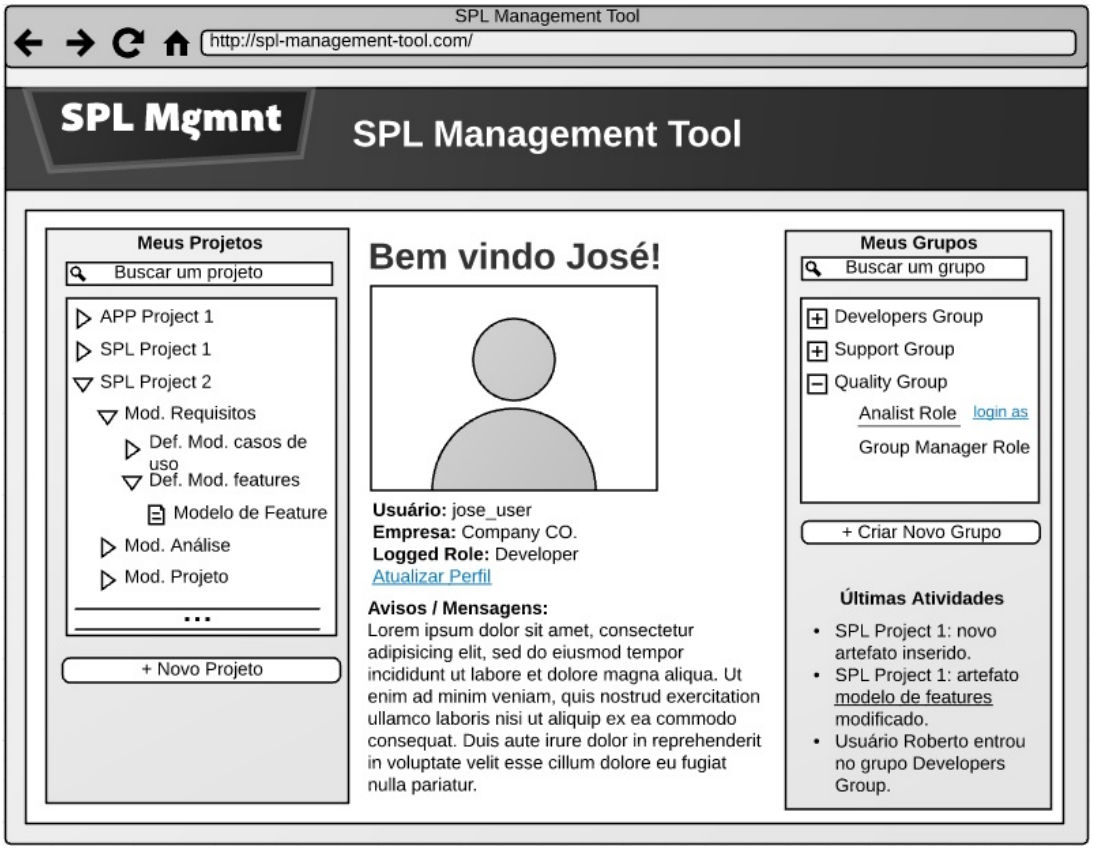


Figura 30 – Exemplo de "Página Inicial" para a implementação *web* da ferramenta.

Uma sugestão no desenvolvimento desta ferramenta é que suas funcionalidades sejam disponíveis de acordo com o perfil selecionado pelo usuário ao realizar o *login*, uma vez que um usuário pode pertencer a vários grupos e possuir diferentes perfis, de acordo com o contexto de cada projeto. Para isso, os perfis disponíveis para o usuário autenticado são apresentados no menu direito, conforme mostra a figura, para que a mudança do perfil em contexto seja facilitada. Esta é uma das sugestões de funcionalidade, na próxima seção são apresentadas sugestões mais completas na modelagem dos casos de uso.

A implementação dessa ferramenta está prevista para ser realizada em um trabalho que está sendo desenvolvido em paralelo a este. O trabalho em desenvolvimento propõe uma Arquitetura Integrada de Reúso de Software (AIRES) (BRAGA *et al.*, 2016) que irá oferecer um repositório online para o compartilhamento dos ativos reusáveis e também irá consumir os serviços desenvolvidos neste trabalho, além de oferecer muitas outras funcionalidades. Em

trabalhos anteriores do grupo de pesquisa da orientadora, uma ferramenta de apoio a reúso de processos de desenvolvimento e padrões foi construída, denominada Peônia (BRAGA; CHAN, 2008)(CHAN ALESSANDRA, 2007)(CHAN, 2008), que foi o passo inicial rumo à AIRES.

A seção seguinte apresenta a modelagem dos casos de uso para esta ferramenta proposta.

6.2 Modelagem dos Casos de Uso

Para ilustrar as funcionalidades da ferramenta proposta foram modelados 24 casos de uso. Como pode ser observado na Figura 31, o Diagrama de Casos de Uso apresentado possui cinco atores: Usuário Regular (ator base), Engenheiro, Engenheiro de Domínio, Engenheiro de Aplicação e Especialista em Processos. Cada ator representa um papel que o usuário pode assumir. Esses atores são uma representação possível, porém outros atores (papéis) podem ser definidos de acordo com o contexto de cada ambiente.

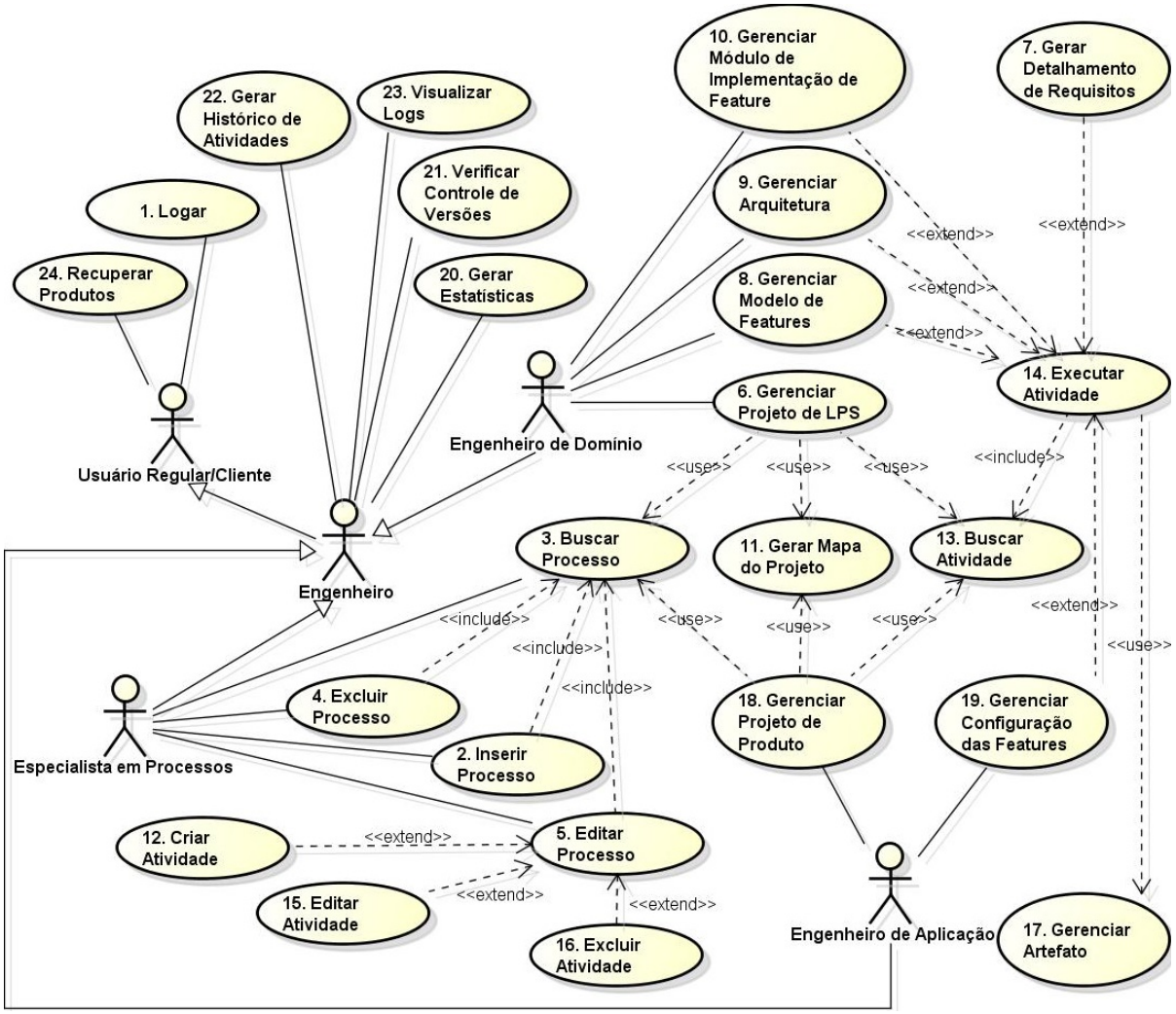


Figura 31 – Diagrama de Casos de Uso.

De acordo com os atores, os casos de uso foram divididos em funcionalidades relativas à

Engenharia de Domínio, à Engenharia de Aplicação, ao gerenciamento dos processos e outras sugestões de funcionalidades do sistema.

Os principais casos de uso relevantes ao contexto das LPS são os relativos às Engenharias de Domínio e de Aplicação. Esses casos de uso são discutidos mais detalhadamente nas próximas subseções, enquanto no Apêndice B pode ser encontrada tabela completa dos casos de uso ilustrados na Figura 31, com o número correspondente do caso de uso, seu nome, o ator, sua descrição e os serviços envolvidos em cada funcionalidade.

6.2.1 Funcionalidades que apoiam a Engenharia de Domínio

As funcionalidades que apoiam a Engenharia de Domínio são aquelas envolvidas na modelagem da LPS. Elas devem apoiar todo o ciclo de desenvolvimento da LPS, desde a criação do projeto, modelagem de *features* e requisitos, até a manutenção da LPS depois de pronta. As subseções seguintes apresentam algumas funcionalidades específicas da fase de Engenharia de Domínio de uma LPS.

6.2.1.1 Gerenciar Projeto LPS

O desenvolvimento de uma nova LPS sempre se inicia com a criação de um projeto e a instanciação de um processo base que será seguido por todo o ciclo de desenvolvimento dessa LPS. Uma das funcionalidades modeladas nos casos de uso é Gerenciar Processos, onde novos processos, tanto para LPS quanto para sistemas configurados, são estabelecidos e mantidos. Após os processos estarem corretamente configurados no repositório, é possível criar um novo projeto LPS.

A Figura 32 ilustra uma possível tela para a criação de um novo projeto LPS. No menu lateral esquerdo é possível verificar os passos a seguir e as informações a fornecer ao criar um novo projeto LPS. Como discutido, é obrigatório selecionar um processo ao se criar um novo projeto. A escolha do processo define se o projeto criado corresponde à Engenharia de Domínio ou à Engenharia de Aplicação.

A funcionalidade de Gerenciar Projeto também inclui buscar e executar as atividades definidas no processo e instanciadas no projeto. Caso necessário, novas fases e atividades podem ser criadas. Ao executar uma atividade, ocorre o Gerenciamento de ativos no repositório, seja armazenando um novo ativo criado, ou recuperando e atualizando ativos já existentes.

O caso de uso Gerenciar Projeto é o mais importante no contexto do desenvolvimento da LPS, pois é a partir dele que todas as fases e atividades são buscadas, recuperadas, executadas e atualizadas, ou seja, é por meio dele que todo o desenvolvimento da LPS é guiado.

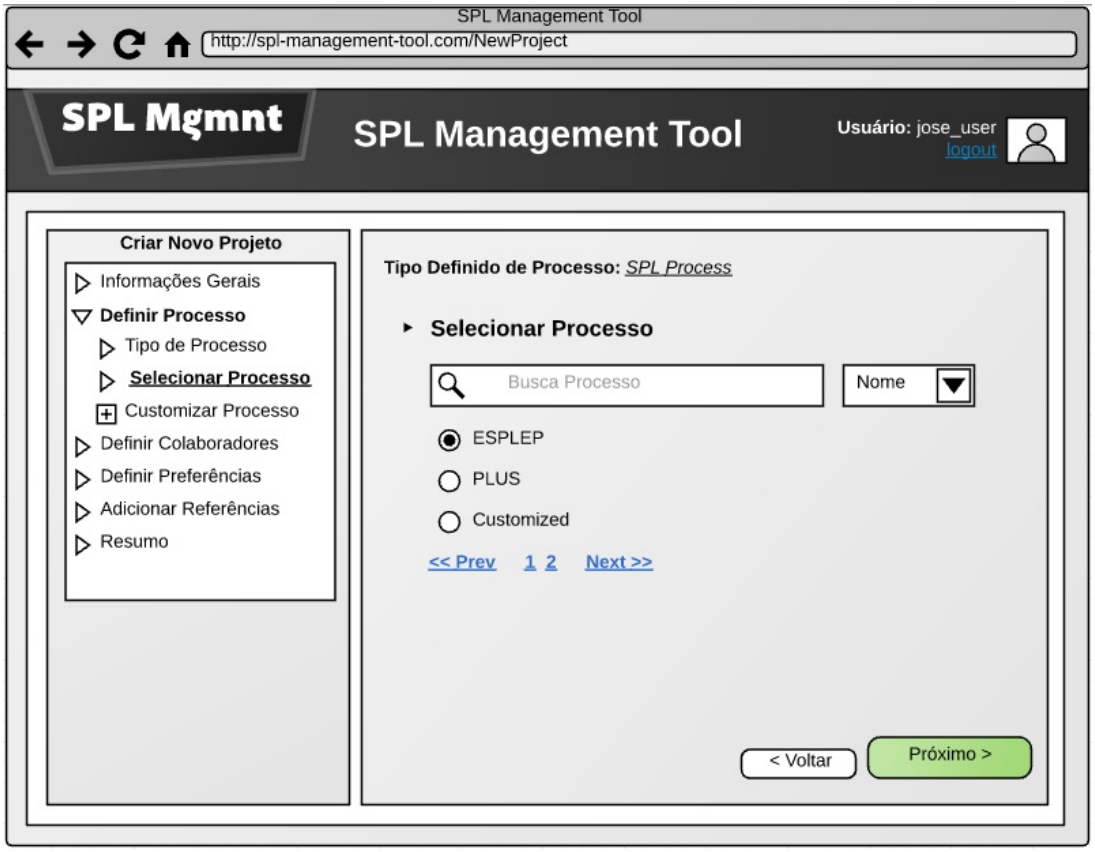


Figura 32 – Exemplo de uma das etapas para a criação de um novo projeto LPS - Selecionar Processo.

6.2.1.2 Gerenciar Modelo de Features

O modelo de *features* é considerado um dos mais importantes artefatos de uma LPS. De acordo com as funcionalidades oferecidas pelos serviços, é possível inserir um modelo de *features* no repositório de duas formas: importar um modelo genérico existente ou adicionar um modelo de acordo com o padrão recomendado neste trabalho. Caso o usuário escolha a primeira opção, todos os elementos que fizerem referência às features, irão referenciar o modelo como um todo, no máximo especificando por meio de descrições qual a *feature* alvo do relacionamento. Todavia, se o usuário optar pela segunda opção, ao armazenar o artefato do modelo de *features* no repositório, cada *feature* é desmembrada do modelo e armazenada de forma independente podendo ser referenciada diretamente por outros elementos, o que potencializa o reúso.

No caso de atualização e remoção de *features*, quando se trata do modelo único, se for necessário remover uma *feature*, então todo o modelo deve sofrer uma atualização para que essa *feature* seja removida. Isso pode gerar inconsistências caso a *feature* removida seja referenciada por outros elementos por meio de descritores textuais, pois seria difícil validar as dependências. No entanto se as *features* são armazenadas individualmente a verificação de dependências é facilmente realizada e somente aquela *feature* sofre alterações, e não todo o modelo.

6.2.1.3 Gerenciar Arquitetura

A arquitetura da LPS é tratada como um artefato estruturalmente comum. É um artefato obrigatório e sua presença é essencial no projeto, mas estruturalmente não tem nada em especial que o difere de outros artefatos.

6.2.1.4 Gerenciar Módulo de Implementação de Feature

A implementação das *features* deve estar obrigatoriamente relacionada, ou diretamente à *feature* implementada, ou ao modelo de *features* como um todo. A *feature* é somente um conceito que representa uma característica (funcionalidade) e a implementação dessa *feature* que realiza as funcionalidades esperadas.

O caso de uso de gerenciamento desse módulo de implementação compreende facilitar o relacionamento dessas implementações com as *features* implementadas, sendo esse relacionamento essencial para o reúso dessas *features*. Uma *feature* também pode possuir mais de uma implementação, assim, esse caso de uso objetiva auxiliar o usuário na criação e manutenção dessas implementações e seus relacionamentos com as *features*.

6.2.2 Funcionalidades que apoiam a Engenharia de Aplicação

6.2.2.1 Gerenciar Projeto de Produto

O projeto de produtos derivados de uma linha de produtos se assemelha muito a um projeto de desenvolvimento comum, com fases como modelagem de requisitos, análise e projeto. Sua maior diferença é que não possui uma fase de codificação, mas sim uma fase de configuração de *features*. A partir do modelo de configuração criado, todo o código do projeto é gerado de acordo com as *features* selecionadas e o módulo de implementação estabelecido para cada uma delas.

Para a criação de um novo projeto de um sistema alvo, ou projeto de produto, também é necessário selecionar um processo correspondente, de forma similar ao ilustrado na Figura 32. As funcionalidades desse caso de uso guiam o usuário de forma análoga ao caso de uso de Gerenciar Projeto LPS, porém possui uma fase específica de configuração do modelo de *features* e não possui as fases específicas de modelagem da LPS.

6.2.2.2 Gerenciar Configuração das Features

Conforme discutido acima, a configuração das *features* é a base para a geração do produto. O objetivo deste caso de uso é auxiliar o usuário a armazenar e manipular a configuração das *features* selecionadas de acordo com os requisitos estabelecidos. O modelo de configuração é armazenado como qualquer outro artefato comum, porém com as especificações claras de dependência de cada *feature* que está sendo referenciada do modelo de *features*.

6.3 Considerações Finais

Neste capítulo foi apresentada a proposta de implementação de uma ferramenta *web* que faz uso dos serviços desenvolvidos e apresentados no capítulo anterior. A ferramenta permite o gerenciamento de LPS e facilita o compartilhamento de ativos reusáveis.

O desenvolvimento dessa ferramenta auxilia usuários e desenvolvedores tanto a seguir corretamente um processo de desenvolvimento, aumentando a qualidade do produto, quanto a descrever e compartilhar artefatos. Se a ferramenta for combinada com um repositório público online, o potencial do reúso que pode ser atingido, pode ir muito além do contexto de LPS, podendo os artefatos armazenados serem reutilizados nos mais variados contextos na *world wide web*.

O próximo capítulo apresenta as conclusões deste trabalho e as projeções de trabalhos futuros.

CAPÍTULO

7

CONCLUSÃO

Este trabalho apresentou uma abordagem baseada em serviços para gerenciar Linhas de Produtos de Software e motivar o reúso. O trabalho foi motivado pela falta de trabalhos existentes na literatura que ofereçam esse gerenciamento de forma *online*, independentemente de plataforma, e que potencializem o reúso como foi a proposta deste trabalho. Essa falta de trabalhos foi evidenciada no Capítulo 3, onde foi apresentado um mapeamento sistemático completo e juntamente com a descrição de alguns trabalhos relacionados.

De forma a desenvolver a abordagem proposta, foi criado um mapeamento dos artefatos gerados pela LPS e também dos elementos de processo para uma estrutura comum proposta pela OMG chamada RAS. Este mapeamento foi apresentado no Capítulo 4 e permite a representação de todos os ativos reusáveis em conformidade com a especificação RAS. Com isso, o reúso é alavancado por se tratar de uma estrutura comum e bem documentada utilizada para representar ativos reusáveis.

Após definido o mapeamento para a representação comum desses ativos, foram desenvolvidos *web services* que implementam requisitos definidos para oferecer o suporte ao gerenciamento proposto e ao mesmo tempo motivar o reúso. A descrição detalhada do processo de desenvolvimento desses serviços foi apresentada no Capítulo 5.

Embora não tenha sido implementada uma ferramenta que ofereça todas as funcionalidades que poderiam ser utilizadas por meio da utilização dos serviços criados, o Capítulo 6 apresentou uma proposta de desenvolvimento de uma ferramenta *web* juntamente com um conjunto de casos de uso que representam essas funcionalidades que podem ser oferecidas consumindo os *web services* desenvolvidos.

Ao longo deste trabalho foram evidenciados alguns benefícios, ou contribuições, e algumas limitações da abordagem apresentada. Essas limitações por sua vez inspiram sugestões para trabalhos futuros. As próximas seções apresentam esses itens.

7.1 Contribuições

As contribuições da abordagem apresentada estão essencialmente no contexto de reúso. Mesmo quando se trata de Linhas de Produtos de Software, o principal conceito envolvido ainda é o Reúso de Software. O fato da abordagem ter sido desenvolvida baseada em serviços contribui para a interoperabilidade das funcionalidades oferecidas, uma vez que a utilização de *web services* é independente de linguagem.

O gerenciamento do desenvolvimento de uma LPS apoiado por um processo de software, também contribui para o aumento da qualidade de software, tendo em vista que a abordagem garante que atividades obrigatórias sejam executadas e artefatos obrigatórios sejam gerados.

Além disso, a contribuição na potencialização do reúso por meio da representação de ativos reusáveis utilizando uma especificação comum, excede o contexto de LPS e promove o reúso de ativos nos mais diferentes contextos. Assim, é possível destacar as principais contribuições deste trabalho:

- Extensão do reúso a nível de elementos de processo a partir do mapeamento desenvolvido;
- Disponibilização de um exemplo documentado de como utilizar a especificação RAS para representar ativos, especialmente os elementos de processo;
- Extensão do reúso além do contexto de LPS tanto para artefatos quanto para elementos de processo utilizando a RAS;
- Serviços independentes que podem ser consumidos individualmente de acordo com a necessidade e o contexto de cada projeto, além da facilidade de acesso por meio da *web*;
- Duas funcionalidades especiais oferecidas para estender o reúso a nível de *features* e requisitos individuais dos modelos de *features* e de requisitos, respectivamente;
- Incentivo à adoção da LPS por meio da apresentação de uma abordagem de gerenciamento do desenvolvimento apoiada por um processo;
- Potencial aumento da qualidade do software pelo fato da abordagem conduzir o usuário a seguir um processo de desenvolvimento; e
- Compartilhamento de processos e elementos de processos diminuindo o tempo de elaboração e instanciação de processo cada vez que um projeto novo é criado.

Além disso este trabalho resultou na publicação de dois artigos científicos, (PACINI; BRAGA, 2015a) e (PACINI; BRAGA, 2015b), o que demonstra a relevância e a originalidade do trabalho desenvolvido. Um terceiro artigo está sendo escrito sobre os serviços implementados e a proposta da ferramenta apresentada.

Outra contribuição deste trabalho é ser utilizado como parte de um ambiente integrado de reúso que está sendo desenvolvido pelo grupo de pesquisa da autora e da orientadora (BRAGA *et al.*, 2016).

7.2 Limitações

Alguma limitações foram observadas quanto à utilização dos serviços e da abordagem proposta. Dentre elas as principais são listadas a seguir:

- A abordagem não apoia o desenvolvimento efetivo da LPS, apenas o gerenciamento do desenvolvimento e dos ativos reusáveis de forma flexível e com facilidade de acesso;
- A abordagem não oferece um banco de dados ou repositório para armazenar os ativos reusáveis, sendo necessário indicar um repositório previamente existente e em funcionamento para a utilização dos serviços propostos;
- O repositório indicado deve oferecer serviços que sirvam para manipular os ativos do repositório e deve ser capaz de importar e exportar esses ativos utilizando a especificação RAS. O Capítulo 3 apresenta alguns repositórios onlines que foram preparados para importar e exportar ativos no formato RAS (ver Seção 3.2); e
- A abordagem não oferece uma interface com o usuário, embora ofereça várias sugestões de como implementar uma ferramenta *web* utilizando as funcionalidades oferecidas pelos serviços desenvolvidos. Assim, o usuário fica livre para implementar somente o que for necessário no contexto de cada projeto, não tendo necessidade de implementar funcionalidades que não serão úteis para o domínio da aplicação.

7.3 Trabalhos Futuros

As limitações observadas neste trabalho, inspiraram trabalhos futuros. Dentre eles se encontram:

- Implementar a ferramenta proposta no Capítulo 6 e realizar avaliações por meio de experimentos. Os resultados obtidos seriam avaliados aplicando métricas relativas à eficiência, eficácia, usabilidade, entre outras;
- Realizar a integração dessa ferramenta à arquitetura integrada de reúso (AIRES) já mencionada no capítulo anterior. Assim, a arquitetura forneceria o repositório e a ferramenta ofereceria o acesso aos serviços e a interface com o usuário;
- Pretende-se também disponibilizar tanto a ferramenta quanto os *web services* em nuvem, a fim de promover sua ampla utilização, facilitar o acesso e potencializar o reúso;

- Espera-se também aplicar conceitos de Computação Colaborativa aos ativos reusáveis do repositório, sendo possível que usuários comentem e classifiquem os ativos com o intuito de recomendá-los. Ainda é possível construir mecanismos de recomendação baseados nessas avaliações de usuários;
- É possível também melhorar o tratamento dos ativos de forma a garantir a segurança da informação, por meio de políticas de autenticação e autorização de acesso, criptografia, e assim por diante; e
- No futuro, é possível também estender o gerenciamento oferecido de forma a construir um ambiente completo de desenvolvimento para LPS e gerenciamento de variabilidades. Este ambiente seria uma solução completa, livre e gratuita disponibilizada para a comunidade científica.

REFERÊNCIAS

ACHER, M.; COLLET, P.; LAHIRE, P.; FRANCE, R. Managing variability in workflow with feature model composition operators. In: SPRINGER. **Software Composition**. [S.l.], 2010. p. 17–33. Citado 2 vezes nas páginas 56 e 57.

ACHER, M.; COLLET, P.; LAHIRE, P.; FRANCE, R. B. A domain-specific language for managing feature models. In: ACM. **Proceedings of the 2011 ACM Symposium on Applied Computing**. [S.l.], 2011. p. 1333–1340. Citado 3 vezes nas páginas 55, 56 e 57.

____. Familiar: A domain-specific language for large scale management of feature models. **Science of Computer Programming**, Elsevier, v. 78, n. 6, p. 657–681, 2013. Citado na página 56.

AJILA, S. A.; KABA, A. B. Evolution support mechanisms for software product line process. **Journal of Systems and Software**, Elsevier, v. 81, n. 10, p. 1784–1801, 2008. Citado 2 vezes nas páginas 56 e 57.

ALVES, V.; CALHEIROS, F.; NEPOMUCENO, V.; MENEZES, A.; SOARES, S.; BORBA, P. FLiP: managing software product line extraction and reaction with aspects. In: **Software Product Line Conference, 2008. SPLC '08. 12th International**. [S.l.: s.n.], 2008. p. 354–354. Citado 2 vezes nas páginas 53 e 56.

BACHMANN, F.; NORTHROP, L. Structured variation management in software product lines. In: IEEE. **System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on**. [S.l.], 2009. p. 1–7. Citado 2 vezes nas páginas 56 e 57.

BARREIROS, J.; MOREIRA, A. Managing features and aspect interactions in software product lines. In: IEEE. **Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on**. [S.l.], 2009. p. 506–511. Citado 2 vezes nas páginas 56 e 57.

BEUCHE, D. Modeling and building software product lines with pure::variants. In: **Software Product Line Conference, 2008. SPLC '08. 12th International**. [S.l.: s.n.], 2008. p. 358–358. Citado na página 54.

____. Modeling and building software product lines with pure:: variants. In: ACM. **Proceedings of the 16th International Software Product Line Conference-Volume 2**. [S.l.], 2012. p. 255–255. Citado 2 vezes nas páginas 28 e 57.

BEUCHE, D.; PAPAJEWSKI, H.; SCHRÖDER-PREIKSCHAT, W. Variability management with feature models. **Science of Computer Programming**, Elsevier, v. 53, n. 3, p. 333–352, 2004. Citado 2 vezes nas páginas 56 e 57.

BIRRELL, A. D.; NELSON, B. J. Implementing remote procedure calls. **ACM Trans. Comput. Syst.**, ACM, New York, NY, USA, v. 2, n. 1, p. 39–59, fev. 1984. ISSN 0734-2071. Disponível em: <http://doi.acm.org/10.1145/2080.357392>. Citado na página 38.

BOOTH HUGO HAAS, F. M. E. N. M. C. C. F. D. O. D. **Web Services Architecture**. 2004. Disponível em: <https://www.w3.org/TR/ws-arch/>. Acesso em: 02/02/2016. Citado 2 vezes nas páginas 38 e 39.

BOTTERWECK, G.; THIEL, S.; NESTOR, D.; ABID, S. bin; CAWLEY, C. Visual tool support for configuring and understanding software product lines. In: IEEE. **Software Product Line Conference, 2008. SPLC'08. 12th International**. [S.l.], 2008. p. 77–86. Citado 2 vezes nas páginas 56 e 57.

BRAGA, R. T.; CHAN, A. Peony: A web environment to support pattern-based development. In: **Web Engineering, 2008. ICWE '08. Eighth International Conference on**. [S.l.: s.n.], 2008. p. 358–361. Citado na página 95.

BRAGA, R. T. V.; FELONI, D.; PACINI, K.; FILHO, D. S.; GOTTARDI, T. Aires: an architecture to improve software reuse. In: **artigo submetido a evento internacional, em análise**. [S.l.: s.n.], 2016. Citado 2 vezes nas páginas 94 e 103.

CAVALCANTI, Y. C.; MACHADO, I. do C.; MOTA, P. A. da; NETO, S.; LOBATO, L. L.; ALMEIDA, E. S. de; MEIRA, S. R. de L. Towards metamodel support for variability and traceability in software product lines. In: ACM. **Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems**. [S.l.], 2011. p. 49–57. Citado na página 56.

CHAN, A. **Peônia: um ambiente web para apoiar processos de desenvolvimento com utilização de padrões de software e requisitos de teste no projeto de aplicações**. Dissertação (Mestrado), 2008. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-18062008-095855>. Citado na página 95.

CHAN ALESSANDRA, C. M. I. M. J. C. B. R. T. V. An environment to support the use of software patterns and test requirements during application development (in portuguese). In: **6th Latin American Conference on Pattern Languages of Programming**. [S.l.: s.n.], 2007. p. 245 – 260. Citado na página 95.

CHINNICI JEAN-JACQUES MOREAU, A. R. S. W. R. **Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language**. 2007. Disponível em: <https://www.w3.org/TR/wsdl20/>. Acesso em: 02/02/2016. Citado na página 38.

CIRILO, E.; KULESZA, U.; LUCENA, C. J. P. de. A product derivation tool based on model-driven techniques and annotations. **J. UCS**, v. 14, n. 8, p. 1344–1367, 2008. Citado na página 28.

CIRILO, E.; NUNES, I.; KULESZA, U.; LUCENA, C. Automating the product derivation process of multi-agent systems product lines. **Journal of Systems and Software**, v. 85, n. 2, p. 258 – 276, 2012. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121211001075>. Citado 3 vezes nas páginas 53, 56 e 57.

CLEMENTS, P.; NORTHROP, L. **Software Product Lines: Practices and Patterns**. Addison Wesley Professional, 2002. ISBN 9780201703320. Disponível em: <http://books.google.com.br/books?id=tHGFQgAACAAJ>. Citado 7 vezes nas páginas 15, 27, 28, 32, 33, 34 e 63.

CURBERA, F.; DUFTLER, M.; KHALAF, R.; NAGY, W.; MUKHI, N.; WEERAWARANA, S. Unraveling the web services web: An introduction to soap, wsdl, and uddi. **IEEE Internet Computing**, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 6, n. 2, p. 86–93,

mar. 2002. ISSN 1089-7801. Disponível em: <http://dx.doi.org/10.1109/4236.991449>. Citado na página 39.

DEHLINGER, J.; LUTZ, R. R. Supporting requirements reuse in multi-agent system product line design and evolution. In: IEEE. **Software Maintenance, 2008. ICSM 2008. IEEE International Conference on**. [S.l.], 2008. p. 207–216. Citado na página 55.

DHUNGANA, D.; GRÜNBACHER, P.; RABISER, R.; NEUMAYER, T. Structuring the modeling space and supporting evolution in software product line engineering. **Journal of Systems and Software**, Elsevier, v. 83, n. 7, p. 1108–1122, 2010. Citado 2 vezes nas páginas 55 e 56.

EL-SHARKAWY, S.; KRöHER, C.; SCHMID, K. Support for complex product line populations. In: **Proceedings of the 15th International Software Product Line Conference, Volume 2**. New York, NY, USA: ACM, 2011. (SPLC '11), p. 47:1–47:1. ISBN 978-1-4503-0789-5. Disponível em: <http://doi.acm.org/10.1145/2019136.2019191>. Acesso em: 02/02/2016. Citado 2 vezes nas páginas 54 e 57.

____. Supporting heterogeneous compositional multi software product lines. In: **Proceedings of the 15th International Software Product Line Conference, Volume 2**. New York, NY, USA: ACM, 2011. (SPLC '11), p. 25:1–25:4. ISBN 978-1-4503-0789-5. Disponível em: <http://doi.acm.org/10.1145/2019136.2019164>. Acesso em: 02/02/2016. Citado na página 54.

ELGEDAWY, I.; RAMASWAMY, L. Rapid identification approach for reusable soa assets using component business maps. In: IEEE. **IEEE International Conference on Web Services (ICWS)**. [S.l.], 2009. p. 599–606. Citado na página 59.

EZRAN, M.; MORISIO, M.; TULLY, C. **Practical software reuse**. [S.l.]: Springer, 2002. Citado na página 27.

FIELDING, R. T.; TAYLOR, R. N. Principled design of the modern web architecture. **ACM Trans. Internet Technol.**, ACM, New York, NY, USA, v. 2, n. 2, p. 115–150, maio 2002. ISSN 1533-5399. Disponível em: <http://doi.acm.org/10.1145/514183.514185>. Citado na página 39.

FILHO, S. M.; MARIANO, H.; KULESZA, U.; BATISTA, T. Automating software product line development: A repository-based approach. In: . [s.n.], 2010. p. 141–144. ISBN 9780769541709. Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-78449273654&partnerID=40&md5=d23375c54254dd7a8a1f1c92b42c7932>. Citado 3 vezes nas páginas 53, 56 e 60.

FLORES, R.; KRUEGER, C.; CLEMENTS, P. Mega-scale product line engineering at general motors. In: ACM. **Proceedings of the 16th International Software Product Line Conference-Volume 1**. [S.l.], 2012. p. 259–268. Citado na página 28.

FOUNDATION, E. **Eclipse Modeling Helios**. 2016. Disponível em: <https://eclipse.org/ide/>. Acesso em: 02/02/2016. Citado na página 77.

FOUNDATION, T. A. S. **CXF 3.0.2 Release Notes**. 2016. Disponível em: <https://cxf.apache.org/cxf-302-release-notes.html>. Acesso em: 02/02/2016. Citado na página 83.

GARCÍA-BORGOÑON, L.; BARCELONA, M.; GARCÍA-GARCÍA, J.; ALBA, M.; ESCALONA, M. J. Software process modeling languages: A systematic literature review. **Information and Software Technology**, Elsevier, v. 56, n. 2, p. 103–116, 2014. Citado na página 64.

GARG, A.; CRITCHLOW, M.; CHEN, P.; WESTHUIZEN, C. Van der; HOEK, A. Van der. An environment for managing evolving product line architectures. In: IEEE. **Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on**. [S.l.], 2003. p. 358–367. Citado na página 56.

GOMAA, H. Designing software product lines with uml 2.0: From use cases to pattern-based software architectures. In: **Reuse of Off-the-Shelf Components**. [S.l.]: Springer, 2006. p. 440–440. Citado 6 vezes nas páginas 15, 28, 33, 35, 36 e 63.

GOMAA, H.; SHIN, M. Automated software product line engineering and product derivation. In: . [s.n.], 2007. ISBN 0769527558; 9780769527550. ISSN 15301605. Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-39749091282&partnerID=40&md5=34bd6a13ca7198265f69a098d6a0a7e0>. Citado 2 vezes nas páginas 52 e 56.

GÓMEZ, A.; RAMOS, I. **Automatic Tool Support for Cardinality-Based Feature Modeling with Model Constraints for Information Systems Development**. [S.l.]: Springer, 2011. Citado na página 56.

GROUP, O. M. **Reusable Asset Specification**. 2005. (Version 2.2). Disponível em: <http://www.omg.org/spec/RAS/2.2/>. Acesso em: 02/02/2016. Citado 10 vezes nas páginas 15, 16, 32, 35, 37, 38, 48, 61, 63 e 123.

____. **RAS Default Profile XML**. 2006. (Version 2.2). Disponível em: <http://www.omg.org/spec/RAS/20060101/DefaultprofileXML.xsd>. Acesso em: 02/02/2016. Citado na página 80.

____. **Software & Systems Process Engineering Metamodel Specification**. 2008. (Version 2.0). Disponível em: <http://www.omg.org/spec/SPEM/2.0/>. Acesso em: 02/02/2016. Citado na página 64.

____. **Unified Modeling Language**. 2011. (Version 2.4.1). Disponível em: <http://www.omg.org/spec/UML/2.4.1/>. Acesso em: 02/02/2016. Citado na página 48.

____. **Common Object Request Broker Architecture (CORBA)**. 2012. Disponível em: <http://www.omg.org/spec/CORBA/3.3/>. Acesso em: 02/02/2016. Citado na página 38.

____. **Common Variability Language**. 2012. Disponível em: <http://www.omgwiki.org/variability/lib/exe/fetch.php?id=start&cache=cache&media=cvl-revised-submission.pdf>. Acesso em: 02/02/2016. Citado 2 vezes nas páginas 48 e 61.

HOLL, G.; VIERHAUSER, M.; HEIDER, W.; GRÜNBACHER, P.; RABISER, R. Product line bundles for tool support in multi product lines. In: ACM. **Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems**. [S.l.], 2011. p. 21–27. Citado 2 vezes nas páginas 56 e 57.

HONG-MIN, R.; ZHI-YING, Y.; JING-ZHOU, Z. Design and implementation of ras-based open source software repository. In: IEEE. **6th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)**. [S.l.], 2009. v. 2, p. 219–223. Citado 2 vezes nas páginas 60 e 63.

JUNIOR, E. A. de O.; GIMENES, I.; HUZITA, E. H. M.; MALDONADO, J. C. A variability management process for software product lines. In: IBM PRESS. **Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research**. [S.l.], 2005. p. 225–241. Citado na página 57.

JUNIOR, E. A. O.; GIMENES, I. M.; MALDONADO, J. C. Systematic management of variability in uml-based software product lines. **Journal of Universal Computer Science**, v. 16, n. 17, p. 2374–2393, 2010. Citado na página 56.

KHOSHNEVIS, S. An approach to variability management in service-oriented product lines. In: **2012 34th International Conference on Software Engineering (ICSE)**. [S.l.: s.n.], 2012. p. 1483–1486. Citado 4 vezes nas páginas 52, 56, 57 e 61.

KITCHENHAM, B. Procedures for performing systematic reviews. **Keele, UK, Keele University**, v. 33, p. 2004, 2004. Citado 4 vezes nas páginas 41, 42, 58 e 60.

KRUEGER, C. New methods in software product line development. In: **Software Product Line Conference, 2006 10th International**. [S.l.: s.n.], 2006. p. 95–99. Citado 3 vezes nas páginas 54, 56 e 57.

____. The BigLever software gears unified software product line engineering framework. In: **Software Product Line Conference, 2008. SPLC '08. 12th International**. [S.l.: s.n.], 2008. p. 353–353. Citado 2 vezes nas páginas 54 e 57.

LAGUNA, M.; MARQUéS, J. Uml support for designing software product lines: The package merge mechanism. **Journal of Universal Computer Science**, v. 16, n. 17, p. 2313–2332, 2010. ISSN 0958695X. Cited By (since 1996)2. Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-78650294973&partnerID=40&md5=10049679947c53bf4832416b3646fedf>. Citado 3 vezes nas páginas 55, 56 e 58.

LISBOA, L. B.; GARCIA, V. C.; LUCRÉDIO, D.; ALMEIDA, E. S. de; MEIRA, S. R. de L.; FORTES, R. P. de M. A systematic review of domain analysis tools. **Information and Software Technology**, Elsevier, v. 52, n. 1, p. 1–13, 2010. Citado na página 58.

MENDONCA, M.; BRANCO, M.; COWAN, D. Splot: software product lines online tools. In: ACM. **Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications**. [S.l.], 2009. p. 761–762. Citado 2 vezes nas páginas 56 e 57.

MICROSOFT. **Distributed Component Object Model (DCOM) Remote Protocol Specification**. 2007. Disponível em: <https://msdn.microsoft.com/library/cc201989.aspx>. Acesso em: 02/02/2016. Citado na página 38.

MITRA, Y. L. N. **SOAP Version 1.2 Part 0: Primer (Second Edition)**. 2007. Disponível em: <https://www.w3.org/TR/soap12-part0/>. Acesso em: 02/02/2016. Citado na página 39.

MOON, M.; CHAE, H. S.; NAM, T.; YEOM, K. A metamodeling approach to tracing variability between requirements and architecture in software product lines. In: IEEE. **7th IEEE International Conference on Computer and Information Technology (CIT)**. [S.l.], 2007. p. 927–933. Citado 2 vezes nas páginas 59 e 63.

MORENO-RIVERA, J. b.; NAVARRO, E.; CUESTA, C. Evolving kobra to support spl for webgis development. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 7046 LNCS, p. 622–631, 2011. ISSN 03029743. Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-81255163086&partnerID=40&md5=725be3d143063297ecdc727ed6499cf6>. Citado 4 vezes nas páginas 49, 53, 56 e 57.

MOURA, D. d. S. **Software Profile RAS: extending RAS and building an asset repository**. Dissertação (Mestrado), 2013. Disponível em: <http://www.lume.ufrgs.br/handle/10183/87582>. Citado 2 vezes nas páginas 60 e 63.

MUTHIG, D.; SCHROETER, J. A framework for role-based feature management in software product line organizations. In: ACM. **Proceedings of the 17th International Software Product Line Conference**. [S.l.], 2013. p. 178–187. Citado 2 vezes nas páginas 56 e 57.

NGUYEN, T.; COLMAN, A.; HAN, J. Modeling and managing variability in process-based service compositions. In: **Service-Oriented Computing**. [S.l.]: Springer, 2011. p. 404–420. Citado na página 56.

O'BRIEN, L.; HANSEN, F.; SEACORD, R.; SMITH, D. Mining and managing software assets. In: IEEE. **Software Technology and Engineering Practice, 2002. STEP 2002. Proceedings. 10th International Workshop on**. [S.l.], 2002. p. 82–90. Citado na página 56.

ORACLE. **Java Remote Method Invocation API (Java RMI)**. 2016. Disponível em: <http://docs.oracle.com/javase/7/docs/technotes/guides/rmi/>. Acesso em: 02/02/2016. Citado na página 38.

PACINI, K. D. R.; BRAGA, R. T. V. An approach for reusing software process elements based on reusable asset specification: a software product line case study. In: IEEE. **Proceedings of The Tenth International Conference on Software Engineering Advances (ICSEA)**. Barcelona, Spain: IARIA XPS Press, 2015. p. 200–206. Citado 2 vezes nas páginas 63 e 102.

____. Supporting tools for managing software product lines: a systematic mapping. In: IEEE. **Proceedings of The Tenth International Conference on Software Engineering Advances (ICSEA)**. Barcelona, Spain: IARIA XPS Press, 2015. p. 470–476. Citado 2 vezes nas páginas 41 e 102.

PAUTASSO, C.; ZIMMERMANN, O.; LEYMANN, F. Restful web services vs. "big"' web services: Making the right architectural decision. In: **Proceedings of the 17th International Conference on World Wide Web**. New York, NY, USA: ACM, 2008. (WWW '08), p. 805–814. ISBN 978-1-60558-085-2. Disponível em: <http://doi.acm.org/10.1145/1367497.1367606>. Citado na página 39.

PEROVICH, D.; ROSSEL, P.; BASTARRICA, M. Feature model to product architectures: Applying MDE to software product lines. In: **Joint Working IEEE/IFIP Conference on Software Architecture, 2009 European Conference on Software Architecture. WICSA/ECSA 2009**. [S.l.: s.n.], 2009. p. 201–210. Citado 4 vezes nas páginas 53, 56, 57 e 58.

PETERSEN, K.; FELDT, R.; MUJTABA, S.; MATTSSON, M. Systematic mapping studies in software engineering. In: SN. **12th international conference on evaluation and assessment in software engineering**. [S.l.], 2008. v. 17, n. 1, p. 1–10. Citado na página 41.

RABISER, R.; DHUNGANA, D.; GRUNBACHER, P.; LEHNER, K.; FEDERSPIEL, C. Involving non-technicians in product derivation and requirements engineering: A tool suite for product line engineering. In: **Requirements Engineering Conference, 2007. RE '07. 15th IEEE International**. [S.l.: s.n.], 2007. p. 367–368. Citado 3 vezes nas páginas 54, 55 e 56.

ROMERO, D.; URLI, S.; QUINTON, C.; BLAY-FORNARINO, M.; COLLET, P.; DUCHIEN, L.; MOSSER, S. Splemma: a generic framework for controlled-evolution of software product

lines. In: ACM. **Proceedings of the 17th International Software Product Line Conference co-located workshops**. [S.l.], 2013. p. 59–66. Citado 2 vezes nas páginas 56 e 57.

ROUILLÉ, E.; COMBEMALE, B.; BARAIS, O.; TOUZET, D.; JÉZÉQUEL, J.-M. *et al.* Leveraging cvl to manage variability in software process lines. In: **Asia-Pacific Software Engineering Conference**. [S.l.: s.n.], 2012. Citado 2 vezes nas páginas 56 e 57.

SARABURA, M.; BOWDEN, P. Solving requirements management challenges in product line development. In: **SPLC**. [S.l.: s.n.], 2008. p. 352. Citado na página 57.

SCHUBANZ, M.; PLEUSS, A.; BOTTERWECK, G.; LEWERENTZ, C. Modeling rationale over time to support product line evolution planning. In: ACM. **Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems**. [S.l.], 2012. p. 193–199. Citado na página 56.

SELLIER, D.; MANNION, M.; MANSELL, J. X. Managing requirements inter-dependency for software product line derivation. **Requirements engineering**, Springer, v. 13, n. 4, p. 299–313, 2008. Citado 2 vezes nas páginas 56 e 57.

SINNEMA, M.; DEELSTRA, S.; HOEKSTRA, P. The covamof derivation process. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 4039 LNCS, p. 101–114, 2006. ISSN 03029743. Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-33746211443&partnerID=40&md5=ff9738ec0d4d20998f54b7060f83256f>. Citado 5 vezes nas páginas 49, 54, 55, 56 e 57.

SOMMERVILLE, I.; MELNIKOFF, S.; ARAKAKI, R.; BARBOSA, E. de A. **Engenharia de software**. ADDISON WESLEY BRA, 2008. ISBN 9788588639287. Disponível em: <https://books.google.com.br/books?id=ifIYOgAACAAJ>. Citado 3 vezes nas páginas 27, 31 e 32.

SUCCI, G.; YIP, J.; LIU, E.; PEDRYCZ, W. Holmes: a system to support software product lines. In: **Proceedings of the 2000 International Conference on Software Engineering, 2000**. [S.l.: s.n.], 2000. p. 786–. Citado 2 vezes nas páginas 53 e 57.

TANHAEI, M.; MOAVEN, S.; HABIBI, J. Toward an architecture-based method for selecting composer components to make software product line. In: **2010 Seventh International Conference on Information Technology: New Generations (ITNG)**. [S.l.: s.n.], 2010. p. 1233–1236. Citado 2 vezes nas páginas 55 e 57.

THAO, C. Managing evolution of software product line. In: IEEE. **Software Engineering (ICSE), 2012 34th International Conference on**. [S.l.], 2012. p. 1619–1621. Citado na página 56.

VISION, I. C. **Astah Professional**. 2016. Disponível em: <http://astah.net/editions/professional>. Acesso em: 02/02/2016. Citado na página 34.

ZHANG, H.; JARZABEK, S. An xvcl approach to handling variants: A kwic product line example. In: IEEE. **Software Engineering Conference, 2003. Tenth Asia-Pacific**. [S.l.], 2003. p. 116–125. Citado 2 vezes nas páginas 56 e 57.

ZHENG, L.; ZHANG, C.; WU, Z.; LIU, M. Managing resource repository of a software product line with feature model. In: IEEE. **Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on**. [S.l.], 2009. p. 1–4. Citado na página 56.

ZHOU, N.; ZHANG, L.-J.; CHEE, Y.-M.; CHEN, L. Legacy asset analysis and integration in model-driven soa solution. In: IEEE. **IEEE International Conference on Services Computing (SCC)**. [S.l.], 2010. p. 554–561. Citado na página 60.

APÊNDICE

A

RELAÇÃO COMPLETA DE SERVIÇOS IMPLEMENTADOS PARA APOIAR O GERENCIAMENTO DE LPS

14.updateProcessTemplate		
input	processTemplateObj	ProcessTemplate
	processTemplateID	string
	tokenID	string
output	ok	boolean
15.updateProcessInstance		
input	processInstanceObj	ProcessInstance
	processInstanceID	string
	tokenID	string
output	ok	boolean
16.updateExpectedPhase		
input	expectedPhaseObj	ExpectedPhase
	expectedPhaseID	string
	mandatory	string
	tokenID	string
output	ok	boolean
17.updatePhaseInstance		
input	phaseInstanceObj	PhaseInstance
	phaseInstanceID	string
	tokenID	string
output	ok	boolean
18.updateExpectedActivity		
input	expectedActivityObj	ExpectedActivity
	expectedActivityID	string
	mandatory	string
	tokenID	string
output	ok	boolean

19.updateActivityInstance		
input	activityInstanceObj	ActivityInstance
	activityInstanceID	string
	tokenID	string
output	ok	boolean
20.updateExpectedArtifact		
input	expectedArtifactObj	ExpectedArtifact
	expectedArtifactID	string
	mandatory	boolean
	tokenID	string
output	ok	boolean
21.updateArtifact		
input	artifactXML	Artifact
	artifactID	string
	tokenID	string
output	ok	boolean
22.updateProject		
input	projectXML	Project
	projectID	string
	tokenID	string
output	ok	boolean

Figura 33 – Serviços para atualização de ativo no repositório.

2.newProcessTemplate		
input	processTemplateObj	ProcessTemplate
	tokenID	string
output	processID	string

3.newProcessInstance		
input	processInstanceObj	ProcessInstance
	processTemplateID	string
	projectID	string
	tokenID	string
output	processID	string

4.newExpectedPhase		
input	expectedPhaseObj	ExpectedPhase
	processTemplateID	string
	expectedPhaseParentID	string
	mandatory	boolean
	tokenID	string
output	phaseID	string

5.newPhaseInstance		
input	phaseInstanceObj	PhaseInstance
	expectedPhaseID	string
	processInstanceID	string
	phaseInstanceParentID	string
	tokenID	string
output	phaseID	string

6.newExpectedActivity		
input	expectedActivityObj	ExpectedActivity
	expectedPhaseID	string
	processTemplateID	string
	expectedActivityParentID	string
	mandatory	boolean
	tokenID	string
output	activityID	string

7.newActivityInstance		
input	activityInstanceObj	ActivityInstance
	expectedActivityID	string
	phaseInstanceID	string
	processInstanceID	string
	activityInstanceParentID	string
	tokenID	string
output	activityID	string

8.newExpectedArtifact		
input	expectedArtifactObj	ExpectedArtifact
	expectedActivityID	string
	processTemplateID	string
	mandatory	boolean
	tokenID	string
output	artifactID	string

9.newArtifact		
input	artifactObj	Artifact
	activityInstanceID	string
	processInstanceID	string
	expectedArtifactID	string
	tokenID	string
output	artifactID	string

10.newProject		
input	projectObj	Project
	type	string
	processTemplateID	string
	tokenID	string
output	projectID	string

12.newFeatureImplementation		
input	artifactFeatureImplementationObj	Artifact
	featureModelID	string
	featureID	string
	activityInstanceID	string
	processInstanceID	string
	tokenID	string
output	artifactID	string

13.newArchitecture		
input	artifactArchitectureObj	Artifact
	activityInstanceID	string
	processInstanceID	string
	tokenID	string
output	artifactID	string

46.importFeatureModel		
input	artifactFMObj	Artifact
	activityInstanceID	string
	processInstanceID	string
	tokenID	string
output	artifactFeatureModelID	string

Figura 34 – Serviços para inserção de ativo no repositório.

23.remove		
input	objectType	ObjectType
	objectID	string
	tokenID	string
output	ok	boolean

Figura 35 – Serviço para remoção de ativo do repositório.

36.find		
input	objectType	ObjectType
	paramType	ParamType
	param	string
	tokenID	string
output	IDObjectList	IDObjectList
47.searchSingle		
input	objType	ObjectType
	searchSingleObj	SearchSingle
	tokenID	string
output	IDObjectList	IDObjectList

48.searchAll		
input	objType	ObjectType
	searchAllObj	SearchALL
	tokenID	string
output	IDObjectList	IDObjectList
49.searchAny		
input	objType	ObjectType
	searchAnyObj	SearchANY
	tokenID	string
output	IDObjectList	IDObjectList

Figura 36 – Serviços para busca no repositório.

24.retrieveProcessTemplate		
input	processTemplateID	string
	tokenID	string
output	processXML	ProcessTemplate
25.retrieveProcessInstance		
input	processInstanceID	string
	tokenID	string
output	processXML	ProcessInstance
26.retrieveExpectedPhase		
input	expectedPhaseID	string
	tokenID	string
output	phaseXML	ExpectedPhase
27.retrievePhaseInstance		
input	phaseInstanceID	string
	tokenID	string
output	phaseXML	PhaseInstance
28.retrieveExpectedActivity		
input	expectedActivityID	string
	tokenID	string
output	activityXML	ExpectedActivity
29.retrieveActivityInstance		
input	activityInstanceID	string
	tokenID	string
output	activityXML	ActivityInstance
30.retrieveExpectedArtifact		
input	expectedArtifactID	string
	tokenID	string
output	artifactXML	ExpectedArtifact

31.retrieveArtifact		
input	artifactID	string
	tokenID	string
output	artifactXML	Artifact
32.retrieveFeatureModel		
input	projectID	string
	processInstanceID	string
	artifactID	string
	tokenID	string
output	artifactXML	Artifact
33.retrieveArchitecture		
input	projectID	string
	processInstanceID	string
	artifactID	string
	tokenID	string
output	artifactXML	Artifact
34.retrieveProject		
input	projectID	string
	processInstanceID	string
	tokenID	string
output	projectXML	Project
35.retrieveProduct		
input	productID	string
	tokenID	string
output	referenceLink	string

Figura 37 – Serviços para recuperação de ativo no repositório.

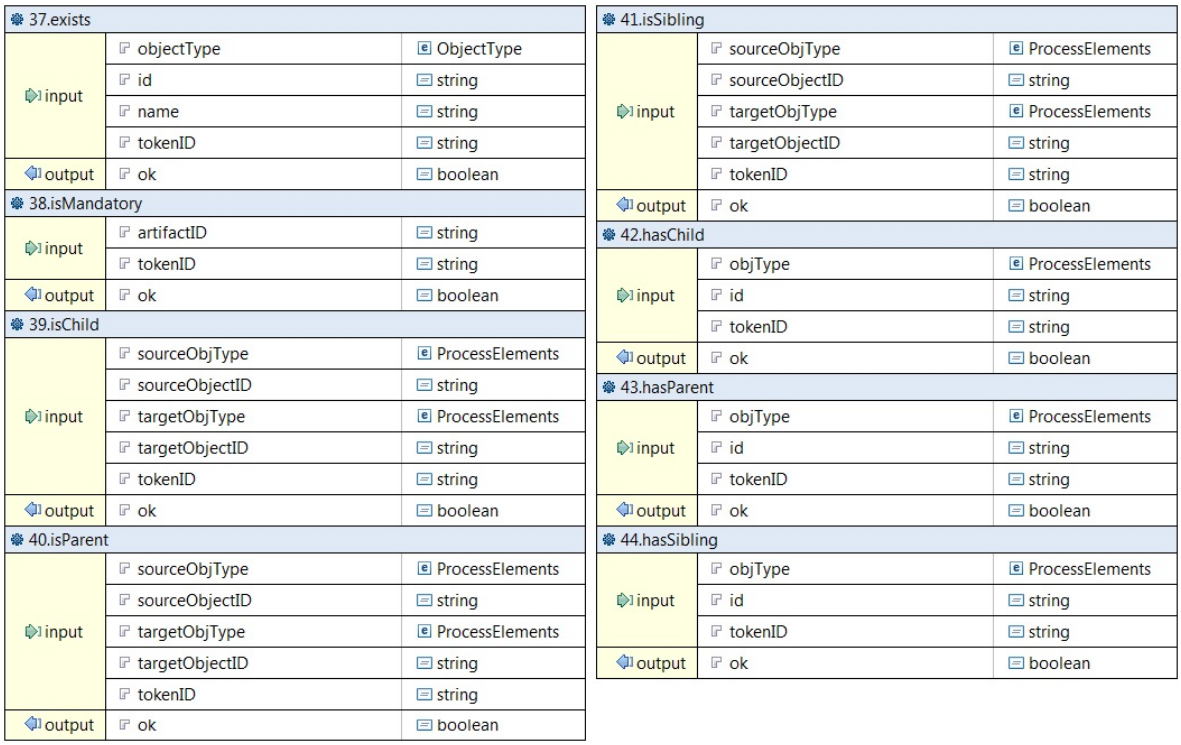


Figura 38 – Serviços para verificação ou validação de dados do repositório.

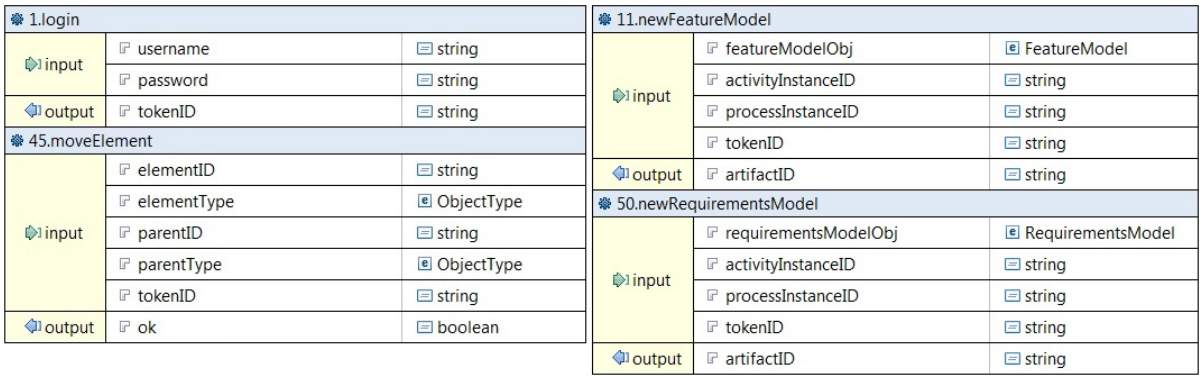


Figura 39 – Outros serviços.

APÊNDICE

B

MODELAGEM COMPLETA DOS CASOS DE USO

Tabela 5 – Associação de Casos de Uso.

Nº	**Ator**	**Caso de Uso**	**Descrição**	**Serviços Envolvidos**
1	Usuário Regular	Logar	O login do usuário é utilizado para autenticação por meio de uma chave denominada *token*. O *token* retornado por este serviço é utilizado na chamada de qualquer outra funcionalidade de acesso a dados do repositório, cabendo à aplicação cliente retornar e validar essa chave de acordo com suas políticas internas.	1
2	Especialista em Processos	Inserir Processo	O engenheiro pode inserir um novo processo para servir de base ao projeto. O processo inserido fica armazenado e deve seguir o meta-modelo proposto na Figura 17 do Capítulo 4.	2, 3
3	Engenheiro	Buscar Processo	O engenheiro pode buscar e selecionar um processo já cadastrado para servir de base ao projeto.	24, 25, 47

Continua na próxima página

Tabela 5 – *Continuação*

Nº	**Ator**	**Caso de Uso**	**Descrição**	**Serviços Envolvidos**
4	Especialista em Processos	Excluir Processo	Um processo que não está mais em utilização por nenhum projeto em andamento pode ser excluído.	23, 37-40
5	Especialista em Processos	Editar Processo	Um processo pode ser alterado no que diz respeito a suas atividades que podem ser incrementadas, reduzidas ou atualizadas.	14, 15
6	Engenheiro de Domínio	Gerenciar Projeto de LPS	Ao criar um novo projeto de LPS, a escolha de um processo cadastrado é obrigatória. Um novo projeto de LPS é criado e vinculado ao usuário/grupo e o usuário é guiado pelo processo selecionado até sua finalização. O usuário também pode querer buscar, editar ou excluir um projeto de LPS já existente de acordo com as restrições e as permissões que este usuário possui.	10, 22, 23, 34, 37-40
7	Engenheiro de Domínio	Gerar Detalhamento de Requisitos	Ao incluir um documento de requisitos, o usuário pode optar por armazenar cada requisito separadamente para potencializar o reúso. Esses requisitos podem ser posteriormente relacionados com as *features*.	50
8	Engenheiro de Domínio	Gerenciar Modelo de *Features*	Armazenar e manipular o Modelo de *Features* de acordo com os requisitos da LPS.	11, 21, 23, 32, 37-40
9	Engenheiro de Domínio	Gerenciar Arquitetura	Armazenar e manipular a arquitetura da LPS.	13, 21, 23, 31, 37-40
10	Engenheiro de Domínio	Gerenciar Módulo de Implementação de *Features*	Inserir artefato(s) que implementam uma *feature*.	12, 21, 23, 31, 37-40

Continua na próxima página

Tabela 5 – *Continuação*

Nº	Ator	Caso de Uso	Descrição	Serviços Envolvidos
11	Engenheiro	Gerar Mapa do Projeto	Funcionalidade sugerida onde o usuário pode requerer do sistema uma visão geral do projeto com suas atividades a realizar e realizadas, bem como todos os artefatos relacionados a essas atividades.	25, 34
12	Engenheiro de Domínio / Engenheiro de Aplicação	Criar Atividade	O usuário tem a opção de incrementar o processo criando atividades de acordo com a necessidade.	6, 7
13	Engenheiro de Domínio / Engenheiro de Aplicação	Buscar Atividade	O usuário pode buscar e selecionar uma atividade para visualização.	36, 47-49
14	Engenheiro de Domínio / Engenheiro de Aplicação	Executar Atividade	Ao executar uma atividade, os artefatos são inseridos e manipulados e as informações sobre a execução são armazenadas.	19, 29
15	Especialista em Processos	Editar Atividade	Após selecionar uma atividade, esta pode ser modificada pelo usuário. As alterações podem compreender desde informações básicas como o nome, até relacionamentos, artefatos esperados e papéis.	19, 29
16	Engenheiro de Domínio / Engenheiro de Aplicação	Excluir Atividade	Uma atividade que não possui artefatos que estão em utilização pode ser excluída.	23, 37-40
17	Engenheiro de Domínio / Engenheiro de Aplicação	Gerenciar Artefato	Manipular artefatos do projeto. Realiza a criação e o armazenamento de artefatos, além da atualização, da exclusão e da recuperação de artefatos armazenados.	9, 21, 23, 31, 32, 37-40

Continua na próxima página

Tabela 5 – *Continuação*

Nº	**Ator**	**Caso de Uso**	**Descrição**	**Serviços Envolvidos**
18	Engenheiro de Aplicação	Gerenciar Projeto de Produto	Ao criar um novo projeto de produto, a escolha de um processo cadastrado é obrigatória. Um novo projeto de produto é criado e vinculado ao usuário/grupo e o usuário é guiado pelo processo selecionado até sua finalização. O usuário também pode querer buscar, editar ou excluir um projeto de produto já existente de acordo com as restrições e as permissões que este usuário possui.	10, 22, 23, 34, 37-40
19	Engenheiro de Aplicação	Gerenciar Configuração das *Features*	Inserir um modelo de configuração existente ou elaborar um novo modelo, caso as *features* estejam inseridas no modelo recomendado por este trabalho, selecionando-as do Modelo de *Features* de acordo com os requisitos do produto alvo. O modelo de configuração é utilizado mais tarde na derivação do produto (funcionalidade não inclusa neste trabalho).	9, 21, 23, 31, 32, 37-40
20	Engenheiro	Gerar Estatísticas	Funcionalidade sugerida para gerar estatísticas de acordo com filtros selecionados (Projetos, Processos, Artefatos, Produtos, ...) para uma estimativa do estado e da utilização dos elementos do repositório.	36, 47-49
21	Engenheiro	Verificar Controle de Versões	Sugestão de funcionalidade para verificar as modificações de itens selecionados de acordo com a granularidade selecionada nos filtros (*Asset* / *Artifact*).	24-36, 47-49

Continua na próxima página

Tabela 5 – *Continuação*

Nº	**Ator**	**Caso de Uso**	**Descrição**	**Serviços Envolvidos**
22	Engenheiro	Gerar Histórico de Atividades	Funcionalidade sugerida para verificar as atividades relacionadas diretamente aos ativos selecionadas de acordo com o tipo de atividade (criação, atualização, exclusão) e possíveis filtros (usuário, grupo, projeto).	36, 47-49
23	Engenheiro	Visualizar Logs	Sugestão de funcionalidade para visualizar logs do sistema. Os logs podem armazenar informações de acesso, operações em banco, etc. (Funcionalidade de armazenar logs não incluída no escopo desta abordagem)	36, 47-49
24	Usuário Regular	Recuperar Produtos	Abre e disponibiliza ao usuário um catálogo de produtos disponível no repositório.	36, 47-49

ANEXO

A

CORE RAS

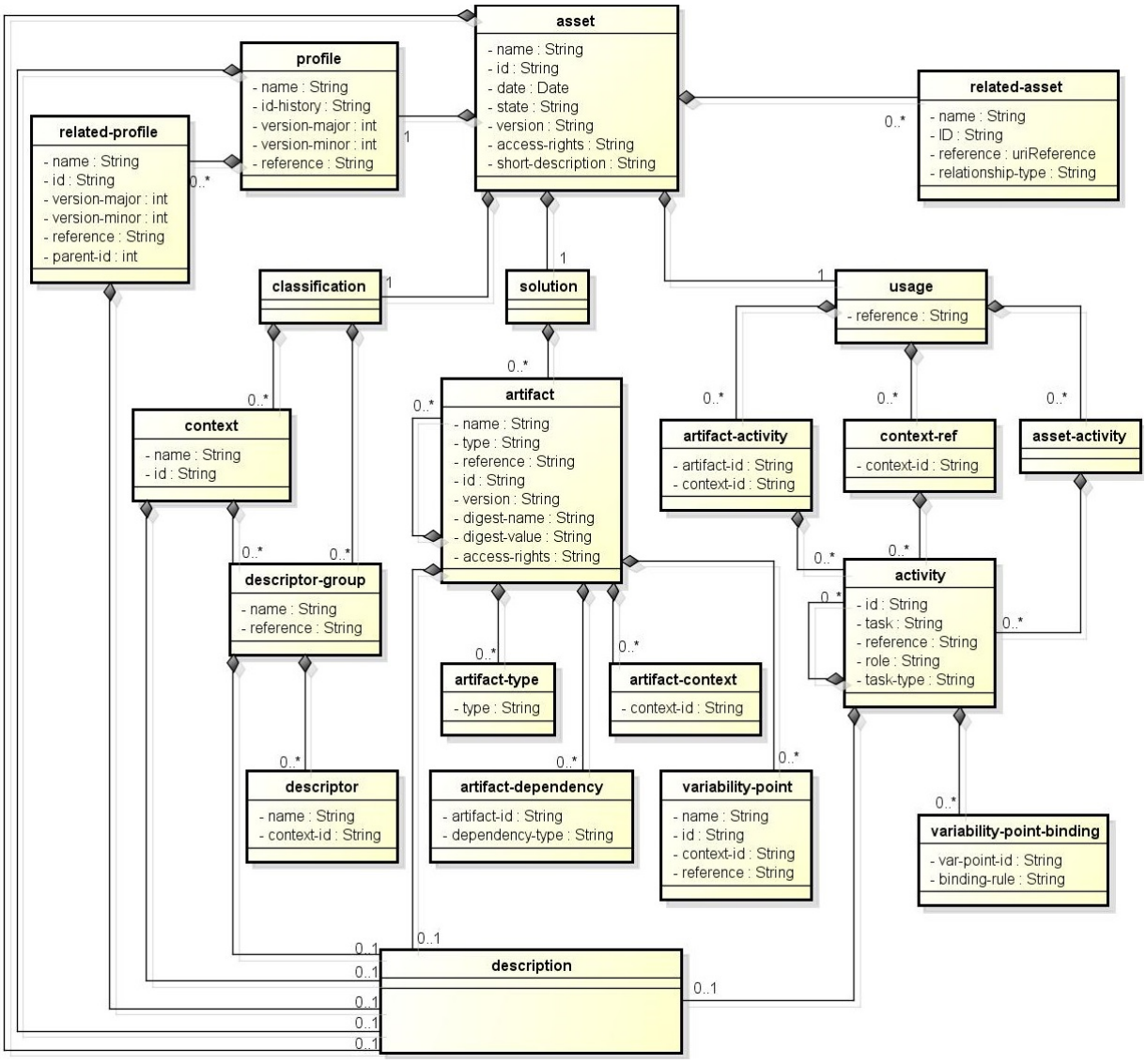


Figura 40 – Estrutura *Core* da RAS. Adaptada de Group (2005).