Universidade Federal do Ceará
Centro de Ciências
Mestrado e Doutorado em Ciência da Computação
Grupo de Redes, Engenharia de Software e Sistemas

Enriquecimento Semântico de Linhas de Produto de Software

Dissertação de Mestrado

João Bosco Ferreira Filho

Rossana Maria de Castro Andrade, PhD.
Orientadora

Windson Viana de Carvalho, DSc.
Co-orientador

Fortaleza, Ceará
2011

João Bosco Ferreira Filho

Enriquecimento Semântico de Linhas de Produto de Software

Dissertação de Mestrado submetida à Coordenação do Programa de Pós-graduação em Ciência da Computação (MDCC) da Universidade Federal do Ceará (UFC) como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientadora: Rossana Maria de Castro Andrade, PhD.

Co-orientador: Windson Viana de Carvalho, DSc.

Fortaleza, Ceará
2011

Enriquecimento Semântico de Linhas de Produto de Software

Dissertação de Mestrado submetida à Coordenação do Programa de Pós-graduação em Ciência da Computação (MDCC) da Universidade Federal do Ceará (UFC) como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Aprovado em ___/___/_____

Banca Examinadora

_____

Profa. Rossana Maria de Castro Andrade, PhD. (Orientadora)
Universidade Federal do Ceará – UFC

_____

Prof. Windson Viana de Carvalho, DSc. (Co-orientador)
Universidade Federal do Ceará – UFC

_____

Prof. José Antônio Fernandes de Macêdo, DSc.
Universidade Federal do Ceará – UFC

_____

Prof. Cidcley Teixeira de Souza, DSc.
Instituto Federal de Educação, Ciência e Tecnologia do Ceará – IFCE

_____

Prof. Jérôme Gensel, DSc.
Université Pierre Mendès France – UPMF

Agradecimentos

Agradeço, necessariamente, a Deus, que sempre me mostrou o caminho certo em tempos de irresolução. É dele o cerne do meu ser e do que me tornarei após mestre, sendo também dele os frutos de meus esforços acadêmicos.

Agradeço aos meus pais, Terezinha e Bosco, não apenas pelo suporte aos estudos, mas também pelo incentivo, interesse e preocupação durante o mestrado. Declaro a eles o mérito de serem o alicerce para a construção de minha formação acadêmica. E com maior relevância, sou grato a eles e ao meu irmão, Gabriel, por sermos uma família unida que somos.

Agradeço a Janaína, minha namorada, a qual me ajudou desde a decisão pela Ciência da Computação até os últimos esforços para terminar esta dissertação. Espero continuar sempre contando com seu companheirismo e compartilhar sempre excelentes momentos como os vividos durante o mestrado.

Agradeço a orientação da professora Rossana, maior responsável pela minha base de pesquisa acadêmica construída até hoje. Além da orientação, agradeço também pela infraestrutura proporcionada pelo grupo sob sua coordenação, o GREat, no qual passei importantes momentos da minha vida. Ainda, sou agradecido pela forte confiança depositada, esperando ter correspondido às expectativas.

Agradeço a co-orientação do professor Windson, sempre com relevantes conselhos e novas ideias para aperfeiçoar o trabalho. Também agradeço pela confiança, mesmo com as dúvidas iniciais sobre o trabalho.

Agradeço aos professores que, com paciência e dedicação, facilitaram a aprendizagem de assuntos essenciais para o desenvolvimento desta dissertação. Em especial, ao professor José Antônio pelo tempo despendido em conversas sobre o uso de ontologias em meu trabalho e orientações pessoais.

Aos demais profissionais, tanto da UFC e MDCC quanto do GREat, em especial ao Orley por sempre, prestativamente, prover as informações e documentos solicitados. Agradeço também a toda secretaria e administração do GREat.

Finalmente, agradeço a todos os amigos que fizeram parte desta caminhada, tanto no GREat quanto fora, cito de forma especial e alfabética: Akemi, Alexandre, Aline, Bruno Góis, Bruno Sabóia, Camila, Carina, Carlos Alberto, Cláudio, Cleilton, Dácio, Darilu, Davi, Davyd, Diana, Fabiana, Fabrício, Fátima, Fernando, Flávio, Joaquim, João Borges, João Marcelo, Karol, Liliane, Lincoln, Lucas, Luís, Márcio, Marcos, Michel, Neto, Paulo Alexandre, Paulo Henrique, Reinaldo, Ronaldo, Rute, Saulo, Smaylle, Sofia, Tales, Teófilo, Thiarley, Valéria e Vanessa.

“O princípio da sabedoria é
adquirir a sabedoria.”

Provérbios 4, 7a

RESUMO

As Linhas de Produto de Software (LPS) evoluíram e ganharam atenção como uma das mais promissoras abordagens para a reutilização de software. A modelagem de características é a principal técnica para representar a variabilidade do domínio em uma LPS. No entanto, existem aspectos do domínio, além da variabilidade, que não podem ser expressos em um modelo de características. Além disso, estes modelos não foram projetados para facilitar a recuperação da informação, interoperabilidade e inferência. Em contraste, as ontologias parecem ser a melhor solução, visto que atendem a esses requisitos. Por isso, este trabalho apresenta uma abordagem para o enriquecimento semântico de LPS baseado em ontologias. Nossa proposta fornece métodos para acrescentar informações sobre o domínio além da descrição da variabilidade, e uma top-ontology que especifica conceitos genéricos e as relações em uma LPS. A abordagem proposta reutiliza o modelo de características existente na LPS, possibilitando a adição de descrições semânticas de uma forma menos intrusiva do que modificar a notação do modelo de características. Um estudo de caso é apresentado para avaliar os benefícios da adoção da abordagem.

Palavras-Chave: Linhas de Produto de Software, Modelagem de Variabilidade, Semântica, Ontologias

ABSTRACT

Software Product Lines (SPLs) have evolved and gained attention as one of the most promising approaches for software reuse. Feature models are the main technique to represent domain variability in SPLs. However, there are other domain aspects, besides variability, which cannot be expressed in a feature model. Moreover, these models were not designed to facilitate information retrieval, interoperability and inference. In contrast, ontologies seem to be the best solution, because they meet these requirements. Therefore, this work presents an approach for semantic enrichment in SPLs based on ontologies. Our proposal provides methods to add domain information besides variability description, and a top-ontology that specifies generic concepts and relations in a SPL. The proposed approach reuses the existing SPL feature model, adding semantic descriptions in a less intrusive way than modifying the feature model notation. A case study is presented to evaluate the benefits of adopting the approach.

Keywords: Software Product Lines, Variability Modeling, Semantic, Ontologies

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

API – *Application Programming Interface*
DL – *Description Logic*
DOM – *Document Object Model*
ED – Engenharia de Domínio
FMP – *Feature Modeling Plug-in*
FODA – *Feature-Oriented Domain Analysis*
KIF – *Knowledge Interchange Format*
LPS – Linha de Produto de Software
LPSS – Linha de Produto de Software Semântica
MDE – *Model-Driven Engineering*
OWL – *Web Ontology Language*
RAR – Repositório de Ativos Reutilizáveis
RDF – *Resource Description Framework*
RDFS – *Resource Description Framework Schema*
SEI – *Software Engineering Institute*
SPLiSEM – *Software Product Line Semantic Enrichment Model*
TTM – *Time to Market*
UML – *Unified Modeling Language*
W3C – *World Wide Web Consortium*
XML – *Extensible Markup Language*

Sumário

1. Introdução

Esta dissertação apresenta uma proposta para o enriquecimento semântico de Linhas de Produto de Software. A Seção 1.1 contém uma contextualização sobre a área na qual este trabalho se insere. Na Seção 1.2, são apresentadas as motivações para inserir semântica dado o contexto descrito. Os objetivos e as contribuições do trabalho estão listados na Seção 1.3. Na Seção 1.4, a estrutura desta dissertação é resumida em tópicos.

1.1 Contextualização

As Linhas de Produto de Software (LPS) são uma das mais promissoras abordagens de reutilização de software. As LPS têm como principal objetivo reduzir a repetição de trabalhos já desenvolvidos, que são a maior parte dos esforços durante o desenvolvimento de um software [1]. Com a redução do retrabalho, as LPS proporcionam uma diminuição dos custos financeiros e de tempo durante o desenvolvimento de um determinado sistema. O paradigma de LPS é baseado principalmente na Engenharia de Domínio (ED) [2], que é parte fundamental na estrutura de uma linha. É através da ED que se define o escopo de uma linha (i.e., quais sistemas de uma determinada área de conhecimento poderão ser produzidos a partir de tal LPS [3]). Além disso, a ED proporciona uma análise e uma especificação da área da família de aplicações considerada pela LPS. As especificações exprimem o conhecimento sobre as funcionalidades comuns e variáveis das aplicações inseridas em tal domínio.

As LPS estão estreitamente ligadas às organizações, por isso é dada uma forte atenção para o conceito *time to market* (TTM), que expressa o tempo decorrido desde a requisição e a concepção do produto até ele estar pronto para ser comercializado ou implantado. Como ilustra a Figura 1, em uma organização que aplica a reutilização de software, a equipe de desenvolvimento publica artefatos para serem reutilizados no futuro em um Repositório de Ativos Reutilizáveis (RAR). O RAR centraliza os artefatos

da organização, os quais podem ser código ou apenas documentos. Ao receber de um cliente uma solicitação de desenvolvimento de um novo produto, a organização deve verificar junto ao RAR quais dos requisitos do cliente podem ser contemplados pelos artefatos já existentes na organização. Após esta análise, é necessário decidir quais dos requisitos exigirão desenvolvimento, compra ou terceirização da empresa.

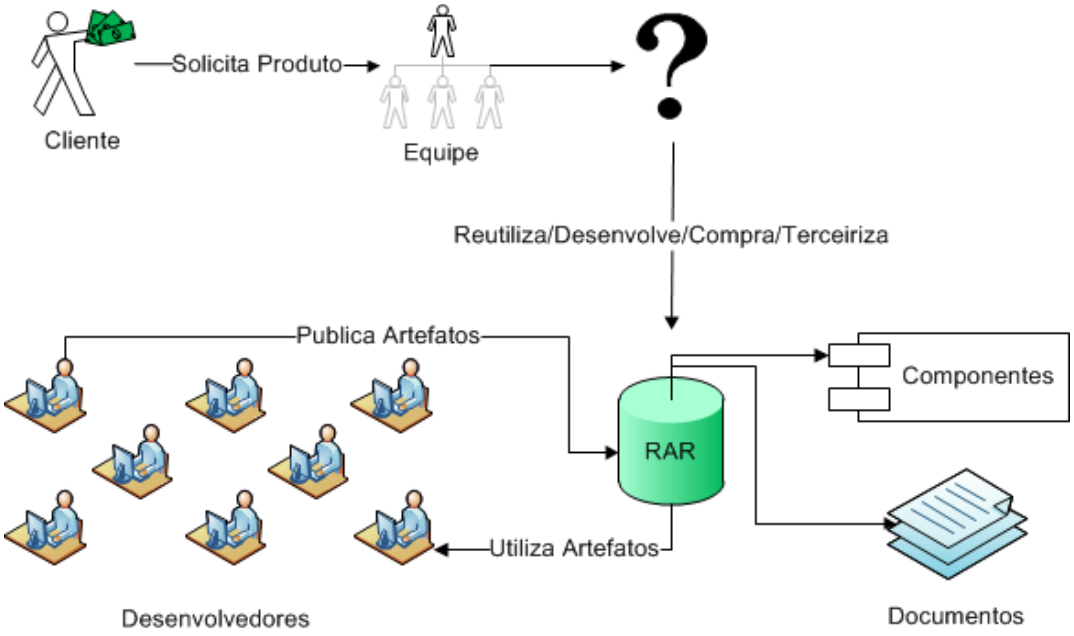


Figura 1. Contexto de uma organização que reutiliza artefatos.

No contexto de agilidade no desenvolvimento, é desejável que se minimize o tempo em que estes produtos são produzidos. A diminuição do retrabalho através da reutilização tem se mostrado uma grande aliada na redução do TTM, juntamente com metodologias de geração automática de modelos e código, dentre as quais se destaca a programação generativa [4] e a engenharia dirigida a modelos (tradução livre de *Model-Driven Engineering* – MDE) e suas transformações [5]. Apesar dos trabalhos emergentes envolvendo linhas de produto e MDE, como em [6], as LPS ainda se encontram distantes da automatização pregada pelas metodologias de geração automática. Este fato decorre, principalmente, devido as LPS ainda exigirem intervenção humana para realizar várias atividades na derivação de um novo produto, desde seus requisitos até a identificação dos artefatos reutilizáveis envolvidos no processo de derivação. Essa intervenção se faz indispensável pelo fato dos sistemas computacionais de apoio das LPS não terem o conhecimento necessário para realizar essas atividades de forma automática. Na verdade, esse conhecimento está retido pelas

pessoas envolvidas ou está expresso de forma limitada em diagramas específicos dos quais muitos são apenas para leitura humana.

1.2 Motivação

Para que se atinjam níveis cada vez maiores de automação nas LPS, é necessário que sistemas computacionais consigam armazenar e processar as informações envolvidas na concepção e no desenvolvimento dos produtos. Isso implica em transformar informação em conhecimento explorável. Viabilizar esse conhecimento é essencialmente representá-lo de forma que o sistema computacional consiga processá-lo. Para isso, podem ser usadas diversas abordagens de representação do conhecimento, dentre elas, as ontologias se apresentam como alternativa mais promissora devido ao bom suporte ferramental e a sua capacidade de expressão de conhecimento. Em termos de linguagens, a *Web Ontology Language* (OWL) é a mais utilizada para implementar ontologias, pois fornece uma grande interoperabilidade pelo fato de ser baseada em XML e por ser uma recomendação da W3C [7]. Essa linguagem, mais especificamente o dialeto OWL DL (*OWL Description Logics*), apóia-se no formalismo de representação do conhecimento da Lógica Descritiva. Além de tornar o conhecimento sobre um domínio legível e processável por um sistema computacional, as ontologias, juntamente com os algoritmos de inferência (*reasoners*), proporcionam a dedução de novas informações e de relações que estavam implícitas na ontologia inicial.

Por outro lado, o conhecimento sobre o domínio nas LPS é geralmente modelado em diagramas de características. Estes diagramas representam as características visíveis ao usuário final bem como seus relacionamentos hierárquicos, suas variabilidades e suas regras de composição [8]. Desde sua concepção, o diagrama de características é alvo de estudo e inspiração para outras modelagens propostas, como ocorre com a notação *Odyssey-FEX* [9]. Outros diagramas podem ser utilizados para a modelagem de variabilidade de características, como é o caso dos diagramas da UML [10] [11] [12] [13], e os modelos de objetivos (*goals*) [14]. O fato é que tais formas de modelagem se concentram em tratar a variabilidade das funcionalidades, o que é bastante válido para o paradigma de LPS, pois a partir deles podemos saber, por exemplo, as combinações possíveis de características na geração de um produto. Entretanto, esses diagramas não são adequados para modelar algumas peculiaridades do

domínio, como, por exemplo, uma relação binária específica entre características do tipo "*troca mensagens com"* (e.g., módulo de localização *troca mensagens com* módulo de controle). Para tanto, seria necessário, por exemplo, o uso de um diagrama de componentes, o qual nem sempre é contemplado na etapa de Engenharia de Domínio. Além disso, é possível termos relações ainda mais específicas e de difícil expressão, tais como, a relação "*procura por",* na qual um módulo de provisão de conteúdo *procura por* serviços de mídias disponíveis na rede. Essa inadequação do diagrama de características é consequência das suas limitações de expressividade, principalmente se o compararmos às ontologias, que são muito mais ricas semanticamente [15].

1.3 Objetivos e Contribuições

Tendo em vista as limitações citadas, este trabalho tem como principal objetivo prover meios para adicionar semântica em Linhas de Produto de Software. Este enriquecimento semântico gera uma nova terminologia utilizada nesta dissertação: **Linhas de Produto de Software Semânticas** (LPSS). Como principal diferencial das LPS, as LPSS têm uma rede semântica que conecta os artefatos da linha entre si e com informações da organização na qual a LPSS está inserida. Artefatos como diagramas de componentes, de casos de uso, de classes, planos de testes, currículos dos profissionais, históricos de produtividade, histórico de erros e assim por diante, estarão relacionados com o modelo de domínio, através de uma ou mais ontologias que descrevem a linha de produto.

Para atingir o objetivo deste trabalho, são propostos e desenvolvidos dois métodos bases para o enriquecimento semântico de uma LPS. O primeiro método é um mapeamento automático de um modelo de características para uma ontologia e é intitulado Fea2Onto. O produto da aplicação do Fea2Onto é uma ontologia com a mesma semântica explícita do diagrama de características, e, por ser uma ontologia inicial, é chamada de esqueleto OWL (tecnologia de representação adotada). O segundo método é um enriquecimento semântico baseado em um modelo criado, chamado *Software Product Line Semantic Enrichment Model* (SPLiSEM). O SPLiSEM é uma *top-ontology* que define a natureza das relações e conceitos que enriquecerão o esqueleto OWL gerado no primeiro método. Os dois métodos, quando utilizados em uma LPS, devem facilitar a obtenção de uma linha que possua uma ontologia para

descrever as suas características de forma interligada com o restante dos seus artefatos, a qual é denominada de LPSS. Na Figura 2, a sequência de aplicação dos métodos propostos é apresentada. Parte-se de um modelo de características, gera-se um esqueleto OWL inicial e, ao final, produz-se uma ontologia que descreve as características de uma linha e as relações entre seus diversos artefatos.

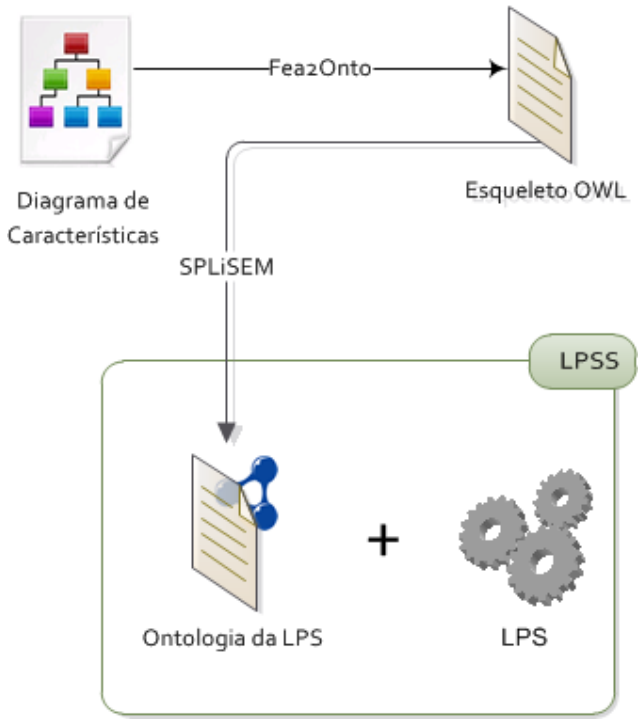


Figura 2. Visão geral da abordagem que propõe o enriquecimento semântico em uma LPS.

Uma vez criada uma LPSS em uma organização, essa poderá se beneficiar de três principais ganhos no processo de configuração de um produto: a recuperação de informação sobre os artefatos, inferência de novas informações e a rastreabilidade de artefatos. Estes três aspectos são analisados no estudo de caso da dissertação.

1.4 Estrutura do Documento

Este capítulo apresentou uma introdução sobre o tema abordado no trabalho. Uma contextualização do tema foi explorada e antecedeu a motivação apresentada, a qual incentivou o desenvolvimento dos objetivos também explicitados neste primeiro capítulo.

As ontologias foram escolhidas como ferramenta para a inserção de semântica nas LPS. Assim, o Capítulo 2 discorre sobre os conceitos fundamentais das ontologias e suas tecnologias que permitiram alcançar os objetivos almejados.

O Capítulo 3 é uma fundamentação sobre o objeto de estudo principal da dissertação, as Linhas de Produto de Software, visto que a inserção de semântica é feita em uma LPS utilizando seu modelo de domínio como base. Os conceitos que fundamentam este paradigma, o próprio paradigma e trabalhos relacionados que estendem o conceito de LPS são apresentados no Capítulo 3.

No Capítulo 4, a proposta propriamente dita é apresentada em duas seções. A primeira seção apresenta o Fea2Onto, e a ferramenta desenvolvida para automatizar o seu mapeamento. A segunda seção contém a descrição do enriquecimento semântico auxiliado pelo SPLiSEM.

Para aplicar o Fea2Onto e o SPLiSEM, no Capítulo 5 é apresentado um estudo de caso que utiliza como base uma LPS existente. Os métodos são utilizados na LPS e uma ontologia enriquecida da linha é obtida. Em seguida, é feita uma análise dos resultados obtidos através do enriquecimento semântico, como a constatação de benefícios como a rastreabilidade dos artefatos e a inferência sobre as novas relações explicitadas.

Finalmente, o Capítulo 6 apresenta as conclusões sobre o trabalho desenvolvido e discorre sobre os trabalhos que poderão dar continuidade no futuro às proposições já desenvolvidas.

2. Ontologias

Neste capítulo, as ontologias são descritas como importantes ferramentas para a especificação conceitual no contexto da Ciência da Computação. Particularmente na Engenharia de Software, as ontologias apresentam grandes contribuições na estruturação e representação de conhecimento. A Seção 2.1 discute as principais motivações e definições sobre as ontologias. Na Seção 2.2, são apresentados os tipos mais comuns de ontologia, bem como os seus principais usos. A Seção 2.3 é um sumário da linguagem de representação do conhecimento que será utilizada no restante do trabalho, a OWL DL, descrevendo seus principais elementos. Um resumo sobre as aplicações de ontologias na área de Engenharia de Software é apresentado na Seção 2.4. Por fim, a Seção 2.5 é uma conclusão sobre essa técnica de representação do conhecimento.

2.1 Introdução

A necessidade de classificar e categorizar as coisas ao seu redor é característica histórica da humanidade. Ao longo do tempo, técnicas para a organização da informação foram desenvolvidas para atender a estes anseios. Um exemplo de estruturação é a organização a partir da utilização de termos, como é o caso dos dicionários e glossários. Outra forma de organizar é considerar os aspectos categóricos e os esquemas de classificação. Este é o foco das taxonomias que ainda são objetos de estudo e pesquisa. Em contrapartida, a conceituação e os relacionamentos entre os conceitos são características de estruturas semânticas mais expressivas, como as ontologias, os *thesaurus* e as redes semânticas. Definir uma ontologia é, essencialmente, organizar a informação em conceitos e relações com o objetivo de distinguir o "ser" como tal [16] [17].

Uma das definições mais conhecidas de ontologia é a de Gruber, que diz que uma ontologia é uma especificação de uma conceituação [18]. Tal definição sofreu atualizações desde sua primeira concepção e adaptou-se para melhor se ajustar à Ciência

da Computação. A definição de ontologia atualizada e publicada em uma enciclopédia da área de banco de dados [19] é: "*uma definição de um conjunto de primitivas de representação com o qual se modela um domínio de conhecimento ou discurso*".

As formas de representar uma ontologia evoluíram ao longo do tempo. As primeiras linguagens de representação do conhecimento foram a KIF [20] e a KL-ONE [21]. A primeira é uma linguagem sintaticamente baseada em *Lisp* para expressar sentenças de lógica de predicados de primeira ordem. Embora seja uma linguagem de alta expressividade, a KIF é considerada como sendo de baixo-nível, dificultando assim sua utilização por profissionais sem muita familiaridade com a notação. A KL-ONE foi uma linguagem que serviu de base para diversos trabalhos sobre ontologias. Tal linguagem basicamente implementou as redes de herança estruturadas, contendo descrições de conceitos e as relações de generalização e especialização entre estes conceitos. A partir da KL-ONE, uma família da lógica chamada de Lógica Descritiva (DL) foi originada. Esta lógica tem como objetivo prover o formalismo necessário para servir de base para sistemas de representação do conhecimento [22].

Paralelo ao desenvolvimento de linguagens baseadas na lógica descritiva, também surgiram estudos para a representação de ontologias em UML. Por ser fortemente adotada pelos profissionais da área de tecnologia da informação, a UML motiva sua utilização para os mais diversos tipos de modelagem, inclusive de ontologias. Em [23], podemos encontrar uma descrição de utilização de UML como linguagem de modelagem de ontologias. Esta representação, entretanto, não se beneficia de mecanismos de inferência, visto que UML é uma notação semi-formal.

Com o advento da Web Semântica, uma nova questão foi adicionada ao uso de linguagens de modelagem de ontologias: a interoperabilidade. Esta questão decorre do fato de que o conhecimento contido na web, em formato de ontologias, deveria ser compartilhado, lido e escrito de forma padronizada por todos os interessados na informação em questão [24], sendo eles humanos ou máquinas. Como XML foi considerada a linguagem que promoveria a interoperabilidade entre sistemas, naturalmente as linguagens seguintes de representação de conhecimento se basearam nela, como é o caso de RDF, RDFS e OWL.

A OWL tem sido utilizada como uma recomendação da W3C para a representação de informações que, além de necessitarem entendimento humano, também devem ser entendidas por máquina [7]. Tal linguagem possui três variações concordantes com o nível de expressividade necessário. A menos expressiva é a OWL *Lite* que, por exemplo, não permite expressar restrições de cardinalidade como no máximo ou no mínimo, a menos que o número inteiro positivo que é o teto ou piso seja zero ou um.

Como alternativa, temos a OWL fundamentada na lógica descritiva: OWL DL. Esta sublinguagem da OWL é mais expressiva e ainda é decidível quanto ao processamento feito pelos algoritmos de inferência sobre perguntas de decisão. A terceira variação, OWL *Full*, é a mais expressiva, porém viola a propriedade da decidibilidade. Em uma ontologia descrita em OWL *Full* os conceitos, além de representarem uma classe de indivíduos, também podem ser tratados, simultaneamente, como indivíduos. Portanto, a fim de utilizarmos o máximo de expressividade, porém sem tornar a ontologia indecidível, esta dissertação adota a OWL DL, que é detalhada na Seção 2.3.

Além da justificativa da escolha da linguagem de ontologia utilizada, outra questão que pode ser levantada é, de forma mais geral, por que adotar ontologias como forma de representar o conhecimento neste trabalho. A justificativa é que, mesmo se fossem adotadas outras formas de representação, como exemplo, diagramas de classe ou mapas mentais, esses diagramas também representam uma ontologia, dado que especificam conceitos e relações entre os eles, de acordo com a definição de ontologias.

2.2 Tipos de Ontologias

Por natureza, as ontologias possuem um nível de expressividade que permite modelar diferentes níveis de abstrações, além de diversos domínios. Essencialmente, tudo que possa ter sua estrutura definida através de conceitos e relações entre estes conceitos pode ser modelado através de uma ontologia. Dada essa abrangência, alguns esforços foram concentrados visando categorizar as ontologias. A Tabela 1 apresenta um compêndio das abordagens que têm como objetivo tipificar as ontologias [16].

Tabela 1. Tipos de ontologias. Adaptado de [16].

Abordagem	Classificação	Descrição
Quanto à Função	De domínio	Fornecem vocabulário sobre conceitos, seus relacionamentos e sobre atividades e regras que o governam.
	De tarefa	Fornecem um vocabulário sistematizado de termos, especificando tarefas que podem ou não estar no mesmo domínio.
	Gerais	Incluem um vocabulário relacionado a coisas, eventos, tempo, espaço, casualidade, comportamento, funções.
Quanto ao grau de formalismo	Altamente Informais	Expressa livremente em linguagem natural.
	Semi-informais	Expressa em linguagem natural de forma restrita e estruturada.
	Semi-formais	Expressa em uma linguagem artificial definida formalmente.
	Rigorosamente formal	Os termos são definidos com semântica formal, teoremas e provas.
Quanto à aplicação	De autoria neutra	Um aplicativo é escrito em uma única língua e depois convertido para uso em diversos sistemas, reutilizando-se as informações.
	De especificação	Cria-se uma ontologia para um domínio, a qual é usada para documentação e manutenção no desenvolvimento de software.
	De acesso comum à informação	Quando o vocabulário é inacessível, a ontologia torna a informação inteligível, proporcionando conhecimento compartilhado dos termos.
Quanto à estrutura	De alto-nível	Descrevem conceitos gerais relacionados a todos os elementos da ontologia (espaço, tempo, matéria, objeto, evento, ação) os quais são independentes do problema ou domínio.
	De domínio	Descrevem o vocabulário relacionado a um domínio, como, por exemplo, medicina ou comércio eletrônico.
	De tarefa	Descrevem uma tarefa ou atividade, como, por exemplo, diagnósticos ou compras, mediante inserção de termos especializados na ontologia.
Quanto ao conteúdo	Terminológicas	Especificam termos que serão usados para representar o conhecimento em um domínio.
	De informação	Especificam a estrutura de registros de bancos de dados
	De modelagem do conhecimento	Especificam conceitualizações do conhecimento, têm uma estrutura interna semanticamente rica e são refinadas para o uso no domínio do conhecimento que descrevem.
	De aplicação	Contêm as definições necessárias para modelar o conhecimento em uma aplicação.
	De domínio	Expressam conceitualizações que são específicas para um determinado domínio do conhecimento.
	Genéricas	Os conceitos que as definem são considerados comuns a vários campos.
	De representação	Explicam as conceitualizações que estão por trás dos formalismos de representação do conhecimento.

Os trabalhos da Tabela 1 levam em consideração diferentes orientações para tipificar as ontologias: quanto à sua função, ao grau de formalismo, à aplicação, à estrutura e ao conteúdo.

2.3 OWL DL

Uma ontologia possui elementos básicos que são utilizados durante a modelagem. A união destes elementos em uma forma bem definida, com a definição de termos, fórmulas e conectores, constitui uma linguagem formal, no caso a OWL DL. Portanto, esta seção é dedicada a apresentar os principais elementos da OWL DL e sua utilização, tendo como base sua especificação por [7] e [22].

Os primeiros elementos fundamentais são os **conceitos**, os quais são utilizados para representar uma classe de indivíduos. Os conceitos podem ter filhos que são conceitos menos abrangentes. Os filhos herdam propriedades e atributos dos conceitos pais. Todos os conceitos em OWL são subclasses da superclasse *thing* ($\top$ é o símbolo utilizado em DL, assim como o *bottom* é utilizado para o vazio $\bot$). A esses conceitos filhos é dado o nome de subclasses. Se um conceito A é subclasse de um conceito B (na sintaxe da DL: $A \sqsubseteq B$), implica que todo indivíduo que é de A é, também, um indivíduo do conceito B. Este é o principal conceito de ontologias, conhecido como subsunção.

Os conceitos podem se conectar para originar novos conceitos, os conectores podem ser binários ou unários. É possível expressar o conjunto que é o complemento de um conceito. Em DL, isto é feito através do operador da negação: $\neg A$. A união entre dois conceitos pode ser expressa através do conector $\sqcup$ entre dois conceitos. O resultado da união é um novo conceito que engloba o conjunto dos indivíduos do primeiro termo adicionado ao conjunto de indivíduos do segundo conceito. De forma análoga, o conectivo $\sqcap$ entre dois conceitos origina o conceito que reúne os indivíduos pertencentes ao primeiro e ao segundo conceito.

Indivíduos são entidades atômicas, no sentido de que eles representam a si, e não possuem uma conotação conceptiva de outros indivíduos. Assim, os indivíduos são sempre elementos que concretizam um determinado conceito. Portanto, um indivíduo é uma instanciação de um conceito. O axioma $a:A$ significa que o indivíduo a é uma instância do conceito A. Isso implica que as características dos indivíduos são definidas

por esta herança, assim como as possíveis relações com outros indivíduos que se restringem a determinados conceitos.

As **propriedades** relacionam um indivíduo a outro, no caso das propriedades entre objetos (*Object Properties*), ou um indivíduo a um tipo de dados, no caso das *Datatype Properties*. Assim como os conceitos, as propriedades também podem herdar as características de outras propriedades. Para expressar a herança entre propriedades em DL, é feita uma sobrecarga do operador *subsumes* (⊑), o qual já era utilizado para definir a hierarquia entre conceitos (e.g., $a \sqsubseteq b$, ilustrando que a propriedade *a* é subpropriedade da propriedade *b*). As propriedades de tipos de dados podem utilizar as literais do RDF e alguns tipos do XML *Schema*, como: *string*, *boolean*, *int*, *long*, *float*, *double*, *date*, entre outros. É importante ressaltar que as propriedades em OWL DL só relacionam indivíduos, embora sua definição seja feita ao nível de conceitos, visto que definimos a imagem e o domínio de uma aplicação utilizando os conceitos. Propriedades que relacionam conceitos entre si só estão disponíveis em OWL *Full*, na qual não é garantido o critério da decidibilidade.

Com o intuito de aumentar a capacidade de inferência sobre as propriedades, é possível definir características para enriquecer o significado destas funções. Tais características podem dizer respeito tanto à própria relação quanto à relação entre duas funções. Na Tabela 2 estão listadas e descritas as características de propriedades que podem ser expressas em OWL, sendo *R*, *R1* e *R2* propriedades quaisquer; e *a*, *b* e *c* indivíduos quaisquer.

Tabela 2. Características das propriedades.

Característica	*Significado*
Transitiva	se *R(a,b)* e *R(b,c)*, então *R (a,c)*
Simétrica	*R(a,b)* se e somente se *R (b,a)*
Funcional	se *R(a,b)* e *R(a,c)*, então $b = c$
Inversa	sejam *R1* e *R2* inversas, *R1(a,b)* se somente se *R2(b,a)*
Inversa Funcional	se *R(a,b)* e *R(c,b)*, então $a = c$

Além das características, também é possível atribuir **restrições** aos indivíduos relacionados por uma propriedade, definindo sua imagem de acordo com estas regras. O primeiro tipo de restrição é a **quantificação**, a qual pode ser expressa pelos operadores

para todo (∀) ou existe (∃). Tais operadores são utilizados em OWL adicionando-os como uma subclasse da classe domínio a ser restringida, e os construtores da linguagem utilizados para o universal e o existencial são: *allValuesFrom* e *someValuesFrom,* respectivamente. Estes quantificadores são adequados para especificar as imagens de propriedades. O axioma $\forall\ R.A \sqsubseteq B$ expressa que, para uma dada relação R que tenha como primeiro termo um indivíduo conceituado por B, terá como segundo termo um indivíduo conceituado por A.

O segundo tipo de restrição consiste em especificar uma **cardinalidade** para as relações. Com isso, é possível limitar o número de elementos que satisfazem uma relação sobre um determinado domínio e imagem. Os axiomas $\geq n\ R$, $\leq n\ R$ e $= n\ R$ expressam as restrições no mínimo *n* elementos, no máximo *n* elementos e exatamente *n* elementos, respectivamente. A cardinalidade em OWL *Lite* é limitada a $n = 0$ ou $n = 1$, diferentemente da OWL DL, na qual *n* pode assumir o valor de qualquer inteiro positivo.

Como citado anteriormente, este trabalho utilizará OWL DL como linguagem de representação do conhecimento. Portanto, na Tabela 3 é apresentada a lista dos axiomas em DL utilizados neste trabalho, bem como seus respectivos significados. Além disso, este trabalho utiliza a *Protégé Tool* [25] como ferramenta para manipulação de ontologias.

Tabela 3. Notação DL utilizada neste trabalho.

DL	*Significado*
$\boldsymbol{A \sqsubseteq B}$	Todo indivíduo que é *A* é *B* ou propriedade A é subpropriedade de B
$\boldsymbol{a : A}$	*a* é um indivíduo da classe *A*
$\boldsymbol{\forall\ R.A}$	Todos os indivíduos sucessores (imagem) da relação *R* são *A*
$\boldsymbol{R(a,b)}$	*a* está relacionado com b pela relação *R*
$\top$	Super classe *thing*
$\bot$	*bottom*
$\boldsymbol{\neg A}$	Não *A*
$\boldsymbol{A \sqcap B}$	*A* e *B*

2.4 Ontologias na Engenharia de Software

Embora a pesquisa sobre ontologias visando o crescimento da própria área seja o principal foco da comunidade de representação do conhecimento, é dada também importância à aplicação destes modelos em outras áreas da Ciência da Computação. No caso da Engenharia de Software, existem possibilidades de se aplicar ontologias em diferentes áreas. Em [25] são definidas quatro diferentes áreas, listadas a seguir, onde as ontologias são geralmente utilizadas na Engenharia de Software.

- **Desenvolvimento Dirigido por Ontologias:** utiliza ontologias em tempo de desenvolvimento e estas são empregadas para descrever o domínio em questão no desenvolvimento. As ontologias nesta área têm o papel de modelos que são parte do desenvolvimento do produto em si. A Engenharia Dirigida por Modelos é um exemplo de área que utiliza as ontologias neste contexto. Um exemplo desta área é descrito em [26].
- **Desenvolvimento Facilitado por Ontologias:** também faz uso das ontologias em tempo de desenvolvimento, porém estas são utilizadas para dar suporte aos desenvolvedores em suas tarefas. Como exemplo, ontologias podem ser utilizadas para facilitar a busca por componentes de software. Em [27] são adicionadas ontologias em ferramentas de desenvolvimento de software para esse fim.
- **Arquiteturas Baseadas em Ontologias:** utiliza a ontologia como um artefato primário em tempo de execução. Neste caso, a ontologia forma a parte central da lógica da aplicação. Abordagens de regras de negócio são um exemplo da utilização de ontologias neste contexto. Um exemplo comum é definir a arquitetura de um framework baseada em ontologias, como realizado em [28].
- **Arquiteturas Facilitadas por Ontologias:** fazem uso das ontologias para prover um suporte de infraestrutura em tempo de execução. Um exemplo são os serviços web semânticos. Nestes serviços, as ontologias provêm uma camada semântica no topo das descrições de serviços existentes. Com isso, é adicionada a característica de descobrimento, composição e pareamento automático de workflows baseados em serviços. Em [29] essa ideia é

utilizada para adicionar uma camada semântica na descrição de serviços multimídia.

Além dessas quatro áreas apresentadas, as ontologias podem ser utilizadas para integrar modelos. Essa integração pode ser chave para suprir a necessidade de se rastrear os artefatos envolvidos na criação de um software. Os trabalhos relacionados descritos na Seção 3.5 são exemplos da inserção de ontologias na Engenharia de Software.

2.5 Conclusão

Este capítulo apresentou os principais conceitos inerentes às ontologias. As motivações para o surgimento das ontologias foram introduzidas, assim como as principais linguagens historicamente utilizadas. Os tipos de ontologias foram elucidados e listados, assim como seus usos na Engenharia de Software. Finalmente, foi exposta a linguagem para descrição de ontologias que será utilizada nesta dissertação, a OWL DL.

É possível constatar que existe uma relação custo versus benefício quando se trata de linguagens que implementam ontologias. Mais expressividade resulta em maior complexidade para processar um modelo ontológico e retornar uma resposta para uma pergunta de decisão. A OWL DL é o dialeto OWL que apresenta maior expressividade, mas sem perder a decidibilidade, sendo esta a principal razão para sua adoção neste trabalho, seguida da disponibilidade de ferramentas.

Essencialmente, este trabalho busca adicionar semântica em uma LPS utilizando novos conceitos e relações entre eles, justificando, portanto, a adequação do uso das ontologias. O próximo capítulo discute sobre as LPS e a notação mais comumente utilizada para representar o seu domínio que é o diagrama de características. Assim, no próximo capítulo é possível analisar a adequação do uso das ontologias em vista da falta de expressividade da notação de características.

3. Linhas de Produto de Software

Neste capítulo é descrito o paradigma que serve como base para fundamentar o trabalho proposto, as Linhas de Produto de Software (LPS), as quais são introduzidas e definidas na Seção 3.1. A Seção 3.2 apresenta o processo da Engenharia de Domínio, bem como suas motivações e suas principais etapas. Em seguida, na Seção 3.3, é discutido o processo de Engenharia da Aplicação. Complementando, na Seção 3.4, é apresentado a modelagem de características como principal notação utilizada para representar o domínio e sua variabilidade. Finalmente, uma conclusão é feita na Seção 3.5.

3.1 Introdução

Para introduzir com sucesso a reutilização de software em uma empresa é necessário considerar técnicas e métodos que auxiliem esta inserção. Em particular, existem questões organizacionais e de gerenciamento relacionadas às técnicas e processos de reutilização e à comunicação entre eles. Existem questões pertinentes a como institucionalizar as novas tecnologias, gerenciar riscos, realizar planejamento e rastrear mudanças [3]. Ainda, é preciso catalogar questões como: análises de mercado e de novas tecnologias, a fim de determinar quais funcionalidades são necessárias e quais poderão vir a ser necessárias. É preciso ter métodos para avaliar se em determinado momento, para determinadas funcionalidades é melhor adquirir, desenvolver ou terceirizar artefatos [30]. Também se fazem necessárias estratégias para medir e aperfeiçoar o processo de desenvolvimento e suas interações. Por fim, é importante também ter técnicas de gerenciamento de configuração e técnicas de implementação eficientes.

Portanto, com o objetivo de promover uma abordagem que conciliasse as técnicas de Engenharia de Domínio e as questões de produção específicas de uma organização, as Linhas de Produto de Software sobressaíram-se perante as outras abordagens de reutilização. Isso se deve ao fato das LPS contemplarem os demais

paradigmas de reutilização, visto que utilizar uma linha de produto é também fazer uso, por exemplo, da Engenharia de Domínio e muitas vezes de *frameworks*, *middlewares* e desenvolvimento baseado em componentes.

O SEI (*Software Engineering Institute*) define uma LPS como sendo um conjunto de sistemas de software intensivos que compartilham um conjunto de características comuns gerenciadas, que satisfazem necessidades específicas de um segmento ou missão particular de mercado e que são desenvolvidos a partir de um conjunto comum de ativos em uma maneira predefinida [31].

De acordo com [32], e como ilustrado na Figura 3, uma LPS deve conter três principais atividades. A primeira é o processo de desenvolvimento do núcleo de artefatos reutilizáveis, também chamada de Engenharia do Domínio (ED). Nessa fase é feita definição, análise e implementação de ativos reutilizáveis segundo o domínio no qual a LPS atua. Embora esteja contida em uma LPS, a ED é estudada antes da existência do paradigma de linhas e provisiona a base para as LPS. A segunda atividade é o desenvolvimento do produto, ou Engenharia da Aplicação, fase na qual é derivado um produto específico utilizando o máximo de artefatos reutilizáveis da LPS. A terceira atividade é a de Gerenciamento da LPS. Essa atividade é a responsável pela devida execução das duas primeiras atividades, coordenando a comunicação entre ambas e solucionando conflitos que possam existir entre, ou dentro de cada uma.

Figura 3. Processos principais das LPS (Adaptado de [32]).

O fluxo cíclico de cada atividade na Figura 3 expressa a continuidade dessas atividades. O desenvolvimento de artefatos reutilizáveis é paralelo ao desenvolvimento de novos produtos, e a sobreposição dos círculos na figura expressa que as fases provêem insumos de uma para outra. A fase de Engenharia de Domínio fornece para a engenharia de aplicação, além dos ativos reutilizáveis em si, o plano de produção da LPS. Esse plano descreve como os ativos podem ser utilizados para gerar novos produtos. Em contrapartida, a engenharia de aplicação fornece *feedback* para a ED, indicando possíveis artefatos para compor o núcleo de ativos reutilizáveis, ou avaliando um ativo utilizado. Uma descrição mais detalhada das atividades citadas será feita nas próximas seções.

3.2 Engenharia de Domínio

A reutilização de software é feita pelo uso de partes de aplicações existentes, ou de aplicações completas, para gerar novas aplicações, ao invés de gerá-las do início [33]. Tais partes de software evoluíram desde a década de 60, onde a unidade de abstração era basicamente a sub-rotina, até os componentes já na década de 90, passando por módulos (anos 70) e objetos (anos 80) [3], e também com a estruturação das soluções para problemas recorrentes em padrões de software [34]. Podemos concluir que aplicações que compartilham de partes reutilizadas também são capazes de processar informações em comum, ou seja, dividem características inerentes à parte de software compartilhada. Assim, o fato de existirem diversas aplicações que se assemelham no que diz respeito a funcionalidades, nos leva a pensar em um agrupamento de aplicações similares. Esse agrupamento é a principal motivação de estudo da Engenharia de Domínio, e é definido, nessa área, como domínio.

Será adotada uma definição bem formulada e aceita de domínio, a qual afirma que um domínio é uma área de conhecimento que foi delimitada para maximizar a satisfação dos requisitos dos *stakeholders*, que inclui um conjunto de conceitos e terminologias entendidas pelos profissionais da área e que inclui o conhecimento de como construir sistemas de software ou partes deles na referida área [4]. Sendo uma área de conhecimento, o domínio diz respeito a um nicho determinado de software ou partes dele, e deve ser delimitado para que se possa definir um escopo de atuação. Essa delimitação se inicia ao visualizarmos onde artefatos deste domínio irão atuar. Se tal

domínio contiver artefatos de software que serão usados como partes de sistemas e em diferentes aplicações (como bancos de dados e links de comunicação), este domínio é chamado de horizontal. Se tal área de conhecimento contiver sistemas completos que compartilham características em comum, como o caso de aplicações para controle de estoque, por exemplo, este domínio é chamado de vertical [35].

Pode-se considerar que existe uma necessidade de cada vez mais diminuir o retrabalho, tanto pelo fato da busca por minimizar custos, quanto pela procura por agilidade no desenvolvimento de aplicações em uma organização, tornando-a assim mais competitiva. Essa necessidade leva as organizações a definirem uma área de atuação de mercado, tendo em vista dominar aquela fatia e sobressair-se em determinado campo. Ao fazer isso, a organização também define um domínio para si como forma de criar expertise em uma determinada área. Portanto, podemos notar que, similar ao que aconteceu na passagem de objetos para componentes, o domínio vem se mostrando como uma nova unidade de abstração no que diz respeito à reutilização de software.

A Engenharia de Domínio é geralmente dividida em três fases principais, as quais são: análise, projeto e implementação do domínio [36], [37]. Tais fases estão ilustradas na Figura 4 e descritas nas subseções seguintes.

Figura 4. Fases da Engenharia de Domínio.

3.2.1 Análise do Domínio

Introduzida por Neighbors em 1980 [38], a análise do domínio é a primeira fase da ED. Segundo esse autor, é nela que serão identificados os objetos e operações de uma classe de sistemas similares em um domínio particular. O principal objetivo a ser alcançado como resultado da análise de domínio é um modelo de representação coerente do domínio, o qual é obtido primeiramente pela seleção e definição do domínio alvo e em seguida pela captura e agrupamento de suas informações relevantes. Segundo Czarnecki (2000), um modelo de domínio é uma representação explícita das

propriedades comuns e variáveis dos sistemas que estão contidos no presente domínio, assim como a semântica de tais propriedades e dos conceitos do domínio, bem como as dependências entre as propriedades variáveis [4].

Para termos um modelo de domínio precisamos, primeiramente, defini-lo. Como exemplo, podemos definir o domínio de comércio eletrônico como todas as aplicações que são capazes de realizar venda de produtos pela internet. Em seguida, é desejável que seja obtido o vocabulário do domínio, ou seja, termos específicos utilizados por profissionais daquela área. Também se faz necessário modelar conceitos do domínio em diagramas, como exemplo um diagrama de fluxo de dados. Por último, é fortemente desejável que um modelo de domínio possua uma forma de expressar a sua variabilidade, e isso é usualmente obtido através de um diagrama de características do domínio. Esse diagrama, primeiramente introduzido em [8], é um modelo representativo dos atributos das aplicações do domínio que estão diretamente relacionados e visíveis ao usuário final (tais atributos são chamados de características). Apesar de ser a primeira definição de característica, a definição de Kang não proporciona a generalidade pretendida para tais conceitos. Assim, é mais aceita a definição generalizada de que são características distintas de conceitos relevantes para qualquer *stakeholder* [35] [63].

Em [39] podemos encontrar um estudo sobre atividades da análise de domínio que foram levantadas baseadas em oito trabalhos sobre essa fase da ED. Este estudo foi compilado e está representado na Tabela 4.

Tabela 4. Atividades da análise de domínio. Baseada em Arango (1994) e Czarnecki (2000).

Atividade Principal	Sub-atividades
Definição do escopo do Domínio	Seleção do domínio.
	Descrição do domínio.
	Identificação de fontes de informação.
	Preparação do inventário.
Modelagem do Domínio (Agrupamento de Dados)	Definição de Abstrações.
	Captura do conhecimento de especialistas do domínio.
	Revisões da Literatura.
	Análises de contextos e cenários.
Modelagem do Domínio (Análise dos Dados)	Identificação de entidades, operações e relacionamentos.
	Modularização.
	Análise de similaridades.
	Análise de variações.
	Análise de combinações.
	Análise de conflitos.
Modelagem do Domínio (Classificação Taxonômica)	Agrupamento de descrições.
	Abstração de descrições.
	Generalização de descrições.
	Construção do vocabulário.
Modelagem do Domínio (Avaliação)	Avaliação do Modelo do Domínio.

3.2.2 Projeto do Domínio

Depois de desenvolvido o modelo de domínio, é necessário definir como tais informações estarão dispostas nas futuras aplicações que serão originadas a partir daquele domínio. Para isso, é necessário projetar uma estrutura base para representar um "esqueleto" do domínio, e isso é feito através da geração de uma arquitetura do domínio. Na engenharia de aplicação, é neste artefato que se descreve os componentes e suas interações [40]. No caso de um domínio, a arquitetura não representará apenas os

conceitos de uma única aplicação, mas sim de todas as aplicações daquele domínio. Com isso, é importante que a arquitetura do domínio seja flexível para sustentar acoplamento ou desacoplamento de novos conceitos.

Definida a estrutura base das aplicações pertencentes ao domínio, é necessário prover as informações de como estas aplicações poderão ser derivadas. O plano de produção é o artefato mais indicado para essa tarefa. Nele estará descrito como os ativos reutilizáveis, produzidos na etapa de implementação do domínio, serão utilizados para produzir uma nova aplicação [41].

3.2.3 Implementação do Domínio

Também chamada de realização do domínio, a fase de implementação é a etapa em que será formado, de fato, um núcleo de ativos reutilizáveis, já iniciado na fase de projeto com os diagramas da linha, que foram identificados na fase de análise. Em uma organização, tais ativos podem ser desenvolvidos, comprados, reutilizados da própria empresa, ou poderão ser produzidos por empresas terceiras[30]. O fato é que, nesta fase, serão implementados os componentes, a arquitetura e o plano de produção descritos nas fases de projeto e análise. Ainda, à medida que novas aplicações forem derivadas utilizando os artefatos produzidos na implementação do domínio, poderão ser identificados novos artefatos que devem participar do núcleo de artefatos do domínio, havendo assim um *feedback* entre a engenharia do domínio e da aplicação.

3.3 Engenharia da Aplicação

A Engenharia da Aplicação é o processo no qual é construído o software baseando-se nos resultados obtidos na Engenharia de Domínio[4]. Durante a análise dos requisitos de aplicações, as quais estão no domínio já previamente analisado, são explorados os modelos já existentes e as características já mapeadas como requisitos reutilizáveis. Esse processo pode ser assistido por ferramentas de gerenciamento de pedidos do usuário, as quais podem auxiliar o usuário na descrição dos requisitos do sistema a ser desenvolvido.

Os requisitos não disponíveis no modelo do domínio, ou simplesmente não desenvolvidos e não armazenados no núcleo de ativos, terão de ser implementados. Assim, os novos requisitos desenvolvidos para um sistema em particular deverão ser

sincronizados com o domínio, se tais requisitos particulares forem considerados comuns a outros sistemas do mesmo domínio. Portanto, é comum que haja um *feedback* entre as duas engenharias com o intuito de aperfeiçoar o modelo e os artefatos do domínio. É na Engenharia de Aplicação que se monta manualmente os componentes ou de forma automática (através de geradores de códigos e auto-configurações). A Figura 5, extraída de [9], ilustra o desenvolvimento do produto na Engenharia de Aplicação a partir dos artefatos da Engenharia de Domínio.

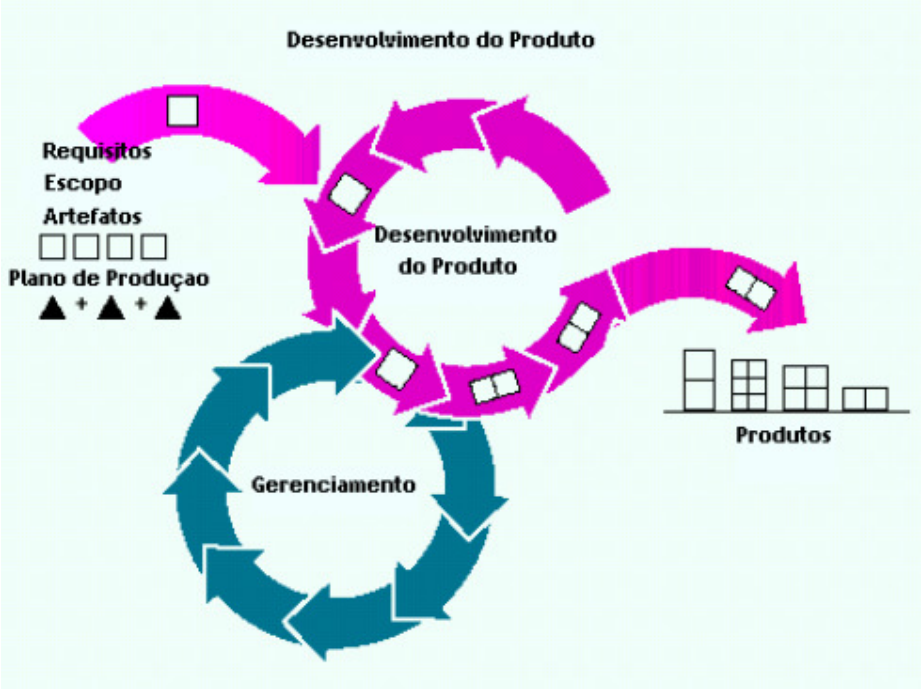


Figura 5. Desenvolvimento do produto a partir dos artefatos da Engenharia de Domínio.

De forma geral, a engenharia de aplicação é composta pelos seguintes sub-processos [42]:

- **Engenharia de Requisitos da Aplicação:** são identificados os requisitos do cliente para um determinado produto, e é desenvolvido um mapeamento entre estes requisitos e o modelo do domínio.
- **Projeto da Aplicação:** é feita a especificação em nível arquitetural da aplicação a ser derivada, espelhando-se na arquitetura de referência desenvolvida. Possíveis relações entre os componentes, e os próprios componentes, são explicitados no modelo arquitetural.

- **Realização da Aplicação:** são selecionados e configurados os componentes que farão parte de uma derivação de produto, bem como desenvolvidas novas funcionalidades até então não contempladas.
- **Testes da Aplicação:** são desempenhados testes de verificação e validação do produto, levando em consideração sua especificação e as necessidades do cliente.

De uma maneira similar, porém mais específica, em [9] são destacadas atividades dentro da engenharia de aplicação:

- **Análise baseada no modelo de domínio:** visa analisar um modelo de domínio, na maioria das vezes um diagrama de características, verificando o quanto esse modelo representa as necessidades do cliente em termos dos artefatos reutilizáveis do núcleo, no que diz respeito à arquitetura genérica e aos outros componentes disponíveis. Um modelo de decisão pode estar relacionado a esta atividade. Normalmente, as necessidades do cliente não estão totalmente cobertas no modelo de domínio, e, portanto, não existem ativos reutilizáveis que cubram estas necessidades, sendo assim necessário implementar do início novos componentes. Ao verificar a necessidade de desenvolver um novo componente não contemplado na fase de ED, e visto que este tem potencial de ser reutilizado, a engenharia de aplicação fornece o *feedback* indicando esta necessidade.
- **Instanciação da arquitetura do produto:** um dos produtos provenientes da fase de ED é a arquitetura genérica, a qual representa a arquitetura base das aplicações contidas no domínio em questão. Portanto, para derivar um novo produto, é necessário instanciar a arquitetura genérica de forma a gerar uma arquitetura específica para o determinado produto, mas que esteja em conformidade com a arquitetura genérica.
- **Povoamento da arquitetura:** refere-se à escolha dos ativos reutilizáveis que irão compor o novo produto, ou seja, a realização do que foi especificado na arquitetura específica da aplicação.

As duas divisões de fases, de [42] e de[9] buscam sistematizar a Engenharia de Aplicação, porém esta fase é dependente da organização na qual está inserida. Uma

organização que utilize sua própria metodologia de desenvolvimento de software deverá adaptar sua metodologia nesta fase.

3.4 Modelo de Características

Com o objetivo de representar o conjunto de características comuns e variáveis de um determinado domínio, o modelo de características, primeiramente introduzido em [8], é um modelo hierárquico de conceitos que se estendem em formato de árvore. As características são, no contexto da Engenharia de Domínio, representações de requisitos reutilizáveis e configuráveis que são relevantes para um *stakeholder* [4]. Quanto à variabilidade, as características são classificadas em quatro tipos:

- **Mandatória**: característica que deve estar presente em todos os produtos derivados a partir da linha. É geralmente representada em diagramas por um círculo preenchido, ou simplesmente o nome da característica.
- **Opcional**: característica que pode ou não estar presente nas derivações de produtos da linha. É comumente representada em diagramas por um círculo vazado.
- **Alternativa:** característica que, para estar presente na derivação de um produto, requer exclusividade. Assim, as características alternativas irmãs possuem um relacionamento mutuamente exclusivo no que diz respeito à seleção destas para derivar um produto. As características alternativas são representadas por um arco vazado entre elas.
- **Ou:** característica que pode ser escolhida juntamente com outras características irmãs para uma derivação de produto. É geralmente representada por um arco fechado ou pela ausência de arco entre as características desse tipo.

A Figura 6 mostra um exemplo de diagrama de características[1] expresso na notação precursora proposta em [8]. As características *Via Sensor* e *Via Serviço Externo* são alternativas e, portanto, mutuamente exclusivas. *Persistência* e *Mostrar*

[1] Este exemplo é baseado na LPS *Mobiline*, descrita em [62] e apresentada na Seção 5.1 desta dissertação.

Compatibilidade de Perfil são características opcionais. As demais características são obrigatórias e fazem parte de uma composição do tipo *Ou*.

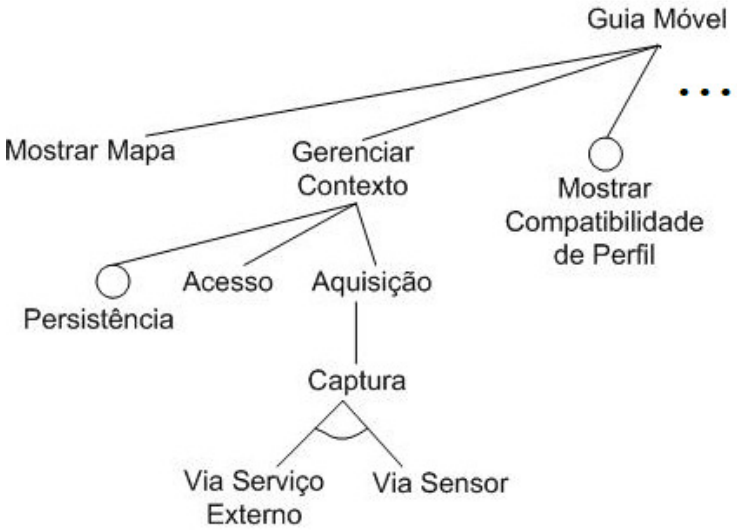


Figura 6. Exemplo de modelo de características na notação proposta em [8].

Outra notação para expressar a variabilidade das características é a proposta em [43]. Esta notação é baseada em cardinalidade, contudo são mantidos os conceitos de opcionalidade e obrigatoriedade e as noções de exclusão mútua e composição "ou". Para cada ponto de variação, ou seja, para a característica pai de um grupo, são atribuídos mínimos e máximos, referindo-se ao número mínimo e máximo de características que podem ser selecionadas em determinado ponto de variação.

Em analogia ao modelo inicial em [8] (FODA), podemos concluir que as características com cardinalidade máxima e mínima igual a 1, são consideradas características alternativas, visto que apenas uma delas poderá ser escolhida em uma derivação, expressando assim a exclusão mútua. A Figura 7 ilustra o mesmo modelo de características da Figura 6, porém na notação de [43] e desenvolvido através da ferramenta proposta em [44].

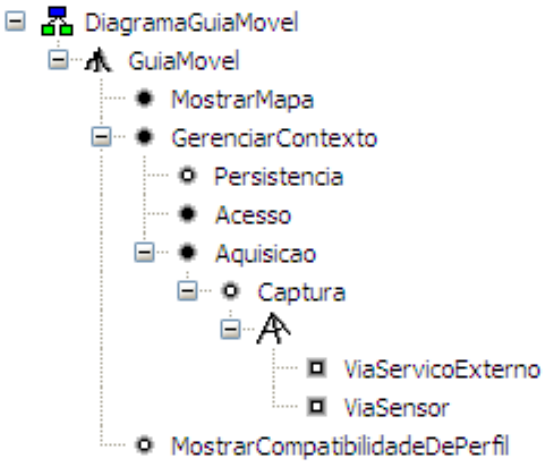


Figura 7. Exemplo de modelo de características na notação descrita em [43].

O modelo de cardinalidade ainda sugere a utilização de relações restritivas entre características que não estão no mesmo ramo. Isso é necessário para expressar a obrigatoriedade de uma característica ser selecionada, ou excluída, ao passo que outra seja escolhida. Como estão em ramos diferentes do diagrama, não é possível utilizar o conceito de características alternativas. Portanto, são utilizadas regras do tipo *requer* e *exclui* para expressar a dependência entre características de ramos diferentes.

3.5 Trabalhos Relacionados

Alguns trabalhos também constataram deficiências quanto a não adequação dos diagramas de características para expressar mais peculiares do domínio. Nestes trabalhos são apresentadas desde discussões sobre essa inadequação, até propostas de substituição destes diagramas e abordagens que utilizam ontologias agregadas ao desenvolvimento de software de uma forma geral, como é descrito a seguir.

Em [15] é descrita a relação entre modelos de características e ontologias. O autor discute a essência dos modelos de características, destacando seu caráter hierárquico e de variabilidade. Após a exploração do cerne do modelo, é apresentada, na forma de espectro, a capacidade semântica entre o modelo de características e as ontologias, concluindo assim que as ontologias são notoriamente mais expressivas que tais modelos. A contribuição central do artigo é a de propor que os modelos de características podem ser interpretados como visões em ontologias. Para concluir isso, o autor apresenta exemplos em modelos de características e suas equivalências em

ontologias. Contudo, o mapeamento direto entre as duas notações não é explorado, ficando claras as relações sintáticas e semânticas dos dois modelos, porém não deixando claro uma correspondência direta entre eles.

Em [26] é proposta uma abordagem para a modelagem de variabilidade em LPS guiada por processo e baseada em ontologias. O trabalho limita-se ao paradigma de orientação a serviços como base na linha de produto. A abordagem é dividida em três fases. A primeira é a construção de um modelo ontológico de domínio, o qual tem como objetivo expressar conceitos básicos do domínio e relacionamentos entre eles. A segunda é uma modelagem de uma ontologia de processo, que é um meta-modelo que representa a estrutura de processos de negócios. A terceira é a modelagem ontológica de serviço, na qual é definido um meta-modelo de descrição de serviços *web*. A grande desvantagem do trabalho em questão é que o ponto inicial para modelagem do domínio na LPS é a ontologia de domínio.

Com o exemplo dado no artigo [26], fica claro que a ontologia do domínio desenvolvida é apenas uma reescrita dos diagramas de características, pois nela somente é expresso um tipo de relação e os conceitos são distribuídos também de forma hierárquica, ou seja, características compostas por características filhas. Portanto, dois pontos podem ser questionados sobre o uso do modelo ontológico como ponto inicial: o primeiro é que não haveria necessidade desse modelo, visto que este representa a mesma semântica que o diagrama de características (da maneira que foi desenvolvido, sem acréscimo de semântica); o segundo ponto é que, além de não apresentar melhoria semântica sobre os diagramas de características, a ontologia substituiria o diagrama de característica, não sendo o ideal, visto que este é largamente utilizado pela comunidade de LPS.

Em [45] é introduzida uma abordagem ontológica para a Engenharia de Domínio. A abordagem é baseada na definição de ontologias para representar o domínio em questão, juntamente com os devidos mapeamentos para modelos de objetos e componentes Java para formar a infraestrutura do domínio. Este mapeamento é composto por um conjunto de diretivas, padrões de projeto e restrições. Assim como em [26], em [45] as ontologias são consideradas como artefatos iniciais para a modelagem do domínio. Além disso, a abordagem é voltada especificamente para a Engenharia de

Domínio, sem levar em consideração sua inserção em uma LPS, onde questões organizacionais estão presentes.

Em [27], é introduzida a ideia de se utilizar ontologias para acrescentar semântica em ambientes de Engenharia de Software, tendo em vista construir ambientes semânticos de Engenharia de Software. O principal objetivo é criar *links* semânticos entre artefatos de ambientes de Engenharia de Software de forma a criar ontologias que os descreva. As mesmas limitações encontradas em [45] e em [26] também podem ser reconhecidas em [27]. A abordagem é um esforço inicial para a utilização de ontologias em ambientes de Engenharia de Software, tendo um caráter geral e desprezando especificidades organizacionais.

A principal inovação apresentada nos trabalhos [26][45][27] é a busca por acrescentar semântica no desenvolvimento de software. A identificação de que é necessário melhorar a descrição dos artefatos em um ambiente de Engenharia de Software é o principal fator motivacional para estender as abordagens para as linhas de produto de software. A especialização da abordagem de LPS, feita nesta dissertação, deve ser feita considerando os esforços anteriores dos referidos trabalhos. Porém, é necessário considerar a reutilização também de notações e processos já consolidados, os quais são essenciais como insumos para o acréscimo de semântica.

Em [46] foi desenvolvida uma ontologia base para formalizar a especificação de modelos de características. O foco do trabalho é prover um framework ontológico para que se possa realizar checagem de consistência do modelo e detecção de conflitos através de regras pré-definidas. Além disso, os autores destacam o ganho em integração entre modelos que o framework proporciona. A ontologia desenvolvida foi originada através de um estudo sobre os principais modelos de características existentes. Visando reutilizar a especificação feita pelos autores, esta será considerada no desenvolvimento do mapeamento automático, descrito na Seção 3. A principal diferença entre o trabalho relacionado e o presente trabalho é que o relacionado visa obter uma formalização dos modelos de características para que se possa validar e checar estes modelos. Já o trabalho a ser proposto utiliza essa formalização para as regras de mapeamento, utilizando a especificação já proposta na construção da *top-ontology*.

Embora sejam relevantes os esforços dos trabalhos relacionados apresentados nesta seção em relação à associação de ontologias ao processo de Engenharia de Domínio, especialmente para modelar variabilidade, estes trabalhos ou não contemplam o aspecto das limitações semânticas dos modelos, ou não se preocupam em reutilizar os modelos de variabilidades na construção da ontologia de domínio e simplesmente descartam os modelos convencionais de variabilidade. Portanto, o diferencial do trabalho proposto será considerar os modelos existentes e acrescentar descrições semânticas de forma menos intrusiva do que modificar o modelo de características.

A Tabela 5 apresenta um resumo da comparação dos trabalhos relacionados, considerando as seguintes características: se reutilizam o diagrama de características, se apresentam algum mapeamento entre esse diagrama e a ontologia, se foi considerada a adição de novas informações para enriquecer a linha e, por último, se são voltados para o paradigma de LPS ou são para reuso de uma forma geral.

Tabela 5. Comparação dos trabalhos relacionados.

	Reuso do Diagrama	*Mapeamento*	*Enriquecimento*	*LPS*
Czarnecki [15]	Sim	--	--	Sim
Bu-qing [27]	--	--	--	Sim
Zaid [48]	Sim	Manual	--	Sim
Wang [49]	Sim	Manual	--	Sim
Falbo [28]	--	--	Sim	--
Trabalho Proposto	Sim	Automático	Sim	Sim

3.6 Conclusão

Este capítulo introduziu os conceitos básicos sobre as Linhas de Produto de Software, as quais se apresentam como um importante paradigma de reutilização de software, tanto pela sua sistematização, quanto pela generalidade no que diz respeito à reutilização de software. Foi dado destaque ao processo de Engenharia de Domínio, visto que este é a base do paradigma de LPS. Ainda, foi descrito o processo de Engenharia da Aplicação, elucidando o processo de derivação de um software em uma LPS. O modelo de características foi introduzido, apresentando-se como o modelo mais comumente utilizado nas LPS para representar a variabilidade do domínio. Finalmente,

foram apresentados trabalhos relacionados ao desenvolvido nesta dissertação, selecionados baseados na adoção de ontologias como ferramenta em abordagens de Engenharia de Software, mais especificamente, em abordagens de reutilização.

As LPS têm sido alvo de estudos em diferentes áreas da Engenharia de Software, porém ainda são necessários esforços para aumentar sua utilização, no que diz respeito à organização e representação das informações da linha. Os modelos comumente utilizados apresentam restrições de expressividade e de suporte de ferramentas para busca e inferência de informações, tendo em vista que foram concebidos apenas para utilização e visualização humana, como é o caso dos modelos de características descritos neste capítulo. Em contrapartida, o Capítulo 2 apresentou as ontologias como uma alternativa para representação do conhecimento, levando em consideração a recuperação da informação e a inferência. Portanto, o Capítulo 4 abordará a proposta deste trabalho de integrar as ontologias às LPS, sendo esta integração menos intrusiva do que a feita nos trabalhos relacionados.

4. Inserção de Semântica em LPS

Neste capítulo, a principal contribuição desta dissertação é apresentada. Primeiramente, na Seção 4.1, é introduzida a proposta de se representar novas informações, além das referentes à variabilidade, em uma LPS por meio de ontologias. Esta inserção de informações origina o termo Linha de Produto de Software Semântica. Na Seção 4.2, é descrito o mapeamento Fea2Onto que define o ponto de partida para a inserção de semântica em uma LPS. O enriquecimento semântico, propriamente dito, é descrito na Seção 4.3 e é realizado a partir de um meta-modelo, o SPLiSEM. Finalmente, a Seção 4.4, apresenta conclusões e considerações sobre o enriquecimento semântico em LPS.

4.1 Introdução

No Capítulo 1, foram apresentadas motivações para se enriquecer semanticamente uma LPS. Transformar as informações da LPS em conhecimento explorável foi uma das motivações apresentadas. Essa motivação é originada do fato de que as informações sobre o domínio são modeladas, principalmente, em diagramas de variabilidades, de forma que não estão em uma notação favorável para a recuperação de informações. Ainda, os diagramas de características expressam bem a variabilidade, entretanto não apresentam semântica suficiente para expressar outras informações relevantes ao processo de Engenharia de Domínio, como exemplo, as informações entre características e artefatos da LPS.

Um engenheiro de domínio de uma LPS detém muito mais conhecimento sobre relações entre características e entre artefatos reutilizáveis da linha do que o que se pode representar nesses diagramas. Grande parte desse conhecimento pode ser traduzida para uma representação compreensiva por máquinas (do inglês, *machine-readable form*), no sentido de que exista uma infraestrutura disponível para que o sistema computacional interprete e "raciocine" sobre esse conhecimento.

Portanto, a contribuição central deste trabalho é prover uma abordagem para enriquecer o conhecimento explorável por máquina de uma LPS. Primeiramente, almeja-se traduzir o conhecimento que esteja expresso em notações não-exploráveis, como é o caso do diagrama de características, em uma ontologia. Em seguida, um modelo é proposto para definir a inserção de novas informações que antes não estavam expressas em diagramas, como é o caso do conhecimento detido pelo engenheiro de domínio sobre relações entre artefatos da linha e características dos produtos. Com a ontologia gerada ao final da aplicação da abordagem em uma dada LPS, esta se torna uma Linha de Produto de Software Semântica (LPSS), termo novo introduzido nesta dissertação.

Uma LPSS é definida como: uma LPS onde artefatos do núcleo de ativos reutilizáveis e artefatos da organização estão relacionados com o modelo de domínio e expressos em uma ou mais ontologias. O ganho em expressividade ao utilizar as ontologias nas LPS através da inclusão de diferentes relações e conceitos é constatado com a aplicação em uma LPS nesta dissertação. Este modelo de representação possibilita, também, a inferência automática de novas relações e de atributos dos artefatos e das características. Além disso, os ganhos na recuperação das informações e na rastreabilidade são outros fatores determinantes para adotar as LPSS.

A abordagem de enriquecimento é dividida em dois métodos, os quais serão as bases para o enriquecimento semântico de uma LPS. O primeiro método é um mapeamento automático de um modelo de características para uma ontologia e é intitulado Fea2Onto. O modelo de características é adotado por ser amplamente utilizado na comunidade de LPS. O produto da aplicação do Fea2Onto é uma ontologia com a mesma semântica explícita e mesma quantidade de conceitos e relações do diagrama de características, e, por ser uma ontologia inicial, é chamada de esqueleto OWL (tecnologia de representação de ontologias utilizada, cuja escolha foi justificada no Capítulo 2). O segundo método é um enriquecimento semântico assistido por um modelo chamado *Software Product Line Semantic Enrichment Model* (SPLiSEM). O SPLiSEM é uma *top-ontology* que define a natureza das relações e conceitos que enriquecerão o esqueleto OWL gerado no primeiro método.

A Figura 8 ilustra a aplicação dos métodos propostos nesta dissertação em uma LPS, partindo de um modelo de características e aplicando os passos necessários.

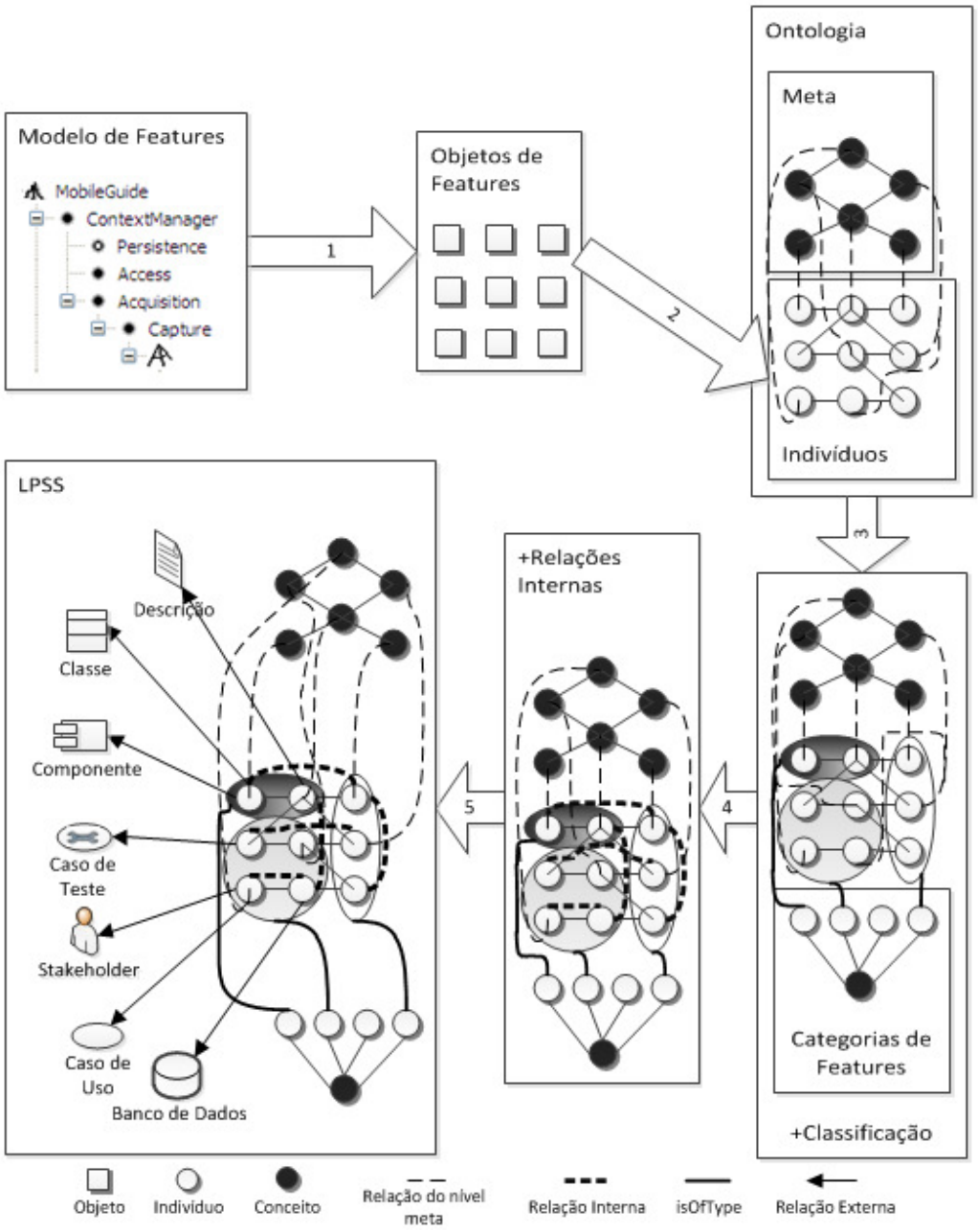


Figura 8. Passos da abordagem de enriquecimento semântico.

O passo 1 é um mapeamento direto entre um modelo de características para objetos, no caso deste trabalho objetos em Java. É feita uma tradução do modelo de características para objetos autocontidos, nos quais não existem relações entre os objetos, porém atributos que indexam as relações entre os objetos. Esse modelo de características em Java funciona como um modelo de objetos intermediário. A partir do conjunto de objetos é feita uma tradução (passo 2) para a linguagem OWL-DL, baseando-se em uma meta-ontologia proposta nesta dissertação, a qual representa a

TBox da ontologia . Esse passo gera indivíduos que são instâncias de conceitos da meta-ontologia proposta. Os passos 1 e 2 constituem o mapeamento Fea2Onto, descrito em detalhes na Seção 4.2. Esses dois passos foram automatizados na ferramenta Fea2Onto.

Os próximos passos não são automáticos e necessitam do engenheiro de domínio para sua realização que é baseada no SPLiSEM, descrito em detalhes na Seção 4.3. O passo 3 consiste na adição de informações classificativas às características, de forma a conseguir criar categorias de características. No passo 4, o engenheiro de domínio define relações entre as próprias características que não foram expressas no modelo de variabilidade tradicional, elas são chamadas de relações internas. Finalmente, no passo 5 o engenheiro de domínio acrescenta relações entre as características e outros conceitos da LPS, como casos de uso, componentes, entre outros. Os passos 3, 4 e 5 formam a fase de enriquecimento semântico assistida pelo SPLiSEM. Após a aplicação de todos os passos, espera-se obter uma LPSS na qual seja favorável buscar, inferir e rastrear as informações que interessem nos processos de Engenharia da Aplicação e Gerenciamento da linha.

4.2 Fea2Onto

Para tornar o conhecimento do domínio disponível para ser explorado, o primeiro passo é prover uma tradução para uma notação formal de descrição do conhecimento que, no caso deste trabalho, é utilizado o dialeto OWL DL. Como a maioria dos trabalhos de LPS utiliza diagramas de variabilidades para expressar o conhecimento do domínio em forma de características, é utilizado o modelo de variabilidade como base da ontologia a ser gerada. Ainda, com base no suporte de ferramentas, foi escolhido o modelo de variabilidades proposto em [43]. Este modelo foi implementado em um *plug-in* para a plataforma Eclipse em [44], chamado *Feature Modeling Plug-in* (FMP), permitindo expressar a variabilidade da linha em características e relações de cardinalidade entre elas.

O mapeamento Fea2Onto consiste, portanto, em uma transformação exógena (entre modelos em notações diferentes) de um modelo de características baseado em cardinalidade para uma instância de uma ontologia representada em lógica descritiva através da linguagem OWL DL. Essa ontologia funciona como um modelo em DL no qual são definidos os conceitos e relações para os quais serão mapeadas as entidades do

modelo de características. Os conceitos criados são referentes ao conceito de características e seus tipos de acordo com sua variabilidade. Por exemplo, o conceito *Feature* na ontologia representa uma característica do diagrama de variabilidade.

As categorias de características estão definidas em (1) como filhas do conceito *Feature*. As características *mandatórias* são as que devem estar presentes em qualquer derivação de produto, e são representadas na ontologia pelo conceito *MandatoryFeature*. Já as *opcionais* podem estar ausentes ou presentes em uma determinada derivação, sendo representadas pelo conceito *OptionalFeature*. As características do tipo "*ou"* podem ser selecionadas em uma derivação sem excluir a seleção das características irmãs do mesmo tipo. Este conceito está intitulado na ontologia como *OrFeature*. As características *alternativas* são mutuamente exclusivas e são representadas pelo conceito *AlternativeFeature*.

$$
\begin{aligned}
&\textit{Feature} \sqsubseteq \top \\
&\textit{MandatoryFeature} \sqsubseteq \textit{Feature} \\
&\textit{OptionalFeature} \sqsubseteq \textit{Feature} \\
&\textit{OrFeature} \sqsubseteq \textit{Feature} \\
&\textit{AlternativeFeature} \sqsubseteq \textit{Feature}
\end{aligned} \tag{1}
$$

As propriedades que relacionam as características, de forma hierárquica, estão definidas no meta-modelo em (2). Para expressar que uma característica é pai de outra, a ontologia proposta contém a relação *consistsOf*, filha da propriedade genérica da OWL *ObjectProperty,* e esta tem como domínio e imagem o conjunto de características.

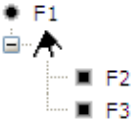
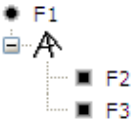
$$
\begin{aligned}
&\textit{consistsOf} \sqsubseteq \textit{ObjectProperty} \\
&\forall\ \textit{consistsOf.Feature} \sqsubseteq \textit{Feature} \\
&\textit{hasOrFeature} \sqsubseteq \textit{ObjectProperty} \\
&\forall \textit{hasOrFeature.OrFeature} \sqsubseteq \textit{Feature} \\
&\textit{hasAlternativeFeature} \sqsubseteq \textit{ObjectProperty} \\
&\forall\ \textit{hasAlternativeFeature.AlternativeFeature} \sqsubseteq \textit{Feature}
\end{aligned} \tag{2}
$$

A relação *consistsOf* é transitiva e possui como inversa a propriedade *consistedBy*. Outra relação, a *hasOrFeature,* indica que uma determinada característica é pai de uma *OrFeature*, da mesma forma que a relação *hasAlternativeFeature* expressa que uma característica é pai de uma *AlternativeFeature.*

As transformações entre o modelo de variabilidade e a ontologia em DL são descritas na Tabela 5. A primeira coluna indica o símbolo na notação do diagrama de

características no FMP, a segunda contém o significado do símbolo e a terceira o axioma correspondente em lógica descritiva.

Tabela 6. Mapeamento de FMP para DL.

Símbolo FMP	***Significado***	***DL***
	*Feature* raiz *F*	*F* : *Feature*
	*Feature* mandatória *F*	*F* : *MandatoryFeature*
	*Feature* opcional *F*	*F* : *OptionalFeature*
	F2 é *subfeature* de F1	*consistsOf(F1, F2)*
	Composição OU	*F2* : *OrFeature* *F3* : *OrFeature* *hasOrFeature(F1,F2)* *hasOrFeature(F1,F3)*
	Composição alternativa	*F2* : *AlternativeFeature* *F3* : *AlternativeFeature* *hasAlternativeFeature(F1,F2)* *hasAlternativeFeature(F1,F3)*

4.2.1 Implementação do Fea2Onto

Para concretizar a automatização proposta pelo Fea2Onto, uma ferramenta baseada neste mapeamento foi implementada. A Fea2Onto *Tool* é uma ferramenta que automatiza as transformações entre os modelos FMP e OWL DL da Tabela 5.

Um diagrama de características desenvolvido no FMP gera um arquivo *.fmp* baseado em XML. Neste arquivo está descrita a estrutura do diagrama, em seu nível meta, assim como estão descritas as características adicionadas ao modelo e suas variabilidades. A hierarquia das características é descrita dentro das *tags* <*model*> do arquivo, dentro das quais estão descritas as *tags* <*children*>, representando, individualmente, uma característica. A hierarquia é definida pelo aninhamento das *tags* <*children*>, as quais, por sua vez, têm um número de identificação único. Através deste número de identificação, é possível recuperar as informações das respectivas características. Cada característica possui uma *tag* <*configuration*>, a qual possui o

referido número de identificação, juntamente às propriedades da característica. É nesta *tag* que se encontram o nome, informação de obrigatoriedade, cardinalidade e descrições da característica.

Portanto, o primeiro módulo considerado no desenvolvimento da Fea2Onto *Tool* foi o de leitura do arquivo gerado a partir do FMP. É este módulo o responsável por carregar um arquivo *.fmp*, realizar a leitura das características e suas propriedades e hierarquia, e instanciar objetos da classe *Feature*. A classe *FMPReader* foi criada para realizar essas tarefas, mais especificamente, os métodos *populateFeatures()* e *populateFeaturesProperties()* são responsáveis por ler a hierarquia das características e suas propriedades, respectivamente, e em seguida instanciar os objetos do tipo *Feature*, guardando-os em um *ArrayList* de *Features*. Esta leitura é feita utilizando a DOM API [47] para leitura de arquivos baseados em XML. Um algoritmo de busca em profundidade foi implementado no método *populateFeatures()*, o qual recebe como parâmetro o nó de partida da busca, para percorrer a hierarquia do modelo. Portanto, a classe *FMPReader* possui uma lista das características e suas propriedades lidas do modelo FMP. O método *getFeatureArray()* pode ser utilizado para recuperar a lista das características.

A classe *Feature*, ilustrada na Figura 9 na notação de diagrama de classes da UML, possui os seguintes atributos: *id* – número de identificação da característica no arquivo *.fmp*; *name* – nome da característica; *fatherId* – número de identificação do pai da característica, *fatherName* – nome do pai da característica; *isMandatory* – identificação de obrigatoriedade ou não da característica; *fatherIndex* – índice do pai da característica no *ArrayList* de *Features*; *isFeatureGroup* – identifica se a característica é um grupo "ou" ou "alternativa" na notação de cardinalidade ou se é apenas uma característica; *minGroup* – identifica a cardinalidade mínima do grupo "ou" ou "alternativa"; *maxGroup* – identifica a cardinalidade máxima do grupo "ou" ou "alternativa". Todos estes atributos são privados e possuem seus respectivos métodos *gets* e *sets*.

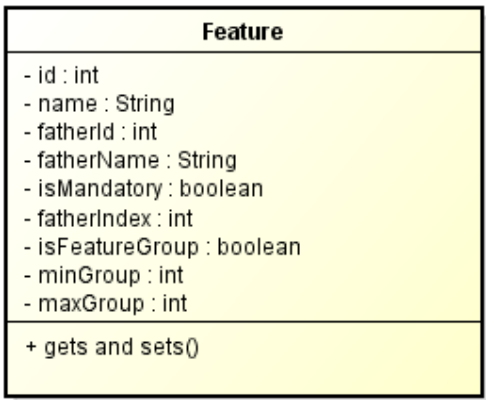


Figura 9. Classe *Feature* e seus atributos.

Com as características traduzidas para objetos, é possível percorrer linearmente o *ArrayList* de *Feature* e traduzir estes objetos para outro modelo, no caso deste trabalho para a ontologia representada em OWL DL. Para isso, foi implementada a classe *OWLWriter*, a qual utiliza a OWL API para gerar a ontologia. Primeiramente, é carregada a ontologia (TBox) e, em seguida, os objetos *Feature* são mapeados como indivíduos dos conceitos da ontologia (ABox), os quais foram definidos em (1). Finalmente, as relações hierárquicas entre os indivíduos, definidas em (2), são escritas como axiomas na ontologia.

Como visto anteriormente, a Fea2Onto *Tool* foi estruturada em três módulos principais: o de leitura, o núcleo e o de escrita. O núcleo é o modelo de características descrito em objetos auto-contidos e funciona como um modelo canônico entre o modelo fonte (FMP) e o modelo alvo (OWL DL). Assim, caso seja necessário adicionar novos modelos, pode-se apenas modificar o módulo de leitura ou escrita.

A Figura 10 mostra a interface da ferramenta, composta por dois campos. O primeiro campo é o caminho de diretório do modelo *.fmp* de entrada. O segundo campo deve conter o caminho de diretório no qual será salvo o arquivo *.owl* de destino. Ambos caminhos podem ser definidos a partir dos botões *Browse*, o qual abre uma janela para explorar os diretórios da máquina. O botão *Translate* realiza a tradução do modelo, que, depois de traduzido, é exibido na caixa de texto.

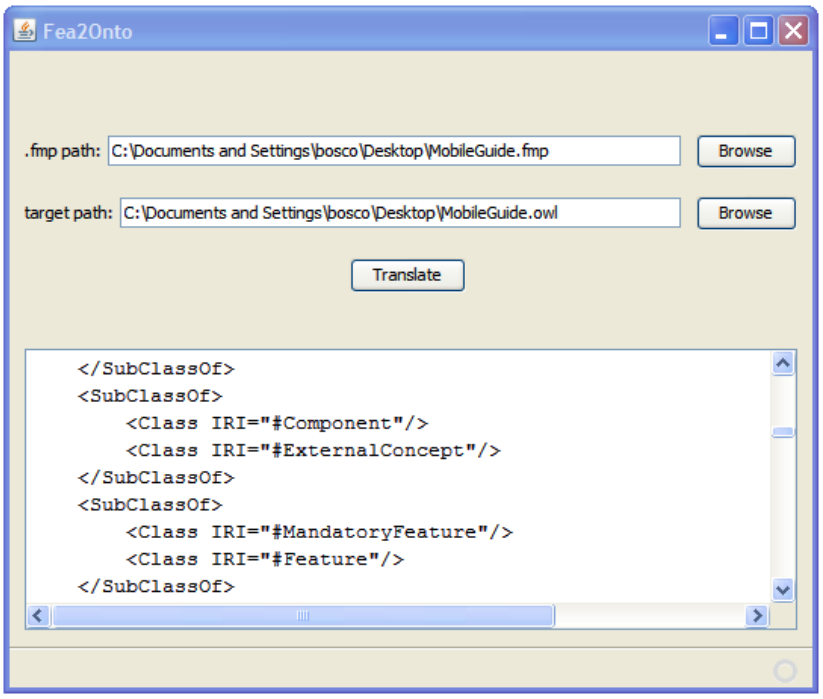


Figura 10. Interface da Fea2Onto *Tool*.

4.3 Enriquecimento utilizando o SPLiSEM

Ao tomar o modelo de características como base para a representação do conhecimento de uma LPS, assume-se que as características são as entidades principais do modelo de representação do conhecimento da linha, assim como as relações de hierarquia entre estas características serão os principais elos do modelo. Isso implica que as informações que serão adicionadas utilizando como base o SPLiSEM são complementares à variabilidade expressa pelas características e suas relações advindas do modelo de características. Portanto, considera-se que as informações relacionadas à variabilidade da linha já estão devidamente expressas e que não devem sofrer alterações decorrentes da adição das informações complementares. Nesta seção, será apresentado um modelo de enriquecimento semântico para inserção de conhecimento em LPS. Este modelo fornece a base para a adição de informações na ontologia da linha e é intitulado SPLiSEM.

Com o mapeamento automático feito a partir da Fea2Onto *Tool*, pode-se obter a base da ontologia da linha, porém as informações contidas no esqueleto OWL são as mesmas que já estavam presentes no diagrama de características. Mesmo já permitindo

inferências, nenhuma nova informação foi adicionada até então e o conhecimento disponível do domínio é o mesmo, porém expresso em outra notação.

Nas subseções a seguir, são descritas as novas informações que serão consideradas na fase de enriquecimento semântico.

4.3.1 Informações Classificativas

Agrupar um conjunto de características em determinada categoria é importante para obter classificações e identificar atributos em comum. Através da tipificação das características, é adicionada informação na linha de produto que pode servir para filtrar características por categoria, por exemplo, no momento da configuração de um produto. Concebeu-se, então, uma forma de representar tal informação na linha de produto.

A tipificação é feita baseada no tipo de informação que a característica representa. Conforme proposto em [48], uma característica pode ser classificada como:

- *Capacidade:* são características que representam uma função, operação ou serviço, que pode ser desempenhado pela aplicação.
- *Ambiente Operacional:* são características que representam um ambiente de execução ou mecanismo no qual uma aplicação é utilizada.
- *Tecnologia de Domínio:* são características utilizadas para representar algoritmos que implementam técnicas de um domínio específico.
- *Técnicas de Implementação:* similar a *Tecnologia de Domínio,* são características que representam implementações de técnicas, porém que podem ser utilizadas em diversos domínios.

Para inserir essa classificação de características no SPLiSEM, foram criados conceitos referentes aos tipos, juntamente com um conceito "pai" chamado *FeatureType*, como mostrado na sintaxe DL em (3).

$$\begin{aligned} &\mathit{FeatureType} \sqsubseteq \top \\ &\mathit{Capability}{:}\mathit{FeatureType} \\ &\mathit{OperationalEnvironment}{:}\mathit{FeatureType} \\ &\mathit{DomainTechnology}{:}\mathit{FeatureType} \\ &\mathit{ImplementationTechnique}{:}\mathit{FeatureType} \end{aligned} \tag{3}$$

Após definidos os tipos na ontologia SPLiSEM, é necessário prover uma propriedade que simbolize a relação entre uma determinada característica e uma

categoria específica, com o intuito de atribuir à característica um tipo. Não foi utilizado o operador ⊑, pois as entidades relacionadas são indivíduos, e não conceitos. Assim, definiu-se a relação *isOfType,* descrita em (4).

$$isOfType \sqsubseteq ObjectProperty$$
$$\forall isOfType.FeatureType \sqsubseteq Feature \quad (4)$$

Outras classificações podem ser feitas baseadas na natureza das características. Embora seja desejável que se mantenha o conceito original de característica proposto em [8], pode-se julgar necessária a adição de conceitos que, por definição, não são características. Um exemplo é o ConIPF [49], no qual é adicionado hardware, como sendo uma característica, ao modelo de variabilidades. É possível estender o SPLiSEM para incluir a categoria hardware de características, adicionando um indivíduo *Hardware* do tipo *FeatureType* ao modelo. Contudo, o mais adequado é que tal conceito seja tratado como externo, conforme é descrito na subseção 4.3.4 deste capítulo.

4.3.2 Relações de Tipos de Dados

Um engenheiro de domínio pode estar interessado em atribuir um valor de um tipo de dado a uma determinada característica. Tais tipos de dados são necessários quando é preciso adicionar informações sobre uma determinada característica. Especificamente, para estes tipos de relação, a natureza da informação a qual se deseja adicionar é bem conhecida e já possui uma estrutura pré-definida. Por exemplo, caso seja necessário acrescentar uma informação de data a uma característica, indicando a data de criação daquela característica no modelo, pode-se adicionar esta informação através de uma relação de tipos de dados entre a característica e o tipo de dado requerido. Os tipos de dados disponíveis em OWL podem ser encontrados em [7], alguns comumente utilizados são: *strings, booleans, int, float, date,* dentre outros.

Uma relação de tipos de dados pode ser definida no modelo como uma relação que herda as características das *DatatypeProperties* da OWL. O domínio de uma relação de tipos de dados é o conceito *Features*, em contrapartida, a imagem deve ser um tipo de dados, citados anteriormente. Em (5) são definidas as relações de tipos de dados, sendo *R* uma relação qualquer e *xsd:Type* os tipos disponíveis em OWL.

$$datatype \sqsubseteq DatatypeProperties$$
$$R \sqsubseteq datatype$$

$$\forall\ R.xsd{:}Type \sqsubseteq Features \tag{5}$$

4.3.3 Relações Internas

Como citado anteriormente, existem relações entre as características de uma linha que são complementares à variabilidade e que não podem ser expressas no modelo de características. A essas relações daremos o nome de internas. Na ontologia SPLiSEM as relações internas herdam as características do tipo *ObjectProperty* da OWL. Em (6), as relações internas estão definidas como as relações que possuem como domínio e como imagem o conjunto de características. Ainda, se relacionarmos qualquer indivíduo que não seja do conceito *Feature*, é obtido o vazio como imagem. *R* é uma relação qualquer.

$$\begin{aligned} &internal \sqsubseteq ObjectProperty \\ &R \sqsubseteq internal \\ &\forall\ R.Feature \sqsubseteq Feature \\ &\forall\ R.\neg Feature \sqsubseteq \bot \end{aligned} \tag{6}$$

No SPLiSEM, são propostos dois tipos de representação para relações internas: as restritivas e as informativas. Estes dois tipos são descritos a seguir.

As **relações internas restritivas** são, geralmente, utilizadas para expressar a inclusão ou a exclusão de um indivíduo B após a seleção de um indivíduo A. Portanto, são acrescentadas à ontologia SPLiSEM duas relações restritivas. A primeira adicionada é a relação *requires*, com o intuito de explicitar a dependência entre duas características, de forma simétrica ou não. A segunda relação é a *excludes*, ela significa a exclusão de uma segunda característica pela seleção de uma primeira. Como exemplo, uma característica responsável por trocas de mensagens requer uma característica responsável pela leitura e escrita de tais mensagens, e uma característica responsável pela alta mobilidade de sistemas celulares exclui uma característica totalmente centralizada de uma rede.

As regras restritivas também podem ser consideradas regras de variabilidade, porém a maioria das notações de variabilidade não apresenta semântica para expressar essas regras. Assim, essas relações são, geralmente, expressas separadas do próprio diagrama, como acontece em [8], [10] e [43]. Portanto, definimos em (7) as ***relações internas restritivas*** como filhas da propriedade *constraint,* a qual por sua vez herda as

características das propriedades internas. As formas de interpretação destas relações são restringidas, isso é feito com os dois últimos axiomas de (7), onde o primeiro restringe que só características que estão selecionadas podem requerer outras características selecionadas e o segundo que apenas características não selecionadas podem excluir características selecionadas. A propriedade *isSelected* é uma propriedade do tipo *datatype* e é definida pelo axioma $isSelected \sqsubseteq datatype$, expressando uma propriedade booleana de uma *Feature* que pode ser selecionada ($isSelected(Video, true)$) ou não ($isSelected(Video, true)$).

$$\begin{aligned}
&constraint \sqsubseteq internal \\
&requires \sqsubseteq constraint \\
&excludes \sqsubseteq constraint \\
&(\forall\ requires.Feature \sqcap \forall\ isSelected.true) \sqsubseteq (Feature \sqcap \forall\ isSelected.true) \\
&(\forall\ excludes.Feature \sqcap \forall\ isSelected.true) \sqsubseteq (Feature \sqcap \forall\ isSelected.false) \qquad (7)
\end{aligned}$$

As **relações internas informativas**, de forma oposta às ***relações internas restritivas***, são utilizadas para acrescentar informações ao modelo, mas sem alterar a sua variabilidade. Podemos, também, chamar essas relações de ***relações internas não-restritivas***. Na ontologia SPLiSEM, essas são definidas como sendo as relações que herdam das relações internas e que são o complemento das relações restritivas. Essa definição é feita utilizando os axiomas $informative \sqsubseteq internal$ e $informative \sqsubseteq \neg constraint$. É perceptível que a quantidade de relações que acrescentam informações não restritivas ao modelo de características que podem ser adicionadas à ontologia SPLiSEM é imensurável. Assim, a ontologia SPLiSEM se limita a prover um nível meta para a adição destas relações, porém serão discutidos limitadores para guiar a adição de relações informativas na Seção 4.3.5.

4.3.4 Relações e Conceitos Externos

As características são os conceitos centrais de uma LPS e, portanto, da ontologia de enriquecimento proposta. Até então, todas as informações já descritas no modelo de variabilidade e as informações adicionais tinham como principal objetivo descrever e relacionar apenas características entre si. Contudo, existem relações entre características e outros conceitos que não são características, os quais nós denominamos de ***conceitos externos***. Estes conceitos estão ligados diretamente ao processo de desenvolvimento de um novo produto da LPS, mas não são características.

As relações entre os conceitos externos e as características são chamadas na ontologia SPLiSEM de relações externas. Os ***conceitos e relações externas*** estão definidos na ontologia SPLiSEM em (8).

$$
\begin{aligned}
&\textit{ExternalConcept} \sqsubseteq \top \\
&\textit{ExternalConcept} \sqsubseteq \neg \textit{Feature} \\
&\textit{external} \sqsubseteq \textit{ObjectProperty} \\
&R \sqsubseteq \textit{external} \\
&\forall\, R.\ \textit{ExternalConcept} \sqsubseteq \textit{Feature} \\
&\forall\, R.\textit{Feature} \sqsubseteq \textit{ExternalConcept}
\end{aligned} \tag{8}
$$

Uma LPS pode apresentar diversos conceitos externos, dependendo da complexidade e do nível de maturidade da organização na qual a LPS se insere. Em geral, quanto maior o nível de maturidade da organização maior é o controle sobre o processo de desenvolvimento de software e maior é o número de artefatos utilizados. Os conceitos externos são entidades desses artefatos. Os conceitos e relações externas são adicionados na ontologia como descrito em (9), sendo *E* um conceito externo qualquer e *R* uma relação externa que tem como imagem *E* e *Feature* como domínio. Além disso, uma entidade *a* qualquer de um dado artefato, é um indivíduo do conceito que o define.

$$
\begin{aligned}
&E \sqsubseteq \textit{ExternalConcept} \\
&R \sqsubseteq \textit{external} \\
&\forall\, R.E \sqsubseteq \textit{Feature} \\
&a{:}E
\end{aligned} \tag{9}
$$

Portanto, a imagem das relações externas é definida de acordo com o conceito externo correspondente. Como um exemplo de conceito e relação externa, a relação *implementsUseCase* denota que uma entidade externa, no caso um *Caso de Uso* é implementado por um determinado indivíduo *Feature*. Ainda, o axioma *implementsUseCase*(*Authentication, Login*) indicaria que a característica *Authentication* implementa o caso de uso *Login.* Esse conceito e essa relação externa pode ser parte de uma extensão da ontologia SPLiSEM em uma LPS que utilize a notação de casos de uso.

4.3.5 Inclusão de relações no SPLiSEM

No momento em que um engenheiro de domínio está estendendo o SPLiSEM para sua LPS. Existem limitadores que devem ser considerados antes da adição de uma determinada relação no SPLiSEM. Um importante limitador é a categorização das

características. Em uma ontologia, as relações estão intrinsecamente ligadas aos seus domínios e imagens, assim, as categorias de características atuam como limitadores do conjunto de relações que podem ser adicionadas. Portanto, não é adequado adicionar à ontologia uma propriedade que relacione uma característica a outra se não houver categorias que conceituem ambas. Como exemplo, na assertiva (*Visualization*, *VisualInstitute*): *wasImplementedAt* é expresso, através da relação *wasImplementedAt*, que a característica *Visualization* foi implementada no local *VisualInstitute,* porém a imagem da propriedade, que é uma localização, não é considerada como uma característica. Portanto *wasImplementedAt* não é uma relação desejável no modelo, a não ser que *Location*:*FeatureType* fosse parte do modelo.

Outro fator limitador da adição de relações no SPLiSEM que deve ser considerado pelo engenheiro de domínio são as extensões e novas notações de diagramas de variabilidade existentes na literatura. Alguns trabalhos que estendem o diagrama de características inicial proposto em [8] acrescentam novas relações ao diagrama original. Um levantamento das técnicas de modelagem de variabilidade foi feito, visando identificar as relações adicionais destes modelos, com o objetivo de fornecer estas extensões na ontologia proposta. As técnicas analisadas foram: FODA[8], Odyssey-FEX[9], Morisio[10], Becker'02 [50], ConIPF [49], CONSUL [51], Kumbang [52], Ye'05 [53], Fey'02[54], van Deursen'02[55], KobrA [56], Reiser'07[57], Loesch'07[58], VPM[59]. A Tabela 7 lista técnicas de modelagem de variabilidade e suas respectivas propriedades adicionais identificadas. Apenas as técnicas que apresentaram relações além das relativas à variabilidade foram listadas, portanto técnicas que apresentaram apenas relações restritivas (*requires* e *excludes*) não constam na tabela.

Tabela 7. Técnicas de modelagem de variabilidade e suas relações adicionais.

Técnicas	***Relações Adicionais***
Becker'02	*complexity*
ConIPF	*has-context, has-features, has-hardware, has-subsystems, has-software*

CONSUL	*recommends, discourages*
Kumbang	*cardinality, value, present, instanceOf, hasInstances, hasPartOfType, connected*
Ye'05	*impacts*
Odyssey-FEX	*implementedBy, communicationLink*
Fey'02	*refine, providedBy, modify*
Loesch'07	*featureUsage*

A propriedade *complexity*, proposta em Becker'02 [50] é uma atribuição de um valor estimado a uma característica que é um ponto de variação, indicando um grau de complexidade para resolução de tal ponto em uma derivação.

Em Kumbang [52], a relação *cardinality* expressa quantas instâncias de uma determinada característica podem estar presentes em uma dada derivação de produto. Assim, essa relação pode ser definida na ontologia SPLiSEM como subpropriedade de *datatype* e sua imagem sendo os inteiros positivos. Este conceito de instância pode ser utilizado para expressar que determinada característica pode ser implementada como mais que um único módulo em uma aplicação. Por exemplo, um módulo que provê acesso a determinado dado pode limitar suas instanciações em um produto, visando garantir desempenho ou disponibilidade de um serviço.

As relações de ConIPF são utilizadas para expressar a composição de uma característica por outras. Para expressar a composição de uma característica com um determinado contexto, é utilizada a relação *has-Context.* Como esta relação simboliza que uma característica possui um determinado contexto definido como um conceito externo e, consequentemente, a relação definida como *external*. O mesmo acontece com as relações *has-hardware, has-subsystems* e *has-software*, porém, os conceitos externos são agora, respectivamente, hardware, subsistema e software, os quais podem ser adicionados ao SPLiSEM estendendo o conceito *Feature*.

Com o intuito de adicionar uma semântica análoga à das relações restritivas de inclusão e exclusão, porém sem o caráter obrigatório atribuído a estas, as relações

recommends e *discourages* são utilizadas em CONSUL [51]. Estas propriedades relacionam características, portanto, são de natureza interna e expressam a recomendação ou não da adição da característica objeto dada a inclusão da característica sujeito em uma derivação de produto. Outra variação das relações restritivas sem caráter obrigatório é a relação *impacts*, que também é interna e expressa um impacto em uma determinada característica devido à escolha de outra. Uma relação que pode ser considerada como uma especialização de *impacts* é a relação *modify*, expressando que uma característica, quando escolhida, modifica o comportamento ou estrutura de outra característica. Essas relações podem ser incluídas na ontologia como subpropriedades da propriedade *internal* definida pelo SPLiSEM.

Em Odyssey-FEX [9], as relações *implementedBy* e *communicationLink* também são internas e representam, respectivamente, que uma característica é implementada por outra e que uma característica possui um link de comunicação com outra. Com uma semântica similar à de *implementedBy*, a propriedade *providedBy* de Fey`02 [54], expressa que uma característica mais conceitual é implementada por outra mais concreta, a qual pode representar um serviço. A relação *refine* expressa que uma característica é uma especialização de outra, a qual representa mais especificidades da característica mais geral.

A propriedade *value*, relaciona um atributo a uma característica, geralmente, sendo uma descrição sobre a característica (e.g., uma definição ou utilização da característica). Portanto, *value* é uma *datatype* que tem como imagem uma *string*. A relação *present* é uma descrição booleana sobre a presença ou não de uma instância de uma determinada característica em uma derivação de produto, permitindo, portanto, identificar as características que foram instanciadas em um determinado produto. Caso seja adicionado o conceito de instância ao SPLiSEM, é possível reproduzir a relação *instanceOf*, a qual relaciona uma instância a uma característica. Adicionalmente, a relação *hasInstances* é uma *datatype* booleana que expressa a existência ou não de instâncias de uma determinada característica. *hasPartOfType* e *connected* são, também, *datatype* booleanas utilizadas para verificar se uma determinada instância tem uma parte transitiva e para verificar se uma dada instância está conectada a outra, respectivamente.

A relação *featureUsage* descreve o grau de utilização de uma determinada característica. Este grau pode ser: sempre usada, nunca usada, somente usada com

exclusão mútua, somente usada em conjunto e somente usada em separado. Assim, para implementar esta descrição em uma extensão do SPLiSEM, é recomendável definir *featureUsage* como filha de *datatype* e definir sua imagem como sendo inteiros positivos, os quais representam o grau de utilização da característica. Outra forma de implementar é definir os graus de utilização como indivíduos filhos de um conceito externo do tipo grau de utilização, sendo, portanto, *featureUsage* definida como uma relação externa.

Com o intuito de facilitar a definição de uma relação a ser adicionada no SPLiSEM, a Figura 11 mostra um diagrama de atividades da UML, o qual expressa o fluxo de decisão para classificar uma relação, sendo, portanto, um guia para o engenheiro de domínio que deseje estender o SPLiSEM.

Primeiramente, verifica-se a imagem da relação (já considerando que o domínio é o conjunto de características), dado que a natureza da imagem foi identificada, navega-se pelo fluxo do diagrama até a escolha do tipo de relação ideal. A *label* [estrutura bem definida] da figura se refere a imagens que podem ser definidas como um tipo de dados, como uma lista de números ou de nomes, ou datas, dentre outros. Já a *label* [inclui ou exclui] faz referência às relações que têm a semântica idêntica as de *includes* e *excludes*.

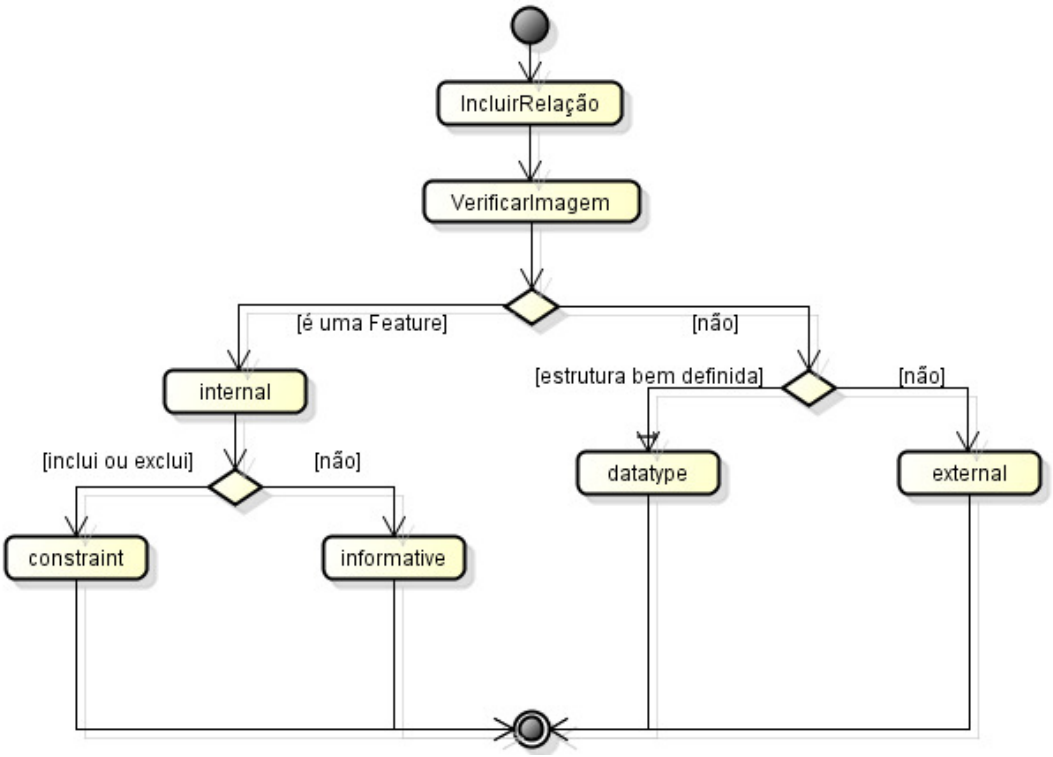


Figura 11. Diagrama de atividades para adição de relações ao SPLiSEM.

4.4 Conclusão

Este capítulo apresentou a abordagem de inserção de semântica em uma LPS. Foram explicados os dois métodos propostos para alcançar as LPSS, denominados Fea2Onto e enriquecimento via SPLiSEM. O primeiro método foi o mapeamento de um modelo de característica para um esqueleto OWL. Este método é importante para extrair a semântica já expressa pelo engenheiro de domínio em relação à variabilidade da LPS. Uma ferramenta para automatizar esse mapeamento foi desenvolvida e apresentada. Embora a ferramenta use modelos específicos no mapeamento, a sua modelagem permite o acoplamento de outros modelos, graças ao modelo canônico de objetos.

O segundo método apresentado foi o de enriquecimento semântico. Esse enriquecimento proporciona a inserção de novas informações acerca do domínio da LPS e de seus artefatos. O método de enriquecimento guia o engenheiro de domínio quanto à inclusão das informações, baseando-se na *top-ontology* SPLiSEM. Foram levados em consideração os tipos de relações que podem ser adicionadas: relações de tipos de dados, internas e externas, bem como novos conceitos diferentes das características, mas

que estão presentes na modelagem de uma LPS. Seguidamente, foram elucidados os limitadores para estender o SPLiSEM adicionando novas relações e conceitos.

O Fea2Onto pode existir sem o enriquecimento semântico utilizando o SPLiSEM, porém o contrário não é verdade, salvo se o diagrama de características for remodelado, manualmente, em OWL. Para ser possível adicionar semântica, é necessária a existência de um modelo alvo, no qual serão inseridas as novas informações. O SPLiSEM é um modelo que proporciona uma classificação de alto-nível dos conceitos e relações em uma LPS, tendo como objetivo principal guiar a adição de informações. A abordagem de se iniciar a partir do diagrama de características proporciona a separação da modelagem de variabilidades das informações adicionais sobre as próprias características ou sobre outros artefatos. No próximo capítulo, são analisados os benefícios da adoção da abordagem em uma LPS.

5. Estudo de Caso

Com o intuito de demonstrar a aplicação dos métodos propostos para o enriquecimento semântico, este capítulo apresenta um estudo de caso com base em uma LPS existente descrita na Seção 5.1 e desenvolvida utilizando o modelo tradicional de características. A Seção 5.2 descreve a aplicação da abordagem proposta, ilustrando a execução e o resultado de cada passo. Na Seção 5.3, são discutidos os benefícios da aplicação da abordagem de enriquecimento na LPS. Finalmente, a Seção 5.4 apresenta as conclusões sobre o estudo de caso.

5.1 LPS Mobiline

A LPS, na qual será aplicada a abordagem de enriquecimento semântico, é uma linha de guias de visita móveis e sensíveis ao contexto, descrita em detalhes em [60]. As aplicações que podem ser derivadas a partir dessa linha são destinadas a auxiliar um determinado visitante de um local específico (e.g., museu, laboratório ou parque), informando-o sobre descrições de itens destes locais (e.g., quadros, equipamentos ou monumentos) através de dispositivos móveis e utilizando informações do contexto, como a localização do usuário.

Os casos de uso da LPS de guias móveis foram elicitados na forma de serviços. Estes serviços são os requisitos de uma aplicação derivada da LPS de guias. Essa aplicação foi intitulada *GREat Tour* e está definida em [61]. Os serviços estão enumerados em códigos de caso de uso e seguidos por suas descrições a seguir:

- UC01 - Cadastrar Perfil Visitante: este serviço permite fazer cadastramento do perfil do usuário visitante.
- UC02 - Listar Roteiros de Visitas: este serviço lista os roteiros de visitação disponíveis. Alguns dos guias de visita móveis estudados no domínio usam informações contidas no perfil do visitante e sua localização para oferecer roteiros de visitação personalizados.

- UC03 - Registrar Roteiro de Visita: este serviço permite ao usuário efetivar a escolha de um roteiro de visitação. Após o roteiro ser cadastrado, a aplicação passa a guiar o visitante durante a visitação.
- UC04 - Recuperar Localização: este serviço faz o mapeamento da posição física do visitante para uma posição relativa dentro do ambiente de visitação. Esse mapeamento permite o sistema inferir quais itens (objetos), presentes no ambiente, estão próximos do usuário e, com base nessa informação, oferecer um detalhamento dos mesmos.
- UC05 - Recuperar Mapa: este serviço fornece uma visualização do local de visitação. Em geral, é utilizado em conjunto com o serviço Recuperar Localização onde o mapa fornecido exibe a localização do visitante.
- UC06 - Recuperar Eventos: este serviço lista os eventos que estão ocorrendo ou ocorrerão no ambiente de visitação. Esses eventos podem ser palestras, apresentações culturais, entre outros. Alguns dos guias de visita móveis fazem uso do perfil do visitante para sugerir os eventos mais apropriados.
- UC07 - Recuperar Perfil do Ambiente: este serviço recupera informações (perfil) do ambiente visitado. Em geral, este serviço provê uma descrição sobre o ambiente e indica se este possui objetos (itens) relevantes para visitação.
- UC08 - Recuperar Itens: este serviço permite fazer a listagem dos itens presentes no local de visitação. Um item pode ser uma obra de arte, um artefato histórico ou qualquer outro objeto presente em um local de visitação. Esse serviço pode fazer uso do perfil do usuário para restringir a exibição de itens.
- UC09 - Recuperar Perfil do Item: este serviço permite recuperar o perfil do item que se deseja obter informações.
- UC10 - Recuperar Texto: este serviço permite acessar um texto associado ao item desejado. Esse serviço pode fazer uso de informações de contexto para adaptar o texto as restrições do dispositivo móvel.
- UC11 - Recuperar Imagem: este serviço possibilita a recuperação de imagens relacionadas aos itens presentes nos locais de visitação. Esse tipo de serviço, em alguns dos guias de visita móveis estudados, faz uso de informações contextuais como o tamanho, resolução e a escala de cores do

display dispositivo móvel do visitante para adaptar a imagem às suas restrições.

- UC12 - Recuperar Vídeo: este serviço é responsável pela recuperação de vídeos relacionados ao item de interesse. Ele leva em consideração informações de contexto do dispositivo móvel (e.g., disponibilidade de biblioteca e memória livre) para adaptar ou negar a exibição do vídeo.
- UC13 - Exibir Vídeo: este serviço envolve a execução de um vídeo na tela do dispositivo, bem como as funcionalidades básicas de um *player*, como pausar, resumir e parar um vídeo.

A LPS de guias móveis também possui artefatos reutilizáveis de código que implementam suas características em forma de componentes de software. Estes componentes estão listados e descritos na Tabela 8.

Tabela 8. Componentes da SPL de guias móveis e suas descrições.

Componente	Descrição
EnvironmentHandler	Responsável por prover e gerenciar as operações relativas ao ambiente físico da visita.
EventHandler	Responsável por prover e gerenciar as operações relativas aos eventos que estão agendados para os ambientes e que são de interesse do visitante.
Guider	Responsável por gerenciar e prover as rotas de visitação.
ItemHandler	Responsável por prover e gerenciar as operações relativas aos itens contidos nos ambientes de visita e suas descrições textuais e de imagem.
Locator	Responsável por prover e gerenciar a informação de localização do visitante nos ambientes e por exibir essa localização no mapa.
VideoPlayer	Responsável por prover a recuperação e reprodução de uma mídia de vídeo no dispositivo móvel.
VisitorManager	Responsável por prover e gerenciar as operações relativas aos visitantes, como suas preferências.

Além da documentação relativa aos casos de uso do produto e dos componentes reutilizáveis da LPS, foram considerados no estudo de caso os casos de teste. Os casos de testes estão ligados às características de forma que a inclusão de uma determinada característica na derivação de um produto implica na execução de determinados casos de teste. A Tabela 9 apresenta os casos de teste utilizados na LPS de guias móveis, com seu código e descrição resumida.

Tabela 9. Casos de testes da LPS de guias móveis e suas descrições.

Código	Descrição
TC01	Inserir e excluir o perfil de um visitante.
TC02	Recuperar e mostrar o perfil de um visitante.
TC03	Inserir novo roteiro de visitação.
TC04	Recuperar e exibir um roteiro de visitação.
TC05	Acessar a localização interna do visitante.
TC06	Exibir mapa com localização do visitante e dos ambientes acessíveis.
TC07	Visualizar eventos disponíveis.
TC08	Visualizar sugestões de eventos para o usuário.
TC09	Acessar o perfil dos ambientes.
TC10	Exibir detalhes do ambiente.
TC11	Acessar o perfil do item de um ambiente.
TC12	Exibir detalhes do item.
TC13	Exibir descrição textual sobre ambiente e item.
TC14	Exibir imagens de um ambiente e item.
TC15	Tocar vídeo e pausar durante a execução.
TC16	Tocar vídeo e parar sua execução.
TC17	Parar a execução de um vídeo pausado.

Além das informações descritas anteriormente, os engenheiros de domínio e aplicação da LPS Mobiline dispõem do modelo de características dessa linha. O modelo de características original do Mobiline, especificamente de sua sublinha para guias móveis, foi modelado utilizando o *Odyssey-FEX* [9].

5.2 Aplicação da Abordagem na LPS Mobiline

Apesar de já estarem contidas na LPS Mobiline, as informações descritas na Seção 5.1 não estão especificadas de forma a facilitar sua consulta, visto que estão em documentos de texto ou diagramas semi-formais e não-computáveis. Com isso, o engenheiro de aplicação que deseje derivar um produto da linha encontrará dificuldades, tais como:

- Encontrar uma característica que esteja relacionada a determinados casos de uso, casos de teste ou componentes;

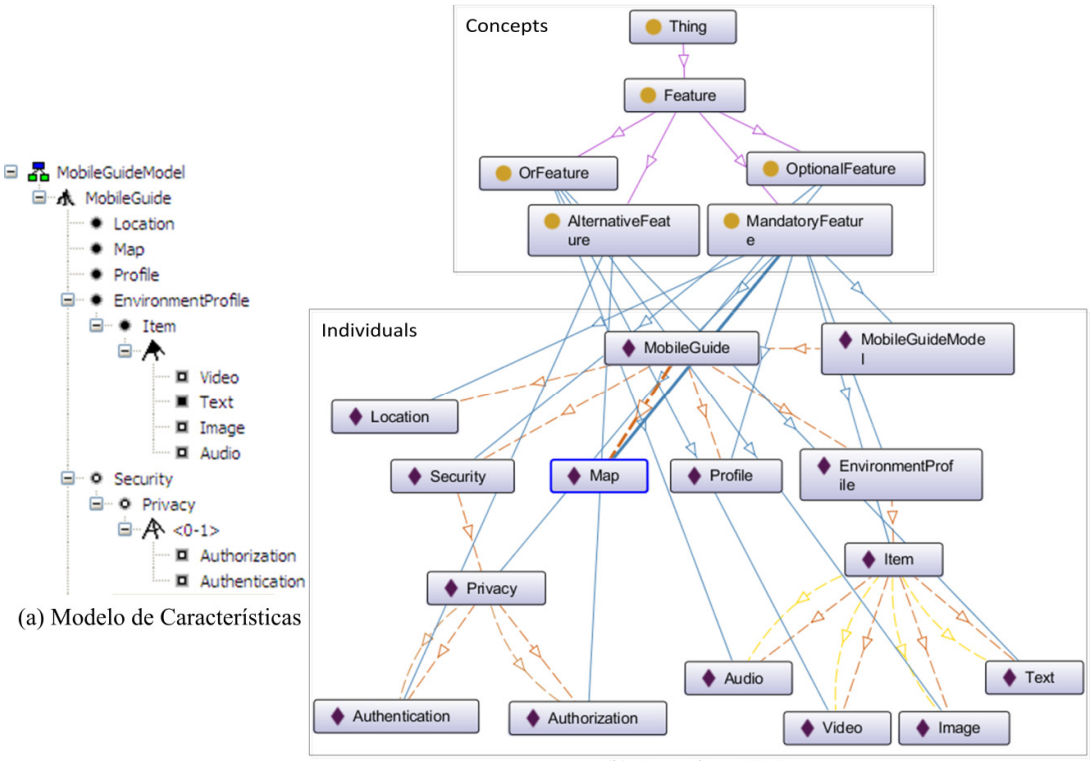
- Identificar características dependentes ou exclusivas entre si;
- Selecionar características por categorias, por exemplo, escolher apenas as características que são técnicas de implementação.

Portanto, a fim de facilitar tarefas como as citadas anteriormente, as quais são atribuídas ao engenheiro de aplicação, é aplicada a abordagem proposta à LPS Mobiline, estabelecendo conexões entre os conceitos da LPS descritos anteriormente e as características da linha.

Inicialmente, o diagrama de características da LPS de guias móveis descrito na notação Odyssey-FEX foi remodelado utilizando o *Feature Modeling Plug-in* [44] que funciona integrado à plataforma Eclipse. O *plug-in* produz um arquivo *.fmp* descrevendo a LPS que possui 42 características e 41 relações de hierarquia e variabilidade entre elas.

O arquivo *.fmp* é dado como entrada para a ferramenta proposta nesta dissertação: Fea2Onto *Tool*. Esta ferramenta fornece como saída o esqueleto *.owl* para ser enriquecido. A Figura 12 mostra um fragmento do modelo de características dado como entrada (a) e o modelo de saída em OWL (b). O modelo de saída é composto de instâncias dos conceitos representados na *top-ontology* proposta nesta dissertação (parte *Concepts* da Figura 9).

O arquivo *.fmp* da LPS está ilustrado no anexo A, enquanto que o arquivo *.owl* está renderizado na sintaxe funcional da OWL (esta sintaxe permite visualizar todos os conceitos e relações em instruções funcionais, o que diminui o espaço de visualização necessário) no anexo B. Os modelos também podem ser baixados em http://great.ufc.br/~splisem, assim como a ferramenta de mapeamento e os demais modelos gerados neste estudo de caso.



(a) Modelo de Características

(b) Esqueleto OWL

Figura 12. Fragmento do modelo de características e sua tradução em OWL.

Após o mapeamento, o método aplicado foi o de enriquecimento semântico. As primeiras informações acrescentadas do SPLiSEM foram as classificativas. Para cada característica, representada como um indivíduo do conceito *Feature* na ontologia, é utilizada a relação *isOfType* para relacionar uma característica ao seu tipo específico (e.g., *isOfType*(*Video, Capability*)).

Seguindo o método de enriquecimento utilizando o SPLiSEM, é necessário, agora, adicionar as relações internas. As primeiras relações internas consideradas foram as restritivas. Como exemplo, a característica *Location*, para ser selecionada em uma derivação, requer a presença da característica *Map*. Assim, foi adicionado ao modelo o axioma *requires*(*Location*, *Map*). Ainda, as relações informativas foram adicionadas observando a Tabela 7. Um exemplo de relação interna informativa adicionada foi referente a expressar uma recomendação para selecionar uma característica se outra estiver selecionada, como é o caso das características *Video* e *Text.* Se uma aplicação de guia de visita prover visualização de vídeos é recomendado que ela também disponha de um mecanismo para mostrar um texto que descreva ou pelo menos intitule o vídeo.

Assim, para expressar essa recomendação foi acrescentado o axioma *recommends*(*Video*, *Text*).

Para concluir o enriquecimento, os conceitos e relações externas foram incluídos. Neste passo, são adicionados à ontologia da linha os conceitos externos e suas possíveis relações com as características da LPS. Para o presente estudo de caso, foram considerados três conceitos externos (casos de uso, componentes e casos de teste), os quais foram devidamente inseridos no modelo de acordo com o SPLiSEM. Os indivíduos filhos desses conceitos representam artefatos existentes na LPS.

Para cada caso de uso descrito anteriormente, foi adicionado uma instância do conceito externo *UseCase* à ontologia da linha. Tais casos de uso são os elementos da imagem da relação *coversUseCase*, a qual é definida como subpropriedade da relação *external*.

Cada componente descrito na Tabela 8 foi adicionado à ontologia da linha como sendo uma instância do conceito externo *Component*. Estes indivíduos são elementos da imagem da relação *isComposibleBy,* uma subpropriedade da relação *external*.

Assim como nos casos de uso e componentes, para cada caso de teste descrito (Tabela 9) foi adicionado um indivíduo instância do conceito externo *TestCase* à ontologia da linha. Estes casos de teste são os elementos da imagem da relação *hasTestCase*, a qual também é definida como subpropriedade da relação *external*.

A Figura 13 ilustra o fragmento da ontologia da linha que descreve a característica *Video* com as relações entre ela e os indivíduos externos. A relação entre *Video* e *VideoPlayer* (*isComposibleBy*) simboliza que a característica *Video* pode ter como componente *VideoPlayer.* Já a relação *coversUseCase* de *Video* com *UC12* e *UC13* expressa que, uma vez selecionada e implementada a característica *Video,* os casos de uso de *ids* 12 e 13 serão cobertos. A relação entre *Video* e os casos de testes *TC15*, *TC16* e *TC17* é a propriedade *hasTestCase*, esses casos de teste são, respectivamente, referentes aos testes de tocar vídeo e pausar, tocar vídeo e parar, e parar vídeo pausado.

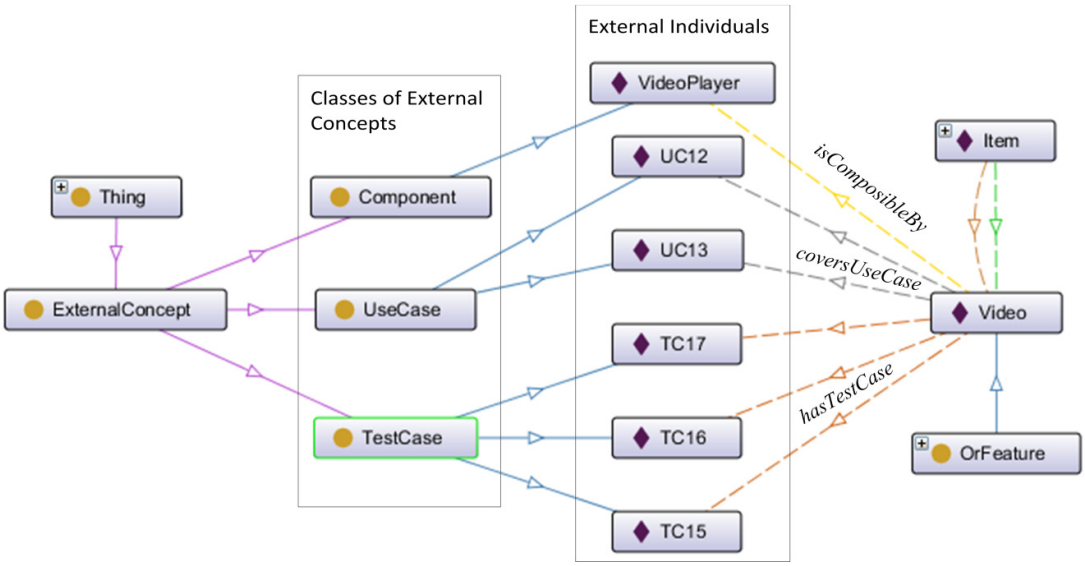


Figura 13. Fragmento da ontologia da LPS para a Feature *Video*.

5.3 Análise dos Benefícios

Ao final da aplicação do modelo de enriquecimento, foi obtida a ontologia da LPS de guias móveis e sensíveis ao contexto. A ontologia da linha possibilita benefícios como: **recuperação de informações**, **inferência** e **rastreabilidade** entre características e conceitos de artefatos. A seguir, serão analisados estes três aspectos benéficos resultantes da aplicação da abordagem de enriquecimento semântico na LPS de guias móveis.

5.3.1 Recuperação de Informações

Com o modelo de características descrito em OWL é possível realizar consultas e obter resultados, o que antes não era possível no modelo *.fmp*. Essa recuperação é facilitada pelas linguagens de consulta existentes. Nesse estudo de caso, usamos a própria *DL Query*, presente na distribuição do Protégé 4.1 [62]. A Figura 14 mostra uma consulta no fragmento do modelo Figura 12(a) e seu resultado. A *query* (*hasAlternativeFeature some Feature*) *or* (*hasOrFeature some Feature*) produz como resultado os indivíduos que são pontos de variação no modelo (*Item* e *Privacy*), ou seja, características que apresentam possibilidade de variação durante a derivação de um produto. A capacidade de recuperar essa informação através de uma consulta é útil em modelos com um grande número de características, proporcionando eficiência a essa busca. De uma forma geral, é possível utilizar as consultas em algoritmos específicos

que utilizem as respostas das consultas. Antes, não era possível obter essas respostas facilmente, visto que a notação de representação do diagrama não possuía um mecanismo de busca atrelado, como acontece com a ontologia.

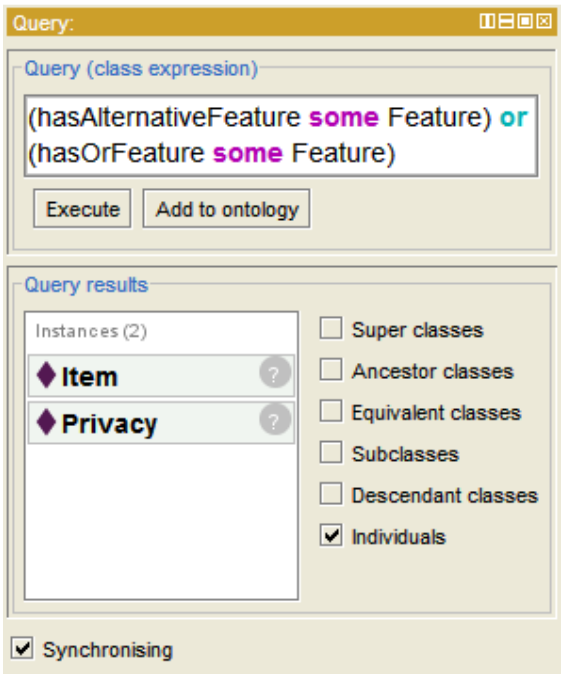


Figura 14. Consulta por pontos de variação.

5.3.2 Inferência

Algumas informações que não foram explicitadas no modelo de enriquecimento podem ser inferidas pela ontologia através de um *reasoner*. Neste estudo de caso foi utilizado o *reasoner HermiT* 1.2.4 também proveniente do Protégé 4.1. Um exemplo de inferência para o modelo da Figura 12(b) é que, se a característica *Authentication* for escolhida em uma determinada derivação, a característica *Security,* apesar de não estar diretamente relacionada com *Authentication,* deverá ser selecionada para também fazer parte da derivação. Essa inferência é feita pela transitividade da relação *consistedBy*, a qual é a inversa da relação *consistsOf.* Portanto, a Figura 15 mostra o resultado da consulta pelas características que são pais ou avós da característica *Authentication*. Inferências mais complexas podem ser introduzidas, como exemplo, é possível inferir que uma vez escolhida a característica *Video* para fazer parte de uma derivação, será recomendável incluir os casos de teste da característica *Text*. Como descrito anteriormente, a característica de vídeo recomenda a inclusão da característica de texto,

e, por transitividade das relações, são incluídos os casos de teste da característica texto, caso a recomendação seja aceita.

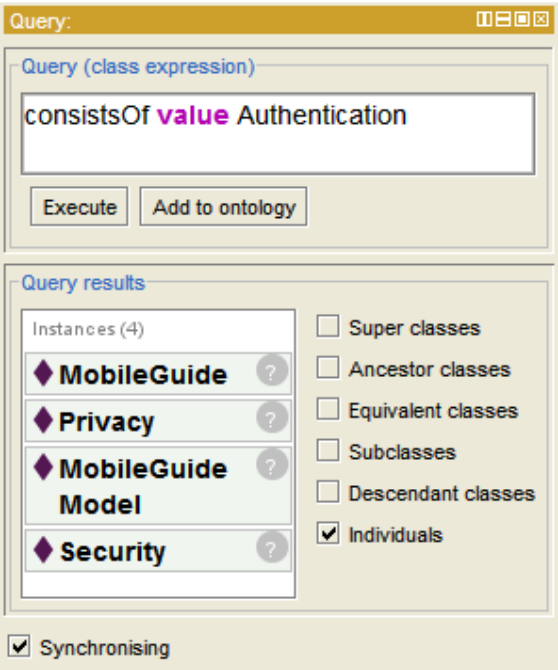


Figura 15. Consulta por características do mesmo ramo.

5.3.3 Rastreabilidade

Dada a aplicação do modelo de enriquecimento, pode-se constatar que relações antes não expressas entre características e artefatos da linha são agora descritas no modelo. Através destas relações são criadas conexões entre os artefatos e suas respectivas características, formando assim um rastro entre eles. Com o fragmento mostrado na Figura 16, podemos verificar que a característica *Video* está relacionada com os três tipos de conceitos externos: componente, caso de uso e caso de teste. Com isso, é possível manter o rastro de qual componente foi utilizado para contemplar os casos de uso *UC12* e *UC13,* que no caso foi o componente *VideoPlayer*. Ainda, é possível verificar que esses casos de uso são testados pelos casos de teste *TC15*, *TC16* e *TC17*.

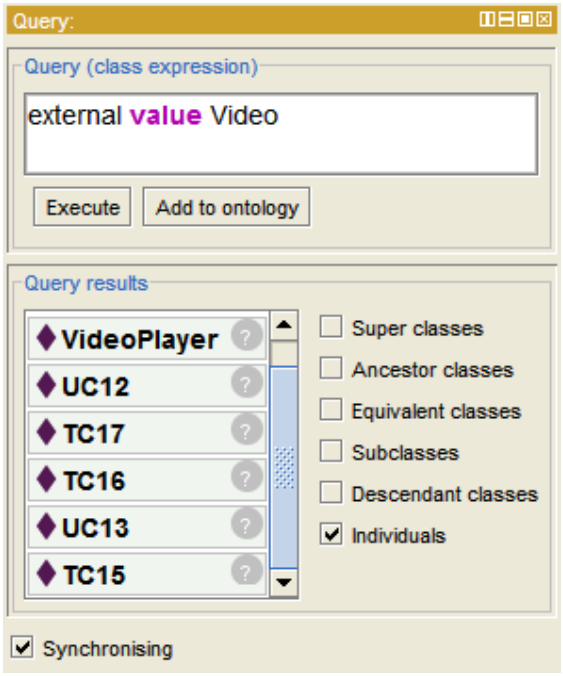


Figura 16. Consulta por conceitos externos relacionados à característica *Video*.

5.4 Conclusão

Este capítulo apresentou a aplicação da abordagem proposta para enriquecer a semântica de uma linha de produto de software de guias móveis e sensíveis ao contexto. Foi utilizado o diagrama de características da LPS escolhida para gerar um esqueleto da ontologia da linha. Em seguida, foram apresentadas as execuções dos passos de enriquecimento, desde as informações classificativas até os conceitos e relações externas. Uma vez gerada a ontologia da LPS, foram analisados os benefícios sob os aspectos de recuperação da informação, inferência e rastreabilidade das informações.

Com o estudo de caso, é possível perceber a aplicabilidade da abordagem em uma LPS real. Os benefícios observados são decorrentes da tecnologia utilizada, as ontologias. Poder-se-ia supor que a abordagem traria ganhos além dos analisados, por exemplo, em produtividade, porém seriam necessários experimentos de desenvolvimento de produtos com e sem a abordagem aplicada, o que não foi realizado nesta dissertação. Contudo, pôde-se observar com o estudo de caso que as informações complementares à variabilidade puderam ser adicionadas, seguindo a estrutura do SPLiSEM, o que não era possível na notação de características, verificando-se, portanto, o enriquecimento semântico proposto. Além disso, este estudo de caso também

demonstra a aplicação da ferramenta criada para o mapeamento entre o modelo de características e a ontologia.

6. Conclusões e Trabalhos Futuros

Este capítulo resume o que foi discutido e desenvolvido nesta dissertação de mestrado. A Seção 6.1 apresenta os resultados alcançados com a realização da proposta. Na Seção 6.2, são apresentados os trabalhos futuros que surgiram como possível continuação deste trabalho.

6.1 Resultados Alcançados

Este trabalho introduziu o conceito de Linhas de Produto de Software Semânticas e propôs métodos para auxiliar a sua viabilização. O primeiro método apresentado foi um mapeamento automático que usa um modelo de características de uma linha como base para a criação de uma ontologia de uma LPSS. Foi desenvolvida e apresentada uma ferramenta que automatiza esse mapeamento, chamada Fea2Onto *Tool*. O segundo método foi um enriquecimento semântico que utiliza um modelo (SPLiSEM) para guiar a inserção de informações no esqueleto OWL. Este esqueleto é originado da aplicação do mapeamento no diagrama de características baseado em cardinalidade. No método de enriquecimento foi discutida a inserção de novas relações para estender o SPLiSEM.

O ganho em expressividade ao utilizar as ontologias nas LPS pôde ser constatado ao observarmos as diferentes relações e conceitos que puderam ser definidos através do modelo de enriquecimento: informações classificativas, relações internas, relações de tipos de dados, conceitos externos e relações externas. Além disso, os ganhos na recuperação das informações é outro fator determinante para adoção da abordagem. A rastreabilidade entre as características e as entidades de outros artefatos é facilitada pelas relações externas.

Pelo fato da ontologia SPLiSEM ser um modelo em alto-nível que baseia o enriquecimento semântico de LPS, ele detalha a natureza das relações e conceitos que serão adicionados na ontologia de uma LPSS. Apesar de algumas relações identificadas

em técnicas de modelagem de variabilidades existentes terem sido instanciadas nesta dissertação, é recomendável que, para cada domínio, seja criada uma extensão do SPLiSEM.

6.2 Trabalhos Futuros

A ferramenta Protégé foi utilizada para inserir as relações e conceitos do estudo de caso no esqueleto OWL gerado pelo mapeamento. Entretanto, esta ferramenta apresenta uma interface, relativamente, complexa, sendo interessante que houvesse a possibilidade do engenheiro de domínio abstrair muitas informações e recursos do Protégé que não interessam para ele. Assim, é importante que se desenvolva uma ferramenta para auxiliar o enriquecimento semântico de forma amigável ao usuário. Uma possibilidade seria utilizar linguagens específicas de domínio visuais para definir as relações e os conceitos de forma mais intuitiva (e.g., o engenheiro poderia arrastar uma caixa representando uma característica para uma área da ferramenta que representasse a categoria dela, ou mesmo ligar dois conceitos para representar uma relação entre eles).

Embora tenha sido projetada para ser extensível para diversos modelos (pelo fato de utilizar um modelo canônico de objetos auto-contidos em Java), a Fea2Onto *Tool* automatiza o mapeamento entre duas notações específicas: modelo de características baseado em cardinalidade e OWL. Portanto, é considerável a implementação de "leitores" de outras notações de diagramas de variabilidade, bem como outros "escritores" em diferentes linguagens de representação de ontologias. Ainda, uma automatização da "volta" do mapeamento é desejável, com isso é possível estabelecer uma sincronização entre a ontologia e o diagrama de características da linha.

Os conceitos externos considerados no estudo de caso desta dissertação apresentam uma complexidade inerente, a qual não foi explorada no trabalho. Por exemplo, um componente, definido como um conceito externo na nossa abordagem, na verdade, pode ser definido em uma ontologia própria de definição de componentes de software. Assim como podem existir ontologias para requisitos de software, casos de teste, *stakeholders*, entre outros. Seria de absoluta relevância que existissem mecanismos de integração dessas ontologias em uma LPS.

Os aspectos benéficos do enriquecimento semântico de uma LPS que foram analisados no estudo de caso desta dissertação foram: recuperação da informação, inferência e rastreabilidade. Tais aspectos podem servir de base para benefícios mais concretos no que diz respeito a uma organização, como: diminuição do tempo de derivação de produtos e minimização de gastos com gerência de documentos. No entanto, estes benefícios somente devem ser constatados com experimentos de utilização da abordagem de enriquecimento semântico em organizações de desenvolvimento de software que utilizam o paradigma de LPS. Para isso, seria necessário conduzir uma pesquisa experimental, de preferência qualitativa, para observar os benefícios concretos da utilização de LPSS.

Referências Bibliográficas

1. WEISS, D. M.; LAI, C. T. R. Software Product-line Engineering: A Family-Based Software Development Process. [S.l.]: Addison-Wesley, 1999.
2. INSTITUTE, S. E. Model-Based Software Engineering, 1997. Disponivel em: <www.sei.cmu.edu/mbse>. Acesso em: 26 de fevereiro 2010.
3. CLEMENTS, P.; NORTHROP, L. Software Product Lines: Practices and Patterns. Boston, MA, USA: Addison-Wesley, 2002.
4. CZARNECKI, K.; EISENECKER, U. Generative Programming: Methods, Tools, and Applications. [S.l.]: Addison-Wesley, 2000.
5. WHITTLE, J.; GAJANOVIC, B. Model Transformations Should be More than Just Model Generators. Workshop on Model Design and Validation at 2005 International Conference on Model Driven Engineering, Languages and Systems (MODELS), 2005.
6. BRAGANÇA, A.; MACHADO, R. J. Model Driven Development of Software Product Lines. Software Engineering Doctoral Consortium, SEDES 2007, 6th International Conference on the Quality of Information and Communications Technology (Quatic 2007), Lisboa, Portugal, Setembro 2007. pp. 199-203.
7. W3C. Owl web ontology language overview, 2004. Disponivel em: <http://www.w3.org/TR/owl-features/>. Acesso em: 1 de Março 2010.
8. KANG, K. C.; COHEN, S. G.; HESS, J. A.; NOVAK, W. E.; PETERSON, A. S. Feature-oriented Domain Analysis Feasibility Study, Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University. Pittsburgh, PA. 1990.
9. OLIVEIRA, R. Formalização e Verificação de Consistência na Representação de Variabilidades, Dissertação de M.Sc., COPPE, UFRJ, Rio de Janeiro, Brasil, Rio de Janeiro, Brasil, 2006.
10. MORISIO, M.; TRAVASSOS, G. H.; STARK, M. E. Extending UML to Support Domain Analysis. Proceedings of the The Fifteenth IEEE International Conference on Automated Software Engineering (ASE'00), p. 321-324, Grenoble France, 2000.
11. CLAUSS, M. Modeling variability with UML. GCSE2001, Young researchers Workshop Proceedings, p. 226–230, 2001.
12. GOMAA, H. Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., 2004.
13. MARINHO, F. G. A Proposal for Consistency Checking in Dynamic Software Product Line Models Using OCL. 32st International Conference on Software Engineering- Doctoral Symposium, 2010.
14. MAIA, M. E. F.; ROCHA, L. S.; ANDRADE, R. M. C. Requirements and Challenges for Building Service-Oriented Pervasive Middleware. Proceedings of the 2009 International Conference on Pervasive Services, New York, NY, USA : ACM Press, 2009. 93-102.
15. CZARNECKI, K.; KIM, C. H. P. Feature Models are Views on Ontologies. 10th International Software Product Line Conference (SPLC'06), 2006.

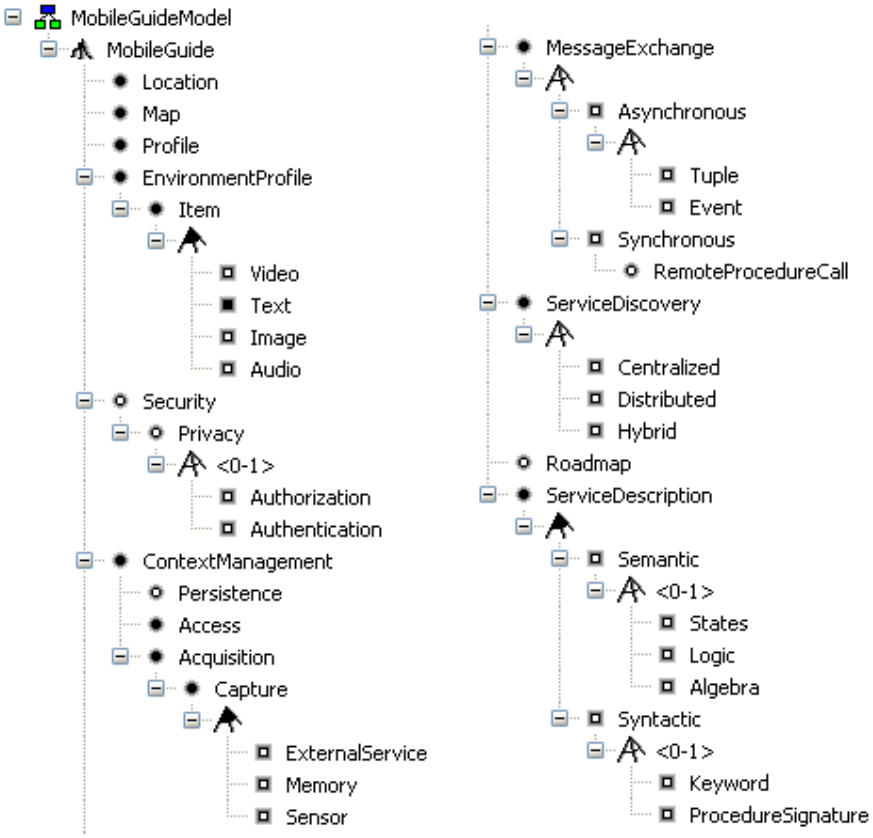
16. ALMEIDA, M. B.; BAX, M. P. Uma visão geral sobre ontologias: pesquisa sobre definições, tipos, aplicações, métodos de avaliação e de construção. Ci. Inf., v. 32, n. 3, Brasilia, Dezembro 2003.
17. VIANA, W.; BRINGEL FILHO, J.; GENSEL, J.; VILLANOVA-OLIVER, M.; MARTIN, H. PhotoMap: from location and time to context-aware photo annotations. J. Locat. Based Serv. 2, 3, Sep. 2008. 211-235.
18. GRUBER, T. R. A translation approach to portable ontologies. Knowledge Acquisition, 5(2), 1993. 199-220.
19. GRUBER, T. R. Ontology. Encyclopedia of Database Systems, Ling Liu and M. Tamer Özsu (Eds.), 2009.
20. NCITS, N. C. F. I. T. S. Draft proposed American national standard for Knowledge Interchange Format. Technical Committee T2 (Information Interchange and Interpretation), 1998.
21. BRACHMAN, R. J.; SCHMOLZE, J. G. An overview of the KL-ONE knowledge representation system. Cognitive Science, 9(2), Abril, 1985. 171–216.
22. BAADER, F.; CALVANESE, D.; MCGUINNESS, D.; NARDI, D.; PATEL-SCHNEIDER, P. The Description Logic Handbook. Cambridge University Press, Cambridge, London, 2002.
23. CRANEFIELD, S.; PURVIS, M. UML as an Ontology Modeling Language. Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on AI (IJCAI-99), Germany, 1999.
24. ALLEMANG, D.; HENDLER, J. Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL. [S.l.]: Morgan Kaufmann, 2008.
25. HAPPEL, H. J.; SEEDORF, S. Applications of ontologies in software engineering. SWESE-ISWC, Novembro 2006. 1-14.
26. BU-QING, C.; BING, L.; QI-MING, X. A Process-Driven and Ontology Based Software Product Line Variability Modeling Approach. Eighth International Conference on Grid and Cooperative Computing, 2009.
27. FALBO, R. A.; RUY, F. B.; DAL MORO, R. Using Ontologies to Add Semantics to a Software Engineering Environment. The Seventeenth International Conference on Software Engineering and Knowledge Engineering (SEKE), 2005. 1-6.
28. VIDAL, V. M. P.; SACRAMENTO, E. R.; FERNANDES DE MACEDO, J. A.; CASANOVA, M. A. An Ontology-Based Framework for Geographic Data Integration. ER 2009, Gramado, Brasil, Novembro 2009.
29. VIANA, W.; FILHO, J. B.; GENSEL, J.; VILLANOVA-OLIVER, M.; MARTIN, H. A Semantic Approach and a Web Tool for Contextual Annotation of Photos Using Camera Phones. Web Information Systems Engineering - WISE 2007, Nancy, France, December 2007.
30. VAN DER LINDEN, F.; SCHMID, K.; ROMMES, E. Software Product Lines in Action, The Best Industrial Practice in Product Line Engineering. Berlin, Heidelberg, Paris: Springer, 2007.
31. SEI, S. E. I. Software Product Lines - Overview, 2011. Disponível em: <http://www.sei.cmu.edu/productlines/>. Acesso em: 15 de março 2011.
32. NORTHROP, L. SEI's Software Product Line Tenets. IEEE Software, v. 19, n.4, July/August 2002. pp. 32-40.
33. KRUEGER, C. W. Software Reuse. ACM Computing Surveys Vol. 24, No. 02, Junho 1992. 131-183.
34. CASTRO, R. N. S.; SOUZA, J.; ANDRADE, R. M. C. A proposal for Discovering Relationships among Software Patterns using Latent Semantic Analysis. Anais da SugarloafPLoP 2008 - 7a Conferência Latino-Americana em Linguagens de Padrões para Programação, 2008.

35. SIMOS, M.; CREPS, D.; KLINGER, C.; LEVINE, L.; ALLEMANG, D. Organization Domain Modeling (ODM) Guidebook, Version 2.0. STARS. [S.l.]. 1996. (STARS-VC-A025/001/00).
36. HARSU, M. A Survey on Domain Engineering. Report 31, Institute of Software Systems, Tampere University of Technology, December 2002.
37. GMV. Domain Engineering Methodologies Survey, Technical Report, CORDET Deliverable GMVCORDET. [S.l.]. 2007.
38. NEIGHBORS, J. M. Software Construction Using Components. PhD thesis, University of California, Irvine, Department of Information, 1980.
39. ARANGO, G. Domain Analysis Methods. Software Reusability,W. Schäfer, R. Prieto-Díaz, e M. Matsumoto (Eds.), Ellis Horwood, New York, NY, 1994. 17-49.
40. DAVID, G.; SHAW, M. An Introduction to Software Architecture. Advances in Software Engineering and Knowledge Engineering, Volume I, V.Ambriola e G.Tortora (Eds.), World Scientific Publishing Company, New Jersey, 1993.
41. CHASTEK, G.; MCGREGOR, J. D. Guidelines for Developing a Product Line Production Plan. Software Engineering Institute (SEI), Technical Report. [S.l.]. 2002. (CMU/SEI-2002-TR-006 ESC-TR-2002-006).
42. POHL, K.; BÖCKLE, G.; VAN DER LINDER, F. Software Product Line Engineering Foundations, Principles, and Techniques. Springer, 2005.
43. CZARNECKI, K.; HELSEN, S.; EISENECKER, U. Formalizing cardinality-based feature models and their specialization. Software Process Improvement and Practice 10 (1) , 2005. pp. 7–29.
44. ANTKIEWICZ, M.; CZARNECKI, K. FeaturePlugin: Feature modeling plug-in for Eclipse. OOPSLA'04 Eclipse Technology eXchange (ETX) Workshop. , 2004.
45. FALBO, R. A.; GUIZZARDI, G.; DUARTE, K. C. An Ontological Approach to Domain Engineering. Proceedings of the 14th Int. Conference on Software Engineering and Knowledge Engineering (SEKE), Ischia, Italy, 2002. 351- 358.
46. ZAID, L. A.; KLEINERMANN, F.; DE TROYER, O. Applying semantic web technology to feature modeling. Proceedings of the 2009 ACM Symposium on Applied Computing (SAC '09), Honolulu, Hawaii. 1252-1256.
47. W3C. Document Object Model (DOM). Disponivel em: <http://www.w3.org/DOM/>. Acesso em: 20 março 2011.
48. KANG, K. C. FORM: a feature-oriented reuse method with domain specific architectures. Annals of Software Engineering, 1998. volume 5, pp. 345-355.
49. HOTZ, L.; KREBS, T.; WOLTER, K.; NIJHUIS, J.; DEELSTRA, S.; SINNEMA, M.; MACGREGOR, J. Configuration in Industrial Product Families – The ConIPF Methodology. IOS Press, July 2006.
50. BECKER, M. Towards a General Model of Variability in Product Families. Proceedings of the 1st Workshop on Software Variability Management, Groningen, Netherlands, February 2003.
51. BEUCHE, D.; PAPAJEWSKI, H.; SCHRÖDER-PREIKSCHAT, W. Variability management with feature models. Science of Computer Programming, 2004. pp. 333–352.
52. ASIKAINEN, T.; SOININEN, T.; MÄNNISTÖ, T. A Koala-Based Approach for Modelling and Deploying Configurable Software Product Families. 5th Workshop on Product Family Engineering (PFE-5), Springer Verlag Lecture Notes on Computer Science, vol. 3014, LNCS 3014, May 2004. pp. 225–249.
53. YE, H.; LIU, H. Approach to modelling feature variability and dependencies in software product lines. IEEE Proc. - Software, 2005. vol.152, pp. 101-109.

54. FEY, D.; FAJTA, R.; BOROS, A. Feature Modeling: A Meta-Model to Enhance Usability and Usefulness. Software Product Lines Conference (SPLC2), 2002. Springer, pp. 198-216.
55. VAN DEURSEN, A.; DE JONGE, M.; KUIPERS, T. Feature-Based Product Line Instantiation Using Source-Level Packages. Software Product Lines Conference (SPLC2) , 2002. Springer, pp. 19-30.
56. ATKINSON, C.; BAYER, J.; MUTHIG, D. Component-based product line development: the KobrA approach. 1st International Software Produt Line Conference, Denver, Colorado, US, 2000. Kluwer Academic Publishers, pp. 289-309.
57. REISER, M. O.; WEBER, M. Multi-level feature trees: A pragmatic approach to managing highly complex product families. Requirements Engineering, 2007. vol. 12, pp. 57-75.
58. LOESCH, F.; PLOEDEREDER, E. Optimization of Variability in Software Product Lines. 11th International Software Product Line Conference (SPLC), 2007. pp. 151-162.
59. WEBBER, D. L.; GOMAA, H. Modeling variability in software product lines with the variation point model. Sci.Comput. Program, , 2004. vol. 53, pp. 305-331.
60. MARINHO, F. G.; LIMA, F. F. D. P.; FILHO, J. B. F.; ROCHA, L. S.; MAIA, M. E. F.; AGUIAR, S. B.; DANTAS, V. L. L.; VIANA, W.; ANDRADE, R. M. C.; TEIXEIRA, E. N.; WERNER, C. M. L. A Software Product Line for the Mobile and Context-Aware Applications Domain. Software Product Line Conference (SPLC), Jeju Island, South Korea, 2010.
61. MARINHO, F. G.; COSTA, A. L.; LIMA, F. F. D. P.; BORGES NETO, J. B.; FILHO, J. B. F.; ROCHA, L. S.; DANTAS, V. L. L.; ANDRADE, R. M. C.; TEIXEIRA, E. N.; WERNER, C. M. L. Uma proposta de arquitetura para linhas de produto de software aninhadas no domínio de aplicações móveis e sensíveis ao contexto. Simpósio Brasileiro de Componentes, Arquiteturas e Reuso de Software (SBCARS), Salvador, Bahia, Brasil, 2010.
62. UNIVERSITY, S. Protégé project. Disponivel em: <http://protege.stanford.edu>. Acesso em: Março, 2011.
63. BAILIN, S. KAPTUR: A Tool for the Preservation and Use of Engineering Legacy. CTA Inc., Rockville, MD, 1992.
64. MYERS, B. A. A Taxonomy of Window Manager User Interfaces. IEEE Transactions on Computer Graphics & Applications 8(5), Setembro, 1988. 65-84.
65. WANG, H.; LI, Y. F.; SUN, J.; ZHANG, H.; PAN, J. Verifying Feature Models Using OWL. Journal of Web Semantics: Science, Services and Agents on the World Wide Web, vol. 5, no. 2, June 2007. pp. 117-129.

Anexo A – Modelo de características baseado em cardinalidade

Este anexo apresenta o modelo da linha de produto de software para aplicações de guias móveis e sensíveis ao contexto.



Anexo B – Esqueleto OWL renderizado em sintaxe funcional

Este anexo apresenta o esqueleto OWL gerado pela Fea2Onto *Tool* a partir do diagrama de características da linha de produto de software de guias móveis e sensíveis ao contexto. O esqueleto é apresentado na linguagem OWL funcional, visto que esta é mais compacta do que a visualização em XML.

```
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(:=<http://www.semanticweb.org/ontologies/2010/7/MetaOntology.owl#>)
Prefix(xml:=<http://www.w3.org/XML/1998/namespace>)
Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)
Prefix(skos:=<http://www.w3.org/2004/02/skos/core#>)


Ontology(<http://www.semanticweb.org/ontologies/2010/7/MetaOntology.owl>
<http://www.semanticweb.org/ontologies/2010/7/MetaOntology.owl>
Declaration(Class(:AlternativeFeature))
SubClassOf(:AlternativeFeature :Feature)
Declaration(Class(:Feature))
Declaration(Class(:MandatoryFeature))
SubClassOf(:MandatoryFeature :Feature)
Declaration(Class(:OptionalFeature))
SubClassOf(:OptionalFeature :Feature)
Declaration(Class(:OrFeature))
SubClassOf(:OrFeature :Feature)
Declaration(ObjectProperty(:consistsOf))
SubObjectPropertyOf(:consistsOf owl:topObjectProperty)
Declaration(ObjectProperty(:hasAlternativeFeature))
SubObjectPropertyOf(:hasAlternativeFeature owl:topObjectProperty)
Declaration(ObjectProperty(:hasOrFeature))
SubObjectPropertyOf(:hasOrFeature owl:topObjectProperty)
ClassAssertion(:MandatoryFeature :Access)
ClassAssertion(:MandatoryFeature :Acquisition)
ObjectPropertyAssertion(:consistsOf :Acquisition :Capture)
ClassAssertion(:AlternativeFeature :Algebra)
ClassAssertion(:AlternativeFeature :Asynchronous)
ClassAssertion(:OptionalFeature :Asynchronous)
ObjectPropertyAssertion(:consistsOf :Asynchronous :Event)
ObjectPropertyAssertion(:consistsOf :Asynchronous :Tuple)
ObjectPropertyAssertion(:hasAlternativeFeature :Asynchronous :Tuple)
```

ObjectPropertyAssertion(:hasAlternativeFeature :Asynchronous :Event)
ClassAssertion(:OrFeature :Audio)
ClassAssertion(:AlternativeFeature :Authentication)
ClassAssertion(:AlternativeFeature :Authorization)
ClassAssertion(:MandatoryFeature :Capture)
ObjectPropertyAssertion(:consistsOf :Capture :Memory)
ObjectPropertyAssertion(:consistsOf :Capture :Sensor)
ObjectPropertyAssertion(:consistsOf :Capture :ExternalService)
ObjectPropertyAssertion(:hasOrFeature :Capture :Sensor)
ObjectPropertyAssertion(:hasOrFeature :Capture :ExternalService)
ObjectPropertyAssertion(:hasOrFeature :Capture :Memory)
ClassAssertion(:AlternativeFeature :Centralized)
ClassAssertion(:MandatoryFeature :ContextManagement)
ObjectPropertyAssertion(:consistsOf :ContextManagement :Access)
ObjectPropertyAssertion(:consistsOf :ContextManagement :Persistence)
ObjectPropertyAssertion(:consistsOf :ContextManagement :Acquisition)
ClassAssertion(:AlternativeFeature :Distributed)
ClassAssertion(:MandatoryFeature :EnvironmentProfile)
ObjectPropertyAssertion(:consistsOf :EnvironmentProfile :Item)
ClassAssertion(:AlternativeFeature :Event)
ClassAssertion(:OrFeature :ExternalService)
ClassAssertion(:AlternativeFeature :Hybrid)
ClassAssertion(:OrFeature :Image)
ClassAssertion(:MandatoryFeature :Item)
ObjectPropertyAssertion(:consistsOf :Item :Audio)
ObjectPropertyAssertion(:consistsOf :Item :Text)
ObjectPropertyAssertion(:consistsOf :Item :Video)
ObjectPropertyAssertion(:consistsOf :Item :Image)
ObjectPropertyAssertion(:hasOrFeature :Item :Video)
ObjectPropertyAssertion(:hasOrFeature :Item :Audio)
ObjectPropertyAssertion(:hasOrFeature :Item :Text)
ObjectPropertyAssertion(:hasOrFeature :Item :Image)
ClassAssertion(:AlternativeFeature :Keyword)
ClassAssertion(:MandatoryFeature :Location)
ClassAssertion(:AlternativeFeature :Logic)
ClassAssertion(:MandatoryFeature :Map)
ClassAssertion(:OrFeature :Memory)
ClassAssertion(:MandatoryFeature :MessageExchange)
ObjectPropertyAssertion(:consistsOf :MessageExchange :Synchronous)
ObjectPropertyAssertion(:consistsOf :MessageExchange :Asynchronous)
ObjectPropertyAssertion(:hasAlternativeFeature :MessageExchange :Synchronous)
ObjectPropertyAssertion(:hasAlternativeFeature :MessageExchange :Asynchronous)
ClassAssertion(:MandatoryFeature :MobileGuide)
ObjectPropertyAssertion(:consistsOf :MobileGuide :Profile)
ObjectPropertyAssertion(:consistsOf :MobileGuide :ServiceDescription)
ObjectPropertyAssertion(:consistsOf :MobileGuide :Roadmap)
ObjectPropertyAssertion(:consistsOf :MobileGuide :Security)
ObjectPropertyAssertion(:consistsOf :MobileGuide :EnvironmentProfile)
ObjectPropertyAssertion(:consistsOf :MobileGuide :Location)
ObjectPropertyAssertion(:consistsOf :MobileGuide :MessageExchange)
ObjectPropertyAssertion(:consistsOf :MobileGuide :Map)
ObjectPropertyAssertion(:consistsOf :MobileGuide :ContextManagement)
ObjectPropertyAssertion(:consistsOf :MobileGuide :ServiceDiscovery)
ClassAssertion(:MandatoryFeature :MobileGuideModel)
ObjectPropertyAssertion(:consistsOf :MobileGuideModel :MobileGuide)
ClassAssertion(:OptionalFeature :Persistence)

```
ClassAssertion(:OptionalFeature :Privacy)
ObjectPropertyAssertion(:consistsOf :Privacy :Authorization)
ObjectPropertyAssertion(:consistsOf :Privacy :Authentication)
ObjectPropertyAssertion(:hasAlternativeFeature :Privacy :Authentication)
ObjectPropertyAssertion(:hasAlternativeFeature :Privacy :Authorization)
ClassAssertion(:AlternativeFeature :ProcedureSignature)
ClassAssertion(:MandatoryFeature :Profile)
ClassAssertion(:OptionalFeature :RemoteProcedureCall)
ClassAssertion(:OptionalFeature :Roadmap)
ClassAssertion(:OptionalFeature :Security)
ObjectPropertyAssertion(:consistsOf :Security :Privacy)
ClassAssertion(:OptionalFeature :Semantic)
ClassAssertion(:OrFeature :Semantic)
ObjectPropertyAssertion(:consistsOf :Semantic :Logic)
ObjectPropertyAssertion(:consistsOf :Semantic :States)
ObjectPropertyAssertion(:consistsOf :Semantic :Algebra)
ObjectPropertyAssertion(:hasAlternativeFeature :Semantic :Logic)
ObjectPropertyAssertion(:hasAlternativeFeature :Semantic :States)
ObjectPropertyAssertion(:hasAlternativeFeature :Semantic :Algebra)
ClassAssertion(:OrFeature :Sensor)
ClassAssertion(:MandatoryFeature :ServiceDescription)
ObjectPropertyAssertion(:consistsOf :ServiceDescription :Syntactic)
ObjectPropertyAssertion(:consistsOf :ServiceDescription :Semantic)
ObjectPropertyAssertion(:hasOrFeature :ServiceDescription :Semantic)
ObjectPropertyAssertion(:hasOrFeature :ServiceDescription :Syntactic)
ClassAssertion(:MandatoryFeature :ServiceDiscovery)
ObjectPropertyAssertion(:consistsOf :ServiceDiscovery :Centralized)
ObjectPropertyAssertion(:consistsOf :ServiceDiscovery :Distributed)
ObjectPropertyAssertion(:consistsOf :ServiceDiscovery :Hybrid)
ObjectPropertyAssertion(:hasAlternativeFeature :ServiceDiscovery :Centralized)
ObjectPropertyAssertion(:hasAlternativeFeature :ServiceDiscovery :Distributed)
ObjectPropertyAssertion(:hasAlternativeFeature :ServiceDiscovery :Hybrid)
ClassAssertion(:AlternativeFeature :States)
ClassAssertion(:AlternativeFeature :Synchronous)
ClassAssertion(:OptionalFeature :Synchronous)
ObjectPropertyAssertion(:consistsOf :Synchronous :RemoteProcedureCall)
ClassAssertion(:OptionalFeature :Syntactic)
ClassAssertion(:OrFeature :Syntactic)
ObjectPropertyAssertion(:consistsOf :Syntactic :Keyword)
ObjectPropertyAssertion(:consistsOf :Syntactic :ProcedureSignature)
ObjectPropertyAssertion(:hasAlternativeFeature :Syntactic :ProcedureSignature)
ObjectPropertyAssertion(:hasAlternativeFeature :Syntactic :Keyword)
ClassAssertion(:OrFeature :Text)
ClassAssertion(:AlternativeFeature :Tuple)
ClassAssertion(:OrFeature :Video)
)
```