UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

THALISSON ALVES OLIVEIRA

FIXONTO: UM MÉTODO PARA ENRIQUECIMENTO SEMÂNTICO E VERIFICAÇÃO DE MODELOS DE CARACTERÍSTICAS EM LPS SENSÍVEL AO CONTEXTO

FORTALEZA

2017

THALISSON ALVES OLIVEIRA

FIXONTO: UM MÉTODO PARA ENRIQUECIMENTO SEMÂNTICO E VERIFICAÇÃO DE MODELOS DE CARACTERÍSTICAS EM LPS SENSÍVEL AO CONTEXTO

Dissertação apresentada ao Curso de Programa de Pós-Graduação em Ciência da Computação do Departamento de Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Engenharia de Software

Orientadora: Profa. Dra. Rossana Maria de Castro Andrade

Co-Orientador: Prof. Dr. Windson Viana de Carvalho

FORTALEZA

2017

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

O52f Oliveira, Thalisson Alves.
FixOnto: um Método para Enriquecimento Semântico e Verificação de Modelos de Características em LPS Sensível ao Contexto / Thalisson Alves Oliveira. – 2018.
85 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2018.
Orientação: Profa. Dra. Rossana Maria de Castro Andrade.
Coorientação: Prof. Dr. Windson Viana de Carvalho.

1. Linhas de Produto de Software. 2. Ontologias. 3. Sensibilidade ao Contexto. I. Título.

CDD 005

THALISSON ALVES OLIVEIRA

FIXONTO: UM MÉTODO PARA ENRIQUECIMENTO SEMÂNTICO E VERIFICAÇÃO DE MODELOS DE CARACTERÍSTICAS EM LPS SENSÍVEL AO CONTEXTO

Dissertação apresentada ao Curso de Programa de Pós-Graduação em Ciência da Computação do Departamento de Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Engenharia de Software

Aprovada em:

BANCA EXAMINADORA

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

Prof. Dr. Windson Viana de Carvalho
(Co-Orientador)
Universidade Federal do Ceará – UFC

Prof. Dr. Vinicius Cardoso Garcia
Universidade Federal de Pernambuco - UFPE

Profa. Dra. Vania Maria Ponte Vidal
Universidade Federal do Ceará - UFC

Prof. Dr. João Bosco Ferreira Filho
Universidade Federal do Ceará - UFC

"O sonho é que leva a gente para frente. Se a gente for seguir a razão, fica aquietado, acomodado."

(Ariano Suassuna)

RESUMO

Linhas de Produtos de Software Dinâmicas (LPSD), projetadas para gerenciar variabilidade de sistemas auto adaptáveis em tempo de execução, podem ser empregadas para sistematizar e maximizar o reuso no desenvolvimento de aplicações sensíveis ao contexto. Para esse fim, existem também as LPSSCs (Linhas de Produtos de Software Sensível ao Contexto), focadas exclusivamente no suporte à sensibilidade ao contexto. Modelos de características são a principal forma de representar as similaridades e variabilidades em LPS tradicional, sensível ao contexto e dinâmicas. Uma LPSSC, foco deste trabalho, contém, por exemplo, o MMSC (Modelo de Características Móvel e Sensível ao Contexto), que inclui o modelo de características e o modelo de contexto. Mesmo sendo a principal representação do conhecimento sobre um domínio em LPS, esses modelos apresentam limitações de expressividade. Por exemplo, aspectos de domínio relevantes (como um artefato UML que está associado a um contexto) não são descritos no MMSC. Além disso, os modelos podem conter inconsistências que levam à derivação de produtos inválidos. Para solucionar esses dois problemas, o objetivo deste trabalho é propor um método para adicionar semântica ao MMSC e realizar verificação automática da corretude e consistência desses modelos. Para avaliar o método é implementada uma ferramenta e realizado uma demonstração de seu uso com uma LPSSC chamada Mobiline. Como resultado foi observado que é possível verificar que os modelos estavam corretos, considerando as regras implementadas, e que o uso de ontologias no processo de enriquecimento semântico permite a realização de buscas semânticas e rastreabilidade de características, contextos e artefatos.

Palavras-chave: Linhas de Produto de Software. Ontologias. Sensibilidade ao Contexto.

ABSTRACT

Dynamic Software Product Lines (DSPL), designed to manage the variability of self-adaptive systems at runtime, can be employed to systematize and maximize reuse in context-aware applications development. To this end, there are also Context-aware Software Product Lines (CASPLs) which are focused exclusively on supporting context-awareness. Feature models are the main way to represent similarities and variabilities in traditional, context aware and dynamic SPL. CASPL, focus of this work, contains, for example, Mobile and Context-aware Feature Model (MCFM), which includes the feature model and the context model. Although these models are the main representation of knowledge about a domain in SPL, they have limitations of expressiveness. For instance, relevant domain aspects (such as a UML artifact that is associated with a context) are not described in the MCFM. In addition, these models may contain inconsistencies that lead to the derivation of invalid products. To solve these two problems, the goal of this work is to propose a method to add semantics to the MCFM and perform automatic verification of the correctness and consistency of these models. A tool is implemented to evaluate the method and it is performed a demonstration of its use with a CASPL called Mobiline. As a result, it is observed the possibility of verifying that the models are correct, considering the implemented rules, and that the use of ontologies in the process of semantic enrichment allows the realization of semantic searches as well as traceability of features, contexts and artifacts.

Keywords: Software Product Lines. Ontologies. Context-awareness.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CASPLs	Context-aware Software Product Lines
DL	Lógica Descritiva
DSPL	Dynamic Software Product Lines
GVM	Guia de Visitas Móveis
LPS	Linhas de Produtos de Software
LPSSC	Linhas de Produtos de Software Sensíveis ao Contexto
MC	Modelo de Contexto
MCFM	Mobile and Context-aware Feature Model
MMSC	Modelos de Características Móveis e Sensíveis ao Contexto
MS	Modelo de Sistema
OWL	Ontology Web Language
RA	Regras de Adaptação
RC	Regras de Composição

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta a contextualização desta dissertação. Nele são introduzidos os conceitos que abrangem a problemática e a solução proposta, bem como a motivação. Neste capítulo são apresentadas também as questões de pesquisa que norteiam este trabalho, as etapas de execução da pesquisa e a organização do restante do documento.

1.1 Contextualização

Os dispositivos móveis possuem vários sensores e alta capacidade de processamento, o que permite o desenvolvimento de aplicações capazes de perceber o ambiente e adaptar-se para satisfazer as necessidades do usuário (MARINHO *et al.*, 2013). Essas aplicações são denominadas sensíveis ao contexto, sendo contexto qualquer informação que possa ser usada para caracterizar a situação de uma entidade. Uma entidade pode ser uma pessoa, lugar ou objeto que é considerado relevante para a interação entre um usuário e um aplicativo, incluindo o usuário e os próprios aplicativos (DEY, 2001). Devido à heterogeneidade e por vezes imprevisibilidade do contexto, o desenvolvimento de aplicações sensíveis ao contexto se torna uma atividade desafiadora. Esse desafio se dá pelo fato da necessidade do projetista prever mudanças contextuais e as adaptações possíveis da aplicação, além de considerar situações excepcionais. Desse modo, há uma necessidade de métodos ou técnicas para suportar o desenvolvimento desse tipo de aplicação, diferentes das aplicações tradicionais.

Diante dessa necessidade, Linhas de Produtos de Software (LPS), comumente representadas por modelos de características - que representam a variabilidade em uma família de sistemas - têm sido usadas para sistematizar e maximizar o reuso no desenvolvimento de aplicações sensíveis do contexto (MARINHO *et al.*, 2013) (FILHO *et al.*, 2012). Já LPSD são LPSs que incluem mecanismos para alterar variantes em aplicações auto adaptáveis em tempo de execução, suportando o desenvolvimento de aplicações sensíveis ao contexto. No entanto, a solução proposta neste trabalho é prevista para o tempo de design. Nesse sentido, este trabalho considera o conceito de Linhas de Produtos de Software Sensíveis ao Contexto (LPSSC) proposto em (MARINHO *et al.*, 2012), sendo que este tipo de LPS também suporta o desenvolvimento de aplicações sensíveis ao contexto. Eles introduzem os Modelos de Características Móveis e Sensíveis ao Contexto (MMSC), que são compostos de um Modelo de Sistema (MS), que é equivalente ao modelo de características em LPS tradicional, e um Modelo de Contexto (MC). É

importante ressaltar que neste trabalho não são considerados aspectos de mobilidade presentes em um MMSC, sendo o foco os aspectos de sensibilidade ao contexto.

Em LPSSC, a modelagem de características e contexto são atividades-chave envolvidas no seu projeto (DERMEVAL *et al.*, 2015). Os Modelos de Características Sensíveis ao Contexto representam a variabilidade de características e também situações de contexto e adaptações predefinidas. No entanto, outros aspectos importantes do domínio (por exemplo, a relação entre contexto e código-fonte) não são expressos (FILHO *et al.*, 2012). Além disso, a modelagem de aplicações sensíveis ao contexto é uma tarefa propensa a erros, pois o projetista deve prever possíveis contextos e adaptações em tempo de projeto. Como consequência, principalmente em LPSs muito grandes como o Kernel do Linux, que possui mais de 5000 características (SHE *et al.*, 2011), contextos inconsistentes podem ser modelados e produtos inválidos podem ser derivados. Dessa forma, a verificação do MMSC se torna uma tarefa essencial. No entanto, a verificação manual em aplicações complexas, tais como aplicações sensíveis ao contexto e em grande escala, é inviável (BENAVIDES *et al.*, 2013). Assim, há também uma necessidade de verificação automática de modelos de características para evitar inconsistências.

Desse modo, considerando a expressividade, os modelos que compõem uma LPSSC (i.e, o MMSC que é composto por um modelo de sistema e um modelo de contexto) têm um grande potencial para serem associados com diversos artefatos importantes no processo de desenvolvimento de uma aplicação. Portanto, esses modelos podem ser considerados como artefatos primários que agrupam artefatos secundários (e.g., casos de uso, casos de teste e até mesmo código-fonte). Para esse fim, Filho (2011) sugere o enriquecimento semântico em LPS, onde os artefatos reusáveis que a compõem e os elementos organizacionais se relacionam ao modelo de domínio, sendo essas relações formalmente representadas por uma ou mais ontologias (FILHO, 2011).

1.2 Motivação

Considerando a importância dos modelos de características para representar uma LPS, existem aspectos além das caraterísticas que não podem ser expressos nesse tipo de modelo, por exemplo, relações das características com artefatos de desenvolvimento tradicionais, como casos de uso (FILHO *et al.*, 2012). A falta de precisão na descrição das características e seus relacionamentos é mais uma de suas limitações. Além disso, como citado anteriormente, a verificação manual dos modelos é inviável. Portanto, existe uma necessidade de verificação de

modelos de características para prevenir erros.

Essas limitações também podem ser encontradas em LPSSCs, pois os mesmos aspectos que não são representados nos modelos de característica estão ausentes em um MMSC. Ademais, como mencionado na seção anterior, é necessário automatizar a verificação desse modelo, que é mais complexo.

O enriquecimento semântico e a análise automática do MMSC em LPSSC trazem benefícios tanto para o profissional responsável pela modelagem do domínio quanto para outros *stakeholders* envolvidos no desenvolvimento de sistemas sensíveis ao contexto. Outros profissionais, como o testador e até mesmo o programador, podem tomar decisões a partir do MMSC, cujas características e contextos modelados estão relacionados com outros artefatos de domínio. A automatização da análise de MMSC oferece uma maior segurança para quem está modelando o domínio, dado que essa modelagem é uma atividade suscetível a erros à medida que o modelo cresce. Como o MMSC é o ponto de partida para o enriquecimento semântico, é necessário garantir que o mesmo está bem formado e consistente.

Por exemplo, considere o domínio de Guia de Visitas Móveis (GVM) como uma família de produtos auto adaptáveis, nos quais as aplicações podem apresentar conteúdo (e.g., texto, imagem e vídeo) de acordo com a localização e configurações do usuário e o estado do dispositivo. Nesse exemplo, o especialista de domínio deve associar a característica de exibição à localização do usuário e ao nível da bateria do dispositivo. Para executar esse tipo de adaptação, o especialista de domínio deve selecionar a característica de troca de mensagens, pois ele o modelou como uma característica obrigatória. No entanto, o especialista de domínio pode modelar uma situação de contexto que não escolha a característica de troca de mensagens, tornando possível a derivação de um produto com uma adaptação inconsistente de acordo com o que foi modelado. Além disso, quando um produto é derivado, pode ser necessário acessar informações como, por exemplo, casos de uso que devem compor a documentação do produto.

Considerando o cenário descrito, há uma ausência de trabalhos que considerem o enriquecimento semântico em LPS que dê suporte ao desenvolvimento de aplicações sensíveis ao contexto, que pode ser verificada no Capítulo 3. Existe também a necessidade de se obter uma solução que realize esse enriquecimento em conjunto com a análise automática dos modelos de característica e de contexto.

Com base no que foi discutido ao longo desta seção, são estabelecidas as seguintes Questões de Pesquisa (QP):

- **QP1:** Como enriquecer semanticamente uma LPSSC, associando características e contextos outros artefatos do domínio, e garantindo que a modelagem permaneça correta após essa associação?
- **QP2:** Como permitir a recuperação e o rastreamento de informações em LPSSC de maneira semântica?

1.3 Objetivos e Resultados Esperados

Este trabalho objetiva prover um método para adição de semântica e verificação automática da corretude em Modelos de Características Sensíveis ao Contexto (MMSC). Para alcançar tal objetivo, as seguintes metas são delimitadas:

- Propor um método que contemple, em conjunto, o enriquecimento semântico e a análise automática de MMSC;
- Criar uma ontologia de domínio para representar uma LPSSC e, consequentemente, os conceitos relacionados a um MMSC;
- Realizar o levantamento de regras para verificação de MMSC e
- Implementar uma ferramenta para automatização do método proposto.

O principal resultado esperado desta dissertação é o método para enriquecimento semântico em LPSSC e a verificação automática de MMSC. A implementação do método proposto traz benefícios diversos para os profissionais envolvidos nos projetos desse tipo de sistema. Como exemplo desses benefícios, há a garantia para o projetista que o modelo está bem formado e consistente pela análise automática dos modelos. Outro exemplo de vantagens do método é a rastreabilidade de informações, que pode ajudar outros profissionais.

Outro resultado esperado deste trabalho é a ferramenta que automatiza parte do método. A ferramenta, dentre outras funcionalidades, faz a verificação automática de um MMSC e busca semântica no MMSC, sendo uma extensão da Fixture (COSTA, 2012) (COSTA *et al.*, 2015).

Vale ressaltar ainda que outra contribuição deste trabalho é uma ontologia de domínio que representa a LPSSC, a qual é utilizada na implementação da extensão da Fixture.

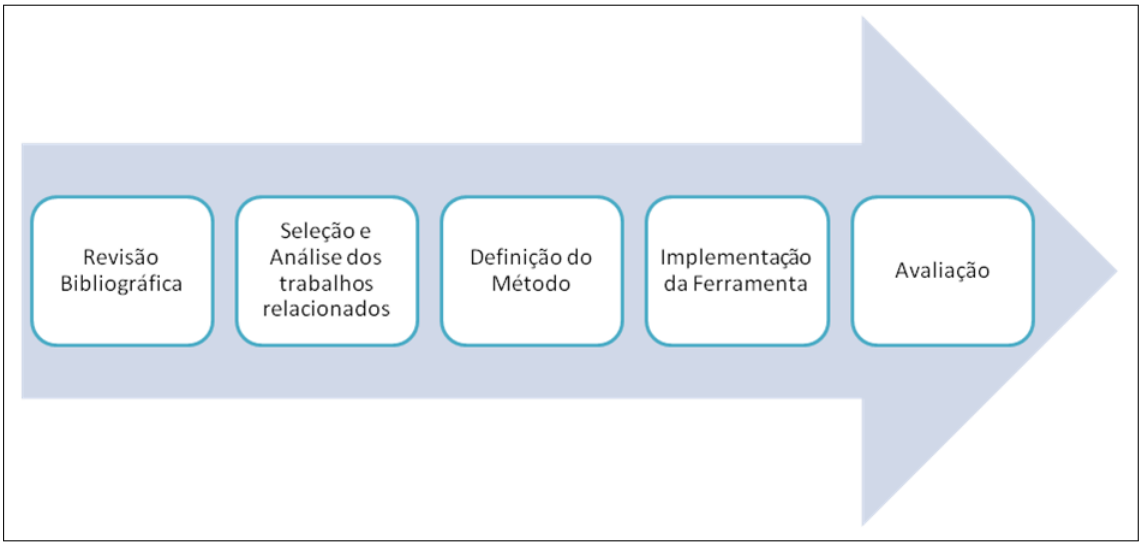
1.4 Etapas de execução da pesquisa

Para atingir o objetivo proposto para este trabalho, as seguintes etapas são executadas (como podem ser vistas na Figura 1):

- **Revisão bibliográfica**: é realizada uma revisão da literatura onde são pesquisados os conceitos de Linhas de Produtos de Software, Computação Ubíqua, incluindo Computação Móvel e Sensível ao Contexto. Outros conceitos pesquisados são LPS em conjunto com sistemas ubíquos e LPS em conjunto com Ontologias, além da avaliação de modelos de características.
- **Seleção e análise dos trabalhos relacionados**: após a revisão da literatura mais ampla, são buscados trabalhos focados nos objetivos desta pesquisa. Nessa etapa, a pesquisa é direcionada para LPSs que suportam desenvolvimento de aplicações sensíveis ao contexto, LPSs que utilizam ontologias e, desse modo, dão suporte a enriquecimento semântico. São pesquisadas também a verificação automática de modelos de características tradicionais e de modelos de características que dão suporte à sensibilidade ao contexto.
- **Definição do Método**: com base nos trabalhos selecionados, é definido o método que atende ao objetivo principal desta pesquisa.

  Elaboração da Ontologia de Domínio: é elaborada uma ontologia de domínio para atender ao método proposto. Essa ontologia serve de base para a geração da ontologia que representa o MMSC modelado.

Figura 1 – Etapas de execução da pesquisa



Fonte – O autor.

Levantamento das Regras: são pesquisadas e selecionadas regras para verificação da boa formação e corretude de um MMSC. Essas regras são utilizadas para a verificação automática.

- **Implementação da Ferramenta**: nessa etapa é realizada a extensão da Fixture (COSTA, 2012; COSTA *et al.*, 2015). São incluídas as funcionalidades relativas à utilização de ontologias e consequente inclusão do enriquecimento semântico e busca semântica (rastreabilidade).
- **Avaliação**: por fim, é realizada uma avaliação do método por meio de uma Prova de Conceito (PoC) (GLASS *et al.*, 2002) utilizando a ferramenta, bem como uma avaliação de usabilidade da mesma.

1.5 Organização da Dissertação

O restante da dissertação está organizado como segue:

- **Capítulo 2 - Fundamentação Teórica** detalha os conceitos envolvidos neste trabalho. Linhas de Produtos de Software e Ontologias estão entre os conceitos discutidos.
- **Capítulo 3 - Trabalhos Relacionados** apresenta e detalha os trabalhos relacionados a esta pesquisa. São apresentadas oportunidades de pesquisa a partir das lacunas observadas.
- **Capítulo 4 - Fixonto: O Método** apresenta o método desenvolvido nesta pesquisa. Também são apresentadas as suas etapas e discutidos os detalhes da ontologia de domínio e das regras implementadas.
- **Capítulo 5 - FixOnto: A Ferramenta** aborda a implementação da ferramenta, onde são apresentadas as tecnologias envolvidas, especificidades da linguagem de ontologias e linguagem de regras, bem como as etapas do método que não foram atendidas e como pode ser conduzidas com o auxílio de outras ferramentas.
- **Capítulo 6 - Avaliação** apresenta a avaliação do método através de uma Prova de Conceito, que demonstra que a ferramenta atende ao que se propõe a realizar.
- **Capítulo 7 - Conclusões e Trabalhos Futuros** conclui este trabalho, detalhando os resultados alcançados e principais contribuições, as limitações da solução proposta e também os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, é apresentado o levantamento bibliográfico, abordando os conceitos dos principais tópicos envolvidos no desenvolvimento da pesquisa bem como os trabalhos recentes que os incorporam. Os tópicos descritos são: Computação Ubíqua, com foco em Sensibilidade ao Contexto, Linhas de Produtos de Software e Ontologias.

2.1 Computação Ubíqua

Mark Weiser preconizava a Computação Ubíqua afirmando que "as mais profundas tecnologias são aquelas que desaparecem. Elas se entrelaçam nas estruturas do cotidiano, até que sejam indistinguíveis" (WEISER, 1991). Com esse paradigma, seria possível visualizar a presença de computadores em toda parte, inseridos no ambiente e até mesmo embarcados no nosso corpo. Outras definições são encontradas na literatura (LIMA *et al.*, 2011) (SPÍNOLA; TRAVASSOS, 2012), e se apresentam claramente como evolução do conceito definido por Weiser conforme descrito a seguir.

Segundo Lima *et al.* (2011), os sistemas ubíquos são sistemas distribuídos abertos, voláteis, heterogêneos e centrados no usuário, apresentando como principais características a interoperabilidade espontânea e a integração com o mundo físico (LIMA *et al.*, 2011).

Já os autores Spínola e Travassos (2012) afirmam que a Computação Ubíqua está presente quando serviços computacionais se tornam disponíveis às pessoas de modo que o computador não é visível nem necessita ser usado como uma ferramenta essencial para acessá-los. Os serviços podem ser materializados em qualquer tempo ou lugar, transparentemente, por meio do uso de dispositivos comuns usados no cotidiano (SPÍNOLA; TRAVASSOS, 2012).

Em resumo, a Computação Ubíqua sugere que a tecnologia faça parte do cotidiano das pessoas de modo que ela seja utilizada sem a necessidade de uma interação implícita, com uma menor frequência de configurações, inicializações ou controle dos sistemas computacionais que cercam as pessoas. Para estar de acordo com os conceitos apresentados, sistemas implementados para compor o cenário da Computação Ubíqua devem possuir algumas características, que são discutidas na Seção 2.1.1.

2.1.1 Características de Sistemas Ubíquos

Lima *et al.* capturam alguns requisitos a serem atendidos ao se desenvolver sistemas ubíquos. Dentre os requisitos capturados pode-se citar coordenação, interoperabilidade, sensibilidade ao contexto e invisibilidade.

Para atenderem ao requisito de **coordenação**, os sistemas ubíquos devem ser essencialmente dinâmicos, e os agentes[1] precisam estar aptos a coordenar suas atividades com outros agentes desconhecidos. Já a **interoperabilidade** entre sistemas pode ser entendida como a capacidade de um sistema se comunicar com outros sistemas, compartilhando dados ou invocando processos comuns, independentes de sua plataforma, arquitetura, linguagem de programação ou sistema operacional.

Considerando a **sensibilidade ao contexto**, os sistemas ubíquos precisam alterar seu comportamento e/ou estado de acordo com informações contextuais, obtidas das mais variadas formas. Assim, devem existir mecanismos que permitam a captura, o processamento e a distribuição de informações contextuais de interesse entre os agentes. A **invisibilidade** pode ser tratada como uma proatividade do sistema, de forma que ele realize suas funcionalidades com a mínima intervenção do usuário.

Já Spínola e Travassos (2012), como produto de uma revisão sistemática, consideram que é necessário que os sistemas que compõem o cenário da computação ubíqua levem em consideração certas características que são discutidas a seguir (SPÍNOLA; TRAVASSOS, 2012):

- **Onipresença de Serviços**: faz com que os usuários sejam capazes de se moverem com a sensação de carregar serviços computacionais junto com eles. Um *webmail* pode ilustrar essa onipresença, pois é possível acessá-lo em qualquer dispositivo conectado à Internet e que possua um navegador compatível.
- **Captura da Experiência**: faz com que sistemas ubíquos sejam capazes de capturar e registrar experiências para uso futuro. Por exemplo, um sistema de refrigeração percebe que sempre às 19h um usuário deixa a temperatura em 18^{o}C e ajusta a temperatura automaticamente nesse horário.
- **Descoberta de Serviço**: representa o mecanismo para permitir a descoberta de serviços proativa pelos sistemas ubíquos de acordo com o ambiente em que estão sendo usados. Isso permite alcançar a funcionalidade desejada pela descoberta de novos serviços ou

[1] O autor Lima (2011) chama de agentes os componentes de softwares que representam qualquer entidade de software presente no sistema capaz de realizar algum processamento e se comunicar.

informações. Por exemplo, um sistema baseado em localização percebe que o usuário está perto de um cinema, verifica a próxima sessão do filme que ele possa se interessar, fazendo a sugestão.

- **Composição de Função**: representa a habilidade de criar um serviço requisitado pelo usuário com base na existência de serviços básicos. Por exemplo, um sistema para celulares precisa de coordenadas de GPS para colher a localização, mas se o celular não tiver GPS, pode procurar por celulares próximos e solicitar a localização deles, obtendo a funcionalidade.
- **Heterogeneidade de Dispositivos**: torna as aplicações ubíquas capazes de adquirir mobilidade entre dispositivos heterogêneos. Assim, o software deve migrar entre dispositivos e se ajustar automaticamente a cada dispositivo. Por exemplo, um *webmail* pode ser acessado no *laptop* ou no *smartphone*.

Algumas características identificadas por Spínola e Travassos (2012) são apontadas por Lima *et al.* (2011). Uma dessas características é a invisibilidade, tratada por Spínola e Travassos (2012) como a habilidade dos sistemas computacionais estarem presentes no nosso cotidiano, utilizando os objetos que nos rodeiam, e diminuindo a sensação de uso explícito do computador, reforçando a percepção de que objetos ou dispositivos podem prestar serviços ou possuir algum tipo de "inteligência". Há portanto uma concordância com o conceito apresentado por Lima *et al.* (2011).

2.1.2 Sensibilidade ao Contexto

A sensibilidade ao contexto e a adaptabilidade, de acordo com Spínola e Travassos (2012), é o mecanismo para coleta de informação do ambiente onde o sistema ubíquo está sendo utilizado, sendo o comportamento adaptativo a capacidade de auto adaptação dinâmica de acordo com as limitações do ambiente para o qual o serviço deve ser oferecido, são apresentadas como uma característica por Lima *et al*.

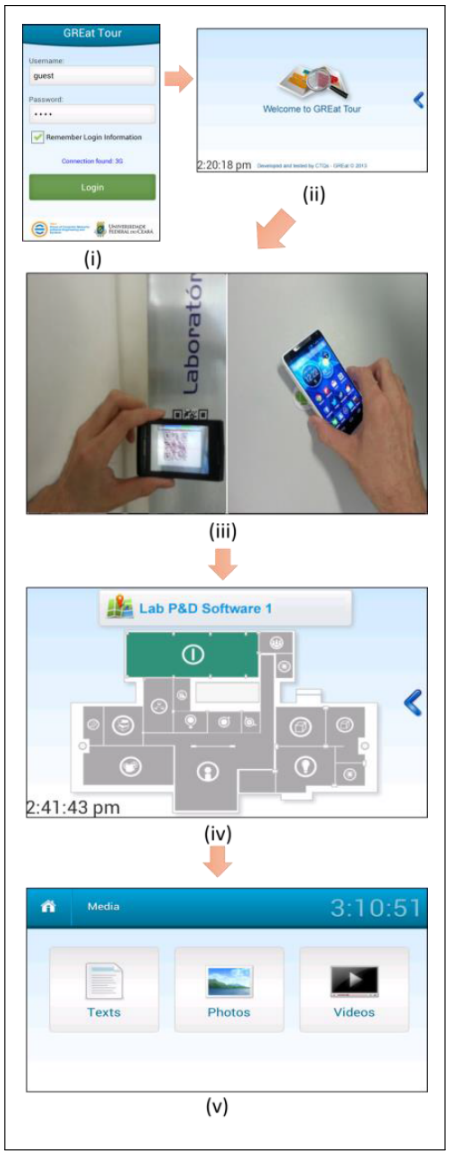
A sensibilidade ao contexto é uma das características de sistemas ubíquos que se destaca por permitir criar aplicações relevantes para o usuário final. Como exemplo de aplicação sensível ao contexto podemos citar o GREat Tour, um guia móvel e sensível ao contexto que é produto da linha Mobiline (MARINHO *et al.*, 2012). Essa aplicação tem adaptação de conteúdo baseado na localização física no prédio do GREat (Grupo de Redes de Computadores, Engenharia de Software e Sistemas). Quando o sistema percebe que mudou de ambiente, saindo, por exemplo,

da recepção para o auditório, a aplicação se adapta e mostra informações do novo ambiente.

Conforme ilustrado na Figura 2, no GREat Tour (i) o visitante fornece um usuário e senha para autenticar-se na aplicação e (ii) estando autenticado, uma tela de boas-vindas é exibida. Após a exibição dessa tela, (iii) o visitante informa sua localização por meio da leitura de QR Codes ou de *tags* NFC. A aplicação então (iv) exibe o mapa do laboratório para o visitante, destacando sua posição atual. (v) O visitante visualiza as informações disponíveis para o ambiente no qual ele se encontra, como textos, vídeos e imagens (LIMA *et al.*, 2013).

Outra aplicação sensível ao contexto é o Ubiprint (LIMA, 2011), cuja adaptação também é baseada na localização. Se um usuário do Ubiprint desejar realizar uma impressão,

Figura 2 – GREat Tour



Fonte – (LIMA *et al.*, 2013)

automaticamente será escolhida a impressora mais próxima dele de modo transparente.

2.2 Linhas de Produtos de Software

Abordagens de reuso de software encorajam o desenvolvimento de software usando artefatos pré-existentes, ao invés de criá-los do zero (HOLMES; WALKER, 2013). Dentre os benefícios do reuso sistemático de software estão o aumento da qualidade dos produtos, diminuição dos custos e maximização dos lucros (FRAKES; KANG, 2005; POHL *et al.*, 2010). Uma maneira de implementar o reuso é através de Linha de Produtos de Software.

2.2.1 Definições

Segundo Gurp *et al.* (2001), uma Linha de Produtos de Software captura as similaridades entre produtos de software para uma família de produtos, permitindo que o desenvolvedor foque em questões específicas dos produtos mais do que no que é comum a todos os produtos (GURP *et al.*, 2001). Para Griss (2000), uma linha de produto é um conjunto de produtos que compartilham um conjunto de requisitos, mas também exibem variabilidades significantes nesses requisitos. Griss (2000) afirma que as similaridades e variabilidades podem ser exploradas tratando o conjunto de produtos como uma família e decompondo o projeto e implementação desses produtos em um conjunto de componentes compartilhados que possuem interesses distintos (GRISS, 2000).

Outro conceito de Linha de Produto de Software é definido pelo SEI (*Software Engineering Institute*), e afirma que uma LPS "é um conjunto de sistemas intensivos em software que compartilham um conjunto de características comuns e gerenciadas, satisfazendo uma necessidade específica de um seguimento particular de mercado, e que são desenvolvidos a partir de um conjunto de ativos de um modo prescrito" (CLEMENTS; NORTHROP, 2002). Esse núcleo de artefatos forma a base para a LPS, e pode se referir a arquiteturas, componentes de software reusáveis, modelos de domínios, requisitos, planos e casos de testes, casos de uso e código-fonte, dentre outros (NORTHROP, 2002).

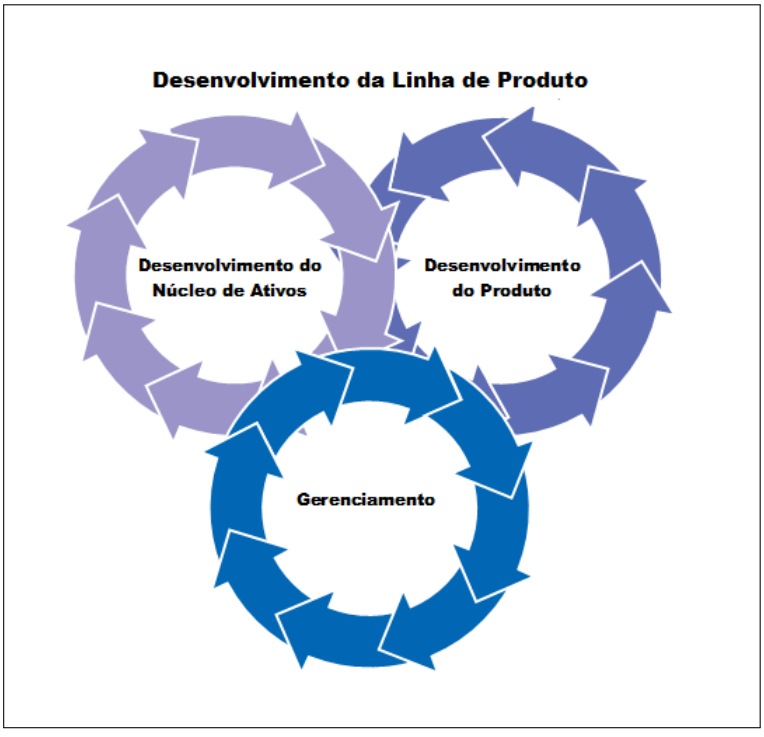
De acordo com Northrop (2002), uma Linha de Produto de Software possui três atividades essenciais e interativas que seguem, e são ilustradas na Figura 3 (NORTHROP, 2002):

- **Desenvolvimento do Núcleo de Artefatos**: envolve atividades de levantamento de similaridades e diferenças dos produtos, determinação das restrições de produção e verificação

de artefatos pré-existentes. Tem por objetivo delimitar o escopo da linha de produto, que descreve quais produtos serão gerados pela linha, além de gerar um plano de produção que descreve como os produtos são produzidos a partir do Núcleo de Artefatos;

- **Desenvolvimento do Produto**: envolve as atividades de geração do produto, observando o escopo da linha e obedecendo o plano de produção. No entanto, essa etapa depende também dos requisitos de cada produto, podendo afetar o escopo, Núcleo de Artefatos e o plano de produção;
- **Gerenciamento**: acontece nos níveis técnico (ou de projeto) e organizacional (ou empresarial). O gerenciamento no nível técnico garante que, na execução das duas atividades anteriores, os grupos se empenhem nas suas atividades, seguindo o processo definido para a linha de produto. O gerenciamento no nível organizacional deve garantir que as unidades organizacionais recebam os recursos corretos (como pessoal bem capacitado) em quantidades suficientes.

Figura 3 – Atividades Essenciais de uma Linha de Produto



Fonte – Adaptado de (NORTHROP, 2002)

Os autores Pohl *et al.* (2010) apresentam uma abordagem de engenharia para LPS, e definem Engenharia de Linha de Produto de Software como "um paradigma para desenvolver *software* usando plataformas e customização em massa" (POHL *et al.*, 2010). Plataforma de

Software é conceituada como “um conjunto de sub-sistemas e interfaces que formam uma estrutura comum do qual um conjunto de produtos derivados podem ser eficientemente desenvolvidos e produzidos” (MEYER; LEHNERD, 1997). Já a customização em massa é “a produção em larga escala de bens adaptados às necessidades do consumidor” (DAVIS, 1987).

Segundo Pohl *et al.* (2010), o paradigma da Engenharia de Linha de Produto de Software separa dois processos:

- **Engenharia de Domínio**: é o processo de engenharia de Linha de Produto de Software em que as similaridades e as variabilidades da linha de produto são definidas e concretizadas;
- **Engenharia de Produto**: é o processo de engenharia de Linha de Produto de Software em que as aplicações da linha de produto são construídas pelo reuso dos artefatos do domínio e explorando a variabilidade da linha.

Northrop (2002) também considera os conceitos de engenharia de domínio e engenharia de aplicação. Para ela, *domínio* é um corpo de conhecimento especializado, uma área de *expertise* ou uma coleção de funcionalidades relacionadas. Afirma também que dois momentos são executados na criação de uma LPS: *engenharia de domínio*, onde ao final é gerado o Núcleo de Artefatos, com base nesse conhecimento e *engenharia de aplicação*, cujo resultado é o produto em si, partindo do núcleo de artefatos (NORTHROP, 2002). Neste trabalho é considerado o conceito proposto por Pohl *et al.* (2010) por se tratar de uma visão mais completa de LPS.

2.2.2 Modelo de Características

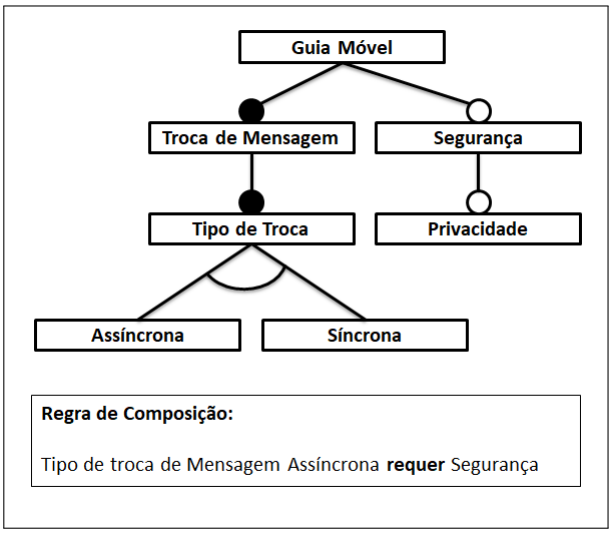
Kang *et al.* (1990) afirma que uma característica pode ser definida como um aspecto, ou uma qualidade visível para o usuário, bem como uma característica de um sistema de software. Para Kang *et al.* (1990), um modelo de características representa as características básicas de uma família de sistemas e os relacionamentos entre elas. A relação estrutural “*consiste de*” representa o agrupamento lógico de características, e características *alternativas* ou *opcionais* devem ser indicadas. Além disso, regras de composição definem a semântica existente entre as características, que não são expressadas no diagrama de características. Para expressar essas regras, podem ser utilizadas sentenças como “mutuamente exclusiva” ou “requer” entre características. (KANG *et al.*, 1990).

Como visto na Seção 2.2.1, no processo de desenvolvimento de uma LPS são capturadas as similaridades e variabilidades dos produtos da linha. Para Marinho (2012), modelos de variabilidade definem os pontos de variação, ou características que podem ou não

serem selecionadas ao se gerar um produto específico de LPS e podem ser representadas através de Modelos de Características (MARINHO, 2012).

No exemplo ilustrado pela Figura 4, as características *Segurança* e *Privacidade* são opcionais. Já a característica *Troca de Mensagem* pode ser escolhida para ser realizada de forma *Síncrona* ou *Assíncrona*. Como exemplo de regra de composição é especificado que, caso o produto possua a característica *Assíncrona*, o mesmo deve possuir a característica *Segurança*.

Figura 4 – Fragmento da LPS Mobiline (MARINHO *et al.*, 2012), um exemplo de modelo de características na representação proposta por (KANG *et al.*, 1990)



Fonte – O Autor.

Existem várias formas de representar a variabilidade em LPS. Dentre as quais pode-se citar o Modelo de Variabilidade Ortogonal, onde são representadas apenas as características variáveis em LPS (POHL *et al.*, 2010) e a Variabilidade Orientada a Aspectos utilizando a linguagem ADORA, que representa características transversais às aplicações (GLINZ *et al.*, 2002).

2.2.3 Linha de Produto de Software Dinâmica

Como visto na Seção 2.1, dentre as características de um sistema ubíquo, está a sensibilidade ao contexto, que envolve aspectos de percepção do ambiente que o cerca, e a capacidade de se auto adaptar conforme esse ambiente muda. Com a crescente necessidade de software adaptativo, que demanda um comportamento autônomo e propriedades auto-gerenciamento tem trazido novos desafios para adaptação dinâmica em família de sistemas.

Desse modo, Linhas de Produtos de Software Dinâmicas (LPSDs) têm emergido como um paradigma para gerenciar variabilidade em tempo de execução em qualquer tempo (CAPILLA *et al.*, 2014a). LPSDs são LPSs que incluem mecanismos para alterar variantes em sistemas auto adaptativos em tempo de execução (HALLSTEINSEN *et al.*, 2008).

Também em (CAPILLA *et al.*, 2014a), os autores definem três propriedades para que uma LPS seja caracterizada como LPSD:

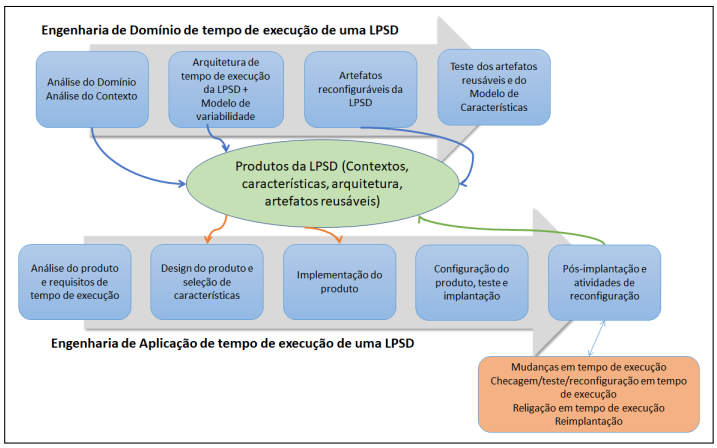
- **Suporte a gerenciamento a variabilidade em tempo de execução**: uma LPSD deve suportar a ativação e desativação de características e mudanças na variabilidade estrutural que pode ser gerenciada em tempo de execução. Reconfiguração em tempo de execução é outra propriedade de variabilidade dinâmica deve obedecer.
- **Ligação múltipla e dinâmica**: são necessários múltiplos tempos de ligação ao invés de um único tempo de ligação, como no momento em que o software adapta suas propriedades de sistema para o novo contexto, características podem ser ligadas várias vezes em diferentes tempos de ligação.
- **Sensibilidade ao contexto e auto adaptação para comportamento autonômico**: LPSs Dinâmicas precisam lidar com propriedades sensíveis ao contexto que são usadas como dados de entrada para mudar os valores das variantes de sistema dinamicamente e/ou selecionar novas opções de sistema dependendo das condições do ambiente.

Como é possível verificar nessas propriedades, o foco de uma LPSD é no gerenciamento de variabilidade em tempo de execução. A Figura 5 mostra o ciclo de vida de desenvolvimento para dar suporte a uma LPSD. O ciclo de vida envolve as engenharias de domínio e de aplicação a exemplo de uma LPS tradicional. No entanto, é importante ressaltar que a última atividade da engenharia de domínio prevê uma etapa de pós-implantação, em que há o gerenciamento de mudanças da aplicação em tempo de execução, bem como a a checagem, teste e reconfiguração em tempo de execução.

2.2.4 Linha de Produto de Software Sensível ao Contexto

Como observado na Seção anterior, uma LPS para ser considerada dinâmica deve tratar a variabilidade de contexto em tempo de execução. A solução proposta neste trabalho envolve raciocínios em tempo de projeto, mesmo considerando a sensibilidade ao contexto como em LPSD. Por outro lado, esta dissertação tem como ponto de partida o conceito de LPS Sensível ao Contexto.

Figura 5 – Ciclo de vida de desenvolvimento para dar suporte a LPSD



Fonte – (CAPILLA *et al.*, 2014a)

O trabalho de de Fernandes e Werner (2008) denomina como Linhas de Produto de Software Sensíveis ao Contexto (LPSSC) as LPSs que suportam o desenvolvimento de aplicações sensíveis ao contexto (FERNANDES; WERNER, 2008). De acordo com Costa (2012), o foco das LPSSCs é gerar produtos capazes de prover serviços de acordo com as necessidades do usuário, analisando o ambiente em que o mesmo se encontra (COSTA, 2012). Em LPSSC existe o MMSC (Modelo de Características Móvel e Sensível ao Contexto) (MARINHO, 2012), que representa as similaridades e variabilidades e as regras de composição entre as características juntamente com as regras de adaptação baseadas no contexto.

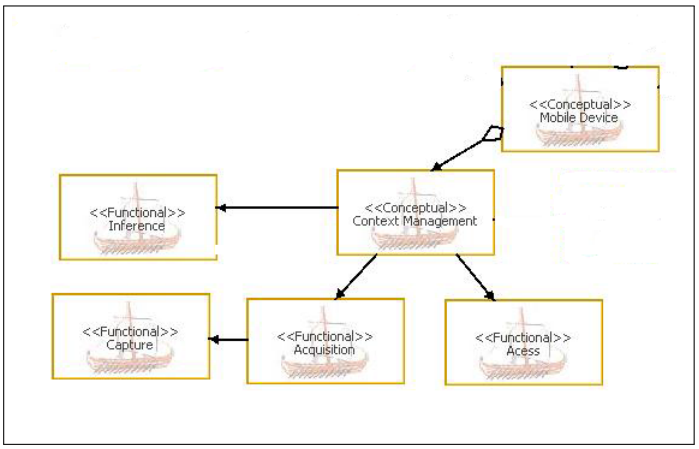
No trabalho de Marinho *et al.* (2010), é proposta uma linha de produto de software para o domínio de aplicações móvel e sensível ao contexto. Para o desenvolvimento dessa linha, foram realizados três ciclos. No primeiro ciclo foram identificadas as similaridades presentes em aplicações móveis e sensíveis ao contexto através de revisão de um conjunto de artigos que consideravam sensibilidade ao contexto e da análise a partir de aplicações desenvolvidas no grupo GREat. Dentre as aplicações pesquisadas foram detectadas características como mobilidade, troca de mensagens, descrição e descoberta de serviços. Como produtos dessa etapa foram obtidos o modelo de características que representavam uma LPS cujos produtos são aplicações sensíveis ao contexto e a arquitetura conceitual de alto nível.

Nos demais ciclos, foram elicitados os requisitos e características de um domínio específico de aplicação, no caso, Guia de Visitas Móvel e, por fim, foi configurado um produto, o *GREat Tour*, que pode ser visto na Figura 2 (MARINHO *et al.*, 2010).

Na Figura 6 é demonstrado um fragmento do modelo de características do Nível Básico da LPS móvel e sensível ao contexto. Nele é prevista a característica Dispositivo Móvel

(*Mobile Device*), que se relaciona com a característica Gerenciamento de Contexto (*Context Management*). Esse gerenciamento se dá através das características Acesso (*Access*) e Aquisição (*Acquisition*). A aquisição acessa as informações de contexto capturadas do ambiente através da característica Captura (*Capture*). Juntamente com a característica Inferência (*Inference*), as características anteriormente citadas formam um subconjunto das características principais que uma aplicação sensível ao contexto deve possuir.

Figura 6 – Fragmento do modelo de características do domínio de aplicações Móvel e Sensível ao Contexto



Fonte – Adaptado de (MARINHO *et al.*, 2010)

2.2.5 Modelo de Características Móvel e Sensível ao Contexto - MMSC

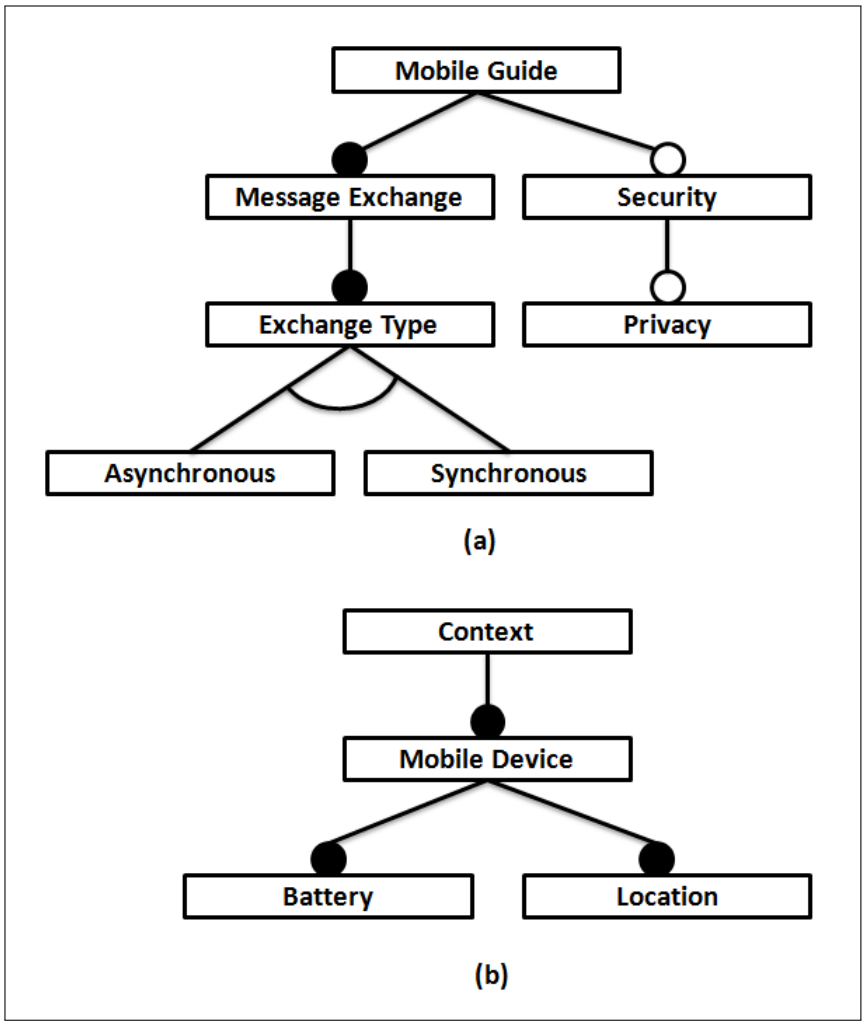
Modelos de características são a principal maneira de representação da variabilidade em LPSs. Em LPSSCs, existem os Modelos de Características Móveis e Sensíveis ao Contexto (MMSCs), que combinam um modelo de características tradicional com um modelo de contexto a fim de proporcionar uma melhor descrição da linha. MMSC é composto por quatro diagramas: (i) Modelo de Sistema (MS), que representam a variabilidade de LPS, (ii) Modelo de Contexto (MC), representando os contextos conhecidos, (iii) regras de composição (RC), relacionada com ao MS e (iv) regras de adaptação (RA) ou regras de contexto, relacionados com o MC (MARINHO, 2012; MARINHO *et al.*, 2012).

Modelo do sistema é um diagrama representado como uma árvore contendo uma única raiz, que representa o domínio. Os outros nós correspondem às características do domínio e as arestas descrevem as relações hierárquicas entre essas características (MARINHO *et al.*, 2012). Um exemplo de MS é encontrado na Figura 7(a). O nó raiz “MobileGuide” corresponde

ao domínio que está sendo representado. O nó “ContextManagement” é uma característica obrigatória e Persistência é um recurso opcional no domínio de guias móveis. As arestas representam, por exemplo, que Aquisição é uma característica obrigatória relacionada com “ContextManagement” e não relacionadas com Persistência.

O Modelo de Contexto é uma especialização de modelo de sistema, que tem quatro níveis. O primeiro nível é o contexto modelado e contém apenas um nó. Os nós no segundo nível representam as entidades de contexto e os nós no terceiro nível e também quarto nível mostram Informação de Contexto e Atributos de Contexto, respectivamente (MARINHO *et al.*, 2012). Na Figura 7(b) está presente um exemplo de MC. O nó raiz “ContextOne” representa um contexto que as aplicações sensíveis ao contexto precisam realizar adaptações através de entidade de contexto. A Entidade de Contexto (por exemplo, “MobileDevice”) é o objeto ou entidade observado pelo contexto em questão. Uma entidade de contexto fornece informações

Figura 7 – Fragmentos do Modelo de Sistema (a) e Modelo de Contexto (b) do MobiLine



Fonte – O Autor.

de contexto. Ela representa as informações da entidade que serão lidas, porque nem todas as informações fornecidas pela entidade de contexto são relevantes para a aplicação. O nível da bateria e a localização são exemplos de informações de contexto capturadas por um dispositivo móvel. No quarto nível existem atributos do contexto, que não estão representados na figura.

Uma Regra de Composição é formalmente especificada como uma implicação de uma expressão antecedente a uma expressão consequente, sendo cada expressão uma fórmula proposicional sobre um conjunto de características e atributos contidos no Modelo de Sistemas (MARINHO *et al.*, 2012). Na expressão (2.1), é apresentada uma simplificação da regra de composição e um exemplo são apresentados. A regra CR1 mostra que a presença de Gestão Contexto e Persistência implicam na presença de Tuplas.

$$\begin{aligned} CR = &< Expr.\ Antecedente > \Longrightarrow < Expr.\ Consequente > \\ CR1 = &\ GestaoDeContexto \wedge Persistencia \implies Tuplas \end{aligned} \tag{2.1}$$

Uma Regra de Adaptação também é formalmente especificada como sendo uma implicação de uma expressão de contexto para uma expressão do sistema (MARINHO *et al.*, 2012). Um evento pode ser um estado ou a combinação de estados de um sistema. Uma ação também pode ser um novo estado ou uma combinação dos estados no sistema. Por exemplo, se o nível da bateria dispositivo móvel atinge valores inferiores a 10%, uma possível ação do sistema é desativar a exibição de vídeos. Em (2.2), expressão simplificada da regra de adaptação e um exemplo são apresentados. A regra AR1 restringe que o nível da bateria do dispositivo atingir um nível inferior a 50% implica em obter a localização a partir das preferências do dispositivo.

$$\begin{aligned} AR = &< Expr.\ Antecedente > \Longrightarrow < Expr.\ Consequente > \\ AR1 = &\ (DispositivoMovel.Bateria < 50) \implies Localizacao.PreferenciasDispositivo \end{aligned} \tag{2.2}$$

Um exemplo de LPSSC é o Mobiline (MOBILINE, 2010) (MARINHO *et al.*, 2013), que é uma LPSSC para o domínio móvel e sensível ao contexto. Esse domínio tem a adaptabilidade como princípio e sugere que durante o tempo de execução, as aplicações devem ser capazes de se adaptar de acordo com as mudanças de contexto (MARINHO *et al.*, 2013). Um fragmento do Mobiline é apresentado na Figura 7.

2.3 Ontologias

Ontologias se apresentam como a representação de um conceito, sendo uma forma de organizar informações. De acordo com Gruber (1993), ontologia é uma especificação explícita de uma conceitualização. Uma ontologia define os conceitos, relacionamentos e outras distinções que são relevantes para o modelar um domínio (por exemplo, domínio de guias móveis e sensíveis ao contexto e domínio de planos de saúde suplementar). Além disso, a sua especificação assume a forma das definições do vocabulário representacional (por exemplo, classes e relações), provendo significado para esse vocabulário e restrições formais para sua utilização coerente (GRUBER, 2009). Segundo Almeida e Bax (2003), podem ser utilizados vários tipos de estruturas para organizar a informação. Como exemplos, pode-se citar as estruturas organizadas a partir de termos (arquivos de autoridade, glossários e dicionários) e as estruturas que se organizam a partir de conceitos e seus relacionamentos. Nesse último tipo de estrutura encontram-se as ontologias, os *thesaurus* e as redes semânticas (ALMEIDA; BAX, 2003).

Filho (2011) apresenta a evolução das formas de representação das ontologias e expõe que as primeiras linguagens foram KIF (GENESERETH, 1998) e KL-ONE (BRACHMAN; SCHMOLZE, 1985). A primeira é baseada em Lisp, usada para expressar sentenças de lógicas de predicados de primeira ordem. Já a segunda implementa redes de herança estruturadas, contendo descrições de conceitos e as relações de generalização e especialização entre esses conceitos. A KL-ONE deu origem à Lógica Descritiva (DL) e, em paralelo, ao seu desenvolvimento surgiram representações de ontologias em UML. Por fim, adicionou-se a questão da interoperabilidade na modelagem de ontologias. Com isso, linguagens baseadas em XML surgiram, como RDF (*Resource Description Framework*), RDFS (*Resource Description Framework Schema*) e Ontology Web Language (OWL) (FILHO, 2011).

Dentre as linguagens baseadas em XML, podemos destacar a OWL. Ela pode ser utilizada por aplicações que necessitam processar conteúdo de informações, ao invés de apenas apresentar essas informações a humanos. A OWL facilita a interpretabilidade por parte da máquina, provendo vocabulário adicional juntamente com uma semântica formal que favorecem os mecanismos de inferência de novos conhecimentos (W3C, 2004). A OWL possui três sub-linguagens que serão detalhadas a seguir, segundo o grau de expressividade (W3C, 2004):

- **OWL *Lite***: fornece suporte aos usuários que necessitam principalmente de uma hierarquia de classificação e restrições simples. Ela apenas permite valores de cardinalidade 0 ou 1. Possui uma menor complexidade formal que os outros tipos de OWL;

- **OWL DL**: fornece suporte aos usuários que querem o máximo de expressividade, mantendo completude computacional (todas as conclusões são garantidas para ser computáveis) e decidibilidade (todas as computações terminarão em tempo finito). OWL DL é assim chamada devido à sua correspondência com lógica descritiva, um campo de pesquisa que estuda as lógicas que formam a base formal da OWL;
- **OWL *Full***: destinada a usuários que querem expressividade máxima mas não necessitem de garantias computacionais.

Dentre as sub-linguagens apresentadas para realizar o enriquecimento semântico de LPS, destaca-se a OWL DL, por ser expressiva e decidível, apresentando-se como a alternativa de se representar o conhecimento e ser processada por máquinas. A OWL *Lite* não é suficientemente expressiva e a OWL *Full* não é decidível. As características da OWL DL serão abordadas a seguir (W3C, 2004; FILHO, 2011).

A maioria dos elementos de uma ontologia OWL faz referência a classes, propriedades, instâncias de classes e relacionamentos entres esses instâncias (W3C, 2004). As classes correspondem aos conceitos básicos em um domínio. Em OWL é importante ressaltar que todas as novas classes definidas são subclasses da classe *Thing*. Outros elementos da OWL são os indivíduos, que representam a concretização de uma classe. São entidades atômicas, no sentido de que eles representam a si, e não possuem uma conotação conceptiva de outros indivíduos (FILHO, 2011). Fazendo uma associação com orientação a objetos, um indivíduo é uma instância da classe.

As propriedades são relações binárias entre indivíduos. São divididas em dois tipos: (i) Propriedades de Objetos (*Object Properties*), que relacionam instâncias de duas classes, e (ii) Propriedades de Tipo de dados (*Datatype Properties*), que relacionam instâncias de classes com literais como por exemplo, XML *Schema* como *string*, *boolean*, *int*, *date* e outros. Além de caracterizar, as propriedades podem prover restrições nos relacionamentos entre os indivíduos. Essas restrições podem se dar por modo quantitativo, por meio de operadores "*para todo*" ou "*existe*", ou de modo a expressar cardinalidade, limitando o número de elementos que satisfazem a relação. (W3C, 2004; FILHO, 2011).

2.4 Conclusão

Neste Capítulo foram apresentados conceitos relacionados a sensibilidade ao contexto, LPS, LPSD, LPSSC e ontologias. Estes conceitos apresentados são importantes para

compreender a solução proposta neste trabalho. Este trabalho utiliza Mobiline como estudo de caso e os exemplos mostrados na Figura 7 da Seção 2.2.5 deste capítulo são utilizados para ilustrar a solução em execução.

3 TRABALHOS RELACIONADOS

Neste capítulo são apresentados trabalhos relacionados à ideia desta pesquisa, que propõe um método e uma ferramenta para análise de modelos de características de LPSSC e seu enriquecimento semântico. Seus principais aspectos são comparados com a solução proposta e são discutidos nas seções deste capítulo. Na Seção 3.1 discute-se o enriquecimento semântico em LPSSC. Já na Seção 3.2, analisa-se a verificação de modelo de características considerando o contexto dos usuários dos produtos. Na Seção 3.3 é realizada uma análise comparativa desses trabalhos e na Seção 3.4 são feitas as considerações finais do capítulo.

3.1 Enriquecimento Semântico

Nesta Seção são apresentados trabalhos que propõem o enriquecimento semântico em LPS ou que permitem o enriquecimento semântico por abordarem a representação de LPS através de ontologias. Também é detalhado o método de enriquecimento semântico que compõe a solução proposta nesta dissertação.

A seleção dos trabalhos foi realizada através de pesquisa em *engines* de busca. As bases consultadas foram ACM DL[1] e IEEEXplorer[2] e foram procurados trabalhos que tratassem de LPS, enriquecimento semântico e ontologias. Além do trabalho de Filho (2011), que já havia sido selecionado previamente com o objetivo de ter estendido o processo de enriquecimento semântico, os seguintes trabalhos foram selecionados:

- O *framework* de (ZAID *et al.*, 2009);
- A comparação de abordagens de modelagem de LPS com ontologias de (DERMEVAL *et al.*, 2015);
- A abordagem de (NESKOVIC; MATIC, 2015) e
- A abordagem de (ERFANI *et al.*, 2016).

O autor Filho (2011) apresentam o conceito de Linha de Produto de Software Semânticas (LPSS). Ele define LPSS como uma LPS onde o núcleo de artefatos reutilizáveis e artefatos de negócios são relacionados com o modelo de domínio e podem ser expressos utilizando uma ou mais ontologias. Assim, ontologias aumentam a expressividade do conhecimento em LPS e permitem inferir novas informações e relações implícitas usando algoritmos de inferência fornecidos pelo motores de inferência.

[1] http://dl.acm.org/
[2] http://ieeexplore.ieee.org/Xplore/home.jsp

Modelo de características podem ser um ponto de partida para a adição de semântica para a LPS. Por exemplo, as relações entre uma característica e outros artefatos da linha (por exemplo, o código fonte e os requisitos de documentos) podem ser especificadas. O autor propõe um processo que envolve duas etapas principais: (i) transformação de um modelo de características em uma ontologia (OWL DL), realizada automaticamente por uma ferramenta chamada Fea2Onto; e (ii) a adição de novas relações com artefatos externos em um processo guiado por uma ontologia de domínio chamado SPLiSEM. No final do processo, um modelo mais expressivo da LPS é produzido.

A solução para adicionar semântica e o processo de criação de uma LPSS está ilustrada na Figura 8, envolvendo dois processos principais. Primeiro é feito uma transformação automática de um modelo de características para uma ontologia, chamada de *Fea2Onto*. Como resultado dessa transformação, é gerada uma ontologia que possui as mesmas informações do diagrama de características. A segunda etapa envolve a aplicação do *Modelo de Enriquecimento Semântico de Linha de Produto de Software* (SPLiSEM). Esse modelo é uma *top-ontology* que define a natureza dos relacionamentos e conceitos usados para enriquecer a ontologia gerado no primeiro processo (FILHO, 2011).

A abordagem ilustrada na Figura 8 é iniciada com um modelo de características. Esse modelo é traduzido em objetos Java que representam essas características. No próximo passo, a tradução desses objetos em uma ontologia é realizada. A ontologia é a representação fidedigna do modelo de características original. Essas traduções são feitas de maneira automática, por meio da ferramenta *Fea2Onto* comentada anteriormente. A ontologia gerada é estudos de caso relacionados ao domínio de evolução do software para ilustrar o benefício de compartilhar e reutilizar o contexto para várias tarefas de engenharia de software.

Comparando esses trabalhos, a proposta de (FILHO *et al.*, 2012) é mais focada em fornecer uma solução para adicionar semântica. Já (ZAID *et al.*, 2009), (DERMEVAL *et al.*, 2015), (NESKOVIC; MATIC, 2015) e (ERFANI *et al.*, 2016) trabalham com modelagem de características usando ontologias, que potencialmente pode ser um ponto de partida para o enriquecimento semântico. No entanto, a exceção de (NESKOVIC; MATIC, 2015), os trabalhos não abordam aspectos contextuais.

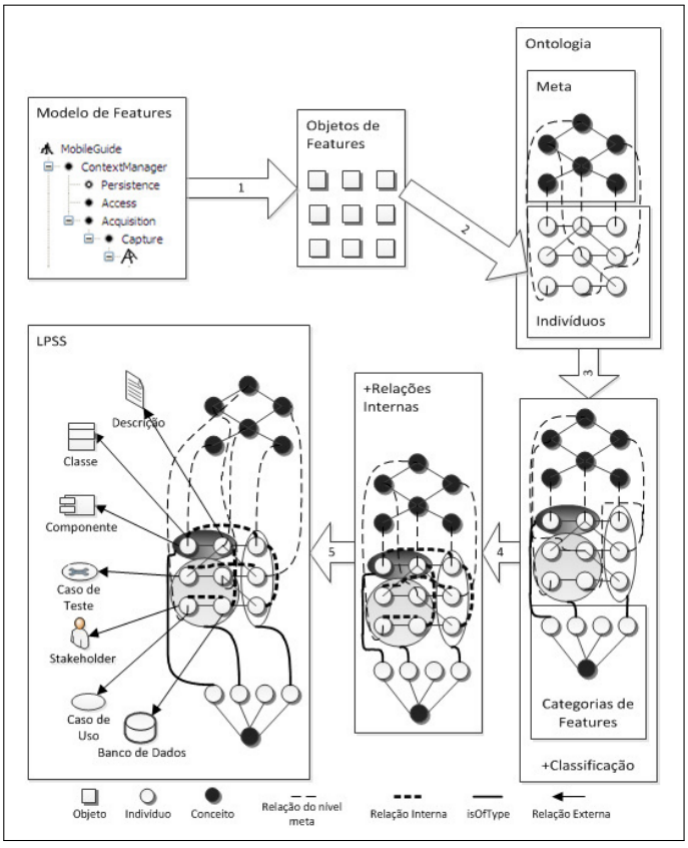
3.2 Verificação de Modelos de Características em LPSSC

Nesta Seção, são apresentados trabalhos que propõem a verificação de modelos de características em LPS tradicionais ou em LPSSC. Também é discutida a solução de verificação automática de modelos de características móveis e sensíveis ao contexto.

A seleção dos trabalhos foi realizada através de pesquisa em máquinas de busca. As bases consultadas foram ACM e IEEE e foram procurados trabalhos que tratassem de LPS, verificação de modelos e sensibilidade ao contexto. Além do trabalho de (COSTA, 2012), que já havia sido selecionado previamente com o objetivo de ter estendida a modelagem e ferramenta, os seguintes trabalhos foram selecionados:

- O *framework* de (ZAID *et al.*, 2009);
- A solução de Asirelli *et al.* (2011);
- A abordagem de (GUO *et al.*, 2012);
- A abordagem de (RINCóN *et al.*, 2014) e
- A abordagem de (AMJA *et al.*, 2016).

Figura 8 – Processo de enriquecimento semântico (FILHO, 2011)



Fonte – (FILHO, 2011)

Zaid *et al.* (2009), cujo trabalho também foi selecionado nos critérios de busca da Seção 3.1, sugerem um *framework* para modelagem de características que consiste de uma ontologia que provê formalmente uma especificação para modelos de características. Além disso, permite a integração de modelos de características segmentados, provendo uma checagem de consistência de modelos e detecção de conflitos baseados em regras. Eles utilizam regras SWRL (*Semantic Web Rules Language*) (HORROCKS *et al.*, 2004) para implementar as regras e um *reasoner* de lógica descritiva para avaliar os modelos e inferir outras informações de interesse considerando a variabilidade do software (ZAID *et al.*, 2009).

Asirelli *et al.* (2011) desenvolveram uma lógica temporal que pode ser interpretada sobre tais estruturas e que podem expressar propriedades sobre famílias e produtos. Os autores propuseram uma lógica para modelo de variabilidade em famílias de produto tirando vantagem do modo em que a lógica deôntica[3]. Desta forma, formalizam conceitos como violação, obrigação, permissão e proibição. Além disso, eles estenderam a lógica para capturar além do comportamento das famílias de produto, usando sistemas de transição modal como o modelo semântico básico. Por fim, Asirelli *et al.* (2011) desenvolveram algoritmos eficientes para derivar produtos válidos da família bem como verificar propriedades sobre produtos e famílias, baseados em *model checkers* existentes (ASIRELLI *et al.*, 2011).

Já Guo *et al.* (GUO *et al.*, 2012) apresentam uma abordagem para manutenção de modelos de características evolutivas. O modelo de características foi formalizado a partir de uma perspectiva ontológica e foi proposto um conjunto de restrições de consistência sintática e semântica, por exemplo, regras de boa formação dos modelos.

Rincón *et al.* (RINCóN *et al.*, 2014) propõem uma abordagem baseada em regras utilizando ontologias para analisar *dead features* e *falsos opcionais* em diagramas de características, identificando as causas e explicando-as em linguagem natural. Para realizar essa análise nove regras foram formalizadas em lógica de primeira ordem implementada em SWRL e Java.

Amja et al. (AMJA *et al.*, 2016) propõem uma abordagem que considera a variabilidade de contexto baseado em Linhas de Produto de Software. A abordagem cria um link semântico entre um modelo baseado em contexto RCA e um modelo de características, e usa o loop de adaptação MAPE-K para determinar as configurações de SPL apropriadas e que consideram mudanças de contexto. Os autores utilizam uma ontologia para representar um modelo combinado de contexto e características. Posteriormente, é realizado reasoning através

[3] lógica que estuda a validade de argumentos nos quais frase regidas por expressões como *É obrigatório que...*, *É permitido que...* desempenham papel relevante (GOMES, 2008)

de lógica descritiva. Também são definidas regras de contexto com base em SWRL aplicadas a configurações válidas da LPS.

Costa *et al.* (2015) propõem uma ferramenta de validação de modelos de características em LPSSC que implementa o processo PRECISE (MARINHO, 2012). Esse processo determina um conjunto de regras de boa formação de modelo de características e modelo de contexto, bem como a análise da obediência desses modelos às regras de composição e adaptação. Na ferramenta de Costa (2012), essas regras foram implementadas em EVL (Epsilon Validation Language), que funciona similarmente a restrições OCL (Object Constraint Language) (COSTA, 2012).

3.3 Discussão

A Tabela 2 apresenta uma comparação entre os trabalhos relacionados, observando os critérios de possibilidade de enriquecimento semântico por meio de ontologias, verificação de regras em modelos de características e de suporte a sensibilidade ao contexto.

Observando os trabalhos discutidos neste capítulo, é possível identificar algumas lacunas considerando o enriquecimento semântico, verificação de modelos e rastreabilidade de informações. Os trabalhos citados anteriormente, com exceção do trabalho de Filho (2011) e Costa (2012), abordam a análise de modelos de características, no entanto, não tratam a questão da sensibilidade ao contexto. O trabalho de Costa (2015) considera a sensibilidade ao contexto na análise dos modelos de características sensíveis ao contexto, porém não prevê enriquecimento semântico para dar suporte a rastreabilidade de informações.

Já os trabalhos de Dermeval *et al.*, Neskovic e Matic (2015) e Erfani *et al.* discutem a representação de LPS em ontologias permitindo, portanto, rastreabilidade de informações de através do uso de *reasoners*. No entanto, não são levados em consideração aspectos contextuais nem a verificação de modelos. O trabalho de Amja *et al.* propõe uma abordagem que envolve a representação com ontologias de uma LPS que trabalha com sensibilidade ao contexto, além de realizar a verificação de modelos, sendo, portanto, o que mais se aproxima da solução proposta nesta pesquisa.

Desse modo, é possível visualizar na Tabela 2 que há uma oportunidade para propor uma solução que envolva enriquecimento semântico com ontologias, leve em consideração a sensibilidade ao contexto dos produtos em uma LPSSC bem como trabalhe com a análise de MMSC.

Tabela 2 – Comparação dos Trabalhos Relacionados

Trabalhos	Verificação de Modelo de Características	Suporte a Sensibilidade ao Contexto	Enriquecimento Semântico
Zaid *et al.* (ZAID *et al.*, 2009)	+	-	+
Asirelli *et al.* (ASIRELLI *et al.*, 2011)	+	-	-
Filho *et al.* (FILHO *et al.*, 2012)	-	-	+
Guo *et al.* (GUO *et al.*, 2012)	+	-	+
Rincón *et al.* (RINCóN *et al.*, 2014)	+	-	+
Costa *et al.* (COSTA *et al.*, 2015)	+	+	-
Dermeval *et al.* (DERMEVAL *et al.*, 2015)	-	-	+
Neskovic e Matic (NESKOVIC; MATIC, 2015)	-	-	+
Amja *et al.* (AMJA *et al.*, 2016)	+	+	+
Erfani *et al.* (ERFANI *et al.*, 2016)	-	+	+

(+) Está presente no trabalho; (-) Está ausente no trabalho.

Fonte – O autor.

3.4 Conclusão

Neste capítulo foram descritas pesquisas que tratam tanto do enriquecimento semântico de uma LPS como de verificação do modelo de características dessas linhas. No capítulo 4 é apresentado o método proposto neste trabalho, que contempla as lacunas discutidas neste capítulo e que é ilustrado através de um exemplo de modelagem de uma LPSSC, a Mobline.

4 FIXONTO: O MÉTODO

Este capítulo descreve o método proposto para contemplar enriquecimento semântico e a verificação de modelos em LPSSC, que é o objetivo principal desta dissertação. Na Seção 4.1, é realizada uma breve justificativa do trabalho e é apresentada a visão geral do método proposto. A Seção 4.2 aborda a etapa de modelagem do método. A Seção 4.3 aborda etapa de tradução automática do modelo para ontologias e detalha os meta-modelos que serviram de base para as ontologias criadas. A Seção 4.4 trata da etapa de verificação do modelo em ontologias e das regras implementadas para avaliação desses modelos. Já a Seção 4.5 aborda a etapa de busca semântica. Por fim, um resumo do capítulo é realizado na Seção 4.6.

4.1 Visão geral do Método

Uma LPSSC pode obter benefícios da integração de enriquecimento semântico e verificação do modelo como, por exemplo, uma maior qualidade ao produto gerado e uma rastreabilidade entre os artefatos criados durante o desenvolvimento.

Este trabalho propõe um método que permite a criação de modelos de características sensíveis ao contexto e o enriquecimento semântico e a verificação automática dos mesmos. Além disso, a solução apresentada neste trabalho permite a recuperação da informação e rastreabilidade de artefatos e conhecimento sobre a modelagem no modelo verificado.

Um aspecto importante que deve ser mencionado, é a utilização de ontologias na grande maioria das etapas do método. A opção por ontologias, especificamente representada em OWL DL, é devido a oportunidade de utilizar apenas uma técnica para contemplar as demais etapas do método além da modelagem (verificação do modelo e buscas semânticas) e por ser uma linguagem formalizada por se basear lógica descritiva, garantindo a decidibilidade através de máquinas. Além disso, é possível utilizar *reasoners* para inferência de conhecimento e buscas semânticas. Portanto, a utilização de ontologias é apenas o meio para executar o método.

A elaboração do método seguiu as etapas que são descritas a seguir:

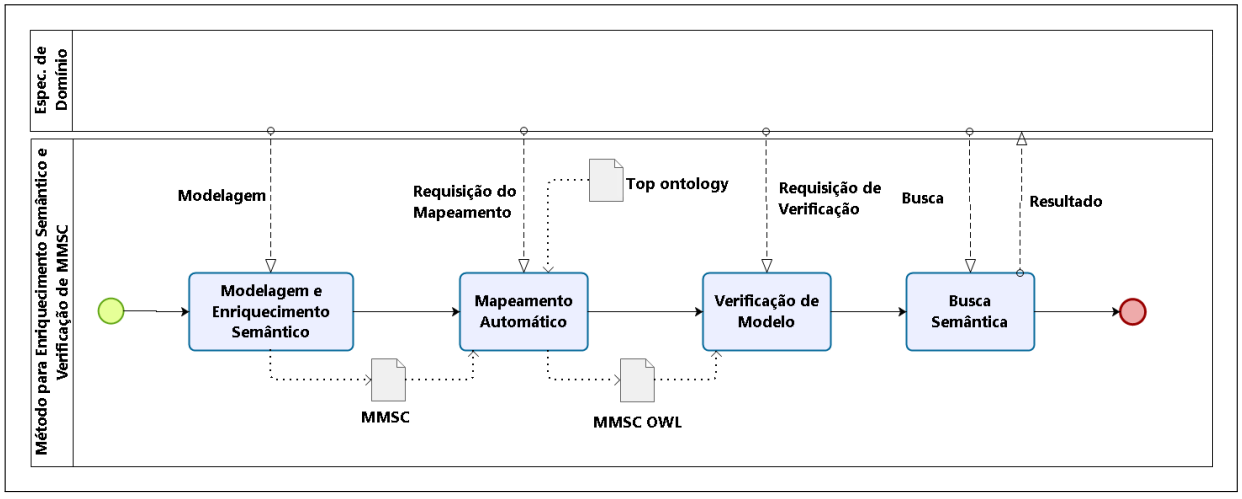
- Análise da literatura: após pesquisa bibliográfica e análise dos trabalhos relacionados, foram selecionadas soluções que contemplavam pelo menos uma parte do objetivo, que envolve enriquecimento semântico de MMSC e também a verificação desse modelo;
- Definição do esboço das etapas do método: foi elaborado um passo-a-passo de uma solução que envolvesse em conjunto o enriquecimento semântico e a verificação de MMSC. Dado

que com o uso de ontologias é possível atingir as duas partes do objetivo, foi definido um esboço do método que considera a utilização deste tipo de representação de domínio. Com os passos definidos, também foi escolhida a linguagem de representação de ontologias, no caso, OWL DL, por ser uma linguagem baseada em XML e em Lógica Descritiva sendo, portanto, interpretável por máquina e decidível;

- Criação de uma ontologia de domínio: uma das etapas do método envolve uma ontologia de domínio para guiar a transformação do MMSC. Foi definido que a ontologia de domínio que representa uma LPSSC seria baseada nos conceitos definidos em (MARINHO, 2012).

O método é composto por quatro etapas principais: (a) modelagem de características e contextos, incluindo o enriquecimento semântico; (b) transformação automática do modelo em ontologia; (c) a verificação automática do modelo de características e de contexto; e (d) a busca semântica. A Figura 9 mostra o método especificado na notação BPMN (OMG, 2011). As etapas são representadas como tarefas (*tasks*) e são detalhadas nas próximas seções.

Figura 9 – Visão Geral do Método FixOnto

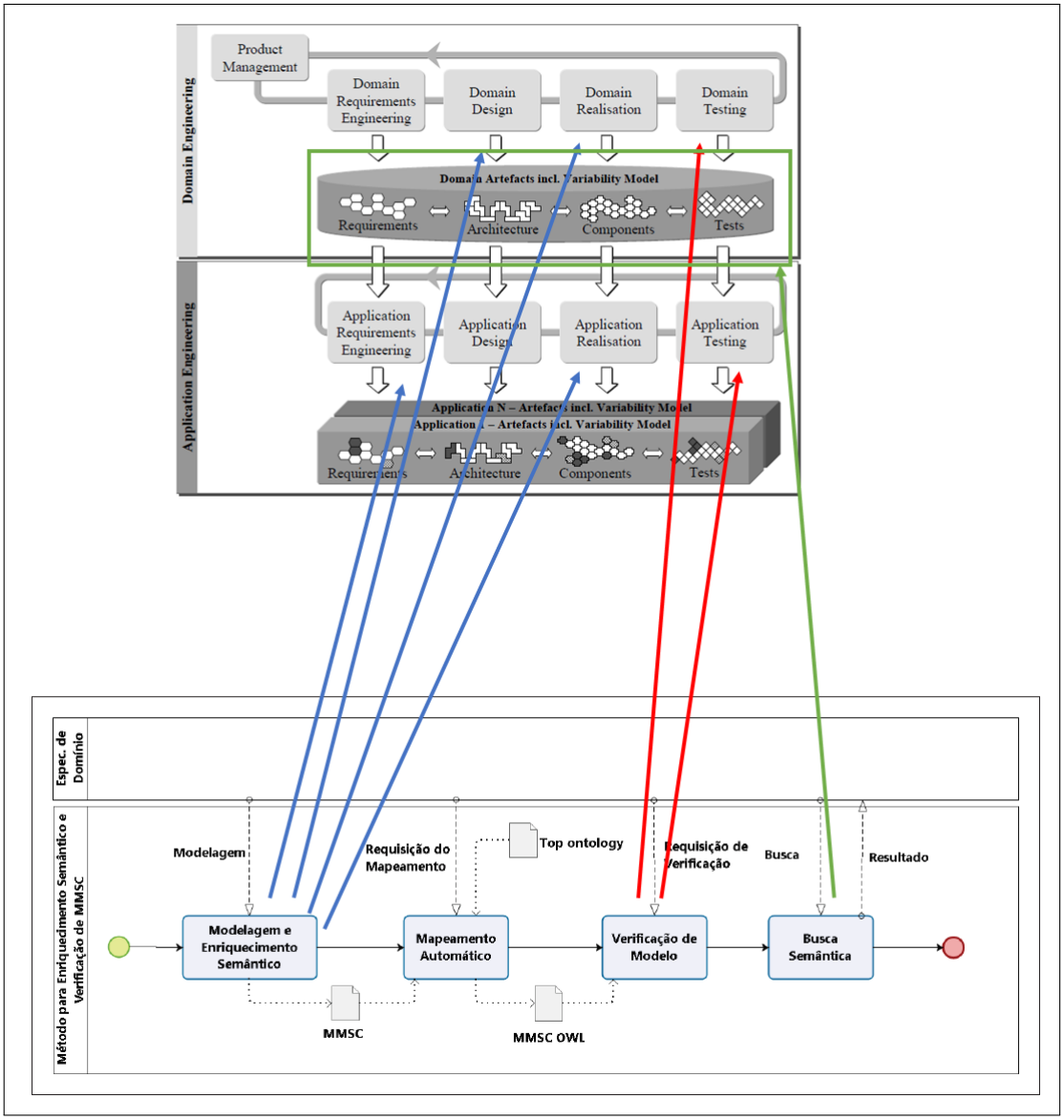


Fonte – O Autor.

Considerando a engenharia de LPS discutida na Seção 2.2, há a etapa de Engenharia de Domínio, onde é realizada a modelagem bem como a construção do núcleo de ativos da linha e a Engenharia de Aplicação, em que é realizada a derivação dos produtos e a atualização do núcleo de ativos. Na Figura 10 é possível visualizar a relação das etapas do método com as engenharia de domínio de uma LPS.

A Figura 10 demonstra que a etapa de Modelagem na FixOnto está associada às atividades de Projeto do Domínio e a sua realização, que são executadas durante a Engenharia de

Figura 10 – Relação entre as etapas do método FixOnto e a engenharia de domínio em LPS



Fonte – (COSTA, 2012)

Domínio. No entanto, pode ser necessário atualizar a modelagem, bem como o enriquecimento semântico, já na Engenharia de domínio. Desse modo, a modelagem na FixOnto também está associada às atividades de Engenharia de Requisitos da Aplicação e na sua realização.

Já a etapa de Verificação de Modelos está associada às atividades de Teste do Domínio e do Teste de Aplicação, que são executas na Engenharia de Domínio e de Aplicação, respectivamente. Já a busca semântica está associada ao núcleo de ativos da linha, tornando possível realizar buscas semânticas nos artefatos que o compõem.

Este trabalho é direcionado, em primeiro lugar, aos profissionais de modelagem de LPSSC, mas não se restringe às pessoas responsáveis pela modelagem, e sim a todos que

utilizarão a modelagem para desempenhar algum papel nas demais etapas de desenvolvimento. Assim, todos os envolvidos no desenvolvimento de aplicações sensíveis ao contexto também são beneficiados por usar um modelo mais expressivo. Esses profissionais são chamados de especialistas de domínio neste documento.

4.2 Modelagem e Enriquecimento Semântico

Na etapa **modelagem e enriquecimento semântico**, o especialista de domínio cria o modelo de características, modela o contexto e associa estes modelos a artefatos de domínio. As restrições entre características e contexto também são modeladas nessa etapa. Um MMSC é a saída da primeira etapa. Neste método, o enriquecimento semântico se refere à associação de características e contextos a outros artefatos do domínio (e.g.: casos de uso, código fonte, documentos de requisitos).

Deve ser ressaltado que essa etapa envolve os modelos que compõem o MMSC (MARINHO, 2012). O especialista de domínio, apresentado na Seção 4.1, deve especificar o Modelo de Sistema, onde são modeladas as características representando a variabilidade entre elas. O especialista também deve especificar o Modelo de Contexto, onde são determinados os contextos presentes no domínio e quais as informações serão observadas. Além desses modelos, devem ser especificadas as Regras de Composição (RC), sendo modeladas as restrições entre as características de MS, e as Regras de Adaptação (RA), relacionando uma situação contextual às características do MS.

A modelagem prevista nessa etapa pode ser realizada em qualquer ferramenta que dê suporte à definição de variabilidades e similaridades de uma família de produtos, bem como contextos, e que também permita associar características e contextos a artefatos que compõem o núcleo de ativos. Um exemplo de ferramenta de modelagem é a FixTure (COSTA, 2012; COSTA *et al.*, 2015).

4.3 Transformação Automática

Na segunda etapa (transformação automática), o MMSC é automaticamente transformado em ontologia, representada em OWL DL. Esta ontologia representa características, contextos, regras e artefatos presentes no MMSC. Nessa etapa, a transformação exige uma ontologia de domínio contendo os conceitos relacionados ao MMSC (por exemplo, característica,

contexto, regras).

Na literatura foram encontradas ontologias que representam diagramas de características (ZAID *et al.*, 2009) e também contexto (VIANA, 2010). No entanto, a ontologia de domínio utilizada no método se baseia nos meta-modelos presentes em (MARINHO *et al.*, 2012) e (COSTA *et al.*, 2015), cujos conceitos e regras norteiam a verificação proposta por este trabalho. A ontologia representa o modelo de sistema e de contexto, bem como as regras de adaptação e de contexto que representam uma LPSSC. As *top ontologies* serão descritas nas próximas seções.

Essa transformação precisa ser implementada de modo a mapear as características, contextos e regras, que são modeladas utilizando a sintaxe específica da ferramenta, com os conceitos presentes na Ontologia de Domínio que representa uma LPSSC. Pode ser utilizada qualquer linguagem de programação ou de Script, contanto que a saída dessa transformação seja uma ontologia descrita em OWL DL.

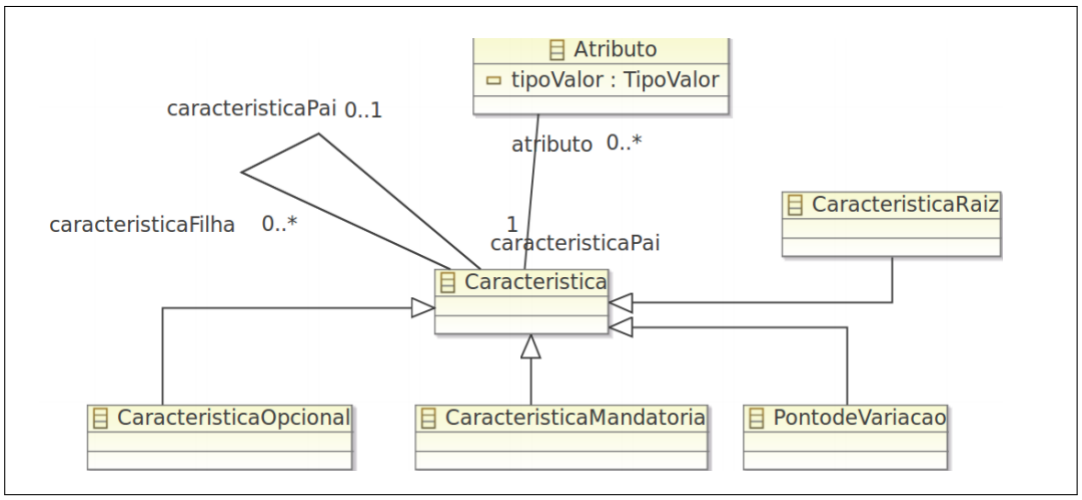
4.3.1 Modelo de Sistema e Modelo de Contexto

Em LPSSC, o modelo de sistema representa as variabilidades e similaridades entre características do domínio que está sendo modelado. Desse modo, é importante que conceitos como característica mandatória e opcional, dentre outros, estejam presentes na ontologia. Já o modelo de contexto é uma especialização do modelo de sistema, que representa os contextos de interesse do domínio, com as entidades e informações contextuais. Portanto, a ontologia também deve incluir esses conceitos. As Figuras 11 e 12 representam os meta-modelos do Modelo de Sistema (MS) e do Modelo de Contexto (MC) e a Figura 13 mostra um fragmento da ontologia de domínio que os representam.

A Figura 13 mostra que uma LPSSC é composta por elementos, que dizem respeito aos diagramas de sistema e de contexto. O modelo de sistema é formado pela hierarquia de característica. Nessa meta ontologia, as subclasses “Característica” podem se relacionar entre si, por exemplo, um indivíduo de “Característica Mandatória” pode ter como subcaracterística um indivíduo de “Ponto de Variação”, e a relação contrária também é possível. As características se relacionam através das propriedades *hasFatherFeature* e *hasChildFeatures*. A primeira indica que dada uma característica C que seja subcaracterística de B, dizemos que *hasFatherFeature(C,B)* e a segunda propriedade indica que B tem como filha C, podendo ser visualizada pelo axioma *hasChildFeatures(B,C)*.

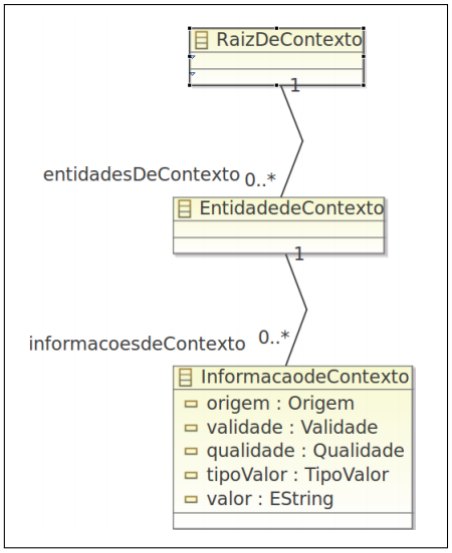
Ainda observando a Figura 13, é possível verificar que um modelo de contexto é

Figura 11 – Meta-modelo do Modelo de Sistema



Fonte – (COSTA, 2012)

Figura 12 – Meta-modelo do Modelo de Contexto



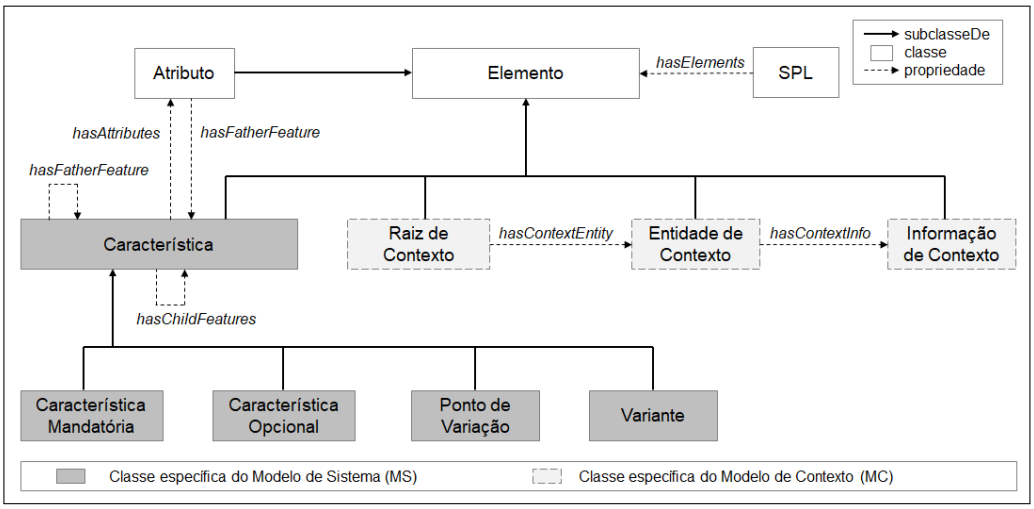
Fonte – (COSTA, 2012)

composto por uma raiz de contexto (*ContextRoot*), que representa o contexto que é de interesse para a aplicação para que sejam realizadas adaptações por uma entidade de contexto (*ContextEntity*), que é o objeto ou entidade observada no contexto em questão, como por exemplo, um dispositivo móvel.

Por fim, um modelo de contexto é composto por informações de contexto, que representa quais informações da entidade serão capturadas, pois nem todas as informações que a entidade pode fornecer são de interesse para determinada aplicação sensível ao contexto. Por exemplo, o nível de bateria e a força do sinal WIFI são informações de contexto que podem ser capturadas em um dispositivo móvel. A seguir, são mostradas as regras de composição e

regras de adaptação que definem restrições às relações entre as características e a modelagem contextual.

Figura 13 – Ontologia de Domínio que representa um Modelo de Sistema e Modelo de Contexto em LPSSC



Fonte – O autor.

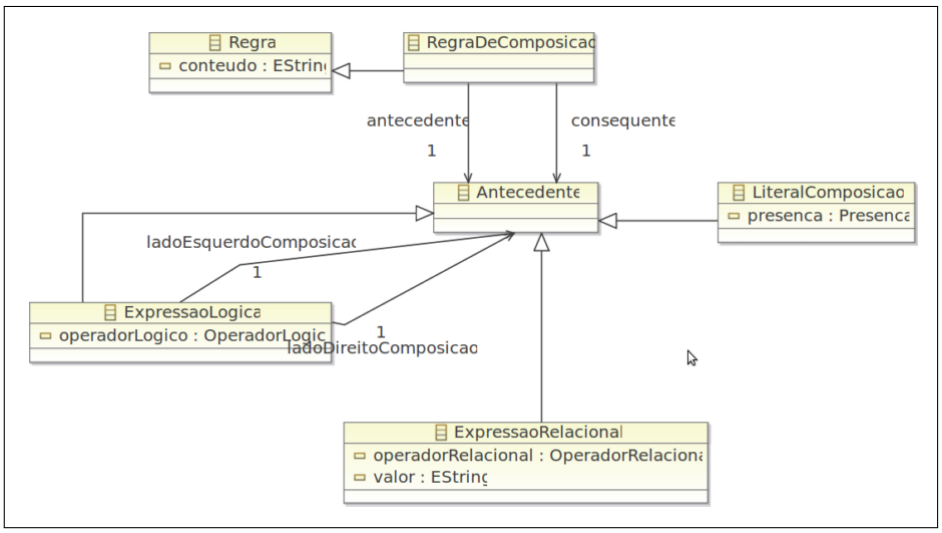
4.3.2 Regras de Composição e Regras de Adaptação

Além dos meta-modelos que representam os diagramas de características e de contexto, existem meta-modelos que representam as regras de composição e adaptação. Como visto na Seção 2.2.4, as regras de composição determinam relações entre características. Já as regras de adaptação determinam as características ativas quando determinado contexto é alcançado. As Figuras 14 e 15 representam os meta-modelos de Regras de Composição e Adaptação, respectivamente. Ambos os modelos são sugeridos no trabalho de (COSTA, 2012) e serviram de base para a construção da ontologia que está ilustrada na Figura 16.

A Figura 16 mostra as regras de composição e adaptação que compõem uma LPSSC, e são representadas pelas classes "Regra de Composição" e "Regra de Contexto", respectivamente. Em (MARINHO *et al.*, 2012), uma regra de composição é uma implicação de uma expressão antecedente para uma expressão consequente, sendo cada expressão uma fórmula proposicional sobre um conjunto de características e atributos contido no MS.

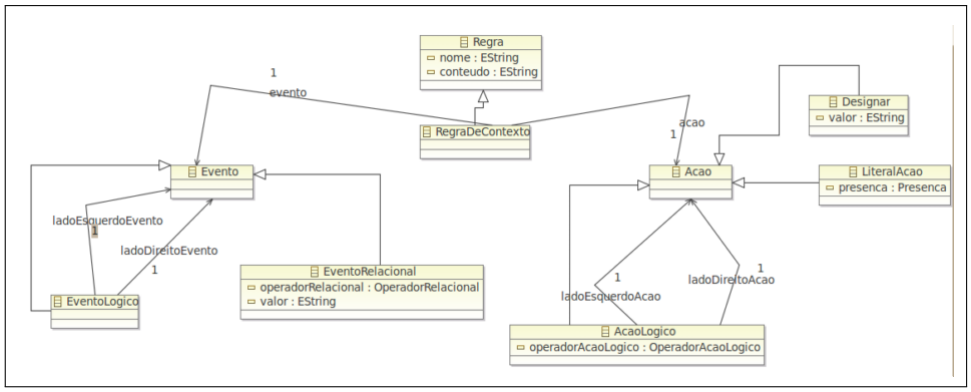
A regra de adaptação é uma implicação de uma expressão de contexto para uma expressão de sistema. Um evento pode ser um estado ou combinação de estados do sistema, e como ação também pode ser um novo estado ou uma combinação de estados no sistema. Por

Figura 14 – Meta-modelo que representa as Regras de Composição em LPSSC



Fonte – (COSTA, 2012)

Figura 15 – Meta-modelo que representa as Regras de Adaptação em LPSSC



Fonte – (COSTA, 2012)

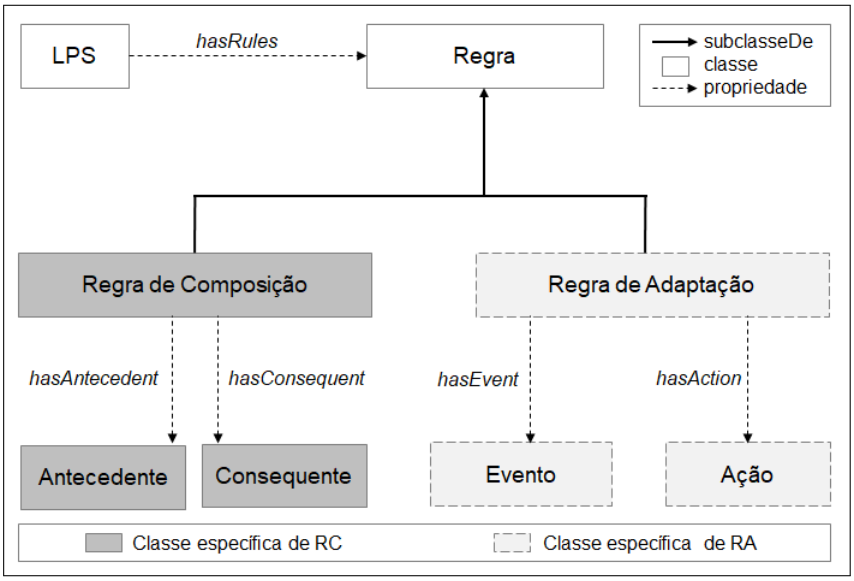
exemplo, se o nível de bateria de um dispositivo móvel adquirir valores inferiores a 10%, como ação, o sistema desabilita a exibição de vídeos.

4.4 Verificação de Modelo

A ontologia em OWL gerada a partir do MMSC é analisada por um *reasoner* na etapa de verificação do modelo. O *reasoner* avalia a corretude e consistência do modelo. Esta verificação é baseada em regras propostas em (MARINHO *et al.*, 2012). Por exemplo, se a ontologia é válida (todas as regras são obedecidas), então ele está pronto para que seja realizada a busca semântica.

Como uma ontologia é o artefato de entrada para a etapa de verificação no método, as regras são implementadas em SWRL (*Semantic Web Rule Language*), que podem ser interpre-

Figura 16 – Parte da ontologia de domínio que representa as Regras de Composição e Adaptação em LPSSC



Fonte – O autor.

tadas por *reasoners*, como por exemplo o Pellet, utilizado no trabalho de (ZAID *et al.*, 2009). Na Figura 17 são mostrados exemplos de regras que foram implementadas.

Uma regra SWRL tem um *body* (antecedente), que na FixOnto representa uma situação inconsistente, e um *head* (consequente) que, na ferramenta, é o indivíduo OWLS que representa a causa da inconsistência. Com a finalidade de identificar as inconsistências, para cada tipo de inconsistência, foram criadas classes OWL correspondentes, bem como propriedades que relacionam indivíduos que desobedecem regras. Assim, quando um indivíduo causa uma inconsistência, o mesmo é assertado como sendo do tipo da inconsistência encontrada. Caso a inconsistência seja na relação entre dois indivíduos, uma relação também será criada.

Como pode ser visto na Figura 17, a regra WFSMR3 indica que um indivíduo tendo ele mesmo como filho caracteriza uma situação excepcional. Caso ocorra essa inconsistência, esse indivíduo será marcado como sendo do tipo WFSMR1. O mesmo entendimento se aplica à outra regra contidas também na Figura 17. Em WFSMR4, se uma característica x for pai de y, e a y for pai de x, caracteriza um ciclo, sendo que x é marcado como sendo da classe OWL de erro WFSMR1, bem como os indivíduos x e y serão associados pela propriedade *hasCicle*. Desse modo, é possível identificar inconsistências na ontologia.

Figura 17 – Exemplo de Regras implementadas em SWRL

```
hasChildFeature(?x,?x) -> WFSMR3(?x)

hasFatherFeature(?x,?y) ^ hasFatherFeature(?y,?x) -> hasCicle(?x,?y) ^ WFSMR4(?x)
```

Fonte – O autor.

4.5 Busca Semântica

A etapa **busca semântica** permite realizar consultas para recuperar informações e rastrear características, contextos e artefatos. A linguagem de consulta deve ser apoiada pelo *reasoner*. Um exemplo de linguagem de consulta é a DL Query fornecida pela ferramenta Protégé (PROTÉGÉ, 2013).

A linguagem DL Query é baseada na Manchester OWL syntax (HORRIDGE *et al.*, 2006), uma sintaxe para o OWL DL que se baseia fundamentalmente na coleta de todas as informações sobre uma determinada classe, propriedade ou indivíduo em uma única construção, chamada *frame*. Essa sintaxe foi construída para que as sentenças criadas fossem com o objetivo de tornar menos complicado o entendimento por uma pessoa. Por exemplo, a expressão 4.1 descreve o conjunto de pessoas que tem pelo menos um filho e que possui filhos que sejam apenas mulheres (ou seja, avós que só têm netas).

$$\begin{gathered} \textit{Pessoa}\ \textbf{AND}\ \textit{temFilho}\ \textbf{SOME} \\ (\textit{Pessoa}\ \textbf{AND}\ (\textit{temFilho}\ \textbf{ONLY}\ \textit{Mulher})\ \textbf{AND}\ (\textit{temFilho}\ \textbf{SOME}\ \textit{Pessoa})) \end{gathered} \tag{4.1}$$

Nesse exemplo, **Pessoa** e **Mulher** são classes OWL e **temFilho** é uma propriedade que associa indivíduos do tipo Pessoa, que pode ser uma Mulher ou Homem. A sentença seleciona indivíduos do tipo Pessoa E que esteja associado através da propriedade *temFilho* a algum indivíduo (Pessoa AND temFilho SOME) que atenda a todas as condições que estão entre parênteses. A sentença entre parênteses indica que o indivíduo seja da classe Pessoa que esteja associado a algum indivíduo Pessoa (temFilho SOME Pessoa) E que esses indivíduos sejam apenas da classe Mulher (temFilho ONLY Mulher).

Desse modo, é possível fazer pesquisas semânticas sobre os modelos de uma LPSSC e extrair informações relevantes tanto nas etapas iniciais do projeto de aplicações sensíveis ao contexto como também ao longo de todo ciclo de vida, ajudando na tomada de decisões durante a evolução dessas aplicações.

4.6 Conclusão

Neste capítulo, foram apresentados detalhes do método proposto para a análise de modelos de características em LPSSC. Uma visão geral foi apresentada e suas principais etapas foram detalhadas neste capítulo. A principal contribuição deste capítulo foi o próprio método, que foi implementado em uma ferramenta que será discutida no Capítulo 5.

5 FIXONTO: A FERRAMENTA

Neste capítulo, a ferramenta FixOnto[1] é detalhada. Suas bases conceituais derivam do método descrito no capítulo 4. Na Seção 5.1, uma visão geral da ferramenta é mostrada. Na Seção 5.2, é descrita a modelagem na ferramenta. Na Seção 5.3, é detalhado o *parser* que transforma os diagramas modelados graficamente para ontologias OWL. A Seção 5.4 apresenta o mecanismo de avaliação da corretude e da consistência da ontologia traduzida. Já a Seção 5.5 aborda a funcionalidade de busca semântica e rastreabilidade de artefatos que estão contidos nos diagramas. Por fim, a Seção 5.6 traz as considerações finais deste capítulo.

5.1 Visão Geral da Ferramenta

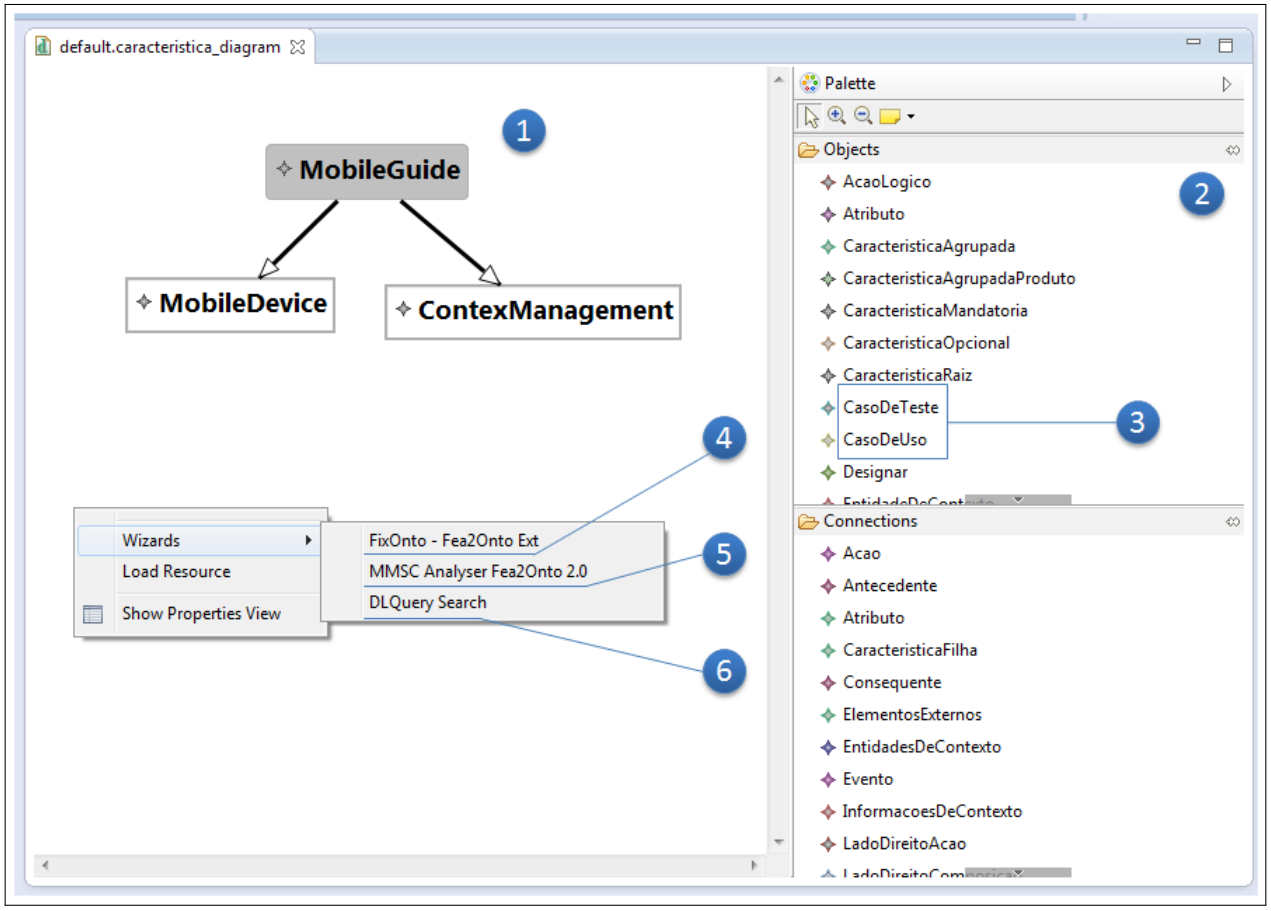
Como discutido no Capítulo 4, o método proposto tem quatro atividades principais: (i) modelagem de características, contextos da aplicação e regras de composição e adaptação e o enriquecimento semântico; (ii) tradução automática do modelo para ontologias; (iii) verificação automática da corretude e consistência do modelo e (iv) o mecanismo de busca semântica.

A Figura 18 apresenta uma tela da FixOnto, onde é possível visualizar a área para modelar graficamente um MMSC, representado por (1), e a paleta com os objetos que podem ser modelados, que é representada por (2). Alguns conceitos adicionais para auxiliar especialistas de domínio no processo de enriquecimento semântico de uma LPSSC são “Caso de Teste” e “Caso de Uso”, estando representados na figura por (3), podendo ser associadas a características e contextos modelados, tornando o MMSC mais rico.

Ainda na Figura 18, o menu com as opções providas pela FixOnto é apresentado. A opção *Fixture - Fea2Onto 2.0* (4) provê a funcionalidade de transformação automática dos modelos, definidos dentro da ferramenta, em OWL. A terceira opção, *MMSC Analyser Fea2Onto 2.0* (5) permite ao usuário verificar se os modelos (MMSC) estão corretos e consistentes, baseados nas regras SWRL que serão discutidas na Seção 5.4. Por fim, a opção *DLQuery Search* (6) provê a funcionalidade de busca semântica, onde é possível pesquisar informações na ontologia utilizando DLQuery.

[1] https://github.com/thalissonoliveira/fixonto

Figura 18 – Tela da FixOnto



Fonte – O autor

5.2 Modelagem e Enriquecimento Semântico

Esta Seção aborda a modelagem de um MMSC na FixOnto e também a implementação dessa funcionalidade. A Subseção 5.2.1, que trata da modelagem, é direcionada para os especialistas de domínio, responsáveis por essa atividade. Já a Subseção 5.2.2 apresenta detalhes de implementação, tecnologias e ferramentas de suporte.

5.2.1 Como modelar e enriquecer semanticamente um MMSC na ferramenta FixOnto

Na FixOnto, um usuário pode modelar graficamente o diagrama de sistema e suas regras de composição bem como do diagrama de contexto, com suas regras de adaptação. É possível também incluir alguns conceitos externos a esses diagramas, especificamente Casos de Uso e Casos de Testes, o que caracteriza o enriquecimento semântico e permite a rastreabilidade.

Voltando à Figura 18, a região representada por (2) contém a paleta com os objetos referentes ao MMSC (e.g., CaracteristicaMandatoria, RaizDeContexto, etc.) e as conexões entre

esses objetos. Por exemplo, um objeto do tipo “CaracteristicaMandatoria” pode ter uma conexão do tipo “CaracteristicaFilha” com uma “CaracteristicaRaiz”. Já a região simbolizada por (1) representa a área de modelagem, para onde são arrastados os objetos e conexões para que seja possível criar o MMSC. Também nessa região representada por (1) é possível acessar os *Wizards* para executar as outras funcionalidades da FixOnto.

O enriquecimento semântico, na versão da FixOnto desenvolvida para este trabalho, é possível por meio da associação entre um objeto “Caso de Teste” ou “Caso de Uso” com características ou contextos através da conexão “ElementosExternos”.

5.2.2 Extensão da FixTure e Implementação do Enriquecimento Semântico

Todo o conceito visual da FixOnto é reutilizado da Fixture (COSTA, 2012) (COSTA *et al.*, 2015). A Fixture é um *plugin* para o ambiente de desenvolvimento Eclipse (ECLIPSE, 2017). Os recursos gráficos presentes na ferramenta, que incluem uma paleta com os conceitos da modelagem e a área para onde são arrastados esses conceitos para criação do modelo, são gerados automaticamente a partir meta-modelos descritos em uma notação semelhante à UML, mas que é específica do *Ecore*, ou *Eclipse Modeling Framework* (*core*) (EMF, 2017), por meio do *Graphical Modeling Framework* (GMF) (GMF, 2017a).

Na Figura 11, que se encontra na Seção 4.3, é possível visualizar um fragmento do meta-modelo relativo aos conceitos do Modelo de Sistemas de uma LPSSC. A representação textual desse modelo encontra-se na Figura 19. Esse modelo é o ponto de partida para a geração automática da ferramenta com GMF.

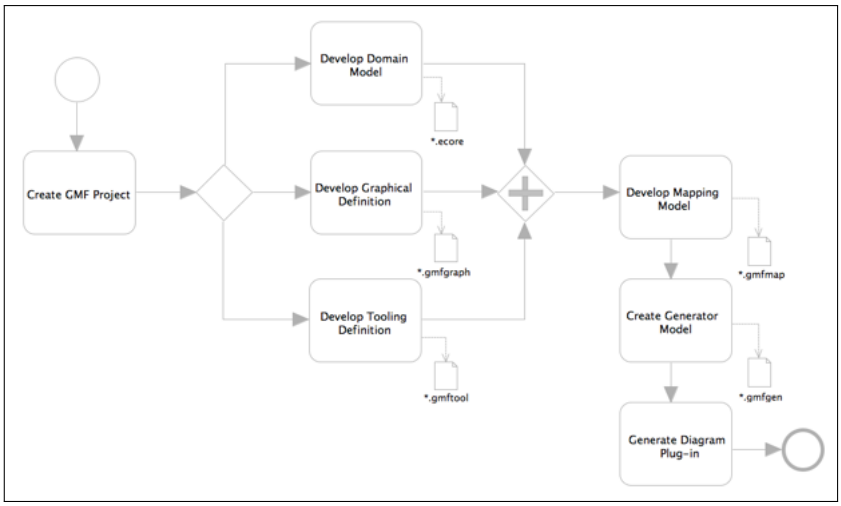
Figura 19 – Meta-modelo do Modelo de Sistema na notação textual (Ecore)

```
49
50 class Caracteristica extends Elemento {
51
52   ref Caracteristica#caracteristicaFilha caracteristicaPai;
53
54   @gmf.link(target.decoration="closedarrow", color="0,0,0", style="solid", width="3")
55   ref Caracteristica[*]#caracteristicaPai caracteristicaFilha;
56
57   @gmf.link(target.decoration="closedarrow", color="0,0,0", style="solid", width="3")
58   ref Variacao[*]#caracteristicaPai variacoes;
59
60   @gmf.link(target.decoration="closedarrow", color="0,0,0", style="solid", width="3")
61   ref Atributo[*]#caracteristicaPai atributo;
62
63 }
64
65 @gmf.node(label="nome")
66 class CaracteristicaRaiz extends Caracteristica {
67   ref LPS[1]#sistema LpsDoSistema;
68 }
69
70 @gmf.node(label="nome", figure="rectangle", border.style="solid", border.width="2")
71 class CaracteristicaMandatoria extends Caracteristica {
72 }
73
74 @gmf.node(label="nome", figure="ellipse", border.style="dash", border.width="2")
75 class CaracteristicaOpcional extends Caracteristica, ElementoCaracteristico {
76 }
77
```

Fonte – O autor.

Essa geração automática é realizada com o auxílio de ferramentas disponíveis no Eclipse Epsilon (EPSILON, 2014). Epsilon é uma família de linguagens e ferramentas para geração de código, transformação modelo-modelo, validação de modelos, comparação, migração e refatoração que funcionam em paralelo ao EMF/GMF e outros tipos de modelos. Um exemplo de ferramenta é a EuGENia, que é um *front-end* para o GMF, que permite aos desenvolvedores gerar um editor totalmente funcional especificando apenas algumas anotações de alto nível no meta-modelo Ecore. Dentre essas linguagens, há a *Epsilon Wizard Language* (EWL), uma linguagem adaptada para transformações nos modelos criado nos editores GMF. Essa geração automática segue o fluxo que é demonstrado na Figura 20.

Figura 20 – Fluxo do *framework* GMF



Fonte – (GMF, 2017b).

O fluxo da Figura 20 ilustra os principais componentes e modelos usados durante o desenvolvimento baseado em GMF. Uma vez definidos os mapeamentos apropriados, o GMF fornece um modelo de gerador para permitir que os detalhes de implementação sejam definidos para a fase de geração. A produção de um editor em formato de *plugin* baseado no modelo do gerador direcionará um modelo final. No tempo de execução é possível preencher os modelos de notação e domínio quando um usuário estiver trabalhando com um diagrama e também prevê a persistência e a sincronização de ambos (GMF, 2017b).

Resumindo, o modelo em sua representação textual, que é um arquivo *.emf, com o auxílio da ferramenta EuGENia, gera os arquivos seguindo o fluxo determinado na Figura 20, sendo que ao final é gerado um *plugin* do Eclipse.

5.3 Transformação do Modelo em Ontologia

Após realizar a modelagem do domínio na FixOnto, o especialista deve acessar a funcionalidade de tradução automática do modelo para ontologias através do *wizard* representado por (4) na Figura 18. Ao invocar essa funcionalidade, a FixOnto percorre o modelo e gera o arquivo SPLiSEM2.owl. Esse processo é guiado por uma ontologia de domínio que contém apenas as classes e as propriedades referentes ao MMSC, como discutido na Seção 4.3. Já a ontologia SPLiSEM2.owl inclui os indivíduos gerados a partir do modelo.

A Tabela 3 apresenta a transformação do Modelo de Sistema e da Regra de Composição para a ontologia em OWL. A primeira coluna mostra a relação na notação disponível na FixOnto e a segunda coluna mostra a expressão em Lógica Descritiva.

Essa transformação é realizada de maneira automática a exemplo da Fea2Onto (FILHO, 2011), onde o diagrama é traduzido em objetos Java, sendo posteriormente realizada a transformação para OWL DL através da OWL API. Para ilustrar essas transformações é utilizada a modelagem contida na Figura 21. A abordagem usada nessa tradução é a de "característica como indivíduo", sendo que cada característica no Modelo de Sistema é traduzido em um indivíduo na ontologia. Essa primeira abordagem é vantajosa por não criar novas classes (conceitos) na ontologia gerada.

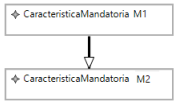
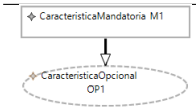
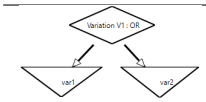
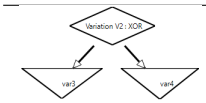
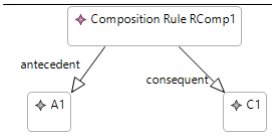
Na Figura 21 tem-se a raiz de contexto *Root* que possui como filhos duas características obrigatórias *ContextManager* e *MobileDevice*. Já a característica *ContextManager* tem o atributo *AttributeOne*. A Caraterística *MobileDevice* possui como filhos a característica GPS e um ponto de variação onde podem ser escolhidas as características *Accelerometer* ou *TemperatureSensor*.

A abordagem utilizada na tradução dos modelos em ontologias implementada na FixOnto foi a de tratar uma característica como um indivídulo OWL. Essa abordagem foi inicialmente escolhida pelo fato da mesma seguir a implementação da Fea2Onto no trabalho de (FILHO, 2011), como é possível verificar na Figura 22.

De acordo com a Figura 22, as características presentes no modelo de características na parte (a) são tratadas como indivíduos OWL que se relacionam às classes OWL, que representam o tipo de características associados (Opcional, Obrigatória, etc.). Já a Figura 23 representa o fragmento da ontologia gerada na tradução feita pela FixOnto a partir do modelo representado na Figura 21.

Na Figura 21, as características *ContextManager* e *MobileDevice* são Indivíduos

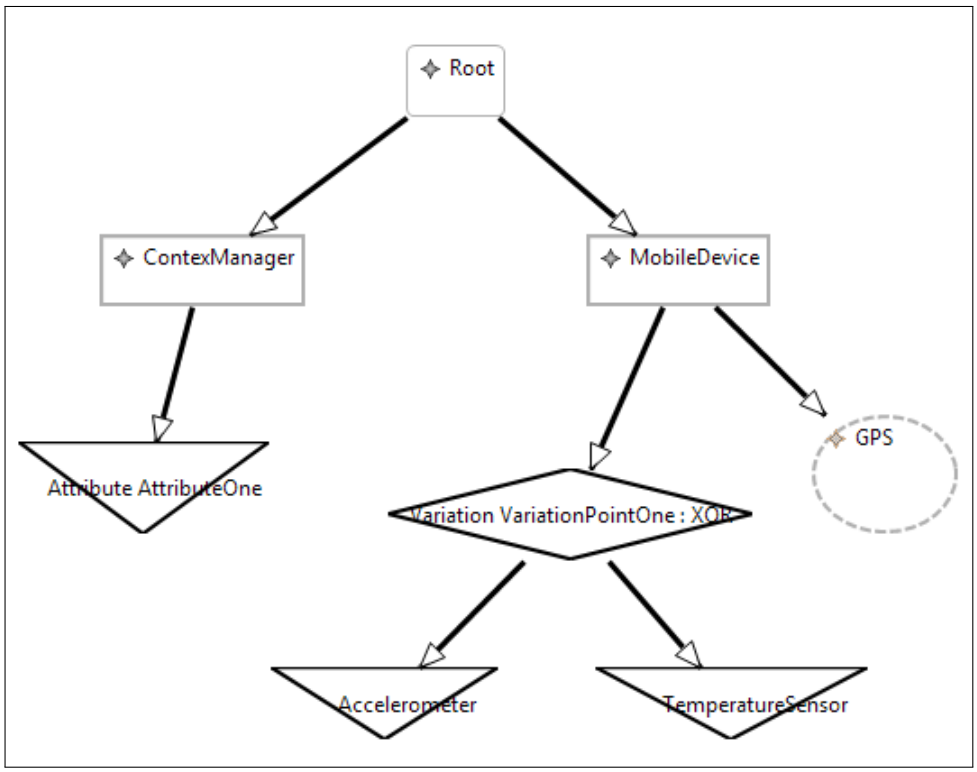
Tabela 3 – Transformação de MMSC para Ontologias (OWL DL)

Relação na FixOnto	Significado	Expressão DL
	*Feature* Obrigatória M2 é filha da *Feature* Obrigatória M1	M1: FeatureObrigatoria M2: FeatureObrigatoria hasFatherFeature(M2, M1)
	*Feature* Opcional OP1 é filha da *Feature* Obrigatória M1	M1: FeatureObrigatoria OP1: FeatureOpcional hasFatherFeature(OP1, M1)
	*Features* Alternativas VAR2 E VAR2 são filhas do ponto de variação V1 e o ponto de variação é do tipo OR	V1: PontoDeVariacao VAR1: FeatureAlternativa VAR2: FeatureAlternativa hasFatherFeature(VAR1, V1) hasFatherFeature(VAR2, V1) hasVariationType(V1, OR)
	*Features* Alternativas VAR2 E VAR2 são filhas do ponto de variação V1 e o ponto de variação é do tipo OR	V2: PontoDeVariacao VAR3: FeatureAlternativa VAR4: FeatureAlternativa hasFatherFeature(VAR3, V2) hasFatherFeature(VAR4, V2) hasVariationType(V2, XOR)
	Regra de Composição R1 tem como antecedente C1 e como consequente C2	R1: RegraDeComposicao C1: Evento C2: Evento hasAntecedent(R1, C1) hasConsequent(R1, C2)

Fonte – Adaptado de (FILHO, 2011; COSTA, 2012).

OWL associados à classe OWL *MandatoryFeature*. Nessa abordagem, as relações entre as características são evidenciadas a partir de *Object Properties* na ontologia. As relações que fazem essas reações são *hasFatherFeature* e *hasChildFeatures* e estão presentes na ontologia de domínio presente na Figura 11.

Figura 21 – Exemplo de Diagrama de Sistema



Fonte – O autor

5.4 Verificação Automática de MMSC

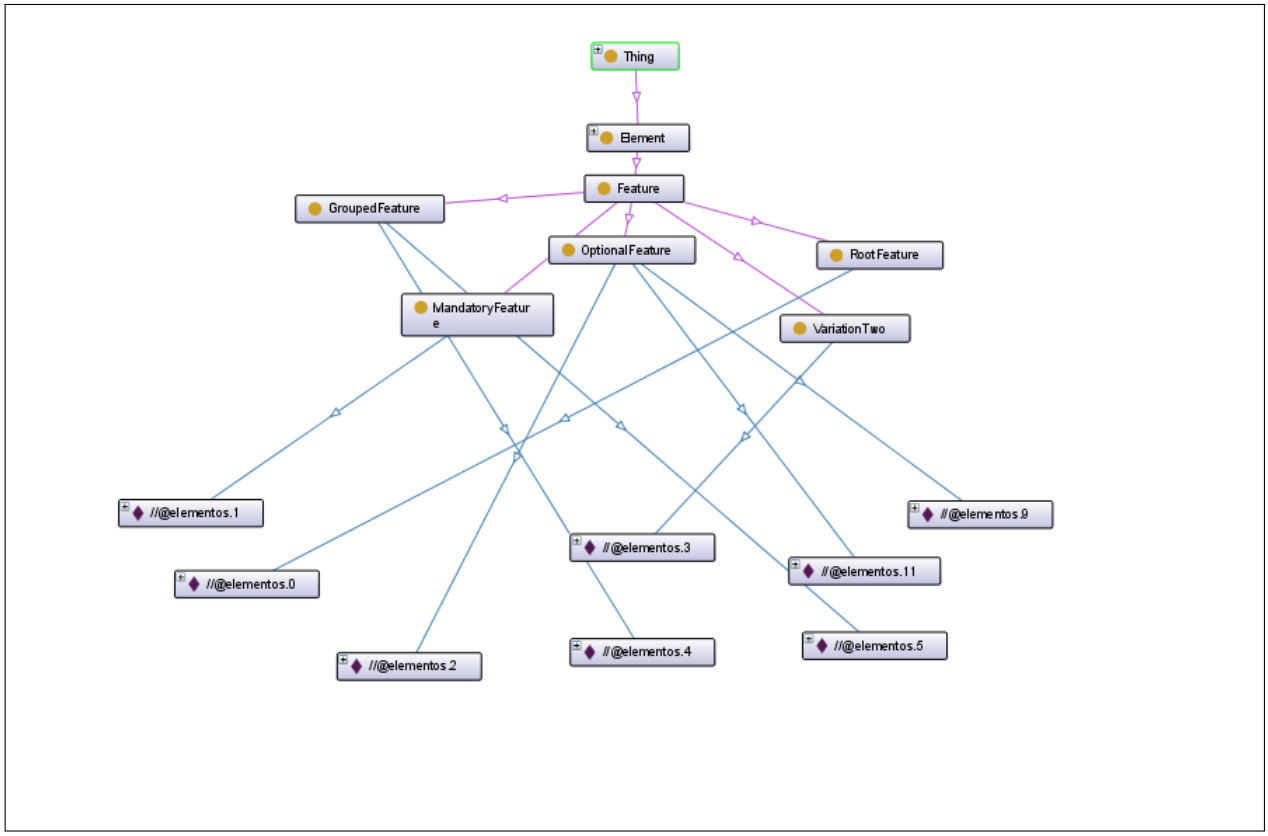
Esta Seção aborda como é realizada a verificação automática de MMSC pela FixOnto. A Subseção 5.4.1 apresenta uma visão geral de como é realizada essa verificação. Já a Subseção 5.4.2 apresenta detalhes de implementação, tecnologias e ferramentas de suporte.

5.4.1 MMSC Analyser

Na FixOnto, para acessar o MMSC *Analyser*, o especialista de domínio deve acessar o *wizard* representado por (5) na Figura 18. Ao clicar nessa funcionalidade, é verificado se a ontologia gerada através do *parser* detalhado na Seção 5.3 obedece um conjunto de regras. Caso nenhuma regra tenha sido desobedecida, é exibida uma mensagem, mostrada na Figura 24. Esta funcionalidade pode ser acessada a qualquer momento durante a modelagem da LPSSC e do enriquecimento semântico ao longo das etapas previstas na Engenharia de Domínio de uma LPS.

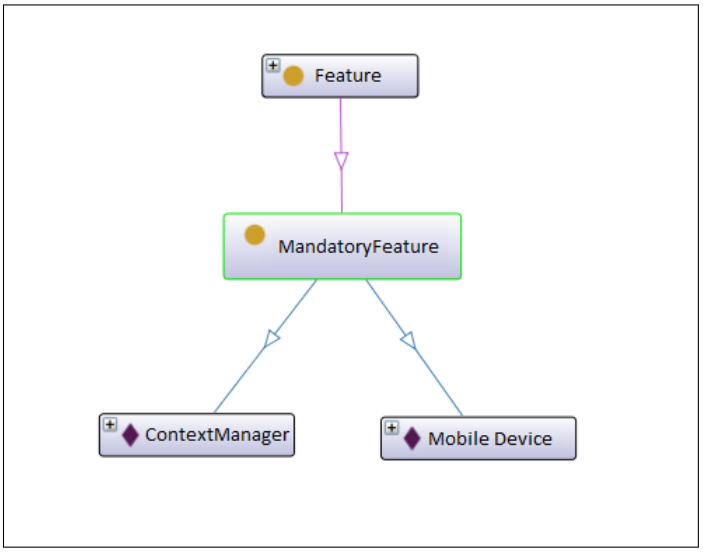
A ferramenta Fixture (COSTA, 2012) (COSTA *et al.*, 2015) implementa o processo PRECISE (MARINHO, 2012), que inclui um conjunto de regras para verificação da corretude e consistência de MMSC. No entanto, para garantir que a ontologia também obedeça a essas regras,

Figura 22 – Feature como Indivíduo OWL



Fonte – O Autor.

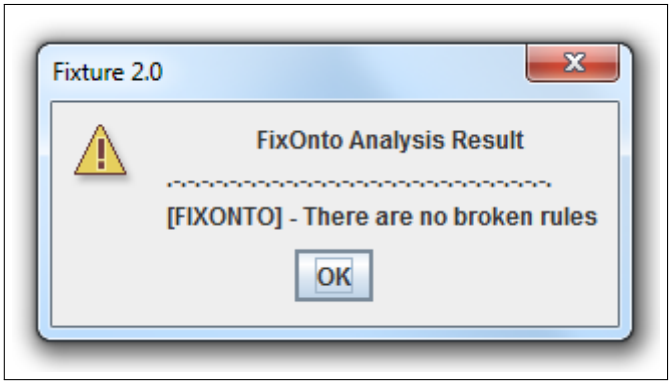
Figura 23 – Feature com Indivíduo OWL



Fonte – O autor.

observou-se a necessidade de implementá-las para que um *reasoner* OWL pudesse verificá-las. Para esse fim, foram implementadas na FixOnto algumas das regras previstas no PRECISE. Como mencionado na Seção 4.4, a linguagem escolhida foi SWRL e o *reasoner Pellet*, cuja

Figura 24 – Resultado da Verificação de Modelo sem erros



Fonte – O Autor.

escolha foi justificada na mesma seção. A Figura 25 mostra o conjunto de regras que foram implementadas.

Figura 25 – Regras implementadas em SWRL.

Código	Regra	Descrição
01	Feature(?x) ^ Feature(?y) ^ hasName(?x,?n) ^ hasName(?y,?m) ^ fix:equalname(?x,?y,?n,?m) -> hasEqualName(?x,?y) ^ WFSMR1(?x)	Verifica se existem Características com mesmo nome
02	Feature(?x) ^ hasAttribute(?x,?m) ^ hasAttribute(?x,?n) ^ hasName(?m,?k) ^ hasName(?n,?l) ^ fix:equalname(?m,?n,?k,?l) -> hasEqualName(?m,?n) ^ WFSMR2(?x)	Verifica se existem Atributos com mesmo nome de alguma Característica
03	hasChildFeature(?x,?x) -> WFSMR3(?x)	Verifica se existe alguma Característica que possui como filha ela mesma
04	hasFatherFeature(?x,?y) ^ hasFatherFeature(?y,?x) -> hasCicle(?x,?y) ^ WFSMR4(?x)	Verifica se existe alguma Característica A que possui como filha uma Característica B, sendo que B possui como filha a característica A
05	MandatoryFeature(?x) ^ hasFatherFeature min 1 Feature(?x) ^ hasFatherFeature max 1 Feature(?x) -> WFSMR5(?x)	Verifica se existe alguma característica com mais de uma característica pai
06	Attribute(?x) ^ hasChildFeature min 1 Feature(?x) -> WFSMR6(?x)	Verifica se existe uma
07	MandatoryFeature(?x) ^ OptionalFeature(?y) ^ hasFatherFeature(?x,?y) -> WFSMR7(?x)	Verifica se uma característica Obrigatória tem como pai uma Característica Opcional
08	MandatoryFeature(?x) ^ GroupedFeature(?y) ^ hasFatherFeature(?x,?y) -> WFSMR7(?x)	Verifica se uma característica Obrigatória tem como pai uma Variante
09	OptionalFeature(?x) ^ MandatoryFeature(?y) ^ hasChildFeature(?x,?y) -> WFSMR8(?x)	Verifica se uma característica Opcional tem como filha uma Característica Obrigatória
10	CompositionLiteral(?x) ^ MandatoryFeature(?y) ^ hasFeaturedElement(?x,?y) -> WFCR1(?x)	Verifica se existe alguma Composição Literal associada a uma Característica Obrigatória
11	ActionLiteral(?x) ^ MandatoryFeature(?y) ^ hasFeaturedElement(?x,?y) -> WFAR1(?x)	Verifica se existe alguma Ação Literal associada a uma Característica Obrigatória

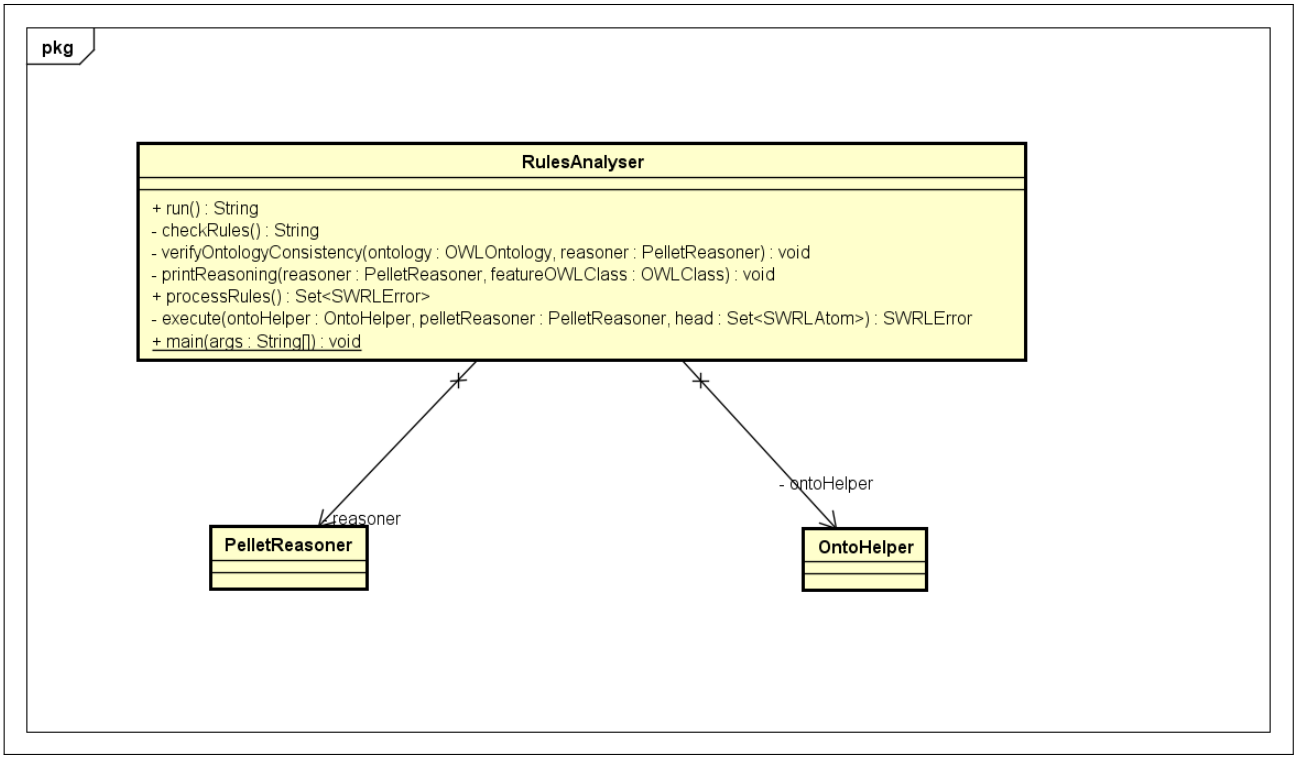
Fonte – O autor

Considerando que algumas regras consideram contexto, como a regra número 11, este trabalho é direcionado para as LPSSC, além de considerar regras direcionas para LPSs tradicionais.

5.4.2 Implementação do MMSC Analyser

Para encapsular a análise das regras dentro da FixOnto foi implementada a classe Java *RulesAnalyser*. A Figura 26 apresenta um *screenshot* da tela, na qual são apresentadas as assinaturas dos métodos e as classes que a compõem. A classe Java *OntoHelper* encapsula as principais operações de criação e manipulação de ontologias que são realizadas pela OWL API. Já a classe *PelletReasoner* contém a implementação que remete ao próprio reasoner, encapsulado no arquivo *pellet-owlapiv3.jar*.

Figura 26 – Diagrama de classes simplificado do MMSC Analyser



Fonte – O autor

Toda a verificação se inicia no método *RulesAnalyser.run()*. Nesse método, a ontologia é recuperada, bem como são incorporadas na ontologia as regras SWRL contidas em um arquivo chamado *rules.fix*. O método *RulesAnalyzer.run()* invoca *RulesAnalyser.checkRules()*, o qual itera sobre todas os axiomas do tipo *Rule* contidos na ontologia. Já o método checkRules(), por sua vez, invoca o método *processRules()*. Por fim, um dos principais métodos é invocado e executado - o método *execute(OntoHelper ontoHelper, PelletReasoner pelletReasoner, Set<SWRLAtom> head)*, onde o conjunto de objetos *SWRLAtom* representa o *head* da regra SWRL. O retorno desse método é um SWRLError, caso a regra seja desobedecida.

Portanto, caso haja um erro, tem-se detalhes do mesmo, como a regra e o indivíduo específico que não atendeu à regra. Ao final dessa etapa, são mostrados todos os erros para o usuário. Caso não haja erros, o usuário está apto a fazer buscas semânticas, utilizando DLQuery (PROTÉGÉ, 2013).

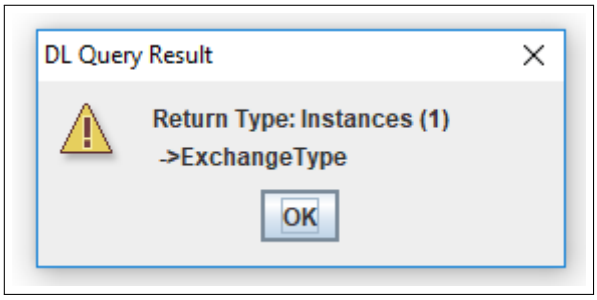
5.5 Busca Semântica e Rastreabilidade

No desenvolvimento de aplicações, sejam sensíveis ao contexto ou não, é de grande importância a comunicação entre os *stakeholders* envolvidos no processo. Um exemplo disso é a interação entre as equipes de desenvolvimento e a equipe de testes de software. No caso de LPSSC, é importante que cada característica tenha um conjunto de testes, e o mesmo teste possa se relacionar com várias características. Quando é detectada uma falha em determinada característica, é importante rastrear informações de quais testes devem ser executados imediatamente após a correção.

Além dessa comunicação, é importante também ter a possibilidade de fazer consultas sobre o modelo da aplicação sendo desenvolvida. Por exemplo, saber se determinado módulo se relaciona com outro ou saber quais regras de adaptação estão relacionadas a determinada característica pode ajudar os especialistas de domínio na tomada de decisões. Considerando essas duas necessidades, dentre outras vantagens que são obtidas através da extração de informações relevantes de um MMSC, foi implementada na FixOnto a funcionalidade de busca semântica, através da consulta utilizando DLQuery. A OWL API, juntamente com o *Pellet*, permite que sejam executadas consultas sobre a ontologia.

Na Figura 27, é apresentado o resultado da DLQuery *hasFatherFeature some MandatoryFeature*, sendo mostrado que “ExchangeType” tem como característica pai uma característica obrigatória.

Figura 27 – Tela com o resultado da DLQuery Search



Fonte – O autor

5.6 Conclusão

Neste capítulo foram discutidas as funcionalidades presentes na FixOnto. A ferramenta é direcionada para os projetistas de sistemas móveis e sensíveis ao contexto. No que concerne à LPS, a FixOnto é utilizada na etapa de engenharia de domínio. Para avaliar a ferramenta foi realizadas uma Prova de Conceito, onde parte de uma LPSSC foi modelada. No Capítulo 6 são apresentados detalhes dessa avaliação.

6 AVALIAÇÃO

Este capítulo descreve uma Prova de Conceito realizada para avaliar o método e a ferramenta propostos e discute os resultados alcançados na avaliação. A Prova de Conceito foi conduzida utilizando uma LPSSC, a Mobiline (MARINHO *et al.*, 2013), detalhada na Seção 6.1. Na Seção 6.2, é demonstrada a modelagem da Mobiline na FixOnto e a execução de cada etapa do método proposto. Logo após, na Seção 6.3, são discutidos os resultados da Prova de Conceito e na Seção 6.4 são apresentadas as ameaças à validade. Por fim, na Seção 6.5, são apresentadas as considerações finais deste capítulo.

6.1 Mobiline

Como discutido na Seção 2, a Mobiline é uma LPSSC para o domínio de aplicações móveis e sensíveis ao contexto. Em (MOBILINE, 2010) estão disponíveis os modelos e outros artefatos. Para este trabalho foi utilizado parte do modelo do Mobiline, especificamente a parte do modelo que compõe o GREat Tour, que é um dos produtos derivados da linha. O modelo utilizado está representado pelas Figuras 28 e 29. A Figura 28 apresenta o Modelo de Características na notação Ubifex (FERNANDES; WERNER, 2008). A Troca de Mensagens é um exemplo de característica obrigatória que deve ser executada de modo assíncrono de duas maneiras: através de tuplas ou alguma abordagem orientada a eventos.

Já a Figura 29 representa o Modelo de Contexto do Mobiline que está presente no GREat Tour. Estão previstos os contextos de dispositivo móvel (*Mobile Device*), cujas informações contextuais observadas são a Memória, bibliotecas disponíveis, a tela, bateria e Localização Indoor.

Considerando as regras, não foram determinadas regras de composição especificamente para o produto GREat Tour. Entretanto, foram delimitadas regras de adaptação, que consideram o contexto. As definições de contexto contidas na documentação da Mobiline são listadas a seguir.

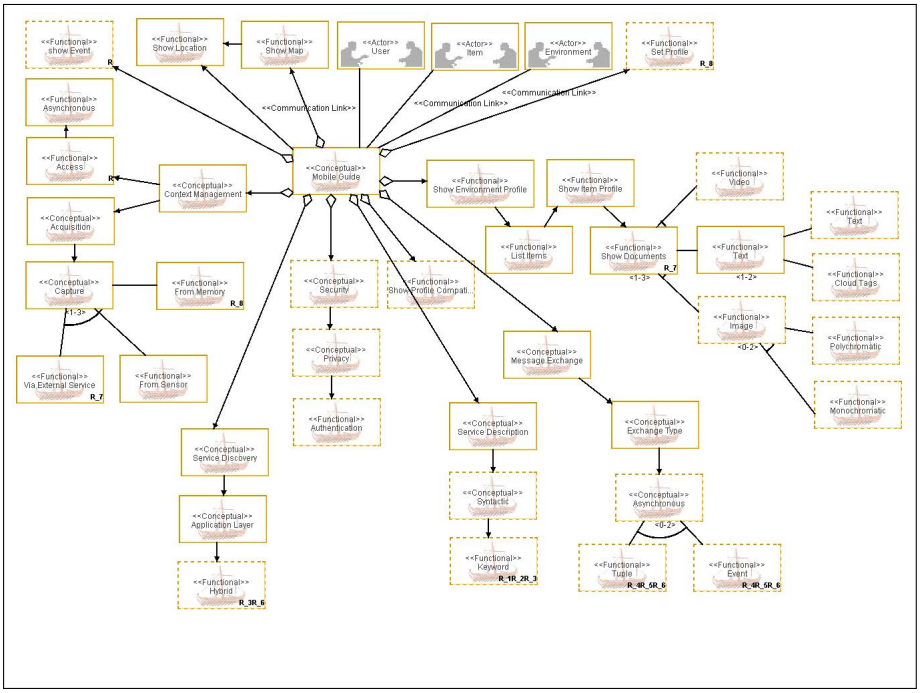
- Nome: Same Environment

  Expressão: $User.LocationIndoor = Environment\_Item$
- Nome: Similar Profile

  Expressão: $User.Profile >= 0,6 * Item\_Profile$

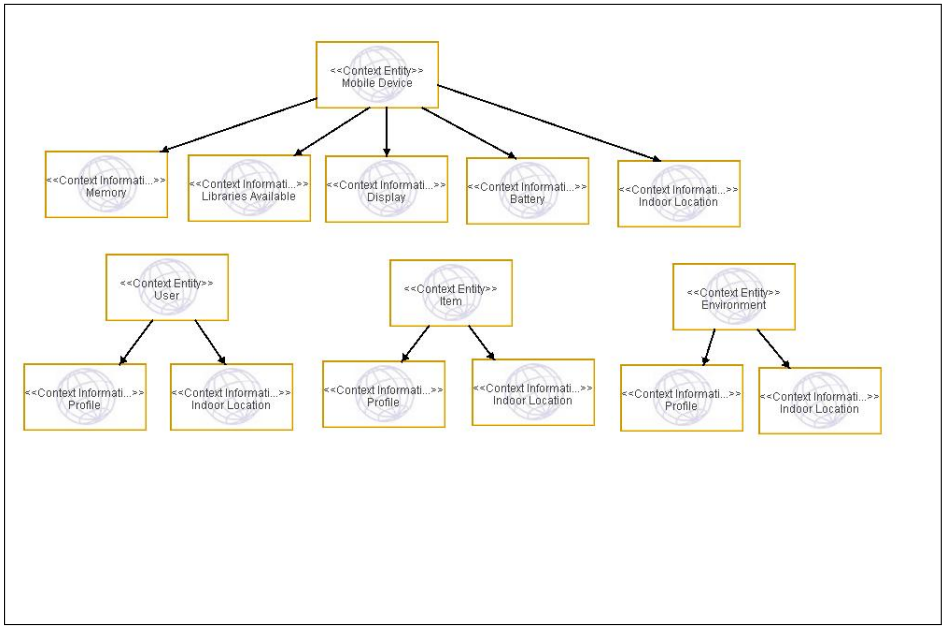
A expressão do contexto denominado *Same Environment* (Mesmo Ambiente) indica

Figura 28 – Modelo de Características do Mobiline



Fonte – Adaptado de (MOBILINE, 2010)

Figura 29 – Modelo de Contexto do GREat Tour



Fonte – Adaptado de (MOBILINE, 2010)

que a localização do usuário (determinada pelo dispositivo móvel) confere com a localização conhecida nos itens do ambiente. Já a expressão do *Similar Profile* (Perfil Parecido) afirma que esse contexto é atingido se o perfil do usuário coincidir pelo menos 60% com os perfis conhecidos pelo GREat Tour. Dadas as definições de contexto, são apresentadas as regras de

adaptação. A expressão 6.1 contém as regras definidas para o GREat Tour.

$$\begin{gathered} SameEnvironmentANDSimilarProfileimplies(ListItems) \\ SameEnvironmentANDSimilarProfileimplies(ShowProfileCompatibility) \end{gathered} \tag{6.1}$$

A regra de adaptação apresentada indica que, caso os contextos *Same Environment* e *Similar Profile* sejam alcançados ao mesmo tempo, as características *List Items* (Listar Itens) e Show Profile Compatibility (Mostrar Compatibilidade de Perfil) devem ser ativadas.

Para atender à etapa de enriquecimento semântico do método proposto, alguns artefatos foram sugeridos e foram associados aos modelos. A Tabela 4 apresenta uma lista contendo casos de uso e casos de teste para essa associação.

Tabela 4 – Casos de Uso e Casos de Teste para o GREat Tour

Tipo	Código	Característica Associada
Caso de Uso	UC1	Característica Context Management
	UC2	Característica Show Location
	UC3	Características Show Map e Show Location
Caso de Teste	TC1	Característica Context Management
	TC2	Característica Show Location
	TC3	Característica Show Map

Fonte – O autor.

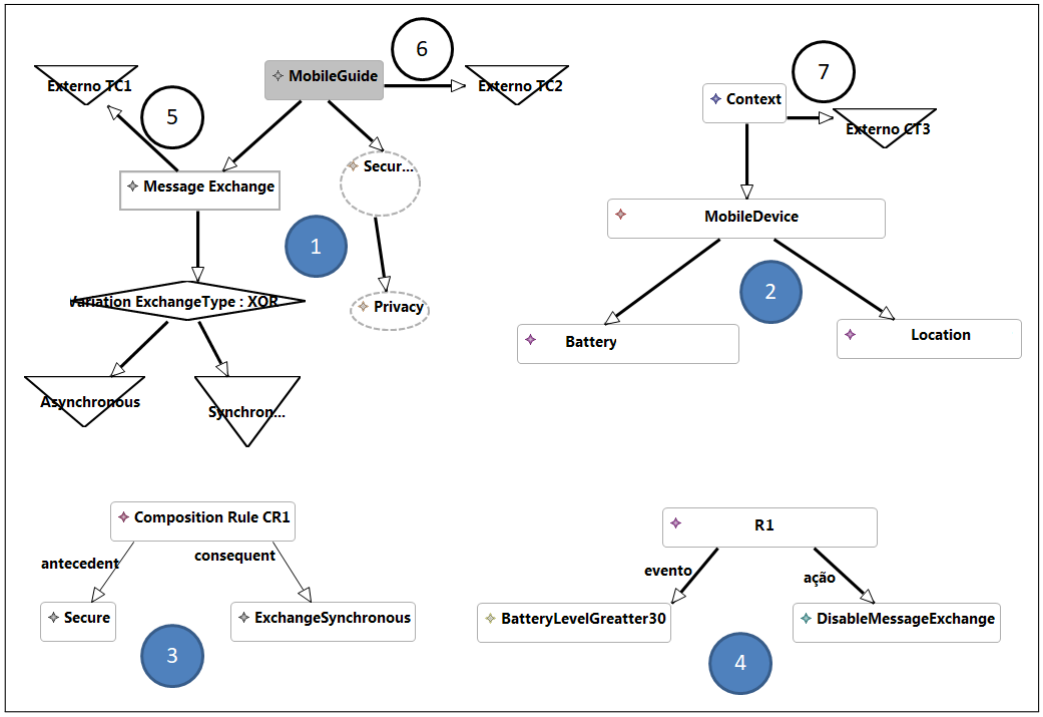
O caso de uso UC1 está relacionado ao módulo que controla a mudança de localização e dos níveis de bateria do dispositivo. O caso de uso UC2 se relaciona ao módulo que mostra a descrição da localização do dispositivo e o UC3 descreve o módulo que mostra um mapa com a localização do dispositivo em destaque. Os casos de testes TC1, TC2 e TC3 representam testes para os casos de uso UC1, UC2 e UC3, respectivamente.

6.2 Modelagem da Mobiline na FixOnto

Nessa etapa da Prova de Conceito, os modelos representados pelas Figuras 28 e 29 foram transcritos na FixOnto. Além disso, foram associados casos de uso e casos de teste ao modelo. A Figura 30 mostra o GREat Tour modelado na FixOnto.

Na Figura 30 estão representados o modelos de características (1), os modelo de contexto (2), uma regra de composição (3) e regra de adaptação (4). Foi realizado também o

Figura 30 – Modelagem do GREat Tour na FixOnto



Fonte – O Autor.

enriquecimento semântico, no qual foram associados casos de teste (5, 6) e casos de uso (7) aos modelos.

Após a fase de modelagem, foi executado o *wizard* que transformou o MMSC em uma ontologia descrita em OWL-DL. A ontologia gerada foi escrita em um arquivo chamado *fixonto.owl*, que pode ser visto no Apêndice A. Em seguida, foi verificada a corretude e consistência do modelo através das regras de boa formação descritas na Seção 4.4. O MMSC modelado não apresentou erro sendo, portanto, necessário inserir inconsistências no modelo para verificar se a detecção de erros funcionava.

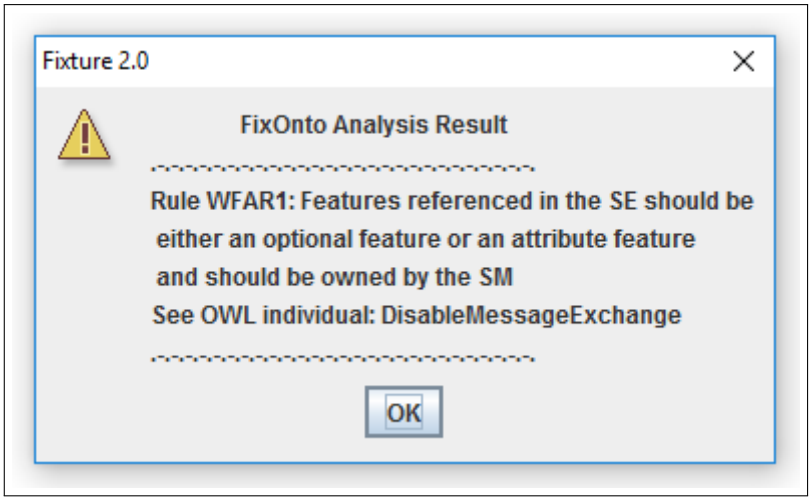
6.2.1 Verificação de Erros e Inserção de Faltas

Para inserir um erro, foi alterado a parte *consequent* da regra de contexto. Foi removido o *consequent* ExchangeType.Synchronous e, depois, foi incluída uma característica obrigatória (ou seja, MessageExchange). O axioma gerado após a alteração foi *hasFeatureElement*(DisableMessage, ExchangeType.Synchronous). A Figura 31 mostra o resultado da ferramenta, portanto, mostra a regra desobedecida.

Outra regra implementada é representada pela expressão $hasFatherFeature\,(?X,?Y) \wedge hasFatherFeature\,(?Y,?X) \rightarrow hasCycle\,(?X,?Y) \wedge WFSMR4\,(?X)$. Se uma característica x

é pai de y e y é pai de x, é dito um ciclo, e x é destacado como um indivíduo do erro WFSMR4 e os indivíduos x e y são associados pela propriedade *hasCycle*. Neste caso, o axioma original foi hasFatherFeature(Privacidade, Segurança). Foi mudado o axioma para $hasFatherFeature(Seguranca, Seguranca)$, e o erro foi lançado.

Figura 31 – Resultado da Verificação de Modelo após inserção de Faltas



Fonte – O Autor.

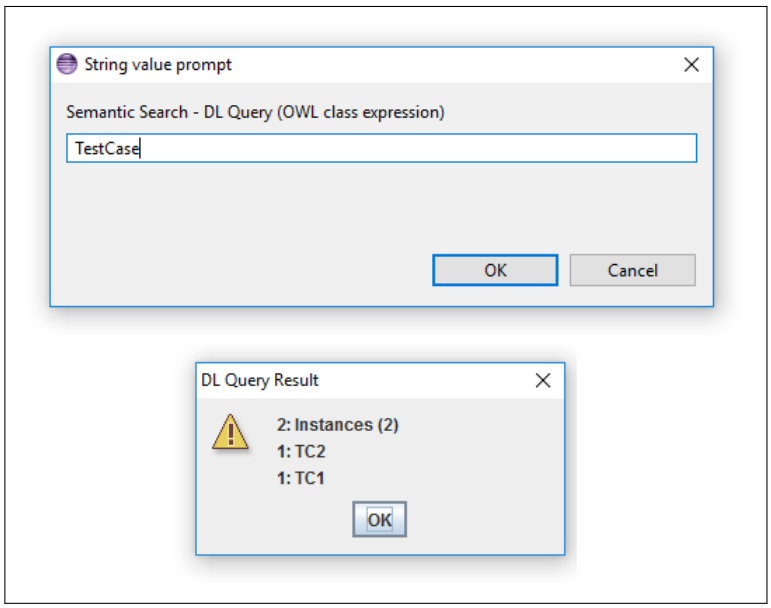
6.2.2 Busca Semântica

Por fim, foi realizada uma busca semântica na ontologia (fixonto.owl), usando DL Query. Foi selecionada a opção DLQuery Search na FixOnto. Para testar a busca dentro da ferramenta, foram selecionados os casos de teste existentes no modelo. As consultas e o resultados podem ser visualizados nas Figuras 32 e 33.

Nessa primeira *query*, é realizada a busca por todos os casos de uso que foram modelados e que, portanto, estão associados a algum componente do modelo de características ou modelo de contexto. Esse tipo de informação pode ser importante para os desenvolvedores verificarem o escopo de uma evolução de software, tendo uma noção de quantos testes foram realizados.

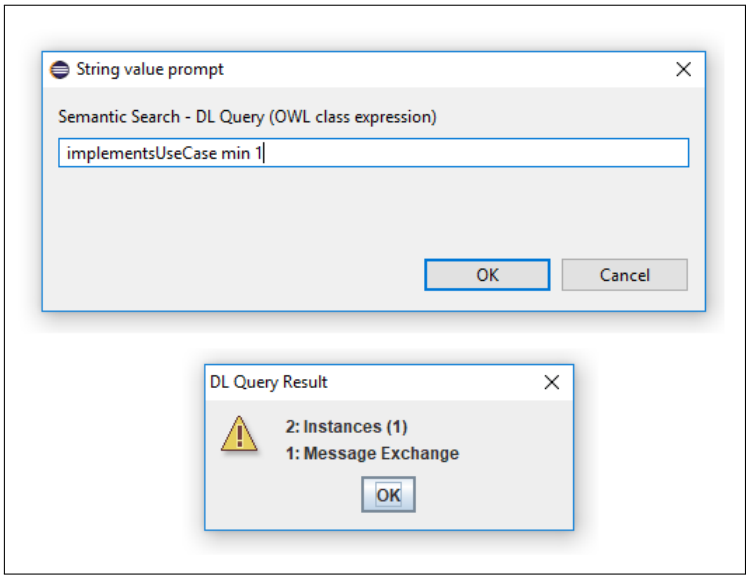
Na segunda *query* é realizada a busca por todos os componentes do modelo que implementam algum caso de uso. A informação fornecida por essa consulta dão uma visão de documentação do modelo, permitindo a avaliação da necessidade ou não de investir em pessoal para gerar novos artefatos de documentação.

Figura 32 – Busca Semântica na FixOnto - Casos de Teste



Fonte – O Autor.

Figura 33 – Busca Semântica na FixOnto - Casos de Uso



Fonte – O Autor.

6.3 Resultados e discussão

O objetivo do método e da ferramenta é permitir que os especialistas de domínio possam modelar e recuperar informações no desenvolvimento de uma LPSSC. A solução proposta inclui também a verificação do modelo, ajudando a evitar erros de modelagem e reduzir custos. A ferramenta em (COSTA *et al.*, 2015), a abordagem de (MARINHO *et al.*, 2012) e abordagem de enriquecimento semântico (FILHO *et al.*, 2012) foram o ponto de partida desta proposta. A

seguir as questões de pesquisa são respondidas.

> **QP1:** *Como enriquecer semanticamente uma LPSSC e garantir que a modelagem está correta?*

Resposta: O método FixOnto tem como primeira etapa a modelagem de um MMSC, que representa uma LPSSC, e o enriquecimento semântico desse modelo. Neste trabalho, o enriquecimento semântico se refere à associação de características e contextos a outros artefatos do domínio (e.g.: casos de uso, código fonte, documentos de requisitos). Essa associação permite extrair informações que podem auxiliar os *stakeholders* na tomada de decisões estratégias ao longo do ciclo de modelagem e desenvolvimento da linha. A segunda etapa do método verifica se modelo criado está bem formado e consistente, podendo, assim, gerar produtos válidos. Essa verificação é realizada sobre um conjunto de regras que garantem essa boa formação e constência. No método FixOnto, a modelagem pode ser realizada em qualquer linguagem, como por exemplo, eFM ou CFM. Já a verificação de modelos exige que os mesmos estejam representados como ontologias especificadas em OWL DL, e as regras estejam descritas em SWRL. Desse modo, há a necessidade de uma etapa intermediária de transformação do MMSC em ontologia na linguagem OWL DL.

Portanto, é possível enriquecer semanticamente uma LPSSC através da associação entre o modelo e outros do domínio. A garantia de que a modelagem está correta acontece através da verificação de um conjuntos de regras que garantem boa formação e consistência.

> QP2: *Como realizar a recuperação e o rastreamento de informações em LPSSC de modo a auxiliar os stakeholders na tomada de decisão?*

Resposta: Quanto mais completa a informação, mais ela pode auxiliar *stakeholders* na tomada de decisão. O método FixOnto tem como última etapa a pesquisa no modelo. Nessa etapa é possível rastrear, por exemplo, os casos de teste modelados. Isso é possível através de buscas semânticas em cima do modelo representado em OWL DL. Utilizando linguagens como *Dl Query* ou SPARQL, é possível realizar buscas semânticas, que possuam um significado relevante para a tomada de decisão. Um exemplo é saber quais contextos serão afetados caso determinado caso de uso seja modificado ou se alguma característica for removida. Desse modo, a busca semântica sobre um MCSC permite a obtenção de informações relevantes para a tomada de decisão ao longo do ciclo de desenvolvimento de uma LPSSC.

Esta pesquisa gerou resultados que destacam os benefícios de realizar uma busca semântica sobre uma ontologia que representa um MMSC e que seja expressiva e válida. A recuperação de informações e a rastreabilidade permitem ao especialista de domínio e outros *stakeholders* tomarem decisões importantes ao longo do desenvolvimento de aplicações sensíveis ao contexto.

6.4 Ameaças à Validade

Após conduzir a avaliação do método FixOnto com o auxílio da ferramenta, algumas ameaças à validade foram detectadas. De acordo com (WOHLIN *et al.*, 2012), foram identificados dois tipos de ameaças à validade: Conclusão e Interna.

A Validade de Conclusão diz respeito à análise estatística dos resultados e à composição do objeto de estudo. Uma ameaça para a validade da conclusão é, no entanto, o baixo número de amostras, o que pode reduzir a capacidade de revelar padrões nos dados. Portanto, como somente um especialista utilizou a ferramenta FixOnto para verificar sua funcionalidade, isso implicando em uma ameaça à validade de Conclusão. Esse número foi suficiente para observar os benefícios do método e da ferramenta, porém, a realização de um experimento controlado com um maior número de especialistas em contexto pode ajudar a generalizar os resultados alcançados.

A Validade Interna diz respeito à questões que podem afetar a variável independente em relação à causalidade, sem o conhecimento dos pesquisadores. Como foi demonstrada a ferramenta usando uma única LPSSC acadêmica, esta pode não apresentar variáveis que possam afetar a solução proposta, tornando os resultados restritos ao seu escopo. A modelagem deve ser realizada com mais LPSSCs, de diferentes domínios, para generalizar os resultados.

6.5 Conclusão

Este capítulo detalhou a Prova de Conceito para demonstrar o método proposto neste trabalho e discutiu os resultados alcançados. Primeiramente foi apresentada a LPS Sensível ao Contexto (Mobiline) utilizada na Prova de Conceito. Depois foram demonstradas as etapas de modelagem da linha na FixOnto e a tradução automática em ontologia. Depois foi detalhada a inserção de erros na ontologia para demonstrar a verificação de modelos e também a busca semântica. Por fim, os resultados foram discutidos e as ameaças à validade apresentadas.

7 CONCLUSÕES E TRABALHOS FUTUROS

Neste capítulo são abordados os resultados alcançados, a produção bibliográfica do período de mestrado, e são evidenciadas as principais limitações deste trabalho. Além disso, os trabalhos futuros desta pesquisa são apresentados.

7.1 Resultados alcançados

A recuperação da informação e a rastreabilidade ajudam os *stakeholders* na tomada de decisão durante o desenvolvimento de aplicações sensíveis ao contexto. A motivação desta pesquisa foi a importância da recuperação de informações de domínio em modelos de características móveis sensíveis ao contexto (MMSC) e a importância da verificação de consistência de uma LPSSC durante as etapas iniciais do projeto. A detecção em tempo de design de inconsistências de modelagem permite economia de recursos durante o ciclo de desenvolvimento de software (MARINHO, 2012).

Na literatura é possível encontrar soluções que verificam LPSs tradicionais (ZAID *et al.*, 2009) (GUO *et al.*, 2012) (RINCóN *et al.*, 2014) e LPSSC (COSTA, 2012) (MARINHO, 2012). No entanto, usando essas soluções existentes, as informações em um modelo de características (as próprias características, os contextos e as regras) ainda podem ser insuficientes. Para preencher essa lacuna, foi proposto um método baseado em ontologias e uma ferramenta para adicionar semântica na LPSSC e para verificar automaticamente sua corretude e consistência.

O método proposto possui quatro etapas. Na primeira etapa, o especialista de domínio realiza a modelagem da aplicação sensível ao contexto. Essa modelagem envolve a criação do modelo de características, modelo de contexto e regras que relacionam esses modelos. Na segunda etapa, é executada a tradução automática do modelo criado para ontologias, para que seja possível executar a terceira etapa. Já a terceira etapa é a verificação automática dos modelos já traduzidos. Por fim, o método prevê a busca semântica sobre o modelo já verificado.

Para a construção do método e da ferramenta, a FixOnto, foram combinadas soluções existentes na literatura. Para o enriquecimento semântico foi utilizada a abordagem proposta por Filho (2011). Para a verificação automática de MMSC foi utilizado o *framework* sugerido por Zaid et al. (2009), no qual foram implementadas as regras propostas pelo processo PRECISE que pode ser encontrado no trabalho de Marinho (2012). A FixOnto é uma extensão da Fixture proposta em (COSTA *et al.*, 2015).

Foi demonstrado o enriquecimento semântico e a verificação automática do MMSC com a FixOnto. Também foi demonstrada a rastreabilidade, que provê informações importantes para os especialistas de domínio, através de pesquisas semânticas sobre o modelo com DL Query. É relevante mencionar que o uso de ontologias, especificadas em OWL DL, permitiu a implementação das funcionalidades mencionadas anteriormente (rastreabilidade, busca semântica e verificação de modelos), sem precisar integrar diferentes técnicas ou soluções para provê-las.

Outra parte importante deste trabalho foi a definição de uma ontologia de domínio com os conceitos envolvidos em LPSSC. Mesmo havendo na literatura algumas ontologias que já continham a maior parte dos conceitos necessários neste trabalho, entretanto, havia muitos conceitos específicos e que não eram relacionados a este trabalho. Por esse motivo, uma nova ontologia foi criada, com conceitos relacionados aos meta-modelos da solução que foi estendida por este trabalho. Desse modo, não houve a necessidade de adaptar a solução estendida às ontologias existentes. Essa ontologia de domínio é o ponto de partida para a tradução do modelo na FixOnto.

foi criada pois foram baseadas produção bibliográfica realizada durante o mestrado. A seguir são relacionadas as publicações:

- Simpósio Brasileiro de Sistemas de Informação 2012: **"Uma Avaliação de Ferramentas para Testes em Sistemas de Informação Móveis baseada no Método DMADV"** Santos, I.S.,Bezerra, C.I.M., Monteiro, G.S., Araújo, I.L.,Oliveira, T.A.,Santos, R.M., Dantas, V.L.L e Andrade, R.M.C.
- Brazilian Symposium on Multimedia and the Web (WebMedia), Workshop on Tools and Applications (WFA) 2013: **"Uma GREat Tour: Um Guia de Visitas Móvel e Sensível ao Context"** Lima, E.R.R., Araújo, I.L., Santos, I.S., Oliveira, T.A., Monteiro, G.S., Costa, C.E.B., Santos, Z. e Andrade, R.M.C.
- SugarLoafPlop 2014: **"Partial View: Um padrão para reutilização de views na arquitetura MVC."** Viana, J.R.M., Oliveira, T.A., Trinta, F.A.M., Andrade, R.M.C.
- ICEIS 2015: **"Fixture: a tool for automatic inconsistencies detection in Context-Aware SPL"** Costa, P.A.S., Marinho F.G., Andrade, R.M.C., Oliveira, T.A.
- ICEIS 2017: **"Semantic Enrichment and Verification of Feature Models in DSPL"** Oliveira, T. A., Andrade, R.M.C., Viana, W.

Esta última publicação refere-se à solução proposta neste trabalho.

7.2 Limitações

O método e a ferramenta propostos neste trabalho possuem limitações devido às soluções que foram usadas. As ontologias, que possuem vantagens como a possibilidade de inferência de informações e permitir buscas semânticas, têm como desvantagem o tempo que demora para realizar essas operações. Em um modelo muito grande essas operações podem se tornar inviáveis.

Outra limitação é a linguagem utilizada para realizar buscas semânticas. A DL Query não é muito intuitiva e exige conhecimentos sobre ontologias. Essa linguagem pode se tornar uma barreira para profissionais da área de Engenharia de Software utilizarem a FixOnto.

Além dessas limitações para o usuário da ferramenta, há algumas limitações para os desenvolvedores de ferramentas que implementem o método. A linguagem escolhida para implementar as regras foi a SWRL, que só permite a criação de sentenças lógicas com o operador $\wedge$ (“e” lógico) e não permite a negação de sentenças. Portanto, algumas regras podem ser inviáveis de serem implementadas.

7.3 Trabalhos Futuros

Como foi discutido ao longo desta dissertação, a solução proposta neste trabalho envolve modelagem e enriquecimento semântico, verificação de modelo e busca semântica no domínio de aplicações sensíveis ao contexto utilizando Linhas de Produtos de Software.

No entanto, lacunas que não foram abordadas neste trabalho, ainda precisam ser preenchidas. Além disso, algumas oportunidades de evoluir este trabalho podem ser exploradas. Essas lacunas e oportunidades são apresentadas e discutidas a seguir:

- Modelar diferentes LPSSCs ou LPSDs, além da Mobiline: para realizar a Prova de Conceito apresentada neste trabalho foi utilizada apenas uma LPSSC, a Mobiline. Portanto, é importante modelar uma segunda LPSSC ou LPSD na ferramenta para analisar diferentes cenários. É possível utilizar outras linhas encontradas na literatura bem como utilizar linhas tradicionais acrescentando modelo de contexto hipotético.
- Avaliar a aplicação do método em LPSs tradicionais: existem evidências de que o método proposto neste trabalho pode ser usado para Modelos de Características tradicionais, pois o MMSC contém o Modelo de Sistemas, equivalente ao Modelo de Características tradicional. No entanto, é necessário realizar avaliações para tal. Na ferramenta S.P.L.O.T

[1], é possível encontrar diversos Modelos de Características que podem ser modelados e avaliados na FixOnto.

- Avaliar a aplicação do método, em especial o enriquecimento semântico, utilizando modelos de características reais, como o Kernel do Linux, que possui mais de 5000 features. Por se tratar de um modelo com grande quantidade de características, é possível verificar a robustez da solução, bem como verificar a viabilidade de sua aplicação em soluções de grande escala.
- Estender o escopo do método para Variabilidade de Contexto: Estender a abordagem para trabalhar com Variabilidade de Contexto (CAPILLA *et al.*, 2014b) em um escopo mais geral, além de LPSSC ou LPSD. A variabilidade de contexto pode ser tratada independente de técnica de reuso. Portanto, o domínio aplicações sensíveis ao contexto e a solução proposta por este trabalho pode ser adaptada para não envolver diretamente conceitos de LPS.

[1] http://www.splot-research.org/

REFERÊNCIAS

ALMEIDA, M. B.; BAX, M. P. Uma visão geral sobre ontologias: pesquisa sobre definições, tipos, aplicações, métodos de avaliação e de construção. **Ciência da Informação**, SciELO Brasil, v. 32, n. 3, p. 7–20, 2003.

AMJA, A. M.; OBAID, A.; MILI, H. Combining variability, rca and feature model for context-awareness. In: **2016 Sixth International Conference on Innovative Computing Technology (INTECH)**. [S.l.: s.n.], 2016. p. 15–23.

ASIRELLI, P.; BEEK, M. H. ter; FANTECHI, A.; GNESI, S.; MAZZANTI, F. Design and validation of variability in product lines. In: **Proceedings of the 2nd International Workshop on Product Line Approaches in Software Engineering**. New York, NY, USA: ACM, 2011. (PLEASE '11), p. 25–30. ISBN 978-1-4503-0584-6. Disponível em: <http://doi.acm.org/10.1145/1985484.1985492>.

BENAVIDES, D.; FELFERNIG, A.; GALINDO, J. A.; REINFRANK, F. Automated analysis in feature modelling and product configuration. In: ____. **Safe and Secure Software Reuse: 13th International Conference on Software Reuse, ICSR 2013, Pisa, June 18-20. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 160–175. ISBN 978-3-642-38977-1. Disponível em: <http://dx.doi.org/10.1007/978-3-642-38977-1_11>.

BRACHMAN, R. J.; SCHMOLZE, J. G. An overview of the kl-one knowledge representation system. **Cognitive Science**, v. 9, n. 2, p. 171 – 216, 1985. ISSN 0364-0213. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0364021385800148>.

CAPILLA, R.; BOSCH, J.; TRINIDAD, P.; RUIZ-CORTéS, A.; HINCHEY, M. An overview of dynamic software product line architectures and techniques: Observations from research and industry. **Journal of Systems and Software**, v. 91, p. 3 – 23, 2014. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121214000119>.

CAPILLA, R.; ORTIZ, O.; HINCHEY, M. Context variability for context-aware systems. **Computer**, v. 47, n. 2, p. 85–87, Feb 2014. ISSN 0018-9162.

CLEMENTS, P.; NORTHROP, L. **Software Product Lines: Practices and Patterns**. Addison-Wesley, 2002. (The SEI Series in Software Engineering). ISBN 9780201703320. Disponível em: <http://books.google.com.br/books?id=tHGFQgAACAAJ>.

COSTA, P. A. d. S.; MARINHO, F. G.; ANDRADE, R. M. d. C.; OLIVEIRA, T. A. Fixture: a tool for automatic inconsistencies detection in Context-Aware SPL. 2015.

COSTA, P. A. S. **UMA FERRAMENTA PARA ANÁLISE AUTOMÁTICA DE MODELOS DE CARACTERÍSTICAS DE LINHAS DE PRODUTOS DE SOFTWARE SENSÍVEL AO CONTEXTO**. Dissertação (Mestrado) — MDCC, Universidade Federal do Ceara, UFC, 2012.

DAVIS, S. **Future perfect**. Addison-Wesley, 1987. ISBN 9780201115130. Disponível em: <http://books.google.com.br/books?id=KZ2zFJOU5Z4C>.

DERMEVAL, D.; TENóRIO, T.; BITTENCOURT, I. I.; SILVA, A.; ISOTANI, S.; RIBEIRO, M. Ontology-based feature modeling: An empirical study in changing scenarios. **Expert Systems with Applications**, v. 42, n. 11, p. 4950 – 4964, 2015. ISSN 0957-4174. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0957417415001190>.

DEY, A. K. Understanding and using context. **Personal Ubiquitous Comput.**, Springer-Verlag, London, UK, UK, v. 5, n. 1, p. 4–7, jan. 2001. ISSN 1617-4909. Disponível em: <http://dx.doi.org/10.1007/s007790170019>.

ECLIPSE. **Eclipse - The Eclipse Foundation open source community website**. 2017. Https://eclipse.org/. Acesso em: 11 jun. 2017.

EMF. **Eclipse Modeling Project - Home**. 2017. Https://eclipse.org/modeling/emf/. Acesso em: 13 jun. 2017.

EPSILON. **Epsilon - Home page**. 2014. Http://www.eclipse.org/epsilon/. Acesso em: 11 jun. 2017.

ERFANI, M.; ZANDI, M.; RILLING, J.; KEIVANLOO, I. Context-awareness in the software domain-a semantic web enabled modeling approach. **J. Syst. Softw.**, Elsevier Science Inc., New York, NY, USA, v. 121, n. C, p. 345–357, nov. 2016. ISSN 0164-1212. Disponível em: <https://doi.org/10.1016/j.jss.2016.02.023>.

FERNANDES, P.; WERNER, C. M. L. Ubifex: Modeling context-aware software product lines. In: **SPLC (2)'08**. [S.l.: s.n.], 2008. p. 3–8.

FILHO, J. a. B. F. **Enriquecimento Semântico de Linhas de Produto de Software**. Dissertação (Mestrado) — MDCC, Universidade Federal do Ceara, UFC, 2011.

FILHO, J. a. B. F.; BARAIS, O.; BAUDRY, B.; VIANA, W.; ANDRADE, R. M. C. An Approach for Semantic Enrichment of Software Product Lines. **Proceedings of SPLC 2012**, II, 2012.

FRAKES, W. B.; KANG, K. Software reuse research: Status and future. **IEEE Trans. Softw. Eng.**, IEEE Press, Piscataway, NJ, USA, v. 31, n. 7, p. 529–536, jul. 2005. ISSN 0098-5589. Disponível em: <http://dx.doi.org/10.1109/TSE.2005.85>.

GENESERETH, M. Knowledge interchange format draft proposed american national standard (dpans) ncits. 1998.

GLASS, R.; VESSEY, I.; RAMESH, V. Research in software engineering: an analysis of the literature. **Information and Software Technology**, v. 44, n. 8, p. 491 – 506, 2002. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584902000496>.

GLINZ, M.; BERNER, S.; JOOS, S. Object-oriented modeling with adora. **Inf. Syst.**, Elsevier Science Ltd., Oxford, UK, UK, v. 27, n. 6, p. 425–444, set. 2002. ISSN 0306-4379. Disponível em: <http://dx.doi.org/10.1016/S0306-4379(02)00015-7>.

GMF. **Graphical Modeling Framework**. 2017. Http://wiki.eclipse.org/Graphical_Modeling_Framework. Acesso em: 17 jun. 2017.

GMF. **Graphical Modeling Framework/Tutorial/Part 1**. 2017. Http://wiki.eclipse.org/Graphical_Modeling_Framework/Tutorial/Part_1. Acesso em: 17 jun. 2017.

GOMES, N. G. Um panorama da lǿgiorgica deôntica. **Kriterion: Revista de Filosofia**, scielo, v. 49, p. 9 – 38, 00 2008. ISSN 0100-512X.

GRISS, M. L. Implementing product-line features with component reuse. In: **Proceedings of the 6th International Conerence on Software Reuse: Advances in Software Reusability**. London, UK, UK: Springer-Verlag, 2000. (ICSR-6), p. 137–152. ISBN 3-540-67696-1. Disponível em: <http://dl.acm.org/citation.cfm?id=645546.658830>.

GRUBER, T. **Ontology (Computer Science) - definition in Encyclopedia of Database Systems**. [S.l.], 2009. Disponível em: <http://tomgruber.org/writing/ontology-definition-2007.htm>.

GUO, J.; WANG, Y.; TRINIDAD, P.; BENAVIDES, D. Consistency maintenance for evolving feature models. **Expert Systems with Applications**, v. 39, n. 5, p. 4987 – 4998, 2012. ISSN 0957-4174.

GURP, J. V.; BOSCH, J.; SVAHNBERG, M. On the notion of variability in software product lines. In: **Proceedings of the Working IEEE/IFIP Conference on Software Architecture**. Washington, DC, USA: IEEE Computer Society, 2001. (WICSA '01), p. 45–. ISBN 0-7695-1360-3. Disponível em: <http://dx.doi.org/10.1109/WICSA.2001.948406>.

HALLSTEINSEN, S.; HINCHEY, M.; PARK, S.; SCHMID, K. Dynamic software product lines. **Computer**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 41, n. 4, p. 93–95, abr. 2008. ISSN 0018-9162. Disponível em: <http://dx.doi.org/10.1109/MC.2008.123>.

HOLMES, R.; WALKER, R. J. Systematizing pragmatic software reuse. **ACM Trans. Softw. Eng. Methodol.**, ACM, New York, NY, USA, v. 21, n. 4, p. 20:1–20:44, fev. 2013. ISSN 1049-331X.

HORRIDGE, M.; DRUMMOND, N.; GOODWIN, J.; RECTOR, A. L.; STEVENS, R.; WANG, H. The manchester owl syntax. In: **OWLed**. [S.l.: s.n.], 2006. v. 216.

HORROCKS, I.; PATEL-SCHNEIDER, P. F.; BOLEY, H.; TABET, S.; GROSOF, B.; DEAN, M. **SWRL: A Semantic Web Rule Language Combining OWL and RuleML**. [S.l.], 2004.

KANG, K. C.; COHEN, S. G.; HESS, J. A.; NOVAK, W. E.; PETERSON, A. S. Feature-oriented domain analysis (foda) feasibility study. **Distribution**, Citeseer, v. 17, n. November, p. 161, 1990.

LIMA, F. F. P. **Um Sistema de Suporte para Computação Ubíqua**. Dissertação (Mestrado) — MDCC, Universidade Federal do Ceara, UFC, 2011.

LIMA, F. F. P.; ROCHA, L. S.; MAIA, P. H. M.; ANDRADE, R. M. C. A decoupled and interoperable architecture for coordination in ubiquitous systems. In: **SBCARS**. [S.l.: s.n.], 2011. p. 31–40.

LIMA, R. R. E.; ARAUJO, L. I.; SANTOS, S. I.; OLVEIRA A. THALISSON, M. S. G. a. E. B. C.; SANTOS, Z.; ANDRADE, R. M. C. Great tour: Um guia de visitas móvel e sensível ao contexto. **Proceeding of WebMedia 2013**, 2013.

MARINHO, F. G. **PRECISE Um Processo de veRificação Formal para modElos de Características de Aplicações Móveis e Sensíveis ao ContExto**. Tese (Doutorado) — MDCC, Universidade Federal do Ceara, UFC, 2012.

MARINHO, F. G.; ANDRADE, R. M.; WERNER, C.; VIANA, W.; MAIA, M. E.; ROCHA, L. S.; TEIXEIRA, E.; FILHO, J. B. F.; DANTAS, V. L.; LIMA, F.; AGUIAR, S. Mobiline: A nested software product line for the domain of mobile and context-aware applications. **Science of Computer Programming**, n. 0, p. –, 2012. ISSN 0167-6423. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167642312000871>.

MARINHO, F. G.; ANDRADE, R. M.; WERNER, C.; VIANA, W.; MAIA, M. E.; ROCHA, L. S.; TEIXEIRA, E.; FILHO, J. B. F.; DANTAS, V. L.; LIMA, F.; AGUIAR, S. Mobiline: A nested software product line for the domain of mobile and context-aware applications. **Science of Computer Programming**, v. 78, n. 12, p. 2381 – 2398, 2013. ISSN 0167-6423. Special Section on International Software Product Line Conference 2010 and Fundamentals of Software Engineering (selected papers of {FSEN} 2011). Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167642312000871>.

MARINHO, F. G.; LIMA, F.; FILHO, J. a. B. F.; ROCHA, L.; MAIA, M. E. F.; AGUIAR, S. B. de; DANTAS, V. L. L.; VIANA, W.; ANDRADE, R. M. C.; TEIXEIRA, E.; WERNER, C. A software product line for the mobile and context-aware applications domain. In: **Proceedings of the 14th international conference on Software product lines: going beyond**. Berlin, Heidelberg: Springer-Verlag, 2010. (SPLC'10), p. 346–360. ISBN 3-642-15578-2, 978-3-642-15578-9. Disponível em: <http://dl.acm.org/citation.cfm?id=1885639.1885671>.

MARINHO, F. G.; MAIA, P. H. M.; ANDRADE, R. M. C.; VIDAL, V. M. P.; COSTA, P. A. S.; WERNER, C. Safe adaptation in context-aware feature models. In: **Proceedings of the 4th International Workshop on Feature-Oriented Software Development**. New York, NY, USA: ACM, 2012. (FOSD '12), p. 54–61. ISBN 978-1-4503-1309-4. Disponível em: <http://doi.acm.org/10.1145/2377816.2377824>.

MEYER, M.; LEHNERD, A. **The Power of Product Platforms**. Free Press, 1997. ISBN 9780684825809. Disponível em: <http://books.google.com.br/books?id=PKJuQjSaHp0C>.

MOBILINE. **Mobiline - a software product line for the development of mobile and context-aware applications**. 2010. Http://mobiline.great.ufc.br/index.php. Acesso em: 11 sep. 2016.

NESKOVIC, S.; MATIC, R. Context modeling based on feature models expressed as views on ontologies via mappings. **Comput. Sci. Inf. Syst.**, v. 12, n. 3, p. 961–977, 2015. Disponível em: <http://dx.doi.org/10.2298/CSIS141031035N>.

NORTHROP, L. M. Sei's software product line tenets. **IEEE Softw.**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 19, n. 4, p. 32–40, jul. 2002. ISSN 0740-7459. Disponível em: <http://dx.doi.org/10.1109/MS.2002.1020285>.

OMG. **BPMN Version 2.0. Technical Report formal/2011-01-03**. [S.l.], 2011. Disponível em: <http://www.omg.org/spec/BPMN/2.0/PDF>. Acesso em: 28 set. 2015.

POHL, K.; BöCKLE, G.; LINDEN, F. J. v. d. **Software Product Line Engineering: Foundations, Principles and Techniques**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2010.

PROTÉGÉ. **Stanford University, Protégé project**: Site. 2013. Disponível em: <http://protege.stanford.edu>. Acesso em: 11 mar. 2013.

RINCóN, L.; GIRALDO, G.; MAZO, R.; SALINESI, C. An ontological rule-based approach for analyzing dead and false optional features in feature models. **Electronic Notes in Theoretical Computer Science**, v. 302, n. 0, p. 111 – 132, 2014. ISSN 1571-0661. Proceedings of the {XXXIX} Latin American Computing Conference (CLEI 2013).

SHE, S.; LOTUFO, R.; BERGER, T.; WaSOWSKI, A.; CZARNECKI, K. Reverse engineering feature models. In: **Proceedings of the 33rd International Conference on Software Engineering**. New York, NY, USA: ACM, 2011. (ICSE '11), p. 461–470. ISBN 978-1-4503-0445-0. Disponível em: <http://doi.acm.org/10.1145/1985793.1985856>.

SPÍNOLA, R. O.; TRAVASSOS, G. H. Towards a framework to characterize ubiquitous software projects. **Inf. Softw. Technol.**, Butterworth-Heinemann, Newton, MA, USA, v. 54, n. 7, p. 759–785, jul. 2012. ISSN 0950-5849. Disponível em: <http://dx.doi.org/10.1016/j.infsof.2012.01.009>.

VIANA, W. **Mobilité et sensibilité au contexte pour la gestion de documents multimédias personnels: CoMMediA**. [s.n.], 2010. Disponível em: <http://books.google.com.br/books?id=ujzTZwEACAAJ>.

W3C. **OWL Web Ontology Language**: Overview. 2004. Http://www.w3.org/TR/owl-features/. Acesso em: 29 mar. 2013.

WEISER, M. The computer for the 21st century. **Scientific American**, v. 265, n. 3, p. 66–75, set. 1991.

WOHLIN, C.; RUNESON, P.; HST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLN, A. **Experimentation in Software Engineering**. [S.l.]: Springer Publishing Company, Incorporated, 2012. ISBN 3642290434, 9783642290435.

ZAID, L. A.; KLEINERMANN, F.; TROYER, O. D. Applying semantic web technology to feature modeling. In: **Proceedings of the 2009 ACM symposium on Applied Computing**. New York, NY, USA: ACM, 2009. p. 1252–1256. ISBN 978-1-60558-166-8. Disponível em: <http://doi.acm.org/10.1145/1529282.1529563>.

APÊNDICEA – ONTOLOGIA DE DOMÍNIO DA FIXONTO REPRESENTADA EM LÓGICA DESCRITIVA (DL)

As expressões descritas de A.1 a A.7 representam as classes da ontologia de domínio criadas com os conceitos envolvidos na FixOnto. Elas estão agrupadas pelos nós da hierarquia, refletindo as modelagens representadas nas figuras 11, 12, 14 e 15.

$$
\begin{aligned}
&\mathit{SPL} \sqsubseteq \top \\
&\mathit{Element} \sqsubseteq \top \\
&\mathit{Expression} \sqsubseteq \top \\
&\mathit{ExternalConcept} \sqsubseteq \top \\
&\mathit{Rule} \sqsubseteq \top \\
&\mathit{OntologyInconsistences} \sqsubseteq \top
\end{aligned}
\tag{A.1}
$$

$$
\begin{aligned}
&\mathit{Attribute} \sqsubseteq \mathit{Element} \\
&\mathit{ContextRoot} \sqsubseteq \mathit{Element} \\
&\mathit{ContextEntity} \sqsubseteq \mathit{Element} \\
&\mathit{ContextInfo} \sqsubseteq \mathit{Element} \\
&\mathit{Feature} \sqsubseteq \mathit{Element}
\end{aligned}
\tag{A.2}
$$

$$
\begin{aligned}
&\mathit{RootFeature} \sqsubseteq \mathit{Feature} \\
&\mathit{MandatoryFeature} \sqsubseteq \mathit{Feature} \\
&\mathit{GroupedFeature} \sqsubseteq \mathit{Feature} \\
&\mathit{VariationTwo} \sqsubseteq \mathit{Feature}
\end{aligned}
\tag{A.3}
$$

$$
\begin{aligned}
&\mathit{CompositionRule} \sqsubseteq \mathit{Rule} \\
&\mathit{ContextRule} \sqsubseteq \mathit{Rule}
\end{aligned}
\tag{A.4}
$$

$$
\begin{aligned}
&Action \sqsubseteq Expression \\
&ActionLiteral \sqsubseteq Action \\
&Designate \sqsubseteq Action \\
&LogicalAction \sqsubseteq Action \\
&Antecedent \sqsubseteq Expression \\
&CompositionLiteral \sqsubseteq Antecedent \\
&LogicalExpression \sqsubseteq Antecedent \\
&RelationalExpression \sqsubseteq Antecedent \\
&Event \sqsubseteq Expression \\
&LogicalEvent \sqsubseteq Event \\
&RelationalEvent \sqsubseteq Event
\end{aligned}
\tag{A.5}
$$

$$
\begin{aligned}
&TestCase \sqsubseteq ExternalConcept \\
&UseCase \sqsubseteq ExternalConcept
\end{aligned}
\tag{A.6}
$$

$$
\begin{aligned}
&WFAR1 \sqsubseteq OntologyInconsistences \\
&WFAR2 \sqsubseteq OntologyInconsistences
\end{aligned}
\tag{A.7}
$$

Já as expressões descritas de A.1 a A.7 representam as propriedades que relacionam indivíduos das classes representadas anteriormente. Por exemplo, indivíduos OWL da classe *Feature* podem se relacionar entre si através da propriedade *hasChildFeature*, na qual uma feature pode ser filha de outra feature.

$$
\begin{aligned}
&relatedToElement \sqsubseteq ObjectProperty \\
&relatedToExpression \sqsubseteq ObjectProperty \\
&relatedToElement \sqsubseteq ObjectProperty \\
&external \sqsubseteq ObjectProperty \\
&hasInconsistency \sqsubseteq ObjectProperty
\end{aligned}
\tag{A.8}
$$

$$
\begin{aligned}
&\textit{hasAttribute} \sqsubseteq \textit{relatedToElement}\\
&\forall \textit{hasAttribute.Attribute} \sqsubseteq \textit{Feature}\\
&\textit{hasChildFeature} \sqsubseteq \textit{relatedToElement}\\
&\forall \textit{hasChildFeature.Feature} \sqsubseteq \textit{Feature}\\
&\textit{hasContextEntity} \sqsubseteq \textit{relatedToElement}\\
&\forall \textit{hasContextEntity.ContextEntity} \sqsubseteq \textit{ContextRoot}\\
&\textit{hasContextInfo} \sqsubseteq \textit{relatedToElement}\\
&\forall \textit{hasContextInfo.ContextInfo} \sqsubseteq \textit{ContextEntity}\\
&\textit{hasFatherContextRoot} \sqsubseteq \textit{relatedToElement}\\
&\forall \textit{hasFatherContextRoot.ContextRoot} \sqsubseteq \textit{ContextInfo}\\
&\textit{hasFatherContextEntity} \sqsubseteq \textit{relatedToElement}\\
&\forall \textit{hasFatherContextEntity.ContextEntity} \sqsubseteq \textit{ContextEntity}\\
&\textit{hasFatherFeature} \sqsubseteq \textit{relatedToElement}\\
&\forall \textit{hasFatherFeature.Feature} \sqsubseteq \textit{Feature}
\end{aligned}
\tag{A.9}
$$

$$
\begin{aligned}
&\textit{hasAction} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasAction.Action} \sqsubseteq \textit{AdaptaionRule}\\
&\textit{hasAntecedent} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasAntecedent.Consequent} \sqsubseteq \textit{CompositionRule}\\
&\textit{hasConsequent} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasConsequent.Consequent} \sqsubseteq \textit{CompositionRule}\\
&\textit{hasContextVariable} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasContextVariable.Attribute} \sqsubseteq \textit{RelationalEvent}\\
&\textit{hasEvent} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasEvent.Event} \sqsubseteq \textit{AdaptationRule}\\
&\textit{hasExpressionVariable} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasExpressionVariable.Feature} \sqsubseteq \textit{RelationalExpression}\\
&\textit{hasFeaturedElement} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasFeaturedElement.OptionalFeature} \sqsubseteq \textit{ActionLiteral}\\
&\textit{hasLeftSideAction} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasLeftSideAction.ActionLiteral} \sqsubseteq \textit{Action}\\
&\textit{hasLeftSideAction} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasLeftSideAction.Designate} \sqsubseteq \textit{Action}\\
&\textit{hasLeftSideAction} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasLeftSideAction.LogicalAction} \sqsubseteq \textit{Action}
\end{aligned}
\tag{A.10}
$$

$$
\begin{aligned}
&\textit{hasRightSideAction} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasRightSideAction.ActionLiteral} \sqsubseteq \textit{Action}\\
&\textit{hasRightSideAction} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasRightSideAction.Designate} \sqsubseteq \textit{Action}\\
&\textit{hasRightSideAction} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasRightSideAction.LogicalAction} \sqsubseteq \textit{Action}\\
&\textit{hasLeftSideComposition} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasLeftSideComposition.CompositionLiteral} \sqsubseteq \textit{Antecedent}\\
&\textit{hasLeftSideComposition} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasLeftSideComposition.LogicalExpression} \sqsubseteq \textit{Antecedent}\\
&\textit{hasLeftSideComposition} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasLeftSideComposition.RelationalExpression} \sqsubseteq \textit{Antecedent}\\
&\textit{hasRightSideComposition} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasRightSideComposition.CompositionLiteral} \sqsubseteq \textit{Antecedent}\\
&\textit{hasRightSideComposition} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasRightSideComposition.LogicalExpression} \sqsubseteq \textit{Antecedent}\\
&\textit{hasRightSideComposition} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasRightSideComposition.RelationalExpression} \sqsubseteq \textit{Antecedent}\\
&\textit{hasLeftSideEvent} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasLeftSideEvent.LogicalEvent} \sqsubseteq \textit{Event}\\
&\textit{hasLeftSideEvent} \sqsubseteq \textit{relatedToExpression}\\
&\forall \textit{hasLeftSideEvent.RelationalEvent} \sqsubseteq \textit{Event}
\end{aligned}
\tag{A.11}
$$

$$
\begin{aligned}
&\textit{hasTestCase} \sqsubseteq \textit{external}\\
&\forall \textit{hasTestCase.TestCase} \sqsubseteq \textit{Element}\\
&\textit{implementsUseCase} \sqsubseteq \textit{external}\\
&\forall \textit{implementsUseCase.UseCase} \sqsubseteq \textit{Element}
\end{aligned}
\tag{A.12}
$$

$$
\begin{aligned}
&\textit{hasCicle} \sqsubseteq \textit{hasInconsistency}\\
&\forall \textit{hasCicle.Element} \sqsubseteq \textit{Element}\\
&\textit{hasEqualName} \sqsubseteq \textit{hasInconsistency}\\
&\forall \textit{hasEqualName.Element} \sqsubseteq \textit{Element}
\end{aligned}
\tag{A.13}
$$