Universidade Federal do Ceará

Centro de Ciências

Programa de Pós-Graduação em Ciências da Computação

Fabiana Gomes Marinho

PRECISE

Um Processo de veRificação Formal para modElos de CaracterístIcas de Aplicações Móveis e Sensíveis ao ContExto

Fortaleza

2012

Fabiana Gomes Marinho

PRECISE

Um Processo de veRificação Formal para modElos de CaracterístIcas de Aplicações Móveis e Sensíveis ao ContExto

Tese apresentada ao Programa de Pós-graduação em Ciências da Computação da UFC, como requisito parcial para a obtenção do grau de DOUTOR(A) em Ciências da Computação.

Orientadora: Rossana Maria de Castro Andrade

PhD em Ciência da Computação

Fortaleza

2012

Marinho, Fabiana Gomes
PRECISE / Fabiana Gomes Marinho - 2012
182.p

Linha de Produtos de Software, Computação Móvel e Sensível ao Contexto, Verificação e Validação, Modelos de Características, Verificação Formal. I.Título.

CDU

Fabiana Gomes Marinho

PRECISE

Um Processo de veRificação Formal para modElos de CaracterístIcas de Aplicações Móveis e Sensíveis ao ContExto

Tese apresentada ao Curso de Pós-graduação em Ciências da Computação da UFC, como requisito parcial para a obtenção do grau de DOUTOR(A) em Ciências da Computação.

Aprovado em

BANCA EXAMINADORA

Rossana Maria de Castro Andrade

Universidade Federal do Ceará (Orientadora)

Cláudia Maria Lima Werner

Universidade Federal do Rio de Janeiro (Avaliadora Externa)

Eduardo Santana de Almeida

Universidade Federal da Bahia (Avaliador Externo)

Vania Maria Ponte Vidal

Universidade Federal do Ceará (Avaliadora Interna)

Francisco Heron de Carvalho Junior

Universidade Federal do Ceará (Avaliador Interno)

Resumo

As Linhas de Produtos de Software (LPSs), além do seu uso em aplicações tradicionais, têm sido utilizadas no desenvolvimento de aplicações que executam em dispositivos móveis e são capazes de se adaptarem sempre que mudarem os elementos do contexto em que estão inseridas. Essas aplicações, ao sofrerem alterações devido a mudanças no seu ambiente de execução, podem sofrer adaptações inconsistentes e, consequentemente, comprometer o comportamento esperado. Por esse motivo, é essencial a criação de um processo de verificação que consiga checar a corretude e a consistência dessas LPSs, bem como checar a corretude tanto dos produtos derivados como dos produtos adaptados dessas LPSs. Sendo assim, nesta tese de doutorado é proposto o PRECISE - um Processo de Verificação Formal para Modelos de Características de Aplicações Móveis e Sensíveis ao Contexto. O PRECISE auxilia na identificação de defeitos na modelagem da variabilidade de uma Linha de Produtos de Software (LPS) para aplicações móveis e sensíveis ao contexto e, assim, minimiza problemas que ocorreriam durante a execução dos produtos gerados a partir dessa LPS. É importante ressaltar que o PRECISE é definido com base em uma especificação formal e em um conjunto de propriedades de boa formação elaborados usando Lógica de Primeira Ordem. Essa especificação é um pré-requisito para a realização de uma modelagem da variabilidade sem ambiguidades. Para avaliar o PRECISE, uma validação é realizada a partir da especificação formal e das propriedades de boa formação definidas no processo. Essa validação tem como objetivo mostrar que o PRECISE consegue identificar defeitos, anomalias e inconsistências existentes em um modelo de variabilidades de uma LPS para aplicações móveis e sensíveis ao contexto. Nessa validação, cinco técnicas diferentes são utilizadas: *Perfil UML*, *Object Constraint Language (OCL)*, *Lógica Proposicional*, *Prolog* e *Simulação*. Além de minimizar os defeitos e inconsistências dos modelos de variabilidades das LPS, o PRECISE ainda se beneficia da generalidade e flexibilidade intrínsecas à notação formal usada na sua especificação.

Palavras-Chave: Linha de Produtos de Software, Computação Móvel e Sensível ao Contexto, Verificação e Validação, Modelos de Características, Verificação Formal.

Abstract

Software Product Lines (SPLs) have been used to develop different types of applications, including the ones that run on mobile devices and are able to adapt when the context elements in which they are located change. These applications can change due to variations in their execution environment and inconsistent adaptations can occur, compromising the expected behavior. Then there is a need for creating a verification process to check the correctness and consistency of these SPLs as well as to check the correctness of both derived products and adapted products from these SPLs. Thus, this work proposes PRECISE - A Formal Verification Process for Feature Models of Mobile and Context-Aware Applications. PRECISE helps to identify defects in the variability modeling of an Software Product Line (SPL) for mobile and context-aware applications, minimizing problems that can take place during the execution of products generated from this SPL. It is worth noting that PRECISE is defined based on a formal specification and a set of well-formedness properties developed using First-Order Logic, which are prerequisites for the achievement of an unambiguous variability modeling. To evaluate PRECISE, a validation is performed from the formal specification and well-formedness properties defined in the process. This validation intends to show that PRECISE is able to identify defects, anomalies and inconsistencies in a variability model of an SPL for mobile and context-aware applications. In this validation, five different techniques are used: *UML Profile*, *OCL*, *Propositional Logic*, *Prolog* and *Simulation*. While minimizing the defects and inconsistencies in the variability models of an SPL, PRECISE still benefits from the generality and flexibility intrinsic to the formal notation used in its specification.

Key-Words: Software Product Line, Mobile and Context-Aware Computing, Verification and Validation, Feature Model, Formal Verification.

Agradecimentos

A Deus, que protegeu e me guiou durante esta longa jornada.

Inicialmente, eu gostaria de agradecer a professora Rossana Maria de Castro Andrade da Universidade Federal do Ceará (UFC), que, apesar de todas as adversidades, acreditou no meu potencial para ser uma aluna do curso de doutorado desta universidade e investiu seu tempo e conhecimento para orientar este trabalho.

Eu também gostaria de agradecer a professora Cláudia Werner por receber-me como pesquisadora visitante no Grupo de Reutilização de Software da Universidade Federal do Rio de Janeiro (UFRJ) e por me conduzir nos primeiros passos da minha pesquisa.

Os resultados desta tese de doutorado não poderiam ser alcançados sem o apoio do Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat) que me ofereceu um ambiente perfeito por meio de sua infra-estrutura e colaboradores. Minha gratidão a todos os membros do GREat por sua paciência e boa vontade em participar de seminários, fornecendo, ouvindo e estimulando discussões sobre vários aspectos do meu trabalho. Este é um "GREat" grupo de pesquisa e eu estou orgulhosa de ter participado dele. Além disso, agradeço o suporte financeiro fornecido pelo grupo para eventos científicos nacionais e internacionais que foram extremamente importantes para o meu desenvolvimento pessoal e profissional ao propiciar a interação com diversos especialistas na área.

Durante este período, alguns colegas mais próximos também foram muito importantes. Especialmente, gostaria de agradecer Valéria, Cidcley, Fátima, Paulo Alexandre, Paulo Henrique e Bosco que contribuíram em muitas discussões.

Meu marido! Eu gostaria de dizer que você é incrível e tudo na minha vida. Você foi o porto seguro que me apoiou durante todo esse período, compartilhando sonhos, problemas, felicidades e conquistas. Muito obrigada por tudo André. O seu carinho, compreensão, paciência, impaciência e amor desempenharam um papel fundamental. Eu também não posso deixar de agradecer a toda a sua família que me recebeu e cuidou de mim durante o meu período no Rio de Janeiro.

Pedro e Esther, obrigada pela existência de vocês na minha vida. Espero ter mais tempo no futuro para desfrutar com vocês.

Mãe e pai, muito obrigada por seu amor e cuidado. Eu admiro todos os seus esforços para apoiar-me neste período. Também agradeço a meus irmãos (e cunhados/-cunhadas) pelo apoio em todos os momentos. Vocês são muito importantes para mim.

Este trabalho é dedicado a todos vocês que de algum modo contribuíram para esta conquista.

Esta pesquisa foi apoiada financeiramente pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

GLOSSÁRIO

BDD Binary Decision Diagram. 48, 49

BPMN Business Process Model and Notation. 21

CSP Constraint Satisfaction Problem. 17, 48, 49, 53

E-C-A Evento-Condição-Ação. 60

FeatuRSEB Feature Reuse-Driven Software Engineering Business. 36

FODA Feature-Oriented Domain Analysis. 16, 36, 37, 69

FORM A Feature-Oriented Reuse Method. 36, 37

GREat Grupo de Redes de Computadores, Engenharia de Software e Sistemas. 92

LPMSC Linha de Produtos de Software Móvel e Sensível ao Contexto. 11, 16, 20, 22, 39–42, 50, 65, 154, 155, 157–159

LPMSCs Linhas de Produtos de Software Móveis e Sensíveis ao Contexto. 16, 24, 26, 43, 69, 154, 156

LPS Linha de Produtos de Software. 1, 15, 23, 24, 32–35, 41–46, 57–59, 62, 70, 81, 85, 156

LPSD Linha de Produtos de Software Dinâmica. 42, 61

LPSDs Linhas de Produtos de Software Dinâmicas. 24, 42, 43

LPSM Linha de Produtos de Software Múltipla. 57

LPSs Linhas de Produtos de Software. 1, 8, 11, 14–17, 21, 23, 24, 26, 34, 39, 40, 42–44, 46, 47, 56–62, 69–71, 85, 154, 159

LPSSCs Linhas de Produtos de Software Sensíveis ao Contexto. 26

MC Modelo de Características. 11, 16–18, 34–39, 43–46, 48, 49, 51–55, 57–59, 61–63, 65, 66, 85, 88, 155

MCC Modelo de Características do Contexto. 11, 39–42, 46, 47, 59, 64, 65, 70, 73–77, 79, 86, 88, 90, 91, 102, 105, 106

MCCs Modelos de Características do Contexto. 75

MCS Modelo de Características do Sistema. 11, 39–41, 46, 47, 59, 64, 65, 70, 73–75, 77–81, 86, 88–92, 96, 98, 99, 101, 102, 104, 105

MCs Modelos de Características. 13, 14, 16–18, 21, 25, 27, 34, 37, 38, 43–63, 66, 68, 70, 85, 86, 88, 91, 155, 157

MDE Model-Driven Engineering. 56–58, 64

MMSC Modelo de Características Móvel e Sensível ao Contexto. 16, 19–22, 24, 25, 39, 40, 46–48, 63, 69, 70, 73, 86–88, 92, 105, 107, 155–158

MMSCs Modelos de Características Móveis e Sensíveis ao Contexto. 16, 18, 20–23, 25, 47, 48, 56, 63, 69–71, 86–88, 107, 154–157

OCL Object Constraint Language. 1, 2, 55, 56, 59, 158

OMG Object Management Group. 22

OVM Orthogonal Variability Model. 34

PA Produto Adaptado. 73, 74, 83–86

PAs Produtos Adaptados. 73

PD Produto Derivado. 73, 81, 82, 84, 86, 100

PDs Produtos Derivados. 73, 156

PLUS Product Line UML-Based Software Engineering. 36

POA Programação Orientada a Aspectos. 58

RA Regra de Adaptação. 73, 77, 79

RAs Regras de Adaptação. 16, 40, 47, 73, 79, 80, 84, 86, 91

RC Regra de Composição. 73, 77, 78

RCs Regras de Composição. 16, 39, 73, 77–82, 84–86, 90

RSEB Reuse-Driven Software Engineering Business. 37

SAT Satisfiability. 48

SGML Standard Generic Markup Language. 30

SPL Software Product Line. 2

SPLs Software Product Lines. 2

UML Unified Modeling Language. 30, 55, 59

V&V Verificação e Validação. 43, 47

VFD Varied Feature Diagrams. 69

XML Extensible Markup Language. 30

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

Capítulo 1

Introdução

Vários desafios e direcionamentos de pesquisas têm surgido nos últimos anos associados à Engenharia de Linhas de Produtos de Software. Um desses desafios está em desenvolver uma representação correta das características comuns e variáveis dos produtos de uma Linha de Produtos de Software. Propor uma nova solução para esse problema é o objeto de estudo desta tese de doutorado que envolve três temas de pesquisa: **Linha de Produtos de Software**, **Aplicações Móveis e Sensíveis ao Contexto** e **Verificação Formal**. Cada um desses temas será abordado nesta pesquisa desde a formulação do problema, análise das soluções existentes, descrição da solução proposta e validação dessa solução.

Neste Capítulo uma contextualização e definição do problema tratado nesta tese de doutorado é apresentada. Na Seção 1.1, a motivação para a escolha do tema de pesquisa é discutida. Na Seção 1.2, as questões de pesquisa que serão investigadas são levantadas. Na Seção 1.3, os objetivos e metas específicas são detalhados, e na Seção 1.4, é apresentada a metodologia adotada para o desenvolvimento da pesquisa. Finalmente, na Seção 1.5 a terminologia de referência adotada é apresentada e na Seção 1.6, a organização do restante deste documento é descrita.

1.1 Motivação

As LPSs têm se destacado nos últimos anos como uma das mais promissoras abordagens de reutilização de software. As LPSs têm como objetivo principal identificar aspectos comuns e variáveis de uma família de produtos [42]. As similaridades entre os produtos membros são configuradas e reutilizadas de modo a trazer melhorias significativas na produtividade e no custo investidos no desenvolvimento de um produto membro. As variabilidades são utilizadas para possibilitar um aumento na flexibilidade e adaptação às necessidades dos usuários, pois permitem tornar explícitos os pontos onde os produtos de uma família diferem entre si, devendo receber tratamento específico [19]. Modelos de variabilidades são utilizados para capturar essas informações e os **MCs** são frequentemente utilizados

com essa finalidade.

Por outro lado, a popularização dos dispositivos móveis (e.g., *smart phones* e *tablets*) tem levado tanto empresas de software quanto a academia a desenvolverem aplicações para esses dispositivos, denominadas **Aplicações Móveis** nesta tese de doutorado. Essas aplicações utilizam as vantagens proporcionadas pelos dispositivos móveis, os quais permitem que seus usuários se movam e sejam capazes de se comunicar e trocar informações e serviços através de alguma tecnologia de comunicação sem fio (e.g., *Bluetooth*, *Wi-fi* e *3G*) a qualquer momento e em qualquer lugar.

Entretanto, os avanços tecnológicos dos dispositivos móveis em termos de conectividade, processamento e recursos de armazenamento têm demandado inovações nessas aplicações levando ao desenvolvimento de aplicações distribuídas e interativas, caracterizadas principalmente pela dinamicidade e incerteza dos recursos disponíveis. Dessa forma, as aplicações móveis devem ser capazes de se adaptarem sempre que mudarem os elementos do contexto em que estão inseridas [58]. Tais aplicações são ditas **Aplicações Móveis e Sensíveis ao Contexto** [95].

Tendo em vista os benefícios das LPSs e os desafios das aplicações móveis e sensíveis ao contexto, Hallsteinsen *et al.* [74] e Fernandes *et al.* [64] afirmam que as LPSs têm se mostrado uma forma eficiente para o desenvolvimento de software móvel e sensível ao contexto, pois conseguem tratar as variações dos requisitos, as necessidades dos usuários e as restrições dos recursos. Por exemplo, em Marinho *et al.* [97], essas características são exploradas na construção de uma LPS para Guias de Visita Móveis e Sensíveis ao Contexto.

Entretanto, é importante ressaltar que a introdução da mobilidade e da sensibilidade ao contexto traz desafios para o desenvolvimento das LPSs. Um desses desafios está relacionado com a necessidade de capturar nos modelos de variabilidades das LPSs as informações de contexto e o impacto dessas informações na adaptação dos produtos derivados a partir das LPSs. Já existem estudos na literatura que consideram o contexto durante a modelagem da variabilidade. Por exemplo, Fernandes *et al.* [63] propõem uma notação para modelagem da variabilidade para LPSs sensíveis ao contexto, chamada UBIFEX-Notation. Ubayashi *et al.* [151] sugerem uma Engenharia para LPSs dependentes do contexto e que usam métodos formais, onde o contexto é tratado como uma LPS separada. As especificações de contexto são construídas como artefatos dessa LPS e são

usados métodos formais para especificar e verificar a corretude dos artefatos construídos.

Além disso, o desenvolvimento dessas novas aplicações exige alto grau de interoperabilidade, desacoplamento, adaptabilidade e autonomia, pois executam em dispositivos móveis portáteis que estão em constante movimento, podendo conectar-se à diversas e diferentes tipos de redes móveis. Esses dispositivos são funcionalmente poderosos e podem se movimentar oferecendo um conjunto de serviços habituais, sendo também capazes de conectar-se, obter dados e fornecê-los a outros usuários, aplicações e sistemas [91]. Outro ponto a ser observado é que essas aplicações devem tratar as mudanças no ambiente de execução de modo a evitar adaptações inconsistentes e, consequentemente, comportamentos inesperados. Nesta tese de doutorado, essas LPSs são denominadas **Linhas de Produtos de Software Móveis e Sensíveis ao Contexto (LPMSCs)** e o MC para essas LPSs é denominado **Modelo de Características Móvel e Sensível ao Contexto (MMSC)**.

Assim como nas LPSs tradicionais, os Modelos de Características Móveis e Sensíveis ao Contexto (MMSCs) possuem restrições entre os seus elementos, chamadas **Regras de Composição (RCs)** que definem relações de dependência ou de exclusividade mútua entre as características do modelo. Contudo, além das RCs, os MMSCs também possuem as **Regras de Adaptação (RAs)** que são responsáveis por definir as situações de contexto que podem levar à adaptações ou reconfigurações dos produtos da LPMSC. A presença dessas restrições pode alterar o conjunto válido de combinações de características de um MMSC, podendo até mesmo levar a combinações inconsistentes. Portanto, é essencial verificar se esses modelos são construídos de forma correta e assegurar que nenhuma das restrições estabelecidas seja violada, uma vez que a corretude e a consistência dos MMSCs permitem a derivação de produtos móveis e sensíveis ao contexto corretos e consistentes.

Uma solução está no uso de uma semântica formal que defina um MMSC, suas propriedades de boa formação e de consistência e quais operações podem ser executadas sobre os mesmos. Uma abordagem adotada na literatura com esse objetivo consiste em mapear os MCs para linguagens formais para as quais existem técnicas e ferramentas de análise disponíveis. A abordagem **Feature-Oriented Domain Analysis (FODA)** [86] foi a primeira a propor o suporte automático para a análise dos MCs de forma a verificar formalmente um modelo. Desde então, uma diversidade de notações e técnicas com poder expressivo variado para representar e analisar MCs têm sido apresentadas, [96], [16], [48],

[20], [148] e [156]. Contudo, não existe um consenso sobre qual semântica ou técnica é a mais adequada.

Mannion [96] descreve os MCs por meio de fórmulas proposicionais e usa Prolog para implementar um protótipo para analisar os modelos. Batory [16] amplia o trabalho de Mannion e transforma os MCs em gramáticas e fórmulas proposicionais que são submetidas a um provador de teoremas com o objetivo de identificar e explicar os defeitos detectados.

Por outro lado, Czarnecki *et al.* [48] propõem uma semântica formal para MCs baseados em cardinalidade e atributos de características, através da definição de uma gramática independente de contexto. Esses MCs são denominados MCs estendidos. Benavides [20] propõe a tradução dos MCs para *Constraint Satisfaction Problem (CSP)* e aborda a análise de MCs estendidos, porém o autor não trata a explicação dos defeitos encontrados. Já Trinidad [148], explora o uso de explicações para os defeitos a partir da extração de informações dos modelos, e Wang *et al.* [156] convertem um MC para ontologia e exploram métodos para detectar e explicar a origem de MCs que não derivam nenhum produto, denominados modelos nulos ou inconsistentes.

Todas essas propostas podem ser consideradas formas de realizar as operações e análises sobre um MC. No entanto, é possível observar que os tipos de análise realizados pelas abordagens existentes varia bastante. Além disso, a maioria delas apenas detecta problemas clássicos dos MCs, tais como características mortas ou falsos opcionais [22]. Contudo, existe uma variedade de operações de análise relativas à combinação das regras de composição,aos produtos derivados e produtos adaptados que podem ser executadas sobre os MCs com o objetivo de obter informações sobre a qualidade, confiabilidade e flexibilidade das LPSs e que permanecem pouco exploradas.

Segundo Benavides *et al.* [22], a principal desvantagem das propostas supracitadas está no baixo nível de abstração, pois essas abordagens são adequadas apenas quando os MCs são analisados usando os formalismos e as ferramentas específicas de cada proposta. Esse fato limita o uso das mesmas, pois em vez das abordagens serem genéricas o suficiente para se adequarem às necessidades dos usuários, esses são obrigados a se adequarem às exigências das abordagens propostas.

Também é importante ressaltar que não foi identificado na literatura nenhum processo de verificação que contemple os vários tipos de análise nem todos os elementos

que compõem um MC. Um processo de verificação tem como objetivo garantir que cada produto de trabalho reflita corretamente as especificações de entrada e envolve a definição de uma estratégia de verificação que pode fazer uso de técnicas como revisões e provas formais que assegurem que os defeitos encontrados serão removidos [81]. Portanto, a criação de um processo de verificação para MMSCs consegue maximizar a qualidade dos MMSCs, pois minimiza a ocorrência de defeitos nos vários elementos de um MMSCs.

Outro ponto a ser destacado é que o suporte para a análise dos MCs estendidos é um ponto falho na maioria dos trabalhos descritos anteriormente. Com exceção de [48] e [20], que abordam cardinalidade e atributo das características em seus trabalhos, os conceitos dos MCs estendidos ainda não estão formalizados na literatura.

A motivação para o desenvolvimento desta pesquisa de doutorado consiste, então, em utilizar processos e linguagens formais para minimizar defeitos e inconsistências nos MMSCs de forma abrangente e flexível o suficiente para serem instanciadas para qualquer outra linguagem de especificação ou plataforma de implementação. Além disso, abranger os conceitos dos MCs estendidos e tratar formalmente as várias operações para análise de MCs identificadas na literatura.

1.2 Hipótese e questões de pesquisa

Diante da motivação apresentada na Seção 1.1, pode ser firmada a seguinte hipótese:

A criação e utilização de um processo de verificação formal para modelos de características móvel e sensível ao contexto aumenta a qualidade dos elementos que compõem esses modelos em termos de corretude e consistência dos mesmos.

Como forma de tratar essa hipótese, esta tese de doutorado objetiva fornecer um processo de verificação para os MMSCs, denominado **PRECISE**, cujo conjunto de operações de análise executadas e propriedades de boa formação e de consistência são documentados formalmente. Construir o PRECISE é uma tarefa desafiadora e suscita as seguintes questões que serão investigadas nesta pesquisa:

1. **Quais são as técnicas de modelagem de características e modelagem de contexto disponíveis na literatura?**

Existem várias abordagens para modelagem de características e modelagem de contexto na literatura. Assim, a investigação deste tema deve verificar as abordagens existentes.

2. **Que critérios usar para selecionar as abordagens a serem estudadas?**

   Relacionada com a primeira questão, é necessário definir critérios para selecionar as abordagens que serão usadas para o desenvolvimento desta tese de doutorado.

   Com relação às abordagens para modelagem de características, os seguintes critérios são usados:

   - a técnica de modelagem é referenciada na literatura;
   - a técnica de modelagem aborda o conceito de cardinalidade e de atributo;
   - a técnica de modelagem aborda o conceito de regras de composição;
   - a técnica de modelagem possui uma definição formal;
   - a técnica de modelagem possui operações de análise automática definidas; e
   - a técnica de modelagem possui o suporte de ferramentas.

   Para a modelagem do contexto são usados os critérios a seguir:

   - a técnica de modelagem é referenciada na literatura;
   - a técnica de modelagem aborda os atributos das entidades de contexto identificados na literatura, tais como heterogeneidade, qualidade e frequência de alteração das informações de contexto;
   - a técnica de modelagem aborda regras de adaptação;
   - a técnica de modelagem possui uma definição formal; e
   - a técnica de modelagem possui o suporte de ferramentas.

3. **Quais propriedades um MMSC bem formado e consistente deve ter?**

   A partir dos elementos que compõem um MMSC, as seguintes propriedades devem ser definidas:

   - como reduzir os defeitos e anomalias de um MMSC;
   - como assegurar que um MMSC é consistente;

- como reduzir os defeitos nas regras de um MMSC;
- como assegurar que as regras de um MMSC são consistentes entre si;
- como reduzir os defeitos de um produto derivado de um MMSC;
- como reduzir os defeitos de um produto adaptado de um MMSC; e
- investigar se as propriedades identificadas acima estão formalizadas na literatura.

4. **Quais operações de análise devem ser realizadas sobre os MMSCs?**

   As seguintes operações de análise são identificadas na literatura para os MMSC:

   - verificar se o modelo e as regras estão corretas;
   - verificar se o modelo é consistente;
   - verificar se as regras são consistentes entre si;
   - verificar se um produto é válido com relação à uma LPMSC;
   - obter as características núcleo;
   - identificar características mortas;
   - identificar opcional falso;
   - identificar cardinalidade errada;
   - identificar se um produto adaptado permanece válido; e
   - investigar na literatura se as operações identificadas acima estão formalizadas.

5. **O que deve ser considerado para desenvolver um processo de verificação formal para um MMSC?**

   Para iniciar a elaboração do PRECISE é necessário investigar na literatura os seguintes pontos:

   - checar a existência de outros processos de verificação formal;
   - investigar o uso de métodos formais para verificação dos modelos; e
   - investigar se existe algum tipo de validação para um processo de verificação formal.

1.3 Objetivo

Esta tese de doutorado possui como objetivo principal propor um processo de verificação formal que visa aumentar, em tempo de desenvolvimento, a corretude e a consistência dos MMSCs, e antecipar problemas que possam ocorrer na adaptação dos produtos desse MMSC em tempo de execução.

Para atingir o objetivo aqui proposto, foram definidos os objetivos específicos, estando os mesmos relacionados às questões de pesquisa apresentadas na Seção 1.2. São eles:

- análise das abordagens existentes para modelagem das características e modelagem do contexto;
- formalização do MMSC com base na análise das especificações formais existentes para MCs;
- identificação e formalização das propriedades que um MMSC deve possuir;
- identificação e formalização das operações que devem ser incluídas para verificar um MMSC;
- criação e formalização do PRECISE, em especial, a formalização da análise da combinação das restrições do modelo; e
- validação do PRECISE.

As principais contribuições desta tese de doutorado abrangem a área de **LPSs**, mais especificamente a **análise dos MCs para aplicações móveis e sensíveis ao contexto**.

1.4 Metodologia para o desenvolvimento da tese

A metodologia para o desenvolvimento desta tese de doutorado é apresentada na Figura 1.1. A notação *Business Process Model and Notation (BPMN)*[1][34] é utilizada no decorrer

deste trabalho por tratar-se de uma notação padrão desenvolvida e mantida pelo *Object Management Group (OMG).*

Após uma extensa pesquisa bibliográfica nos temas envolvidos nesta tese de doutorado, o **primeiro passo** consiste em construir o processo de verificação formal para MMSCs que tem por finalidade checar a corretude e a consistência dos MMSCs, bem como a corretude dos produtos derivados e produtos adaptados desses modelos e, assim, aumentar a corretude das LPMSC. Além disso, inclui várias operações de análise identificadas na literatura. O resultado do levantamento bibliográfico realizado está descrito nos Capítulos 2 e 3 e a definição do processo pode ser encontrada no Capítulo 4.

O **segundo passo** consiste em definir um MMSC usando uma especificação formal baseada na Lógica de Primeira Ordem. Para a elaboração dessa especificação formal, é necessário analisar técnicas para modelagem das características e para a modelagem do contexto presentes na literatura, com o objetivo de extrair as contribuições de cada técnica, bem como identificar as lacunas a serem preenchidas, tais como maior independência da tecnologia utilizada e abrangência dos conceitos dos MCs estendidos. A especificação formal elaborada encontra-se documentada no Capítulo 5.

Uma vez formalizado o MMSC, o **terceiro passo** consiste em definir as propriedades de boa formação e as propriedades de consistência para esses modelos. Para definir essas propriedades, é preciso analisar as propriedades e os critérios de boa formação e de consistência para MCs e para modelos de contexto disponíveis na literatura. As propriedades definidas estão formalizadas em Lógica de Primeira Ordem e estão documentadas no Capítulo 6.

O **quarto passo** consiste em validar o processo de verificação proposto. Para isso, a especificação formal e as propriedades de boa formação e de consistência de um MMSC são usadas pelo processo de verificação formal proposto para verificar a corretude e a consistência de algum MMSC. A validação realizada está descrita no Capítulo 7.

[1]A BPMN é uma notação desenvolvida pelo OMG para se tornar um padrão para a documentação de processos e facilitar o entendimento por todos os usuários, desde os analistas de negócio até os desenvolvedores técnicos responsáveis pela implementação que irá executar os processos. Assim, a notação BPMN cria uma ponte entre a concepção de um processo de negócios e sua implementação.

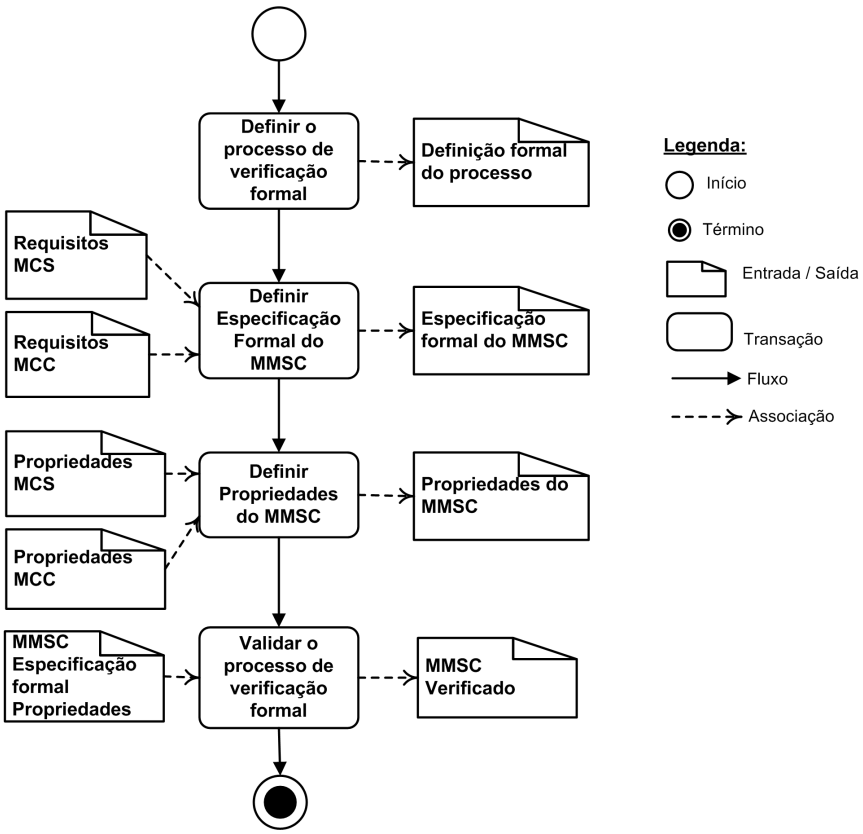

Figura 1.1: Metodologia para o desenvolvimento da tese.

1.5 Terminologia utilizada

Nesta seção, é apresentada a terminologia de referência adotada para esta pesquisa que tem como objetivo tornar homogênea a apresentação da nomenclatura utilizada no processo de verificação dos MMSCs. Esta terminologia foi baseada em parte no trabalho de Moura & Nikolaj [54] que afirmam que a verificação de MCs implica no processo formal de determinar se um modelo satisfaz um conjunto de critérios definidos.

- **Propriedades de boa formação:** conjunto de restrições sintáticas que determinam a estrutura de um modelo ou regra.
- **Corretude:** até que ponto o modelo atende às propriedades de boa formação especificadas [54].
- **Consistência:** grau de uniformidade, padronização e ausência de contradições entre regras. É importante ressaltar que na terminologia específica das LPSs, o termo **consistência** é utilizado como uma possibilidade de configurar um produto a partir da LPS. Sendo assim, a primeira definição é usada nesta tese para verificar a consistência entre regras, enquanto a segunda definição é usada para verificar se pelo

menos um produto correto pode ser configurado a partir de uma LPS.

- **Anomalia:** presença de informações contraditórias em um modelo.
- **Falha:** um evento que ocorre em algum momento, quando o sistema não fornece um serviço conforme esperado por seus usuários [10].
- **Erro:** um estado errôneo de sistema que pode levá-lo a um comportamento inesperado por seus usuários [10].
- **Defeito:** uma característica do sistema de software que pode levar a um erro de sistema [10].

De acordo com a terminologia definida, um MMSC é **correto** se atende aos critérios ou propriedades de boa formação estabelecidas para esse modelo e é **consistente** se é possível configurar ao menos um produto a partir desse modelo. Defeitos e anomalias inseridos na sua construção tornam o MMSC **incorreto**.

As regras de composição e as regras de adaptação definidas para um MMSC são **corretas** se sua estrutura satisfaz às propriedades de boa formação definidas para essas regras. Além disso, essas regras são consideradas **consistentes** se não existem contradições entre elas.

O MC de um produto configurado ou de produto adaptado é **correto** se satisfaz todas as propriedades de boa formação e regras de composição definidas.

1.6 Organização do texto

Este trabalho está organizado em outros sete capítulos, além deste capítulo de introdução, a seguir:

Capítulo 2 - Fundamentação Teórica, apresenta os conceitos relacionados com o desenvolvimento da tese. Primeiro, esse Capítulo apresenta uma revisão da literatura sobre Computação Móvel e Sensível ao Contexto, discutindo os principais conceitos envolvidos (mobilidade, sensibilidade ao contexto e adaptabilidade). Além disso, é apresentado o paradigma das LPSs, LPMSCs e Linhas de Produtos de Software Dinâmicas (LPSDs). O processo de Engenharia para LPSs baseado em dois ciclos e o

conceito de variabilidade também são descritos. Outro ponto abordado diz respeito aos conceitos básicos relativos às operações de análise em MCs.

Capítulo 3 - Trabalhos Relacionados, descreve e discute, com base em uma revisão da literatura nos temas envolvidos, os trabalhos relacionados com a pesquisa desta tese de doutorado.

Capítulo 4 - PRECISE, apresenta as entradas, atividades e saídas do processo de verificação formal proposto para MMSCs.

Capítulo 5 - Especificação Formal do Modelo de Características Móvel e Sensível ao Contexto, apresenta a formalização de um MMSC. Essa formalização é documentada por meio de uma especificação utilizando Lógica de Primeira Ordem.

Capítulo 6 - Propriedades de um Modelo de Características Móvel e Sensível ao Contexto, descreve as propriedades necessárias para um MMSC ser considerado correto e consistente.

Capítulo 7 - Validação do PRECISE, descreve a estratégia utilizada para validar o processo proposto e os resultados obtidos com a realização dessa validação.

Capítulo 8 - Conclusão, onde as considerações finais desta pesquisa, incluindo as principais contribuições, resultados alcançados, dificuldades e limitações, bem como direcionamentos futuros são apresentados.

Capítulo 2

Fundamentação Teórica

Neste Capítulo é apresentada a fundamentação teórica relacionada a esta tese de doutorado. Na Seção 2.1, é apresentada uma breve introdução relacionada aos conceitos abordados. Na Seção 2.2, o conceito de mobilidade é introduzido. Na Seção 2.3, são apresentados conceitos relativos com a sensibilidade ao contexto. Na Seção 2.4, a teoria necessária para o desenvolvimento desta tese com relação à software adaptável é descrita. Na Seção 2.5, as definições de Linha de Produtos de Software, Linha de Produtos de Software Móvel e Sensível ao Contexto e Linha de Produtos de Software Dinâmica são apresentadas. Por fim, na Seção 2.6, os conceitos básicos de Verificação e Validação de software com foco em Linhas de Produtos de Software e as operações de análise de MCs disponíveis na literatura são descritas.

2.1 Introdução

As aplicações móveis e sensíveis ao contexto são desenvolvidas para dispositivos móveis e caracterizadas por serem distribuídas, interativas e pela incerteza de recursos disponíveis [95]. Mobilidade, sensibilidade ao contexto e adaptabilidade são requisitos essenciais e que estão fortemente relacionados a essas aplicações. A mobilidade expõe os usuários e aplicações a situações de contexto diferentes. Já a sensibilidade ao contexto captura as informações do contexto atuais, enquanto a adaptabilidade utiliza essas informações para adaptar o comportamento das aplicações adequadamente.

Em geral, projetar (*design*) e implementar esses requisitos torna o desenvolvimento do software móvel e sensível ao contexto uma tarefa complexa. Entretanto, o processo de desenvolvimento desse tipo de aplicação pode se beneficiar do paradigma das LPSs em termos de reusabilidade e configurabilidade [63], uma vez que as LPSs conseguem minimizar problemas intrínsecos às aplicações móveis e sensíveis ao contexto, como a fragmentação, causada pela heterogeneidade dos dispositivos móveis. A fragmentação está relacionada ao número de versões de uma mesma aplicação que deve ser criada devido

às peculiaridades de cada tipo de dispositivo móvel.

As LPSs que fornecem suporte para o desenvolvimento de aplicações sensíveis ao contexto são denominadas **Linhas de Produtos de Software Sensíveis ao Contexto (LPSSCs)** [64]. Segundo Fernandes *et al.*, essas LPSs conseguem capturar tanto as similaridades e as variabilidades das aplicações, como as informações do ambiente do contexto que podem influenciar o comportamento dos produtos gerados. No entanto, esse termo não é utilizado nesta tese de doutorado por ser abrangente, podendo levar o leitor a pensar em aplicações sensíveis ao contexto que não executam em dispositivos móveis. Sendo assim, neste trabalho de pesquisa, as LPSs específicas para o domínio de aplicações móveis e sensíveis ao contexto são denominadas **Linhas de Produto de Software Móveis e Sensíveis ao Contexto** (LPMSCs).

2.2 Mobilidade

Mobilidade pode ser definida como a capacidade de poder se deslocar ou ser deslocado facilmente [92]. No contexto da computação móvel, mobilidade se refere ao uso de dispositivos móveis portáteis, funcionalmente poderosos, que podem se movimentar, oferecendo um conjunto de serviços habituais, sendo também capazes de conectar-se, obter dados e fornecê-los a outros usuários, aplicações e sistemas [91]. A diversidade dos dispositivos móveis, tais como a variedade das plataformas de desenvolvimento, os tipos das aplicações, os mecanismos de transferência de dados e a mobilidade do usuário, entre outros, introduz limitações para o desenvolvimento de aplicações para os mesmos. Considerando esse cenário, Ballard [13] afirma que o desenvolvimento de aplicações para dispositivos móveis deve considerar três fatores fundamentais: o *contexto móvel*, a *aplicação móvel* e o *usuário móvel*.

O *contexto móvel* corresponde a qualquer informação que caracterize a interação entre o usuário e a aplicação móvel e o ambiente que os cerca [159]. É importante destacar que o contexto móvel está associado às limitações dos dispositivos relativas à memória, ao poder de processamento, à bateria, à capacidade de armazenamento, dentre outras. Apesar da evolução do hardware estar conseguindo minimizar algumas dessas limitações, a grande maioria dos dispositivos móveis portáteis disponíveis no mercado ainda as apresenta.

As *aplicações móveis* correspondem às aplicações que executam em dispositivos móveis. Em geral, as aplicações móveis podem ser independentes ou dependentes. As aplicações independentes executam sem qualquer contato com outro usuário ou sistema (e.g., relógio e calculadora). As aplicações dependentes precisam conectar-se a outro usuário ou sistema para executarem (e.g., calendário, e-mail, agenda e contatos). Todavia, a finalidade primária dos dispositivos móveis está associada com a conectividade. Um dispositivo móvel deve conectar as pessoas ou sistemas bem como transmitir e receber informações.

A característica *usuário móvel* deve-se ao fato do usuário poder se mover enquanto executa uma aplicação. Porém, a aplicação deve estar sempre disponível, independente da localização do usuário. Outra característica do usuário móvel é que o usuário é passível de interrupção, pois ao usar o dispositivo, uma única aplicação pode ser executada por vez.

2.3 Sensibilidade ao contexto

Sensibilidade ao contexto é a capacidade de tornar as aplicações cientes das informações que descrevem a situação de contexto do usuário, as quais podem ser usadas para aumentar (ou manter) a interação entre os aplicativos e o usuário, ou melhorarem a sua usabilidade e desempenho [12].

2.3.1 Contexto

Na literatura, podem ser encontradas várias definições para o termo contexto. Por exemplo, Schilit e Theimer [131] definem contexto como localização, identificação de pessoas e objetos próximos, e mudanças nesses objetos. Os autores também classificam contexto em três categorias: contexto computacional (e.g., memória disponível, capacidade de processamento, largura de banda), contexto do usuário (e.g., localização, pessoas próximas, hora) e contexto físico (e.g., temperatura, nível de luminosidade, barulho).

Brown *et al.* [35] se referem a contexto como localização, identificação das pessoas ao redor do usuário, hora do dia, estação do ano, temperatura, entre outros. Abowd *et al.* [2], de forma semelhante a Schilit e Theimer [131], sugerem as mesmas três categorias para contexto citadas anteriormente.

O contexto computacional contém principalmente propriedades relacionadas ao hardware, como também propriedades que são geradas pelo hardware com o objetivo de processamento ou comunicação. Exemplos típicos incluem o processador, tamanho da memória, dispositivos de entrada/saída e conexão da rede. O contexto do usuário está relacionado com propriedades como a localização do usuário, a localização de pessoas e sistemas próximos, e a situação social do usuário. O contexto físico contém propriedades conceituais que são geradas pelo ambiente, tais como nível de barulho, iluminação e temperatura de um determinado local.

Dey [57] conceitua contexto como qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade pode ser uma pessoa, um lugar ou um objeto considerado relevante para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a aplicação.

Vieira [154] distingue contexto de elemento contextual. Contexto representa um conjunto de elementos contextuais instanciados que são necessários para apoiar uma tarefa. Os elementos contextuais, por sua vez, representam qualquer dado ou informação capaz de caracterizar uma entidade em um domínio. Além disso, contexto possui uma dimensão dinâmica e deve ser construído em tempo de execução, enquanto que elementos contextuais são estáveis e podem ser definidos em tempo de projeto.

Neste trabalho de pesquisa, a definição de contexto adotada é semelhante à definição apresentada por Vieira [154], pois os elementos contextuais também são considerados para definir uma situação de contexto específica. **Contexto** é definido como um conjunto de valores de atributos que representam uma situação particular das entidades de contexto. As entidades de contexto podem ser computacionais (e.g., dispositivos de hardware e rede), físicas (e.g., usuários e objetos de um ambiente) ou ambientais (e.g., sala e prédio).

2.3.2 Modelo de contexto

Um modelo de contexto é necessário para definir e armazenar as informações de contexto em uma forma processável por uma máquina. Strang e Linnhoff-Popien [138] realizaram um levantamento bibliográfico sobre as técnicas para representação do contexto e classificaram essas técnicas com base na estrutura de dados utilizada para representar as informações de contexto. A seguir são descritas as principais técnicas identificadas neste

estudo:

Modelo chave-valor utiliza a estrutura de dados mais simples para modelar as informações de contexto. As informações de contexto são descritas por meio de pares compostos por uma chave que identifica a informação e um valor associado a essa chave. Caso as informações de contexto obtidas apresentem os valores das chaves, as ações correspondentes devem ser tomadas.

Modelo baseados em marcação consiste em uma abordagem de modelagem baseada em marcação que corresponde a uma estrutura de dados hierárquica composta por marcações (*tags*) com atributos e valores. Eles geralmente são baseados em uma serialização de uma derivação de uma *Standard Generic Markup Language (SGML)*, que é a superclasse de todas as linguagens de marcação, tais como a *Extensible Markup Language (XML)*. Uma linguagem baseada em XML permite a representação de informações de contexto e dependências, definindo padrões de interação em uma escala limitada.

Modelo gráfico consiste em modelos como, por exemplo, os modelos da *Unified Modeling Language (UML)*[111], [112]. A UML é apropriada para a modelagem de contextos, pois é uma linguagem de modelagem de propósito geral, além de possuir uma estrutura genérica. Isso pode ser verificado em [78]. Algumas abordagens baseadas em UML utilizam extensões dessa linguagem, como perfis e estereótipos, para representar as informações de contexto [135].

Modelo orientado a objetos refere-se à informações de contexto modeladas com base no paradigma da orientação a objeto que exploram o poder de conceitos como encapsulamento, reusabilidade e herança. Os detalhes de processamento de contexto são encapsulados ao nível de objetos. O acesso às informações de contexto é feito apenas por meio de interfaces.

Modelo baseado em lógica usa a lógica para definir as condições sob as quais uma expressão resultante ou fato pode ser derivado a partir de um conjunto de outras expressões ou fatos (processo conhecido como inferência). Para descrever essas condições em um conjunto de regras, um sistema formal é aplicado. Em um modelo de contexto baseado em lógica, o contexto é definido como fatos, expressões e regras.

Geralmente as informações de contexto são adicionadas, atualizadas e removidas em termos de fatos ou inferidas a partir de regras do respectivo software.

Modelo baseado em ontologia utiliza ontologias para representar os conceitos e suas relações. O uso de ontologias provê uma maneira uniforme para especificar os conceitos do núcleo do modelo, assim como uma quantidade arbitrária de sub-conceitos e fatos, possibilitando o compartilhamento e reúso do conhecimento contextual. Esse conhecimento contextual é avaliado através dos analisadores de ontologias.

2.4 Software adaptável

Um software é adaptável quando possui a capacidade de se adequar às mudanças nas circunstâncias em que executa [122]. Assim, se o ambiente de execução mudar, o software deve ser capaz de se modificar para executar de uma forma mais eficiente nas novas condições. O processo de adaptação pode levar em conta de informações que caracterizam a situação de contexto do usuário ao acessar o software ou não. O conceito de software adaptativo está relacionado a um conjunto de situações mais abrangente que a sensibilidade ao contexto e que podem levar à adaptação do mesmo. Por exemplo, o software pode detectar a necessidade de corrigir um componente que não está executando adequadamente.

Viana *et al.* [153] afirmam que no domínio de aplicações para dispositivos móveis, a adaptação pode ocorrer de várias formas e em diferentes momentos. Dessa maneira, acrescentam que se faz necessário diferenciar aspectos importantes da adaptação, tais como:

O objeto de adaptação (o que deve ser adaptado): os objetos de adaptação frequentemente mencionados pelos autores são conteúdo, apresentação e funcionalidade (serviço). O conteúdo representa as informações e arquivos multimídia apresentados aos usuários. A redução, filtragem e transformação de conteúdo são os mecanismos de adaptação mais considerados. Com relação à apresentação, o processo de adaptação permite a modificação de como o conteúdo é percebido pelos usuários. A redução da qualidade de um vídeo para exibição em um dispositivo móvel com memória limitada é um exemplo típico desse tipo de adaptação.

O momento da execução (quando o mecanismo é iniciado): a adaptação pode ser classificada em estática e dinâmica. O primeiro tipo de adaptação sugere a criação de versões de um aplicativo ou recurso antes da interação com o usuário. Ao instalar o aplicativo, o processo de adaptação corresponde a uma escolha entre uma das versões disponíveis de recursos ou de conteúdo que permanecerá o mesmo durante a execução da aplicação. A adaptação dinâmica, por sua vez, ocorre durante a execução do aplicativo. As transformações são feitas em tempo real, de acordo com a execução da aplicação e o comportamento do aplicativo varia durante a execução.

O mecanismo utilizado (como a adaptação é realizada): quando o usuário intervém na alteração da aplicação, é utilizado o termo **adaptável**. O termo **adaptativo** é usado quando o aplicativo por si só toma a decisão de se adaptar, sem qualquer intervenção do usuário. Os termos adaptável e adaptativo são constantemente usados como sinônimos. No entanto, Viana *et al.* enfatiza a diferença entre estes termos. Além do usuário e da aplicação, uma adaptação pode ser iniciada e controlada por outro software. Essa é uma abordagem especial para adaptação de aplicações móveis que executam em cima de algum *middleware* que monitora o ambiente e é responsável pela adaptação automática às mudanças nesse ambiente. Nesse caso, considerando a definição de [153], a adaptação é considerada adaptativa.

2.5 Linha de produtos de software

Uma LPS é uma coleção de sistemas de software que compartilham um conjunto de características comuns e gerenciadas, satisfazendo necessidades específicas de um segmento de mercado ou objetivo particular, e que são desenvolvidas de maneira pré-definida a partir de um conjunto comum de recursos-base. As similaridades compartilhadas pelos produtos de software podem ser exploradas para alcançar economias na produção e assim os produtos podem ser construídos a partir de artefatos comuns [42].

Griss [70] destaca a importância da existência das variações entre os produtos e define LPS como um grupo de produtos que compartilham um conjunto de requisitos comuns. Porém, esses produtos também possuem variabilidades significativas nos requisitos. De forma semelhante, Weiss e Lai [157] definem uma LPS como um processo para se obter vantagens das características comuns e das variabilidades previsíveis de uma família

de produtos.

Segundo Clements e Northrop [42], as atividades para a construção de uma LPS são: desenvolvimento do núcleo de artefatos, desenvolvimento dos produtos e gerenciamento da LPS. A atividade de desenvolvimento do núcleo de artefatos é denominada Engenharia de Domínio e a atividade de configuração dos produtos a partir desse núcleo de artefatos é denominada Engenharia de Aplicação. Dessa forma, a Engenharia de Domínio tem como objetivo elaborar artefatos núcleo que serão reutilizados pela Engenharia de Aplicação durante a construção dos produtos [42].

Clements e Northrop [42] também afirmam que para introduzir com sucesso as engenharias de domínio e de aplicação em uma empresa, é necessário considerar questões organizacionais e de gerenciamento relacionadas e a comunicação entre eles. Existem questões pertinentes a estabelecer formas de institucionalizar as novas tecnologias, gerenciar riscos, realizar planejamento e rastrear mudanças. Essas atividades correspondem ao Gerenciamento da LPS.

Um enfoque para a engenharia de uma LPS baseado em um *Modelo de Dois Ciclos de Vida* é descrito por van der Linden *et al.* [93]. O primeiro ciclo abrange as atividades da **Engenharia de Domínio**. Durante esse ciclo, a infra-estrutura da LPS composta de todos os artefatos relevantes para o desenvolvimento de um software, incluindo requisitos, arquitetura, codificação e testes, é construída, possibilitando o desenvolvimento dos produtos individuais.

O segundo ciclo de vida abrange as atividades da **Engenharia de Aplicação** que são responsáveis pela construção dos produtos finais a partir da infra estrutura da LPS construída no primeiro ciclo de vida. A infra estrutura da LPS geralmente contém a maioria das funcionalidades requeridas pelo novo produto. Portanto, os vários artefatos (e.g., arquitetura e código) podem ser instanciados, levando a uma versão de um produto.

2.5.1 Variabilidade em linhas de produtos de software

A variabilidade em uma LPS pode ser definida como a possibilidade de um sistema ou artefato ser alterado, customizado ou configurado para um contexto em particular [32]. Por tratamento da variabilidade, entende-se, desde o momento de reconhecimento de um ponto de variação até o mapeamento desse ponto para uma instância de variação. Existem

vários mecanismos disponíveis na literatura para o tratamento das variabilidades [86], [79], [157] e [48].

Os modelos de variabilidade definem as características que representam os pontos de variação, ou seja, aquelas características que podem ou não ser selecionadas durante a configuração de um produto específico. Esses modelos são importantes na Engenharia das LPSs devido ao seu papel na documentação e gerenciamento da variabilidade. Diversas abordagens para modelagem da variabilidade foram propostas, tais como tabelas, modelos de variabilidade ortogonais (OVM) e modelos de características (MCs).

No uso de tabelas [157], as características são listadas na vertical e os produtos são listados na horizontal. A alocação de características aos produtos é feita pela marcação da intersecção na tabela.

O *Orthogonal Variability Model (OVM)* proposto por Pohl *et al.* [146] também é usado para definir a variabilidade de uma LPS. Os elementos básicos do OVM são os pontos de variação e suas variantes (quais são as escolhas possíveis em cada ponto de variação). Os pontos de variação e variantes podem ser declarados como obrigatórios ou opcionais. Para um conjunto de variantes relacionadas a um mesmo ponto de variação, é possível definir um intervalo para o número de variantes selecionáveis desse grupo. Além disso, dois tipos de restrições (requer e exclui) podem ser especificadas para pares de variantes, pontos de variação ou pares mistos. O OVM enfatiza uma distinção clara entre os pontos de variação e variantes.

Pelo fato do MC ser uma das notações mais frequentemente utilizada para representar graficamente os modelos de variabilidade, essa notação é adotada nesta tese de doutorado. Um produto é uma combinação válida de características e um MC denota o conjunto de produtos válidos para a LPS que o MC representa. Nesse modelo, as características são os blocos de construção e representam os aspectos variáveis ou os pontos comuns a todos os produtos da LPS. Combinando as características de um MC de maneiras diferentes, respeitando as restrições de composição impostas pelo MC, obtém-se um conjunto de produtos.

2.5.2 Modelo de características

O MC é um modelo de alto nível utilizado para expressar os produtos da LPS, representando as características de um domínio específico, suas semelhanças, variabilidades e relações [86]. Quando diferentes produtos são analisados na mesma LPS ou domínio de problema, eles são comparados com base nas características. O MC permite realizar a derivação ou configuração de um produto que corresponde ao processo de construir um produto a partir de uma LPS.

Diversas definições para o termo **característica** são encontradas na literatura. Segundo Kang *et al.* [86], as características representam atributos das aplicações do domínio que estão diretamente relacionados e visíveis ao usuário final. Essa definição foi refinada em trabalhos subsequentes para abordar as necessidades de outras partes interessadas, além do usuário final. Kang *et al.* [88] definem uma característica como uma abstração essencial que tanto os clientes como os desenvolvedores conseguem compreender. Gomaa e Shin [68] afirmam que uma característica representa requisitos reutilizáveis para uma LPS.

Nesta tese de doutorado, a definição de **característica** adotada é dada por Griss [70]: "uma **característica** é um requisito que os usuários e clientes veem como sendo de importância na descrição e distinção dos membros de uma LPS". A escolha dessa definição deve-se ao fato desse trabalho de pesquisa considerar os usuários/clientes como principais envolvidos na definição dos produtos de uma LPS.

Outras definições do termo **característica** incluem:

- Qualquer coisa que usuários ou programas do cliente podem querer controlar sobre um conceito [47];
- Uma unidade lógica de comportamento especificado por um conjunto de requisitos funcionais e não funcionais [31];
- Uma característica do produto a partir de visões do usuário ou cliente, que consiste, essencialmente, de um conjunto coeso de requisitos individuais [41];
- Um incremento na funcionalidade do produto [17];
- Uma estrutura que amplia e modifica a estrutura de um determinado programa, a fim de satisfazer um requisito das partes interessadas, para implementar e encapsular

uma decisão de *design* e oferecer uma opção de configuração [7];

- Um aspecto importante para o cliente [126]; e
- Características importantes do produto [130].

O primeiro MC foi proposto por Kang *et al.* [86] como parte do método FODA. Desde então, vários outros foram propostos, por exemplo, A Feature-Oriented Reuse Method (FORM) [85], Feature Reuse-Driven Software Engineering Business (FeatuRSEB) [71], van Gurp *et al.* [72], Product Line UML-Based Software Engineering (PLUS) [61], Programação Generativa [47], Riebisch *et al.* [126] e Diagramas Baseados em Cardinalidade [48].

No método FODA, um modelo é composto de dois elementos: características e relacionamentos entre essas características. As características são organizadas em uma estrutura hierárquica de árvore. Uma das características é a raiz que representa o sistema como um todo. Os relacionamentos podem ser de dois tipos:

Relacionamento hierárquico é definido entre uma característica ancestral e suas características descendentes. Uma característica descendente só pode fazer parte dos produtos em que a sua característica ancestral aparece. Os relacionamentos hierárquicos podem ser mandatórios, opcionais ou alternativos. Um relacionamento mandatório entre uma característica ancestral e uma característica descendente indica que, quando a característica ancestral faz parte de um produto específico, a característica descendente também deve fazer parte desse produto. Um relacionamento opcional entre uma característica ancestral e uma característica descendente indica que, quando a característica ancestral faz parte de um produto específico, a característica descendente pode ou não fazer parte desse produto. Um relacionamento alternativo entre uma característica ancestral e uma característica descendente indica que, quando a característica ancestral faz parte de um produto específico, uma e somente uma característica descendente deve fazer parte desse produto.

Relacionamento de restrição (ou **regra de composição**) é definido entre duas ou mais características. As restrições podem ser inclusivas ou exclusivas. Uma restrição inclusiva (ou regra de composição inclusiva) estabelece uma relação de dependência entre as características. Por exemplo, se uma característica X requer uma característica Y, implica que se a característica X é incluída em um produto, a

característica Y também deve ser incluída. O contrário não é válido. Uma restrição exclusiva (ou regra de composição exclusiva) estabelece uma relação de exclusividade mútua entre as características. Por exemplo, se uma característica X exclui uma característica Y, implica que se a característica X é incluída em um produto, a característica Y não deve ser incluída. O contrário também é válido.

Griss *et al.* [71] propõem a *Reuse-Driven Software Engineering Business (RSEB)*, uma extensão da abordagem FODA que adiciona um novo tipo de relacionamento hierárquico entre as características denominado *OR*. Esse relacionamento estabelece que, se uma característica pai faz parte de um produto específico, uma ou mais características filhas devem pertencer a esse produto.

Em seguida, Kang *et al.* [85] propõem uma extensão chamada FORM que inclui o conceito de *camadas de abstração* nos MCs para documentar a variabilidade em cada fase do ciclo de desenvolvimento. Para isso, eles adicionaram quatro camadas de abstração e novas relações entre as características. As camadas de abstração incluídas são Camada de Capacidade, Camada do Ambiente Operacional, Camada do Domínio da Tecnologia e Camada da Técnica de Implementação. Cada característica deve pertencer a uma dessas quatro camadas.

A relação *Generalização/Especialização* consiste em um relacionamento que permite definir explicitamente uma característica filha como uma especialização de sua característica pai e uma característica pai ser uma generalização das suas características filhas. A relação *Implementada-por* permite que características das camadas mais altas sejam conectadas às características que as implementam nas camadas mais baixas.

Os MCs definidos com base nos conceitos definidos anteriormente são denominados **modelos de características básicos**.

Em [11] e [33], os autores adicionam aos MCs o relacionamento de multiplicidade para substituir os relacionamentos alternativo e *OR*. Em seguida, Czarnecki *et al.* [18], [16] propõem o MC baseado em *cardinalidade*. Nesses trabalhos, os autores introduzem um relacionamento hierárquico que generaliza os relacionamentos mandatório e opcional. Um relacionamento de cardinalidade indica que, quando uma característica pai pertence a um produto específico, a inclusão das suas características filhas depende da cardinalidade. Esses MCs são denominados **modelos de características baseados em cardinalidade**.

A cardinalidade equivalente ao relacionamento mandatório ([1..1]), significa que a característica filha deve ser parte do produto. A cardinalidade equivalente ao relacionamento opcional ([0..1]), significa que uma característica filha pode ou não ser parte do produto. Além disso, a cardinalidade é adicionada a um grupo de cardinalidade que deve ter um valor mínimo (m) e um valor máximo (n) (card[m..n]). Por exemplo, se um grupo de características possui quatro características filhas, a cardinalidade desse grupo pode ser $card_4[2..3]$, que implica que no mínimo 2 e no máximo 3 características podem ser parte do produto.

Em [46], Czarnecki *et al.* introduzem os *atributos* como uma forma de representar uma seleção de valores a partir de um domínio finito ou infinito que define o tipo do atributo. Benavides *et al.* [21] fazem uma distinção entre atributos que são diretamente relacionados com uma característica e atributos derivados, também chamados características extra funcionais, que são atributos cujo valor é calculado baseado nos valores de outros atributos. Atributos derivados também são associados a uma característica. Benavides [20] define atributo como qualquer informação da característica que pode ser medida.

Os MCs que incluem o conceito de atributo são denominados **modelos de características estendidos**. Esses modelos são utilizados como base para o desenvolvimento dessa pesquisa de doutorado por serem mais ricos semanticamente. O MC ilustrado na Figura 2.1 é um exemplo de um MC estendido.

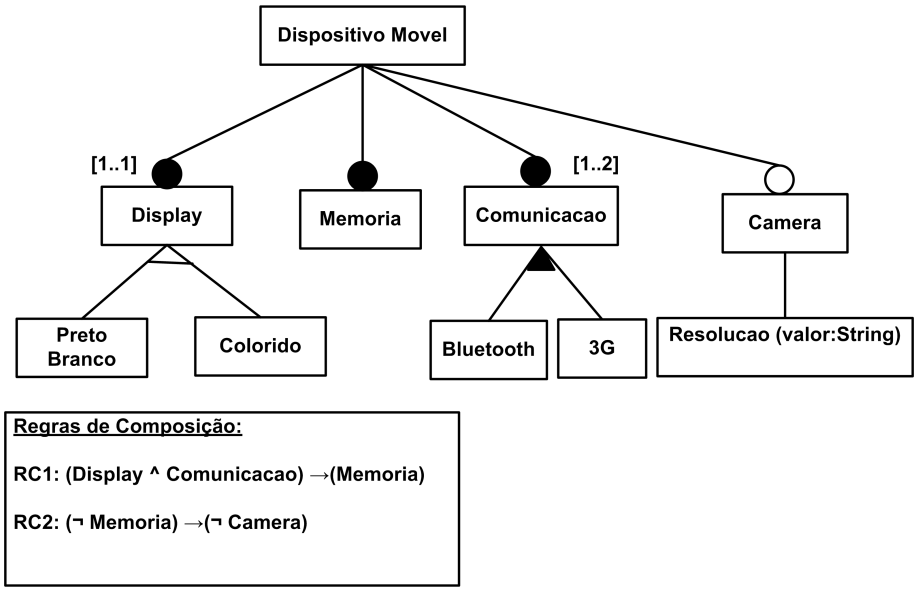

Figura 2.1: Exemplo de um MC estendido.

De acordo com a Figura 2.1, um *Dispositivo Movel* é composto por: *Display*, *Memoria*, *Comunicacao* e, opcionalmente, *Camera*. O *Display* é uma característica al-

ternativa, podendo assumir apenas um dos valores: *Colorido* ou *Preto Branco*. Já a característica *Comunicacao* é opcional e pode assumir os valores *Bluetooth* e / ou *3G* simultaneamente. Por fim, a característica opcional *Camera* possui um atributo denominado *Resolucao* que define o valor da resolução da câmera, caso essa exista. A Figura 2.1 também ilustra duas regras de composição, sendo **RC1** uma regra de composição de dependência (requer) e **RC2** uma regra de composição de mútua exclusividade (exclui) entre as características.

A regra de composição RC_1 afirma que a presença das características *Display* e *Comunicacao* no MC requer a presença da característica *Memoria*. Já a regra de composição RC_2 restringe que a ausência da característica *Memoria* no MC exclui a presença da característica *Camera*.

2.5.3 Linha de produtos de software móvel e sensível ao contexto

Os sistemas de software móveis e sensíveis ao contexto são submetidos a variações das necessidades dos usuários e às evoluções das restrições dos recursos que se modificam ao longo do tempo. Em geral, esses requisitos tornam o desenvolvimento de software móvel e sensível ao contexto uma tarefa complexa. Na literatura, podem ser encontrados trabalhos que buscam tratar tais variações [153], [95], [106], [115], [117] e [152].

Por outro lado, estudos têm mostrado que o desenvolvimento desse tipo de software pode se beneficiar do paradigma das LPSs em termos de reusabilidade e configurabilidade [74], [64] e [97].

2.5.4 Modelo de características móvel e sensível ao contexto

Nesta pesquisa de doutorado, o MC para uma LPMSC é denominado *Modelo de Características Móvel e Sensível ao Contexto* (MMSC) e é composto por dois modelos: MCS e MCC.

O MCS expressa as variabilidades, similaridades e restrições definidas entre as características do domínio modelado. O MCC representa as entidades do contexto consideradas relevantes para o domínio considerado. De acordo com a Seção 2.5.2, relacionamentos de restrição podem ser definidos entre duas ou mais características. Esses relacionamentos são denominados regras de composição RCs e podem ser inclusivas ou

exclusivas. Já as regras de adaptação RAs definem situações de contexto que podem levar à adaptações dos produtos de um MMSC. Portanto, as RAs associam os dois modelos (MCS e MCC), de modo a guiar a adaptação dos produtos gerados, uma vez que as mesmas definem as políticas de adaptação, ou seja, as situações de contexto que podem levar à adaptações dos produtos da LPMSC.

É importante destacar que o MCC proposto nesta tese de doutorado é independente do domínio de aplicações sensíveis ao contexto e utiliza a mesma estrutura de um MCS. Contudo, devido à dificuldade em encontrar LPSs sensíveis ao contexto, uma LPMSC é usada como estudo de caso.

A LPMSC usada como exemplo é resultado do projeto MobiLine[1]. Essa LPMSC produz Guias de Visita Móveis e Sensíveis ao Contexto que são executados no dispositivo móvel dos visitantes e fornecem informações sobre o ambiente, as pessoas ao redor e ambientes que são visitados. As funcionalidades desses guias podem ser adaptadas de acordo com o contexto atual do visitante, incluindo a localização deste, o seu perfil ou preferências, as características de seu dispositivo móvel e informações sobre outras pessoas presentes na mesma sala.

Para definir o escopo da LPMSC, um conjunto de aplicações que se encaixam neste domínio foram analisadas e as informações consideradas relevantes foram organizadas usando um MCS. Alguns requisitos identificados foram *Ambiente de Execução Dinâmica*, *Adaptabilidade* e *Sensibilidade ao Contexto*. Características derivadas da natureza móvel e distribuída dessas aplicações, tais como *Troca de Mensagens*, *Descrição do Servico* e *Descoberta do Servico*, também foram incluídas [95]. Devido à restrição de espaço, a Figura 2.2 mostra apenas parte do MCS construído no projeto MobiLine.

O Guia de Visita Móvel e Sensível ao Contexto pode mostrar os mapas do ambiente e a compatibilidade do perfil do visitante. Além disso, gerencia o contexto, controla a troca de mensagens e exibe o perfil do ambiente em que o visitante se encontra. A persistência, assim como o vídeo e a imagem, possuem atributos que definem, respectivamente, a quantidade de memória disponível para armazenar as informações do contexto e a resolução das imagens e vídeos. A Figura 2.2 também ilustra três regras de composição.

[1]O projeto MobiLine foi fomentado pelo CNPq (MCT/CNPq 15/2007 Universal - processo número 484523/2007-4). Esse projeto foi desenvolvido pelo Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat) [69] da Universidade Federal do Ceará em parceria com o Grupo de Reutilização da COPPE - Universidade Federal do Rio de Janeiro.

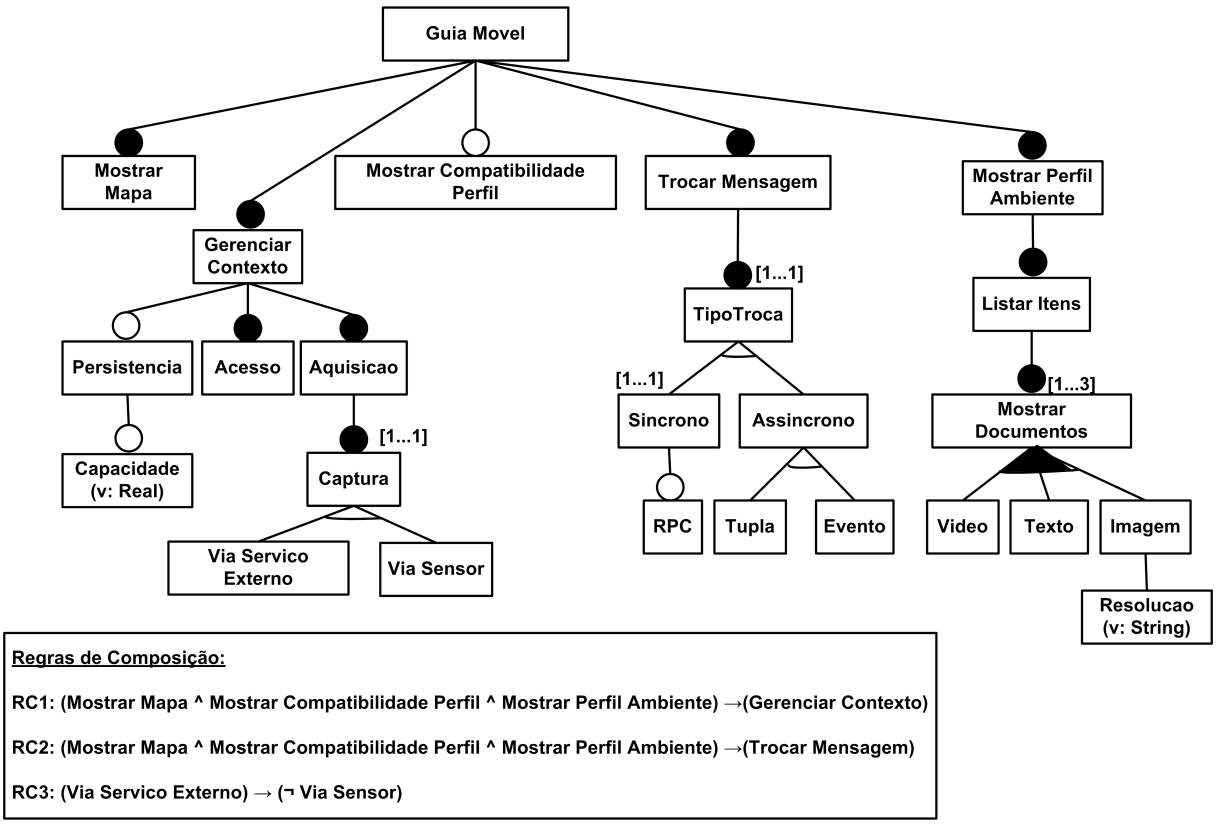

Figura 2.2: Parte do MCS da LPMSC do projeto MobiLine.

A Figura 2.3 ilustra parte do MCC desenvolvido no projeto MobiLine. De acordo com a Figura 2.3, as entidades de contexto são *Dispositivo* e *Rede*. A entidade de contexto *Dispositivo* possui como informações de contexto *Display*, *Bateria*, *Bibliotecas* e *Memoria*. A entidade de contexto *Rede* possui como informações de contexto *Latencia* e *Tipo*. Além disso, cada uma das informações de contexto armazena valores para os atributos *valor*, *origem*, *alteracao* e *qualidade*. Por exemplo, a informação de contexto *Memoria* possui alta qualidade, sofre alterações frequentemente e é obtida a partir de sensores.

2.5.5 Linha de produtos de software dinâmica

O ciclo de vida de um produto em uma LPS tradicional possui o objetivo de produzir aplicações que compartilhem um conjunto comum de artefatos em um domínio específico em que a adaptação dos produtos é realizada durante o desenvolvimento da LPS. Porém, a crescente necessidade de adaptação dinâmica do comportamento das aplicações em tempo de execução tem exigido o desenvolvimento de aplicações capazes de adaptar suas propriedades e recursos em tempo de execução [40], [87].

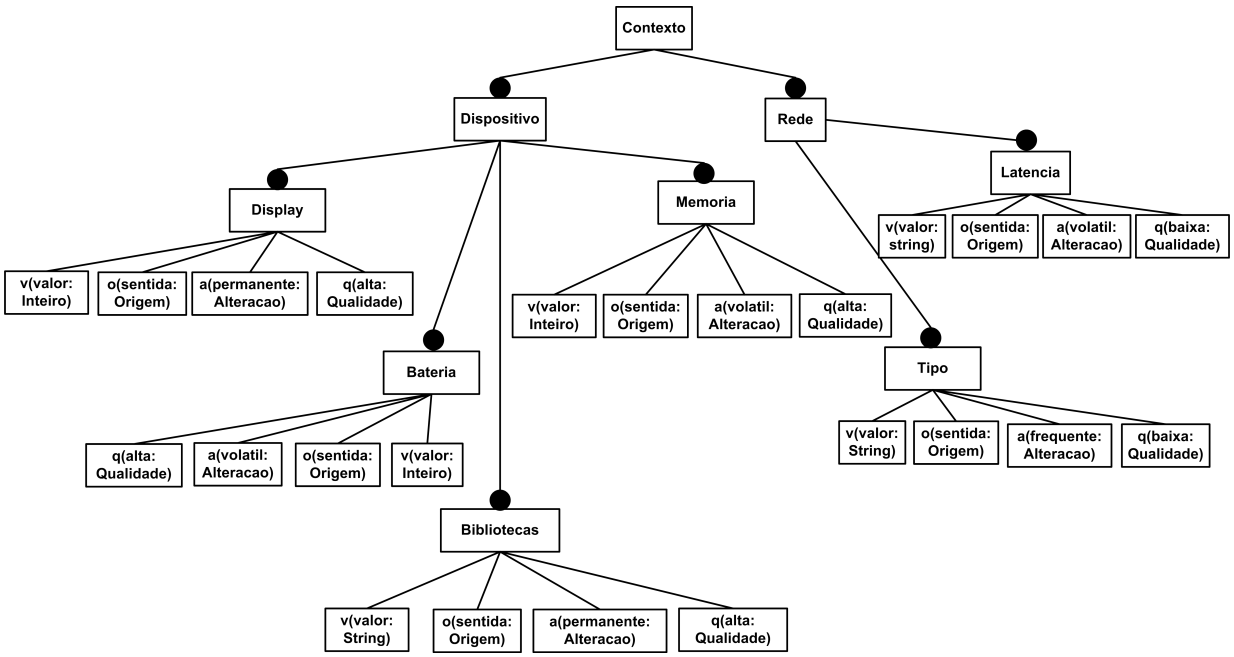

Figura 2.3: Parte do MCC da LPMSC do projeto MobiLine.

Nesse contexto, surgiram as *Linhas de Produtos de Software Dinâmicas* (LPSDs) [73]. Apesar de ainda não existir um consenso na comunidade de LPS com relação à definição de Linha de Produtos de Software Dinâmica (LPSD), em geral, pode-se afirmar que os produtos gerados por uma LPSD são softwares adaptativos, ou seja, os produtos podem se adaptar proativamente quando mudanças são realizadas no seu ambiente.

As atividades essenciais das LPSDs são monitorar a situação atual do ambiente e dos recursos computacionais e controlar a adaptação dos produtos. Segundo Hallsteinsen *et al.* [73], algumas características específicas das LPSDs podem ser destacadas:

- configuração e associação dos pontos de variação em tempo de execução;
- associações modificadas várias vezes durante o tempo de vida da aplicação;
- adição de pontos de variação ou extensão de um ponto de variação;
- constante necessidade de tratar mudanças;
- sensibilidade ao contexto; e
- tomada de decisão automática.

As LPSDs têm atraído a atenção de vários pesquisadores de LPSs. Apesar de possuírem a característica de sensibilidade ao contexto, o escopo das LPSDs é mais

abrangente que o escopo das LPMSCs discutidas nesta tese de doutorado, em que as mudanças no contexto são tratadas em tempo de projeto. As LPSDs possuem interseções com outras áreas, como sistemas auto-adaptativos, sistemas auto-gerenciáveis, sistemas *self-healing*, dentre outros, o que as torna capazes de identificar automaticamente mudanças de contexto no ambiente de execução e realizar a adaptação necessária, sem que as situações que levem a essas adaptações tenham sido previamente definidas em tempo de desenvolvimento.

2.6 Verificação e validação

Os defeitos de um software podem ser introduzidos em qualquer instante durante o processo de desenvolvimento. Portanto, é essencial que eles sejam detectados o mais cedo possível, uma vez que o custo relativo para a sua correção é maior quanto mais tarde eles forem detectados [14]. Sendo assim, as atividades de verificação e validação ocorrem em todos os estágios do processo de desenvolvimento de um software, começando desde a revisão dos requisitos e continuando até o teste do produto para certificarem que esse produto atende à sua especificação e entrega a funcionalidade esperada pelo cliente.

Em geral, os termos **Verificação e Validação (V&V)** são utilizados como sinônimos, mas essa utilização não corresponde à realidade, pois os dois termos possuem abrangências diferentes. A verificação envolve a garantia de que o software está de acordo com suas especificações. A validação, por outro lado, deve assegurar que o sistema de software atenda às expectativas do cliente [120], [136], [14], [118].

No que tange a V&V para LPSs, este trabalho de pesquisa foca essencialmente a modelagem dos MCs. Nesse caso, a verificação corresponde ao processo de determinação de que um modelo representa uma descrição conceitual da especificação dos requisitos. A validação corresponde ao processo de determinação do grau em que um modelo é uma representação precisa do mundo real a partir da perspectiva da utilização prevista [124] e [107]. De acordo com Salinesi *et al.* [127], a dificuldade existente na verificação dos MCs resulta do fato de que a sua semântica é representada por um conjunto de MCs de produtos implícitos e qualquer erro introduzido no MC pode afetar esses produtos ou a habilidade de uma LPS em gerar produtos corretos.

A checagem da corretude e da consistência dos MCs e da corretude dos pro-

dutos de uma LPS, com relação às variabilidades e restrições especificadas, permanece desafiadora, em especial, quando a checagem ou verificação automática dessas configurações é considerada [23].

Além disso, verificar a corretude e a consistência dos MCs é uma tarefa propensa a erros e inviável para modelos de grande escala. Por esse motivo, vários trabalhos de pesquisa têm surgido nessa área de LPSs [16], [23], dentre outros. Benavides *et al.* [22] realizaram uma revisão na literatura, cujo objetivo é investigar os estudos que propõem abordagens para analisar os MCs. De acordo com essa revisão, os tipos de operações que podem ser realizadas durante a verificação dos MCs são:

Modelo nulo recebe como entrada um MC e retorna um valor informando se tal modelo é nulo ou não. Um MC é nulo se ele não representa nenhum produto.

Produto válido recebe como entrada um MC e um produto, e retorna um valor que determina se o produto pertence ao conjunto de produtos representado por um MC. Essa operação também é chamada na literatura de verificação de uma configuração válida, sistema único válido, consistência da configuração, compatibilidade de características, verificação do produto, e completude da especificação do produto.

Configuração parcial válida recebe como entrada um MC e uma configuração parcial e retorna um valor informando se a configuração é válida;

Todos os produtos recupera todos os produtos que podem ser configurados a partir de um MC. Apesar de ser considerada uma operação simples quando o número de produtos é pequeno, em geral, é inviável realizar essa operação com um número elevado de produtos. Essa operação também é referenciada na literatura como todas as configurações válidas e lista de produtos.

Calcular o número total de produtos retorna o número total de produtos de um MC e revela a flexibilidade e complexidade da LPS, pois quanto maior o número de produtos, mais flexível e mais complexa é a LPS. Essa operação também é referenciada na literatura como grau de variação.

Filtro tem como entrada um MC e uma configuração, e retorna o conjunto de produtos que pode ser derivado do modelo, incluindo a configuração de entrada.

Detecção de anomalias tem como entrada um MC e retorna as anomalias detectadas. Foram identificados três tipos principais de anomalias em MCs relatados na literatura. São eles:

1. característica morta, nunca aparece em nenhuma configuração válida de um MC.
2. característica opcional falsa, quando é incluída em todos os produtos da LPS, apesar de ser modelada como opcional.
3. cardinalidade errada, quando não pode ser instanciada.

Explicações retorna uma explicação quando um MC é considerado incorreto ou inconsistente. Em geral, as ferramentas automatizadas são capazes de responder se um MC é correto, porém, encontrar a origem do problema é um grande desafio. Fornecer explicações não é um problema trivial, pois é necessário informar a origem mínima do problema e fornecer uma explicação para a correção do mesmo.

Explicações corretivas tem como entrada um MC e uma operação de análise e retorna um conjunto de explicações corretivas indicando alterações a serem feitas nas entradas originais, a fim de alterar a saída da operação. Normalmente, uma explicação corretiva oferece sugestões para solucionar um problema.

Relações do MC tem como entrada dois MCs diferentes e retorna um valor informando como os modelos estão relacionados. O conjunto de características em ambos os modelos não é necessariamente o mesmo. Essa operação é útil para determinar como um modelo evoluiu ao longo do tempo. As possíveis relações entre os dois MCs incluem refatoração, generalização, especialização e edição arbitrária.

Características núcleo recebe um MC como entrada e retorna o conjunto de características que fazem parte de todos os produtos da LPS.

Características variantes recebe um MC como entrada e retorna o conjunto de características variantes do modelo. Características variantes são aquelas que não aparecem em todos os produtos da LPS.

Conjuntos atômicos recebe um MC como entrada e retorna os conjuntos atômicos do modelo. Um conjunto atômico é um grupo de características que podem ser tratadas como uma unidade. A ideia intuitiva por trás dos conjuntos atômicos é que as

características obrigatórias e suas características pai sempre aparecem juntas em produtos e, portanto, os mesmos podem ser agrupados sem alterar o resultado de certas operações. Uma vez que os conjuntos atômicos são calculados, podem ser usados para criar uma versão reduzida do modelo, substituindo cada característica pelo conjunto atômico que o contém.

Análise das dependências recebe um MC e uma configuração parcial como entrada e retorna uma nova configuração com as características que devem ser selecionadas ou removidas como resultado da propagação de restrições no modelo.

Normalização retorna um MC normalizado. A justificativa é que o mesmo conjunto de produtos pode ser especificado usando MCs diferentes. Isso leva à necessidade de um processo de simplificação ou normalização em que qualquer MC pode ser traduzido em um modelo canônico.

2.7 Considerações finais

Neste capítulo foram discutidos os conceitos relacionados à mobilidade, sensibilidade ao contexto e adaptabilidade, considerados relevantes para esta tese de doutorado. Dentre esses conceitos, várias definições de contexto foram apresentadas, incluindo a definição de contexto adotada nesta tese. Também foi enfatizada a importância da representação dos elementos de contexto e do impacto que esses elementos podem causar nas aplicações sensíveis ao contexto. Além disso, foram apresentados os conceitos relacionados às LPSs, bem como o conceito de variabilidade em LPS foi descrito e foram descritas e analisadas as principais notações encontradas na literatura para a modelagem da variabilidade. No entanto, o domínio considerado nesta tese de doutorado não requer apenas a modelagem da variabilidade do sistema, mas também a modelagem do contexto e das regras de adaptação que especifiquem as adaptações do sistema de acordo com as mudanças em seu contexto. Portanto, neste trabalho o modelo de características é denominado Modelo de Características Móvel e Sensível ao Contexto (MMSC) e é constituído do modelo de características do sistema MCS e do modelo de características do contexto MCC.

O MCS captura as similaridades e variabilidades de uma coleção de aplicações e possui um conjunto de regras de composição que expressam as relações de dependência e de mútua exclusividade entre os elementos desse modelo. O MCC captura as informações

contextuais do domínio das aplicações. Além disso, o MMSC possui um conjunto de RAs que correspondem às decisões de adaptação com base nas informações de contexto. As RAs são responsáveis por associarem o MCS ao MCC, pois as adaptações que são realizadas no MCS, dependem das mudanças de contexto observadas no MCC.

Por outro lado, é importante ressaltar que esta tese de doutorado tem como objetivo desenvolver LPSs confiáveis a partir da checagem da corretude e consistência dos MMSCs e da corretude e consistência das configurações e adaptações dos produtos derivados desses modelos. Portanto, nesse Capítulo, os conceitos relacionados com as atividades de V&V em MCs foram descritos. Nesse sentido, diversas operações existentes na literatura para verificar a corretude dos MCs das LPSs foram apresentadas e os métodos encontrados na literatura para dar suporte para as atividades de verificação de MCs descritos.

Capítulo 3

Trabalhos Relacionados

Neste Capítulo, os trabalhos relacionados com esta tese de doutorado são investigados e estruturados em três categorias. Na primeira, os estudos que abordam a análise dos MCs são descritos. Na segunda categoria, são agrupados os trabalhos relativos à modelagem e checagem da corretude e consistência dos MMSCs, e por fim, na terceira, os estudos que focam na integridade da composição de modelos que compõem um MMSC são apresentados. Na Seção 3.2, os trabalhos da primeira categoria são discutidos. Na Seção 3.3, os trabalhos da segunda categoria são apresentados e, na Seção 3.4, os trabalhos da terceira categoria são descritos.

3.1 Introdução

Visando possibilitar a análise dos MCs, estudos na literatura incluídos na primeira categoria propõem a transformação desses MCs para notações formais. Segundo [22], as quatro propostas mais promissoras para a análise de um MC são baseadas no mapeamento dos MCs para *Satisfiability (SAT)* [53], *CSP* [30], *Binary Decision Diagram (BDD)* [37] e *Ontologia* [55].

Na Ciência da Computação, "satisfazibilidade" (SAT) [53] corresponde ao problema de estabelecer se as variáveis de uma fórmula Booleana ou Proposicional podem ser atribuídas de modo a fazer a mesma avaliar como verdadeira. Igualmente importante é a questão de determinar se não existem atribuições, o que implicaria que a função expressa pela fórmula é falsa para todas as atribuições possíveis variáveis. Nesse último caso, a função é "insatisfazível" [53].

Outros estudos propõem a transformação de um MC para CSP [30] que são problemas matemáticos definidos como um conjunto de objetos cujo estado deve satisfazer uma série de restrições ou limitações. CSP representa as entidades em um problema como um conjunto homogêneo de restrições finitas sobre variáveis, que é resolvido por métodos

de satisfação de restrição [29]. Em geral, o uso de CSP está associado a técnicas de programação restritiva. A programação restritiva é um paradigma que se refere ao uso de restrições na construção de relações entre variáveis. Consiste em especificar para uma solução que critérios (restrições) esta tem de cumprir. De uma forma geral, as restrições são implementadas como uma extensão de uma linguagem já existente. Essas operam sobre domínios específicos, sendo os mais usuais: booleanos, números inteiros e racionais, lineares, finitos e mistos (vários dos anteriores). Apesar dos vários domínios disponíveis, o mais usado é o domínio finito. Na prática, o interpretador cria inicialmente o domínio para as variáveis do problema e restringe cada domínio à medida que avalia as restrições. Ao final desse processo, obtêm-se uma ou várias soluções que satisfaçam as restrições ou, caso essas não sejam "satisfazíveis", nenhuma solução [29].

Uma estratégia diferente propõe transformar um MC para um BDD [37], que é um grafo acíclico dirigido com vértices não terminais, arcos e dois vértices terminais, o qual permite representar e manipular funções booleanas. Os vértices não terminais possuem variáveis binárias e os arcos que saem desses vértices são rotulados com 0 ou 1. Um vértice terminal é rotulado com 0 e o outro com 1, representando os valores booleanos falso e verdadeiro. Como cada vértice não terminal representa uma variável da função, para descobrir o valor atribuído a uma determinada entrada da função basta percorrer o grafo a partir da raiz até um terminal. Os valores das variáveis binárias definem um caminho no grafo que pode levar ao terminal rotulado com 1 ou ao terminal rotulado com 0. Ao longo de um caminho, se uma variável de controle vale 0, deve-se seguir o arco rotulado com 0 e, se vale 1, o arco rotulado com 1. Daí a expressão decisão binária, porque cada nó não terminal só tem dois sucessores possíveis [36].

Outra abordagem propõe um conjunto de regras para traduzir os MCs em ontologias que são processadas usando ferramentas específicas para executar a análise automática das representações dos MCs. Uma ontologia, denominada OWL-DL [55], foi definida para formalizar os MCs, checar a consistência e detectar conflitos no MC por meio de regras pré-definidas. Por exemplo, Zaid *et al.* [158] fornecem uma especificação formal para MCs baseada em ontologia. Esses autores fornecem meios para integrar os modelos, verificar a consistência e detectar conflitos. Eles propõem o uso de OWL para modelagem e o *Reasoner* Pellet [116] para a análise.

Essas soluções trazem benefícios, uma vez que existem vários *Solvers* ou *Pro-*

vadores de Teoremas[1]. Exemplos de *Provadores de Teoremas* incluem *SAT Solvers*, exclusivos para problemas especificados usando lógica proposicional (SAT), *CSP Solvers*, voltados para problemas descritos usando programação restritiva, ou *BDD Solvers*, usados para solucionar problemas especificados usando diagramas de decisão binária [65].

Os trabalhos relacionados na segunda categoria abordam o desenvolvimento de aplicações geradas a partir uma LPMSC e que permitem o acesso transparente à informação através de dispositivos diferentes e heterogêneos, visando possibilitar a **adaptação** [152]. A adaptação de software está relacionada ao processo necessário para modificar uma aplicação, incluindo a detecção de mudanças no contexto, tais como eventos e informações que podem levar à uma mudança, planejamento e execução dessas mudanças na aplicação. Em geral, a literatura classifica a adaptação de software em dois tipos: **adaptação estática** e **adaptação dinâmica**. A adaptação estática se refere à mudanças que são realizadas em tempo de desenvolvimento. A adaptação dinâmica refere-se à mudanças que ocorrem enquanto a aplicação está em execução [101].

Na terceira categoria, estão agrupadas as abordagens que relacionam as técnicas de modelagem que têm sido usadas para a construção e manutenção de MCs complexos. Essas técnicas consistem em dividir os MCs em modelos menores e mais gerenciáveis. No entanto, uma necessidade intrínseca à essa técnica está em garantir que as relações semânticas entre os elementos das diferentes visões sejam preservadas. Dessa forma, é necessário garantir a corretude e a consistência dos modelos construídos, a consistência dos relacionamentos semânticos entre esses modelos e a corretude dos produtos configurados e dos produtos adaptados devido à mudanças no contexto.

3.2 Abordagens para análise dos modelos de características

Nesta Seção, os trabalhos relacionados com a análise dos MCs são discutidos. Na literatura são propostas diversas abordagens para analisar se os MCs estão corretos e consistentes com os requisitos de variabilidades de uma LPS. Algumas abordagens são baseadas em

[1]Os *Solvers* ou Provadores de Teoremas analisam automaticamente as notações formais. Um *Solver* é um termo genérico para um software matemático que resolve um problema matemático, tomando descrições de problemas em algum tipo de forma genérica e calculando a sua solução. Em um *Solver*, a ênfase está na criação de um programa ou biblioteca que pode ser facilmente aplicado a outros problemas do mesmo tipo.

provas matemáticas, outras utilizam elementos formais menos rigorosos.

3.2.1 Abordagens baseadas em lógica proposicional

Em lógica e matemática, a lógica proposicional corresponde a um sistema formal no qual as fórmulas representam proposições que podem ser formadas pela combinação de proposições atômicas usando conectivos lógicos e um sistema de regras de derivação que permite que certas fórmulas sejam estabelecidas como teoremas do sistema formal [1]. As fórmulas proposicionais foram introduzidas no domínio das *LPSs* por Mannion [96] que usou os MCs como modelo de requisitos, definiu as regras para traduzir tais modelos em fórmulas proposicionais e forneceu algumas operações para a sua análise automática. A ideia principal é que as variabilidades e as similaridades definidas no MC sejam traduzidas em uma fórmula proposicional, onde os termos representam as características. O modelo correspondente é correto e consistente se, e somente se, a fórmula proposicional é avaliada como "satisfazível".

Essa solução também foi adotada por Batory [16], que estabeleceu uma conexão entre MCs, gramáticas e fórmulas proposicionais, possibilitando a construção de compiladores e linguagens de domínio específico. Batory possibilitou que *SAT Solvers* identificassem predicados contraditórios ou inconsistentes em um modelo e verificassem se uma determinada combinação de características define um produto válido. Além disso, Batory explica detalhadamente como verificar se um MC é inconsistente e se uma configuração é válida para esse modelo.

Já Zhang *et al.* [160] sugeriram o uso de uma ferramenta baseada no *Sistema SVM* [141]. Em sua proposta, Zhang *et al.* conseguiram identificar características mortas em MCs. Czarnecki *et al.* [48] também utilizaram gramáticas livres de contexto para analisar os MCs.

Sun *et al.* [139] propuseram uma formalização de MCs usando a linguagem Z [137] e a ferramenta *Alloy Analyzer* [82] para o suporte automatizado da análise dos MCs. As maiores contribuições do trabalho de Sun *et al.* estão na identificação e no tratamento de explicações, uma operação específica cujo objetivo é retornar uma explicação quando um MC é considerado incorreto ou inconsistente.

Czarnecki e Wasowski [52] propuseram a extração de MCs a partir de fórmulas

proposicionais. Os autores caracterizam uma classe de fórmulas lógicas equivalentes para MCs e identificam estruturas lógicas correspondentes a esses elementos sintáticos. Dessa forma, enquanto vários MCs diferentes podem ser extraídos de uma única fórmula, o modelo computado expõe graficamente a estrutura lógica original, enquanto minimiza redundâncias na representação.

Elfaki *et al.* [60] propõem um método baseado em Lógica de Primeira Ordem que define e fornece um mecanismo automatizado para as seguintes operações: propagação (o mesmo que Análise das Dependências descrito no Capítulo 2), explicação, identificação de características mortas, e inconsistências entre regras de composição em um MC.

Mendonça *et al.* [102] mostram por meio de um experimento que a análise dos MCs usando *SAT Solvers* é computacionalmente viável. Em [103], os autores propõem uma técnica para MCs de larga escala. Nesse sentido, duas heurísticas foram definidas para compilar MCs usando *Solvers BDD*. Os autores validaram a técnica proposta usando um *benchmark* que contém vários MCs com até duas mil características. Os modelos utilizados por Mendonça *et al.* [103] foram obtidos da indústria, literatura, ou foram gerados automaticamente. Durante a validação, os autores usaram dez MCs automaticamente gerados com duas mil características cada. Conforme afirmado pelos autores, o processo de compilação é equivalente a verificar se o MC é "satisfazível".

Segundo Gheyi *et al.* [67], as propriedades provadas por Sun *et al.* [139] também podem ser verificadas usando a linguagem *Alloy* [82]. O principal objetivo do trabalho de Gheyi *et al.* é mostrar como a ferramenta *Alloy Analyzer* pode ser usada para verificar meta-propriedades dos MCs.

No entanto, nenhuma das abordagens descritas anteriormente é adequada para capturar toda a semântica dos MCs. Por exemplo, não fica claro como modelar dependências entre as características e dependências entre atributos. A proposta apresentada nesta pesquisa de doutorado é baseada em Lógica de Primeira Ordem, área estabelecida na comunidade científica e com forte fundamentação teórica, possibilitando especificar e analisar os MCs, incluindo as lacunas citadas anteriormente, em um nível abstrato e independente da notação adotada.

3.2.2 Abordagens baseadas em programação restritiva

O uso de CSP para a análise automática dos MCs é investigado nos trabalhos de Benavides *et al.* [21] e Trinidad *et al.* [147]. Nesses trabalhos, um MC é transformado em um CSP e, usando técnicas de programação restritiva, é possível automatizar a análise dos MCs.

Benavides *et al.* [21] propõem um modo automático para analisar propriedades dos MCs, tais como o número de configurações possíveis, enumerando todas as configurações e verificando se um MC é consistente. Os autores apresentam um mapeamento para transformar um MC estendido em um CSP. Os autores também estenderam o metamodelo das características para incluir características atributo e definiram restrições entre esses.

Trinidad *et al.* [147] sugerem uma técnica para automatizar o tratamento de erros em MCs. Os autores modelam os problemas de detecção, explicação dos erros e fornecimento das soluções. Eles consideram restrições do tipo requer e exclui, e definem um método para detectar características mortas e opcionais falsas com base na teoria do diagnóstico.

Apesar de Benavides *et al.* [21] terem identificado a necessidade de incluir atributos e relacionamentos entre os atributos nos MCs, os autores definem atributos apenas como pares atributo-valor, e não especificam os tipos de relações possíveis entre esses atributos ou como essas relações podem influenciar nas operações de análise. Outro ponto não abordado por Benavides *et al.* diz respeito ao tratamento das operações de modificação, denominadas no Capítulo 2 como *Relações do MC* e que incluem as operações de refatoração, generalização, especialização e edição arbitrária. Nesta tese de doutorado, os atributos são definidos formalmente, as relações entre os atributos são consideradas, assim como o impacto desses atributos na adaptação dos produtos sensíveis ao contexto.

Trinidad *et al.* [147] afirmam que existem lacunas na detecção das causas dos problemas e no fornecimento de soluções para esses problemas e apresentam uma solução baseada para algumas situações particulares, tais como características mortas e opcionais falsas. Contudo, a detecção da origem específica dos defeitos e o fornecimento de explicações para corrigir esses defeitos permanece um desafio para as abordagens de análise de MCs. Nesta pesquisa de doutorado, é possível detectar a origem dos defeitos, no entanto a explicação contendo sugestões para correção desses defeitos não é tratada.

3.2.3 Abordagens baseadas em lógica descritiva

Alguns estudos na literatura propõem o uso de lógica descritiva para analisar MCs. Wang *et al.* [155] foram os primeiros a propor a análise de MCs usando lógica descritiva. Os autores introduziram um conjunto de regras de mapeamento para traduzir os MCs em ontologias que são processadas usando a ferramenta *Protégé-OWL* [121].

Em Wang *et al.* [156], os autores estenderam sua proposta inicial para dar suporte para explicações por meio de uma ferramenta de depuração de OWL [55]. Uma ontologia *OWL-DL* para formalizar os MCs, checar a consistência e detectar conflitos do MC por meio de regras pré-definidas é proposta. A *OWL-DL* é uma sub-linguagem expressiva e decidível da OWL. Os autores sugerem o uso de ferramentas de *Reasoning* de lógica descritiva, tal como *RACER* [123], para executar a análise automática das representações dos modelos.

Zaid *et al.* [158] fornecem uma especificação formal para MCs baseada em ontologia. Esses autores fornecem meios para integrar os modelos, verificar a consistência e detectar conflitos com base nas regras $SWRL$ [143]. Eles propõem o uso de OWL para modelagem e o *Reasoner Pellet* [116] para a análise. Fan *et al.* [62] também propuseram traduzir MCs para lógica descritiva e usar a ferramenta $RACER$ para executar as análises.

Wang *et al.* [156] e Zaid *et al.* [158] afirmam que as abordagens propostas também devem explicar a origem do problema encontrado. Embora a abordagem de Zaid *et al.* utilize a ferramenta *Pellet* que fornece algum suporte para depuração, o formato adotado por essa ferramenta não possui uma interface amigável. Dessa forma, os autores comentam que uma interface amigável para consultar a ontologia correspondente a um MC é necessária. Já Wang *et al.* usam um depurador OWL que analisa automaticamente uma instância inconsistente. O depurador fornece dicas e sugestões para a causa do problema. A abordagem desta tese de doutorado utiliza uma ferramenta, denominada **Fixture** [45], que disponibiliza uma interface de alto nível para os usuários criarem e verificarem os MCs. Essa ferramenta possui a funcionalidade de indicar a causa dos defeitos e inconsistências identificados.

3.2.4 Outras abordagens

Para analisar a consistência de um MC, Cechticky *et al.* [39] formalizam um metamodelo para MCs. Além desses, Gomaa e Shin [68] também propõem a definição de um metamodelo que é uma extensão do metamodelo da UML [111], com o objetivo de facilitar a verificação de consistência entre as múltiplas visões de uma LPS, em que cada visão apresenta uma perspectiva diferente do software em desenvolvimento, representada pelos múltiplos diagramas da UML, como, por exemplo, diagramas de caso de uso, diagramas de classes, diagramas de estados, dentre outros. Essas duas abordagens focam apenas na análise de um MC como um todo e não abrangem a verificação dos produtos.

Czarnecki e Pietroszek [51] abordam o problema de garantir a validade dos MCs por meio de *templates* e restrições de boa formação especificadas em OCL [110]. Eles disponibilizam um procedimento de verificação automática que estabelece que dada uma configuração correta do *template* do MC, nenhuma instância mal formada do *template* será produzida. Esse trabalho de Czarnecki é o inverso do trabalho anterior ([68]), pois foca na verificação da corretude das instâncias (produtos) de um MC e não no modelo em si.

Schobbens *et al.* [132] propõem uma semântica formal para MCs baseados em cardinalidade, usando uma formalização genérica da sintaxe e da semântica desses modelos. Esse trabalho resultou na linguagem *Varied Feature Diagrams* (VFD) que é expressivamente completa, tornando-a capaz de expressar várias construções, uma vez que é independente da notação dos MCs. A notação adotada nesta tese de doutorado tomou como base a notação de Schobbens *et al.*, com o objetivo de agregar o aspecto genérico da VFD. No entanto, nesta pesquisa, a notação proposta é enriquecida com construtores novos, tais como atributos e os elementos necessários para a modelagem do contexto.

Ziadi *et al.* [145] usam um Perfil UML em conjunto com restrições OCL para assegurar as restrições estruturais de um MC. Um Perfil UML fornece um mecanismo de extensão genérica para customizar modelos UML para domínios específicos. Esse trabalho se assemelha à abordagem usada para validar esta pesquisa de doutorado, pois um Perfil UML com restrições OCL é usado para auxiliar na criação da ferramenta Fixture usada durante a validação. No entanto, a diferença entre o trabalho de Ziadi *et al.* da abordagem adotada nesta tese está no fato de que o Perfil UML elaborado corresponde a uma

pequena parte da validação realizada, uma vez que também são adotadas as seguintes estratégias: fórmulas proposicionais e BDD para verificar a consistência dos MMSCs, linguagem Prolog para testar a verificação formal dos produtos derivados a partir das LPSs, e técnicas de simulação de cenários de contexto que pré-definem situações que podem levar a adaptações nos produtos.

Já Heidenreich [76] foca no desenvolvimento de Glslpss baseado em *Model-Driven Engineering (MDE)* [113] e expressa os MCs em termos de metamodelos. A MDE é uma metodologia de desenvolvimento de software que foca na criação e exploração de modelos de domínio. O autor também usa a linguagem OCL para garantir a boa formação dos MCs e das suas configurações.

3.3 Abordagens para adaptação das LPSs móveis e sensíveis ao contexto

Segundo Viana *et al.* [153], as aplicações podem ser adaptáveis ou adaptativas. Quando o usuário intervém na adaptação de uma aplicação, o termo **adaptável** é utilizado, enquanto o termo **adaptativo** é usado quando o aplicativo por si só toma a decisão de adaptar-se, sem qualquer intervenção do usuário. As abordagens usadas para a adaptação usam a variabilidade, assim como as restrições dos MCs para a derivação dos produtos. As abordagens **adaptativas** são consideradas nesta tese de doutorado.

No Capítulo 2, as abordagens de adaptação são classificadas de acordo com o **objeto de adaptação** (o que deve ser adaptado), **o mecanismo utilizado** (como a adaptação é realizada) e **o momento da execução** (quando o mecanismo é iniciado) [153]. O objeto de adaptação refere-se à granularidade da adaptação que neste trabalho envolve a inclusão ou remoção de características. Os mecanismos considerados para definir e implementar a adaptação incluem transformações de modelos, composição de modelos e variabilidade. Já o momento da execução, que define se a adaptação é realizada em tempo de desenvolvimento (estática) ou em tempo de execução (dinâmica), nesta tese de doutorado é realizado em tempo de desenvolvimento, portanto, a adaptação estática é considerada.

Na literatura podem ser encontrados diversos estudos para o desenvolvimento de aplicações adaptativas, como pode ser verificado em [108], [95] e [152]. Por exemplo,

Borges *et al.* [108] explicam como os avanços das redes de sensores sem fio podem ser úteis para ajudar no desenvolvimento e na criação de ambientes inteligentes. Por outro lado, Maia *et al.* [95], destacam as condições e desafios para construir um *middleware* orientado a serviço que seja interoperável e sensível ao contexto. Neste trabalho de pesquisa, a corretude e consistência dos modelos é verificado por meio da conjunção das regras de composição com as regras de adaptação.

3.3.1 Abordagens para adaptação estática

Perrouin *et al.* [117] propõem uma abordagem baseada na derivação de produtos de uma LPS em tempo de desenvolvimento. Os autores iniciam com um MC e, para cada característica, é criado um modelo parcial. Uma operação de composição é realizada para combinar os modelos parciais das várias características selecionadas para uma configuração particular. A adaptação resultante corresponde aos modelos, uma vez que a composição das diferentes características resulta em um modelo composto. Perrouin *et al.* combinam variabilidade, usada para definir o MC com MDE [113], usada para definir os modelos e a operação de composição. O trabalho de Perrouin *et al.* possui uma complexidade adicional com relação à abordagem apresentada nesta tese de doutorado, pois um modelo é elaborado para cada característica e a composição desses vários modelos precisa preservar as restrições e a semântica dos modelos individuais.

Hartmann e Trew [75] introduziram o modelo de variabilidade do contexto para representar informações de contexto de produtos de software. A combinação do modelo de variabilidade do contexto com o MCs resulta na chamada *Linha de Produtos de Software Múltipla (LPSM)*. O modelo de variabilidade do contexto restringe o MC, tornando possível modelar múltiplas LPSs que suportam várias dimensões do contexto. O conceito de modelo de variabilidade do contexto difere do MC do contexto proposto nesta tese de doutorado, pois para dar suporte às variações de contexto, em vez de suportar uma única LPS que permita a configuração de vários produtos, Hartmann e Trew adotam a criação de diversas LPSs.

Em [63], Fernandes *et al.* propõem a abordagem *UbiFEX* que visa tratar a análise de característica para LPSs sensíveis ao contexto. Os autores usam MCs para representar o contexto e o sistema. Regras de contexto especificadas pelo Engenheiro de Software capturam as situações de contexto que podem levar à adaptações nos produtos de

software. Os autores afirmam que um dos desafios para a construção de uma LPS sensível ao contexto está em desenvolver mecanismos para incorporar informações de contexto e adaptação em um MC. Sendo assim, essa abordagem propõe a *UbiFEX-Notation*, uma extensão da notação do MC para representar explicitamente informações de contexto em um MC. A *UbiFEX-Notation* proposta por Fernandes *et al.* é uma notação específica para modelagem de características.

Arboleda *et al.* [8] sugerem uma abordagem baseada em modelos de variabilidades e modelos de restrições em conjunto com *Programação Orientada a Aspectos (POA)* para derivar produtos. Todas as operações para derivar um produto ocorrem em tempo de desenvolvimento (composição de modelos e geração de código). Essa abordagem enfatiza a adaptação no nível de modelos e de código. A adaptação modifica os modelos usados para representar o produto. Além disso, uma vez que a POA é usada, o código fonte também é o alvo das modificações durante o processo de derivação. A abordagem proposta por Arboleda *et al.* define a variabilidade da LPS e usa MDE para definir modelos intermediários e POA para compor as regras de transformação de modelos. Apesar dos autores trabalharem a adaptação em tempo de desenvolvimento e partirem do nível dos modelos, a MDE é aplicada até o nível de código.

Sánchez *et al.* [129] definem uma linguagem para composição de artefatos em LPSs chamada $VLM4$. Essa linguagem pode ser usada para gerar regras de transformação de modelos e automatizam o processo de derivação em tempo de desenvolvimento. Essa abordagem tem como objetivo a criação de transformação de modelos que no final produzem um modelo que representa a configuração da LPS. Os autores usam variabilidade para definir os artefatos a serem compostos e transformações MDE que são geradas a partir da sua própria linguagem. Da mesma forma que Arboleda *et al.* [8], os autores trabalham em tempo de desenvolvimento, mas a MDE é aplicada até o nível de código.

Em [3], Acher *et al.* investigam o uso de MCs para representar o contexto e as variantes do software, assim como suas inter relações com o objetivo de configurar sistemas adaptativos com relação a um contexto específico. A modelagem de um sistema adaptativo requer a modelagem da variabilidade do sistema, dos modelos de seu ambiente e algumas regras de adaptação que especificam qual configuração do sistema adaptativo deve ser executada em cada contexto específico. Nesse trabalho, os autores representam o modelo de contexto como um MC. A ideia é modelar o sistema adaptativo e o seu ambiente como duas LPSs.

Na LPS *CANDEL* (*Context As dyNamic proDuct Line*) [84], proposta por Jaroucheh *et al.*, um gerenciador do contexto é construído com base em um modelo de contexto genérico. Os autores afirmam que as técnicas de gestão das similaridades e variabilidades das LPSs podem ser aplicadas para lidar com variabilidades do contexto para customização e adaptação do software. O MC da LPS *CANDEL* é semelhante ao MCC proposto nesta tese de doutorado, uma vez que é independente de domínio.

Ubayashi *et al.* [151] propõem um método que trata o contexto como uma LPS separada da LPS do sistema. Os autores usam métodos formais para especificar e verificar a corretude dos artefatos das duas LPSs. A abordagem proposta por Ubayashi *et al.* difere do MCC proposto nesta tese de doutorado, pois para dar suporte às variações de contexto os autores desenvolvem duas LPSs.

Em [98], Marinho *et al.* apresentam um perfil UML enriquecido com especificações OCL que define os elementos necessários para construir o MCS e o MCC. MCs são usados para construir esses dois modelos e regras de adaptação pré-definidas pelo usuário em tempo de desenvolvimento definem as mudanças de contexto que podem levar à adaptações das aplicações.

As abordagens para a adaptação estática descritas anteriormente abordam a reconfiguração dos produtos em tempo de desenvolvimento. No entanto, com exceção de [63], [151] e [98] essas abordagens não consideram nenhum tipo de verificação dos modelos usados nem a corretude dos produtos adaptados. O mecanismo de verificação proposto por Fernandes *et al.*, denominado *UBIFEX-Simulation*, tem como objetivo minimizar defeitos nos produtos reconfigurados e garantir a corretude e a consistência das regras, porém não existe nenhuma checagem da corretude ou da consistência dos MCs.

A abordagem proposta por Ubayashi *et al.* [151] é específica para o domínio de aplicações embarcadas. Os autores tratam o contexto como uma LPS separada e as especificações do contexto são construídas como artefatos dessa LPS que podem ser compartilhados entre várias LPSs. Métodos formais são adotados para verificar a corretude desses artefatos. Porém, a verificação da corretude dos produtos gerados ou dos produtos reconfigurados a partir da LPS não é tratada.

O perfil UML proposto por Marinho *et al.* [98] minimiza os defeitos de um MC sensível ao contexto e dos produtos configurados a partir desse modelo, mas não trata a consistência do modelo nem a corretude dos produtos reconfigurados.

3.3.2 Abordagens para adaptação dinâmica

Trinidad *et al.* [150] propõem mapear os MCs para modelos de componentes. Para cada característica existe um ou mais componentes que o implementam. Existe um componente adicional chamado *Configurador*, que é encarregado de criar as ligações em uma arquitetura que representa uma configuração particular. A abordagem foca no relacionamento entre as características e os componentes de software.

Bastida *et al.* [15] introduzem uma abordagem para composição de serviços sensíveis ao contexto. Os autores apresentam uma metodologia que tem como objetivo a definição de um modelo executável composto de vários serviços que representam um conjunto de variantes selecionados para vários pontos de variação. Em seguida, a composição ocorre em tempo de execução com base em regras *Evento-Condição-Ação (E-C-A)* para conectar novos serviços. Os autores usam informações de contexto definidas como uma propriedade dinâmica.

Em [25], Bencomo *et al.* propõem LPSs para sistemas adaptativos. Nessa abordagem, uma especificação completa do contexto e das mudanças suportadas é fornecida por meio de uma máquina de estados. Cada estado representa uma variação particular do sistema e as transições entre os estados definem adaptações dinâmicas que são disparadas por eventos que correspondem à mudanças de contexto. O trabalho de Bencomo *et al.* define políticas de reconfiguração na forma de regras E-C-A, onde as ações correspondem à mudanças na configuração dos componentes e eventos representam as notificações enviadas a partir do ambiente e processadas por um *motor* de contexto.

Em [105], os autores apresentam o K@RT, um *framework* para LPSs voltadas para a criação de sistemas adaptativos. Os autores adotam modelos em tempo de execução para a variabilidade dinâmica e para lidar com a explosão de estados. Em [106], um trabalho relacionado apresenta uma estratégia para adaptação dinâmica em que os modelos são mantidos em tempo de execução como parte da aplicação executada. A configuração alvo é calculada com base nas condições atuais do ambiente de execução a partir de um *script* de reconfiguração que transforma a configuração atual na configuração alvo. Esse *script* é gerado com base nas diferenças entre o modelo atual e o modelo alvo.

Parra *et al.* [115] apresentam uma Linha de Produtos de Software Dinâmica e Sensível ao Contexto (*CAPucine*) para construir aplicações orientadas a serviços e adaptá-las em tempo de execução de acordo com o contexto. A *CAPucine* é baseada em dois

processos para a derivação de um produto. O primeiro utiliza artefatos que representam as características da família de produtos. Esses artefatos, representados como modelos, são compostos e transformados para gerar o produto. O segundo processo diz respeito à adaptação dinâmica e apresenta artefatos sensíveis ao contexto que atuam em tempo de execução.

Dinkelaker *et al.* [59] propõem uma LPSD usando aspectos em tempo de execução para definir características e restrições. Esta abordagem mescla princípios da derivação de produtos das LPSs com a noção de variabilidade dinâmica. A variabilidade dinâmica é intrínseca aos sistemas adaptativos, pois estes apresentam graus de variabilidade dependentes dos seus contextos em tempo de execução. Os sistemas adaptativos descobrem os serviços dinamicamente a partir de opções que podem ser desconhecidas em tempo de desenvolvimento [24].

Nenhuma das abordagens para adaptação dinâmica analisadas anteriormente aborda a verificação da corretude ou consistência dos modelos, a corretude dos produtos configurados ou a corretude produtos reconfigurados. Como a abordagem de adaptação considerada nesta tese de doutorado é estática, não é possível analisar vantagens ou desvantagens da proposta apresentada neste trabalho com as abordagens dinâmicas supracitadas.

3.4 Abordagens para checagem da integridade na composição de visões dos modelos

As pesquisas incluídas na terceira categoria de trabalhos relacionados correspondem às abordagens para verificação da composição de visões de MCs.

Griss [71] reforça o uso de diferentes visões de um MC, pois as visões exibem níveis de detalhes diferentes dos modelos. Já Pohl *et al.* [119] afirmam que partes interessadas (*stakeholders*) diferentes percebem de forma diferente o que é variável em um MC. Considerando desenvolvedores e clientes como duas partes envolvidas, Pohl *et al.* introduzem os conceitos de variabilidade externa e variabilidade interna. A primeira é visível para o cliente, enquanto a segunda é oculta.

Outras pesquisas nessa área [56] abordam transformações de modelos entre

diferentes visões, onde as visões são representadas por diferentes tipos de modelos (e.g., máquinas de estados, diagramas de classes, dentre outros). Por exemplo, Segura *et al.* [133] usam transformações gráficas e formalismos para automatizar a composição de MCs. Eles apresentam um catálogo de regras independentes de tecnologia para descrever como mesclar os modelos que podem incluir atributos e regras de composição.

Hubaux *et al.* [80] apresentam uma abordagem para especificar visões em MCs, em que as visões correspondem à conjuntos de características. Eles identificam problemas e desafios chave quando diferentes *stakeholders* participam da seleção das características de uma determinada LPS a partir de uma visão parcial do sistema. Dentre esses problemas, destacam-se: como especificar as visões, como checar a cobertura de uma visão e como tratar dependências associadas à características pertencentes a diferentes visões.

Em [4], Acher *et al.* introduzem uma técnica para compor MCs a partir de modelos menores. De acordo com os autores, os MCs podem ser separados e compostos com o objetivo de facilitar o gerenciamento da complexidade. No entanto, as propriedades relevantes dos modelos originais devem ser preservadas no momento da separação e da composição.

Acher *et al.* [4] afirmam que em seu trabalho as restrições existentes nos modelos que compõem um MC não foram consideradas, pois o grau de dificuldade para garantir a integridade da composição das visões aumenta na presença dessas restrições.

Lopez-Herrejon e Egyed [94] apresentam o $C2MV2$, um projeto em andamento cujo objetivo é aplicar e estender pesquisas sobre gerenciamento de consistência incremental em LPSs que são desenvolvidas em abordagens composicionais. Os autores pretendem incluir restrições de vários modelos e assegurar a consistência entre esses modelos.

Em [5], Acher *et al.* propõem uma nova técnica para a partição de MCs. Nesse trabalho, MCs são relacionados semanticamente à lógica proposicional. Apesar dos autores considerarem as restrições existentes nos modelos, eles não abordam sensibilidade ao contexto ou a implicação dessas restrições na configuração e adaptação dos produtos de uma LPS.

O trabalho de Lopez-Herrejon e Egyed [94] apresenta um avanço nesse sentido, pois os autores afirmam que sua abordagem para composição de modelos considera diversos tipos de modelos e restrições existentes entre esses.

3.5 Considerações finais

Neste capítulo, a razão pela qual esta tese foi desenvolvida é justificada. Os problemas envolvidos na análise dos MCs foram analisados e foi concluído que nenhuma das propostas anteriores encontradas na literatura consegue resolver todos os problemas existentes e referenciados. Esse fato mostra que essa tese de doutorado é original e contribui para o avanço do estado da arte pesquisado.

Nesse sentido, foram apresentados os trabalhos relacionados com esta tese de doutorado. Inicialmente, os estudos que abordam a análise dos MCs foram descritos. Em seguida, os trabalhos relativos à modelagem e checagem da corretude e consistência dos MMSCs são apresentados, e por fim, os estudos que focam na integridade da composição de modelos que compõem um MMSC são apresentados.

O Quadro 3.1 mostra a relação entre as operações de verificação dos MCs, descritas no Capítulo 2 e tratadas pelas abordagens de análise dos MCs. Além disso, a análise realizada especifica qual a notação usada por cada abordagem (**MC** - Modelo de Características, **BC** - Baseado em cardinalidade e **OVM** - Modelos de variabilidade ortogonal), se a abordagem suporta MCs estendido, e se a mesma é descrita formalmente. Uma marcação (•) no quadro indica se a abordagem trata a operação apresentada.

A análise das abordagens descritas anteriormente mostra que a verificação dos MCs não é tratada de maneira uniforme na literatura. De acordo com o Quadro 3.1, pode-se observar que algumas operações permanecem desafios para os pesquisadores, tais como *Explicação corretiva*, *Características variantes*, *Conjuntos atômicos* e *Análise das dependências*. Por outro lado, as operações de verificação que são tratadas por praticamente todos os métodos de análise correspondem às operações modelo vazio e produto válido. O tratamento das demais operações varia com relação aos métodos de análise.

Com relação ao tipo, é possível verificar que as abordagens baseadas em lógica proposicional possuem uma maior cobertura de operações, enquanto as abordagens baseadas em lógica descritiva possuem um menor escopo. Outro ponto observado diz respeito à notação utilizada. A maioria das abordagens é especificada formalmente (onze das vinte analisadas), usa uma notação baseada no MC proposto por [86] (treze das vinte analisadas) e poucas adotam a semântica do MC estendido (apenas seis das vinte analisadas).

A proposta desta tese de doutorado, denominada PRECISE, cobre nove das

dezessete operações consideradas, tendo, portanto, a maior cobertura das operações de verificação entre as abordagens analisadas, além de ser a única abordagem a tratar todos os tipos de anomalias (*características mortas*, *opcional falsa* e *cardinalidade errada*). Outros pontos importantes são: (i) notação descrita formalmente, viabilizando uma modelagem sem ambiguidades; e (ii) a notação utilizada é a dos MCs estendidos, mais ricos semanticamente.

Outro aspecto a ser observado é que todos os métodos de análise baseados em metamodelos cobrem exatamente o mesmo conjunto de operações: modelo vazio, produto válido e cardinalidade errada.

No Quadro 3.2 são apresentadas e analisadas as abordagens de adaptação descritas com relação aos critérios escopo de adaptação (modelo, arquitetura e código) e mecanismo de adaptação (MDE, variabilidade e sensibilidade ao contexto). Uma marcação (•) indica se a abordagem propõe soluções para os diferentes critérios avaliados.

A primeira categoria de abordagens analisadas abrange a adaptação estática (em tempo de desenvolvimento). A maioria delas inclui um processo de derivação completo por meio da definição da variabilidade nos primeiros estágios do desenvolvimento. Essas abordagens combinam gerenciamento da variabilidade com técnicas de desenvolvimento de software, tal como MDE nos casos em que ocorrem transformações entre modelos. No entanto, devido à falta de suporte para as adaptações dinâmicas, a configuração é pré-definida para cada produto em tempo de desenvolvimento e as abordagens não são capazes de criarem novas configurações quando o produto é executado.

Na segunda categoria, são apresentadas abordagens que oferecem suporte para as adaptações dinâmicas. Em geral, são baseadas em plataformas que são utilizadas para modificar as aplicações em tempo de execução. Algumas das abordagens usam informações de contexto e regras de adaptação para disparar adaptações sempre que ocorre um determinado contexto. O ponto de partida dessas abordagens é geralmente um conjunto de aplicações já desenvolvidas. Portanto, elas precisam ser complementadas para oferecer apoio ao processo de desenvolvimento inicial que ocorre antes da execução das aplicações.

A proposta apresentada nesta tese de doutorado considera as fases iniciais do processo de desenvolvimento, fornecendo suporte para a construção e verificação dos modelos, os quais são o MCS e o MCC. É importante ressaltar que o MCC é genérico e independente de domínio de aplicações sensíveis ao contexto. Além disso, as funcionali-

dades de definição das variabilidades e configuração dos produtos também são oferecidas. As adaptações ocorrem em tempo de desenvolvimento, com base em regras de adaptação que definem as situações de contexto relevantes para o domínio considerado.

Por fim, no Quadro 3.3 são apresentadas as abordagens para checagem da integridade na composição de visões descritas com relação aos critérios restrição (regras de composição e restrições do modelo) e sensibilidade ao contexto. Uma marcação (•) indica se a abordagem propõe soluções para os critérios avaliados.

De acordo com o Quadro 3.3, apenas os trabalhos mais recentes passaram a considerar as restrições existentes entre os modelos, pois garantir a integridade da composição dos modelos torna-se mais complexa quando essas restrições são consideradas. Por exemplo, as abordagens [56] e [4] não dão suporte para a composição das regras de composição nem para as restrições existentes nos modelos que compõem um MC. Além disso, nenhuma das abordagens analisadas considera a sensibilidade ao contexto na composição dos modelos.

Na proposta desta tese de doutorado, são consideradas composições entre regras de composição, entre regras de adaptação, e entre regras de adaptação e regras de composição. Além dessas, as restrições implícitas aos modelos (e.g., mandatoriedade e cardinalidade) são tratadas no momento da composição dos modelos. A sensibilidade ao contexto é obtida por meio do MCC, que composto com o MCS, determina as possíveis adaptações dos produtos de uma LPMSC.

De acordo com o resumo contido no Quadro 3.3, a única abordagem que trata os dois pontos avaliados é a proposta nesta tese de doutorado.

Quadro 3.1: Comparação entre as abordagens de análise dos MCs

Operação	**Lógica Proposicional**								**CSP**		**Lógica Descritiva**				**Outras**						**Proposta da tese**
	[96]	[16]	[160]	[48]	[139]	[60]	[102]	[67]	[21]	[149]	[155]	[62]	[156]	[158]	[39]	[68]	[50]	[51]	[132]	[145]	PRECISE
Modelo nulo	•	•	•	•	•		•		•	•	•	•	•		•				•		•
Produto válido	•	•		•	•			•			•		•			•			•	•	•
Configuração parcial válida		•		•															•		•
Todos os produtos	•	•			•			•	•												
Número de produtos	•			•					•												
Filtro				•					•												
Característica morta			•			•	•			•				•							•
Opcional falsa			•							•											•
Cardinalidade errada						•				•							•	•			•
Redundância			•			•															•
Explicação		•			•	•				•	•		•	•						•	•
Explicação corretiva																					
Relações do modelo					•			•	•	•	•		•								
Características núcleo						•	•													•	•
Características variantes																				•	
Conjuntos atômicos			•				•								•						
Análise das dependências						•															
Notação usada	MC	MC	*BC*	*BC*	MC	*OVM*	MC	MC	*BC*	MC	MC	MC	MC	MC	*BC*	MC	*BC*	*BC*	MC	MC	*BC*
Modelo estendido								•	•					•		•	•	•			•
Especificação formal			•		•	•	•	•	•	•		•					•	•	•		•

Quadro 3.2: Comparação entre abordagens para adaptação

Critérios	**Abordagens Estáticas**									**Abordagens Dinâmicas**						**Proposta da Tese**
	[117]	[75]	[63]	[8]	[129]	[3]	[84]	[151]	[98]	[150]	[15]	[25]	[105]	[115]	[59]	PRECISE
Modelo	•	•	•	•	•	•	•	•	•				•	•	•	•
Arquitetura				•						•	•	•	•	•	•	
Código				•										•	•	
MDE	•			•	•						•	•				
Variabilidade	•	•	•	•	•	•	•	•	•	•				•	•	•
Regras de adaptação			•			•		•	•	•	•					•
Sensibilidade ao contexto		•	•			•		•	•		•		•	•	•	•

Quadro 3.3: Comparação das abordagens de adaptação

Critérios	[4]	[71]	[119]	[5]	[94]	[80]	[133]	PRECISE
Restrições				•	•	•		•
Sensibilidade ao contexto								•

Capítulo 4

PRECISE

Neste Capítulo, a proposta desta tese de doutorado, denominada PRECISE - Processo de veRificação Formal para modElos de CaracterístIcas de Aplicações Móveis e Sensíveis ao ContExto, é apresentada. Este processo tem como objetivo mostrar em tempo de desenvolvimento que um MMSC possui as propriedades de corretude e consistência.

Na Seção 4.1, uma breve introdução dos requisitos e características do PRECISE são apresentadas. Na Seção 4.2, uma visão geral do PRECISE com relação à engenharia de dois ciclos de vida das LPSs é mostrada. Na Seção 4.3, as entradas, saídas e procedimentos definidos para o processo de verificação formal PRECISE são descritas.

4.1 Introdução

Neste trabalho de pesquisa de doutorado, é proposto um processo de verificação formal para MMSCs, seus produtos derivados e seus produtos adaptados. O processo de verificação formal proposto, denominado PRECISE, tem como objetivo aumentar a corretude e a consistência dos MMSCs e, consequentemente, gerar LPMSCs corretas e consistentes.

O PRECISE atende os seguintes requisitos:

- verificar a corretude e a consistência de MMSCs;
- verificar, em tempo de desenvolvimento, a corretude e a consistência das adaptações dos produtos derivados a partir do MMSC geradas por alterações de contexto;
- ser genérico o suficiente para verificar qualquer MMSC elaborado com base na notação FODA, pois, conforme mencionado no Capítulo 3, a notação adotada nesta tese de doutorado tomou como base a *Varied Feature Diagrams (VFD)* [132], que corresponde a uma generalização das abordagens para modelagem de características construídas a partir da notação FODA;
- ser independente de ferramenta ou linguagem de programação; e

- ser customizável, de modo a atender as necessidades específicas dos usuários.

Conforme discutido no Capítulo 1, os MCs foram escolhidos como base para o processo de verificação formal por serem construídos nos primeiros estágios da Engenharia das LPSs e, portanto, defeitos incluídos nesses modelos podem se propagar para os demais artefatos da LPS.

Conforme mencionado no Capítulo 2, no contexto desse trabalho, os MMSCs são compostos por dois modelos: MCS e MCC. Essa separação introduz uma complexidade extra para a verificação da corretude e consistência desses modelos, pois é necessário definir uma estratégia para compor esses modelos e as restrições existentes entre eles, e checar a consistência destas composições. No entanto, essa separação do MMSC em dois modelos é necessária, uma vez que cada modelo representa um tipo de informação diferente. Além disso, a junção desses dois modelos dificultaria a visualização do MMSC como um todo.

A revisão da literatura apresentada no Capítulo 3 mostra que pesquisas existentes têm se concentrado em garantir que os MCs sejam corretos e consistentes com os requisitos de uma LPS [17], [16], [96], [22], [21].

4.2 Visão geral do processo de verificação

O modo como o PRECISE se relaciona com a Engenharia de Dois Ciclos de Vida das LPSs (**Engenharia do Domínio** e **Engenharia da Aplicação**) é ilustrada na Figura 4.1.

De acordo com a Figura 4.1, no ciclo da **Engenharia de Domínio**, o PRECISE é inserido logo após a atividade Engenharia de Requisitos, com o intuito de checar a **corretude** e a **consistência** dos MMSCs elaborados. Caso esses modelos sejam avaliados como **incorretos** ou **inconsistentes**, o fluxo retorna para a atividade Engenharia de Requisitos para que os modelos sejam corrigidos. Caso contrário, o modelo é classificado como **correto** e **consistente**, o processo de verificação formal finaliza e a próxima atividade da Engenharia de Domínio é iniciada.

Na fase de **Engenharia de Aplicação**, o PRECISE é incluído para checar a **corretude** dos produtos derivados na fase Engenharia de Requisitos. Caso o produto seja analisado como **incorreto**, o fluxo do processo retorna para a atividade Engenharia

de Requisitos para que o modelo do produto derivado seja corrigido. Caso contrário, o modelo do produto é considerado **correto**, o processo de verificação formal finaliza e a próxima atividade da Engenharia de Aplicação é iniciada.

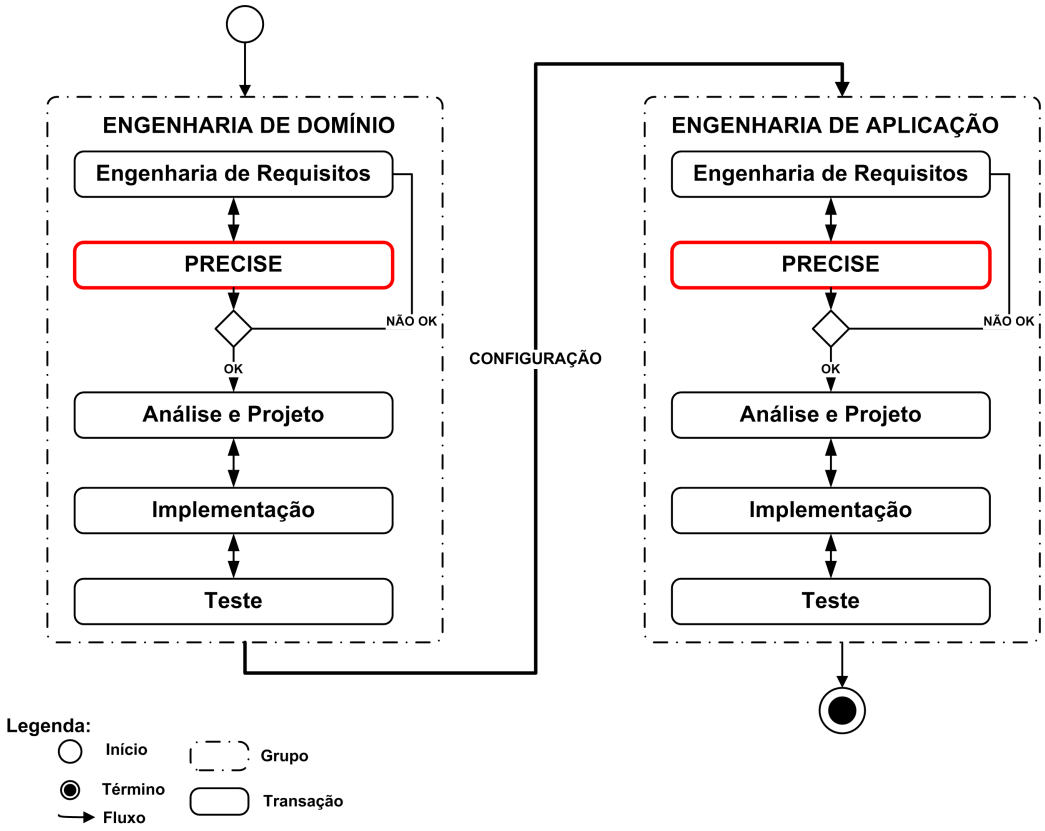

Figura 4.1: Inclusão do processo de verificação formal na engenharia das LPSs.

4.3 Processo de verificação

Em resumo, o PRECISE tem como objetivo minimizar defeitos e inconsistências dos MMSCs, bem como identificar defeitos nos produtos derivados e nos produtos adaptados devido a mudanças no contexto. Para isso, um conjunto de propriedades de boa formação e de consistência definidos posteriormente no Capítulo 6 são utilizadas. Os tipos de verificação estudados para modelos de características, que são considerados no PRECISE, incluem:

- **verificação da consistência** [28] - analisa os modelos para identificar se é possível configurar ao menos um produto a partir do modelo;
- **verificação de propriedades específicas de um domínio** [148], [102], [99], [90] - consiste em identificar propriedades indesejáveis do domínio, tais como informações

contraditórias ou informações redundantes; e

- **verificação da conformidade ou corretude** [114] - checa, através das propriedades de boa formação, se um modelo satisfaz às restrições estabelecidas na sua especificação formal e, portanto, corresponde a uma instância válida da especificação.

Na Figura 4.2 é usada a notação BPMN e apresenta uma visão geral do fluxo, entradas/saídas e relacionamentos do PRECISE que é realizado em duas partes. A primeira contém os sub-processos necessários para a execução das verificações. A segunda parte contém as transações e sub-processos relacionados ao Engenheiro de Software que utiliza o PRECISE.

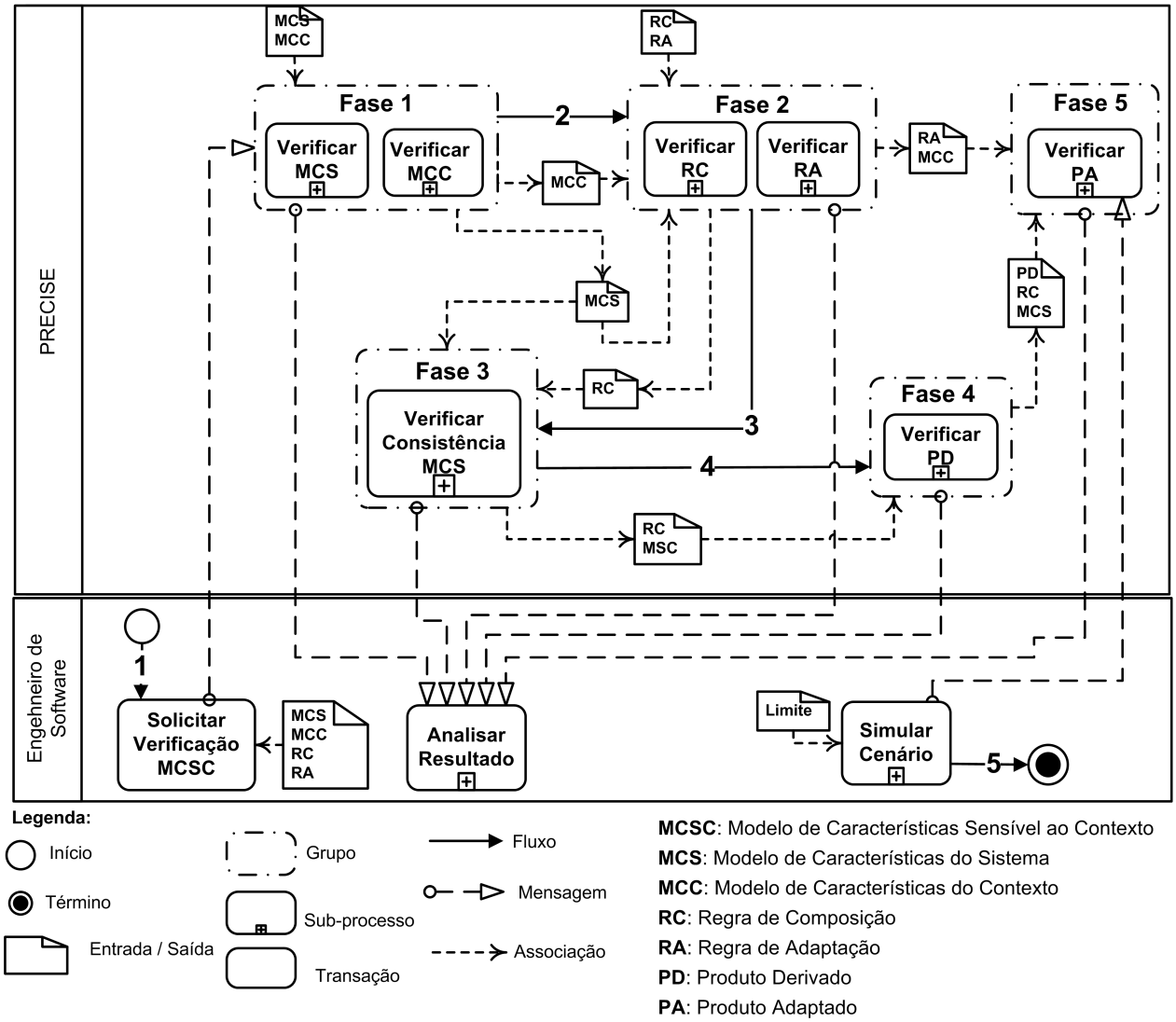

Figura 4.2: PRECISE

De acordo com a Figura 4.2, o PRECISE checa um MMSC, considerando seus seis elementos: MCS, Regra de Composição (RC), Produto Derivado (PD), MCC, Regra de Adaptação (RA), e, por fim, Produto Adaptado (PA). Os tipos de verificação realizados estão descritas a seguir: corretude do MCS, corretude do MCC, corretude das RCs, corretude das RAs, consistência do MCS, consistência entre as RCs, consistência entre as RAs, corretude dos Produtos Derivados (PDs) e corretude dos Produtos Adaptados (PAs) gerados a partir de mudanças no contexto.

O fluxo do processo inicia (**1**) quando uma solicitação é realizada por um Engenheiro de Software para a transação **Solicitar Verificação MMSC** e um MMSC é fornecido como entrada. Essa solicitação é encaminhada para a **fase 1** do processo que recebe como entrada um MCS e um MCC. A **fase 1** é composta pelo sub-processo **Verificar MCS**, que checa a corretude do MCS, e pelo sub-processo **Verificar MCC**, que checa a corretude do MCC. Caso algum modelo seja avaliado como incorreto, a origem do problema é reportada para o Engenheiro de Software por meio do sub-processo **Analisar Resultado**, que analisa o resultado de cada verificação. Caso contrário, o processo prossegue para a próxima fase. É importante ressaltar que o sub-processo **Analisar Resultado** é executado após todas as fases do PRECISE.

Após a execução da **fase 1**, o fluxo (**2**) prossegue para a execução da **fase 2**, que recebe como entrada o MCS e MCC resultantes da **fase 1**, o conjunto de RCs e o conjunto de RAs. A **fase 2** é composta pelo sub-processo **Verificar Regras** que verifica a corretude e a consistência das regras (RCs e RAs).

O fluxo do processo (**3**) prossegue para a execução da **fase 3**, que recebe como entrada o MCS e o conjunto de RCs resultantes das fases 1 e 2, respectivamente. A **fase 3** é composta pelo sub-processo **Verificar Consistência MCS**, que verifica a consistência do MCS, checando se é possível derivar ao menos um produto a partir desse MCS.

Em seguida, o fluxo (**4**) prossegue para a fase **4**, que utiliza como entrada o conjunto de RCs e o MCS, resultantes da **fase 3**. A fase **4** corresponde à verificação da corretude do PD (configurado pelo engenheiro de software) por meio do sub-processo **Verificar PD**. Por fim, na **fase 5**, é usado um processo de simulação para verificar, em tempo de desenvolvimento, a corretude dos produtos adaptados devido à mudanças no contexto. Essa fase recebe como entrada o conjunto de RCs, o conjunto de RAs, o MCC e o MCS, resultantes da **fase 2**, e o PD, resultante da **fase 4**.

A **fase 5** é composta pelo sub-processo **Verificar Produto Adaptado** que subscreve uma solicitação para a execução do sub-processo **Simular Cenários**. O sub-processo **Simular Cenários** é executado e retorna uma simulação de um cenário de adaptação. Então, o sub-processo **Verificar PA** verifica a corretude do PA de acordo com a simulação recebida. A **fase 5** é executada enquanto o limite estabelecido pelo engenheiro de software para o número de simulações não é atingido. Caso contrário, o fluxo do PRECISE é finalizado (**5**).

Apesar de existir uma sequência definida para o processo, indicada pelo fluxo de execução do PRECISE, o Engenheiro de Software não precisa necessariamente executar todas as fases sempre que utiliza o processo, pois é possível especificar quais fases devem ser executadas. Por exemplo, se apenas a corretude dos modelos iniciais e a corretude das regras precisam ser verificadas, o PRECISE permite que apenas a **fase 1** e parte da **fase 2** sejam executadas. No entanto, não é possível "pular" fases. Por exemplo, para executar a **fase 3** o Engenheiro de Software deve executar as duas fases anteriores.

4.3.1 Fase 1 - Verificar os modelos

Na fase **Verificar os modelos**, o PRECISE realiza duas verificações: corretude do MCS e corretude do MCC. Caso seja identificado algum defeito, as possíveis causas do problema são fornecidas para o Engenheiro de Software.

Verificar MCS

A Figura 4.3 apresenta o detalhamento do sub-processo **Verificar MCS** da **fase 1**. O fluxo do sub-processo inicia (**1**) com a execução da transação **Verificar Boa Formação** que recebe como entrada um MCS, retorna o resultado da verificação realizada (**2**) para o sub-processo **Analisar Resultado** e envia o MCS para a transação **Verificar Anomalias**. A transação **Verificar Anomalias** checa a presença de anomalias no MCS, retorna o resultado (**3**) para o sub-processo **Analisar Resultado** e o fluxo do sub-processo finaliza (**4**).

Entradas:

1. Modelo de características do sistema (MCS).

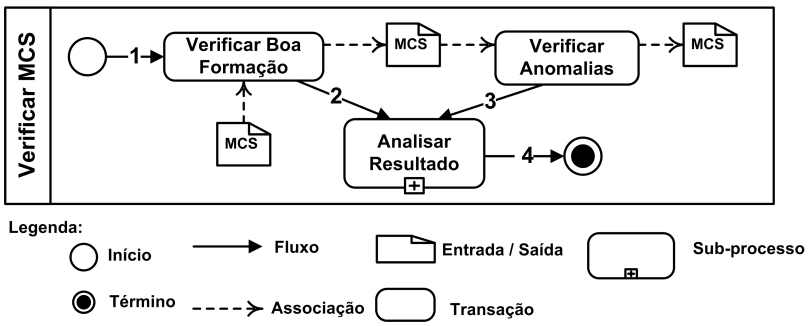

Figura 4.3: Verificação do MCS.

Procedimento:

1. Verificar o MCS com relação às propriedades de boa formação descritas na Definição 23 do Capítulo 6.

2. Analisar os resultados.

3. Verificar o MCS com relação às propriedades de presença de anomalias descritas na Seção 6.2 do Capítulo 6.

4. Analisar os resultados.

Saídas:

1. Modelo de características do sistema (MCS).

Verificar MCC

A Figura 4.4 apresenta o detalhamento do sub-processo **Verificar MCC** da **fase 1**. O fluxo do sub-processo **Verificar MCC** inicia com a execução da transação **Verificar Boa Formação** (**1**) que recebe como entrada um MCC, retorna o resultado da verificação (**2**) para o sub-processo **Analisar Resultado** e o fluxo do sub-processo finaliza (**3**). Não foi identificada na literatura nenhuma definição para anomalia em Modelos de Características do Contexto (MCCs). Por esse motivo, o detalhamento do sub-processo **Verificar MCC** não inclui a verificação de anomalias.

Entradas:

1. Modelo de características do contexto (MCC).

Procedimento:

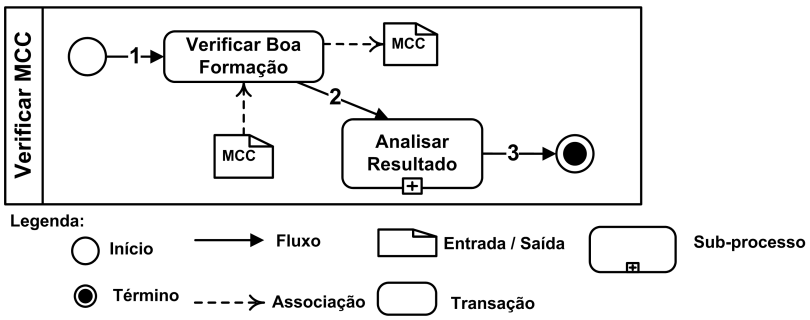

Figura 4.4: Verificação do MCC.

1. Verificar o MCC com relação às propriedades de boa formação descritas na Definição 26 do Capítulo 6.

2. Analisar os resultados.

Saídas:

1. Modelo de características do contexto (MCC).

Analisar Resultado

A Figura 4.5 apresenta o detalhamento do sub-processo **Analisar Resultado** que é executado ao final de todas as fases do PRECISE. O fluxo do sub-processo **Analisar Resultado** inicia com a execução da transação **Identificar Defeito** (**1**) que recebe como entrada o **Resultado da Verificação** realizada em cada fase, retorna a **Lista de Resultados** dos problemas identificados e o fluxo do sub-processo finaliza (**2**).

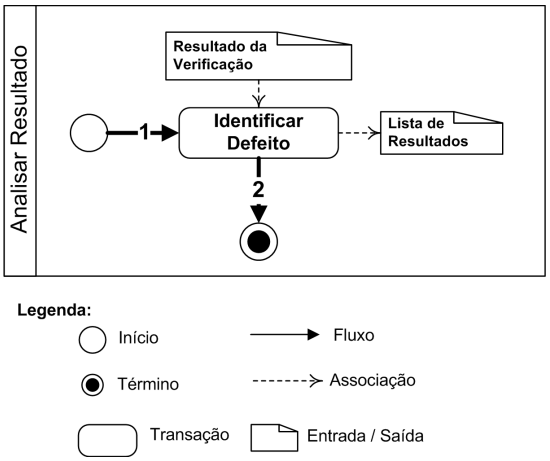

Figura 4.5: Sub-processo analisar resultado

Entradas:

1. Resultado da verificação.

Procedimento:

Se o MCS ou MCC não satisfaz a pelo menos uma das propriedades de boa formação descritas na Definição 26 (MCS Bem Formado) ou na Definição 23 (MCC Bem Formado) do Capítulo 6

Então a propriedade violada é citada como causa do defeito

Senão Se o MCS satisfaz a pelo menos uma das propriedades de presença de anomalia descritas na Seção 6.2 do Capítulo 6

Então a propriedade identificada é citada como causa do defeito

Senão Se alguma regra não satisfaz a pelo menos uma das propriedades de boa formação descritas na Definição 24 (RC Bem Formada) ou na Definição 27 (RA Bem Formada) do Capítulo 6

Então A propriedade violada é citada como origem do defeito

Senão Se alguma inconsistência entre as regras é identificada com base na Definição 34 do Capítulo 6

Então a situação que descreve o tipo da inconsistência é citada como origem da inconsistência

Senão Se a conjunção das RCs com o MCS é inconsistente, com base na Definição 36 do Capítulo 6

Então o conjunto de RCs não é satisfeito e é citado como origem da inconsistência

Saídas:

1. Próxima fase, caso nenhum problema seja identificado.

2. Lista de resultados, caso algum problema seja identificado

4.3.2 Fase 2 - Verificar as regras

Na fase **Verificar as regras**, o processo utiliza os resultados da **fase 1** para checar a corretude e a consistência das regras definidas pelo engenheiro de software. Os modelos

da **fase 1** são necessários, pois as regras utilizam elementos desses modelos. Caso algum problema seja identificado, as possíveis causas são apresentadas.

Verificar RCs

A Figura 4.6 apresenta o detalhamento do sub-processo **Verificar RC** da **fase 2**. O fluxo do sub-processo **Verificar RC** inicia com a execução da transação **Verificar Boa Formação** (**1**) que recebe como entrada o MCS resultante da **fase 1** e o conjunto de regras de composição (RCs), retorna o resultado da verificação realizada (**2**) para o sub-processo **Analisar Resultado** e envia RCs para a transação **Verificar Consistência**. A transação **Verificar Consistência** checa a presença de inconsistências entre as regras que compõem RCs, retorna o resultado (**3**) para o sub-processo **Analisar Resultado** e o fluxo do sub-processo finaliza (**4**).

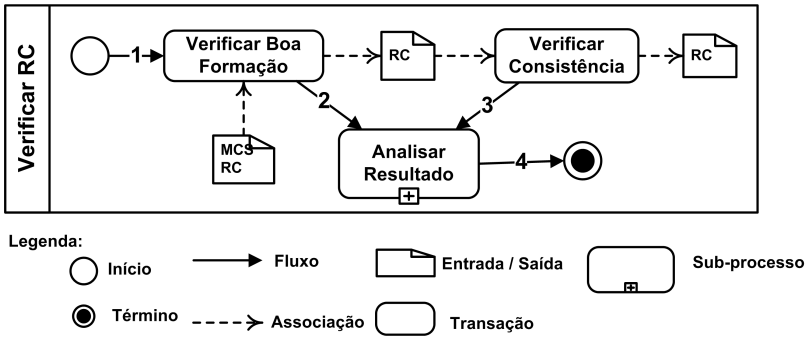

Figura 4.6: Verificação das RCs.

Entradas:

1. Modelo de características do sistema (MCS).
2. Regras de Composição (RCs).

Procedimento:

1. Verificar RCs com relação às propriedades de boa formação descritas na Definição 24 do Capítulo 6.
2. Analisar os resultados.
3. Combinar as regras pertencentes a RCs na forma de uma conjunção.
4. Verificar a conjunção resultante do passo anterior.

5. Analisar os resultados.

Saídas:

1. Regras de Composição (RCs).

Verificar RAs

A Figura 4.7 apresenta o detalhamento do sub-processo **Verificar RA** da **fase 2**. O fluxo do sub-processo **Verificar RA** inicia com a execução da transação **Verificar Boa Formação** (**1**) que recebe como entrada o MCC, o MCS e RAs. A transação **Verificar Boa Formação** retorna o resultado da verificação realizada (**2**) para o sub-processo **Analisar Resultado** e envia RAs para a transação **Verificar Consistência**, que checa a presença de inconsistências entre as regras que compõem RAs, retorna o resultado (**3**) para o sub-processo **Analisar Resultado** e o fluxo do sub-processo finaliza (**4**).

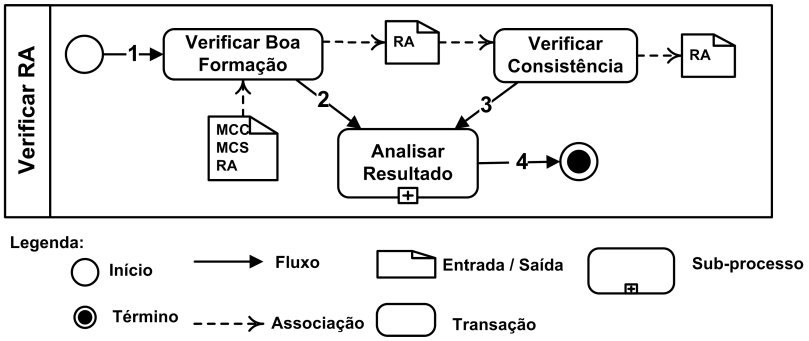

Figura 4.7: Verificação das RAs.

Entradas:

1. Modelo de características do contexto (MCC).
2. Modelo de características do sistema (MCS).
3. Regras de Adaptação (RAs).

Procedimento:

1. Verificar RAs com relação às propriedades de boa formação descritas na Definição 27 do Capítulo 6.
2. Analisar os resultados.

3. Combinar as regras pertencentes a RAs que podem ser ativadas simultaneamente na forma de uma conjunção.

4. Verificar a conjunção resultante do passo anterior.

5. Analisar os resultados.

Saídas:

1. Regras de Adaptação (RAs).

4.3.3 Fase 3 - Verificar consistência do MCS

A Figura 4.8 apresenta o detalhamento do sub-processo **Verificar Consistência MCS** da **fase 3**. O fluxo do sub-processo inicia com a execução da transação **Verificar Consistência** (**1**) que recebe como entrada o MCS, resultante da **fase 1**, e o conjunto de regras de composição RCs resultante da **fase 2**. A transação **Verificar Consistência** retorna o resultado da verificação realizada (**2**) para o sub-processo **Analisar Resultado** e o fluxo do sub-processo **Verificar Consistência MCS** finaliza (**3**).

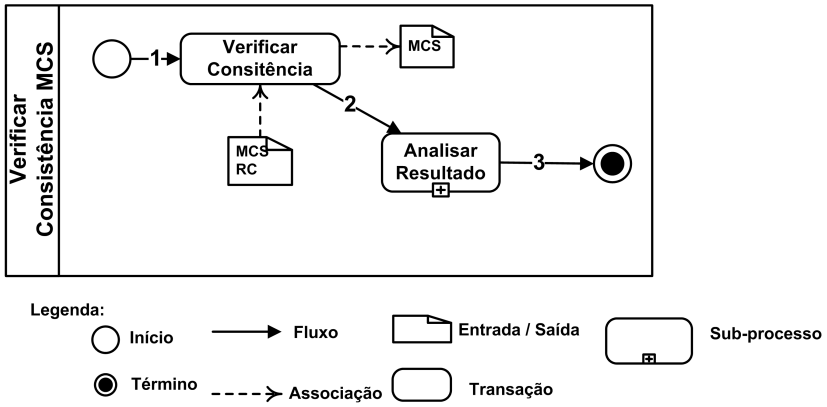

Figura 4.8: Verificação da consistência do MCS.

Entradas:

1. Modelo de características do sistema (MCS).

2. Regras de composição (RCs).

Procedimento:

1. Combinar RCs com MCS na forma de uma conjunção.

2. Verificar a consistência da conjunção construída.

3. Analisar os resultados.

Saídas:

1. Modelo de características do sistema (MCS).

4.3.4 Fase 4 - Verificar PD

Na Engenharia de Aplicação tradicional de uma LPS, todos os pontos de variação de um MCS de um PD são selecionados pelo engenheiro de software em tempo de desenvolvimento. Nesses casos, é possível checar a corretude do produto em relação às propriedades de boa formação logo após a derivação do produto.

A Figura 4.9 apresenta o detalhamento do sub-processo **Verificar PD** da **fase 4**. O fluxo do sub-processo **Verificar PD** inicia com a execução da transação **Configurar PD** (**1**) que utiliza como entrada o conjunto de RCs resultante da **fase 2** e o MCS resultante da **fase 3**, pois o PD deve satisfazer as restrições do MCS e das RCs. O sub-processo **Verificar Boa Formação** (**2**) que recebe como entrada um PD, retorna o resultado da verificação realizada (**3**) para o sub-processo **Analisar Resultado** e o PD para a transação **Verificar Anomalias**.

A transação **Verificar Anomalias** checa a presença de anomalias no PD e retorna o resultado (**3**) para o sub-processo **Analisar Resultado**. A transação **Verificar Consistência** utiliza como entrada o produto PD resultante da transação **Verificar Anomalias** e o RCs, e checa a presença de inconsistências entre PD e RCs. O resultado da transação **Verificar Consistência** é entrada para o sub-processo **Analisar Resultado** (**4**) e o fluxo do sub-processo **Verificar PD** finaliza (**5**).

Entradas:

1. Modelo de características do sistema (MCS).

2. Regras de composição (RCs).

3. Produto derivado (PD).

Procedimento:

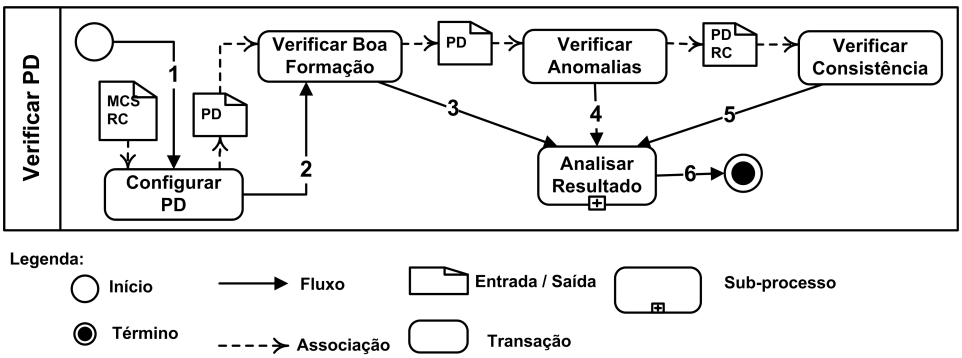

Figura 4.9: Verificação do PD.

1. Verificar o PD com relação às propriedades de boa formação descritas na Definição 25 do Capítulo 6.

2. Analisar os resultados.

3. Verificar o PD com relação às propriedades de presença de anomalias descritas na Seção 6.2 do Capítulo 6.

4. Analisar os resultados.

5. Verificar a consistência do PD com relação às RCs com base na Definição 33 apresentada no Capítulo 6.

6. Analisar os resultados.

Saídas:

1. Produto configurado (PD).

4.3.5 Fase 5 - Verificar PA

Nas aplicações móveis e sensíveis ao contexto, adaptações podem ocorrer em um produto devido às variações dos elementos do contexto. No entanto, essas variações podem causar adaptações incorretas dos produtos. Com o objetivo de detectar esses problemas, o PRECISE define um processo de simulação que identifica, em tempo de desenvolvimento, adaptações incorretas resultantes das mudanças de contexto. No entanto, para garantir uma verificação abrangente da corretude das adaptações dos produtos, implica que o PRECISE seja executado para todos os cenários que possam levar à reconfigurações do produto, o que é considerado inviável na prática, pois o número de adaptações pode crescer exponencialmente (problema conhecido como explosão de estados) [109].

Simular cenário

A Figura 4.10 apresenta o detalhamento do sub-processo **Simular Cenário** que é executado pelo engenheiro de software para verificar, em tempo de desenvolvimento, a corretude dos produtos adaptados devido a mudanças no contexto em que o produto está inserido. O fluxo do sub-processo **Simular Cenário** inicia com a execução da transação **Verificar Limite Simulações** (**1**) que recebe como entrada o número limite de simulações informado pelo Engenheiro de Software (*Limite*) e verifica se esse valor ainda não foi atingido (**2**). Caso o retorno seja positivo, a transação **Gerar Valores do Contexto** é executada (**3**) e o retorno é entrada para o sub-processo **Verificar PA**. Caso contrário, o fluxo do sub-processo **Simular Cenário** finaliza (**4**).

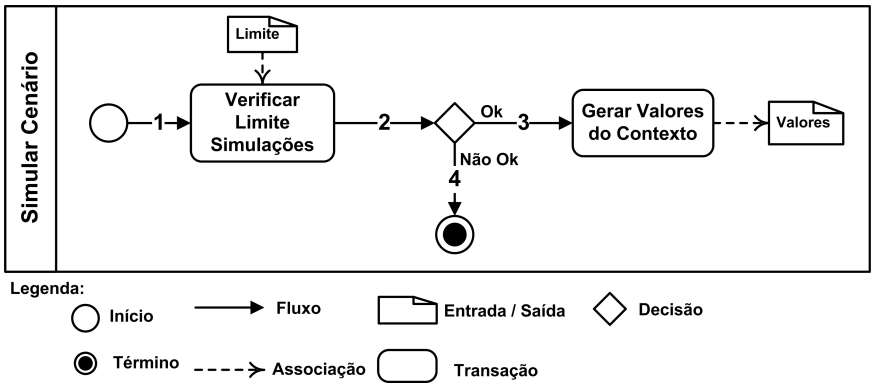

Figura 4.10: Fluxo do sub-processo simular cenário.

Entradas:

1. Número limite de simulações (*Limite*)

Procedimento:

Se o limite informado foi atingido

Então finaliza o sub-processo

Senão gera valores de contexto para o sub-processo **Verificar PA**

Saídas:

1. Valores de contexto.

Verificar o produto adaptado

A Figura 4.11 apresenta o detalhamento do sub-processo **Verificar PA** da **fase 5**. O fluxo desse sub-processo inicia com a execução da transação **Selecionar e Combinar RAs Ativadas** (**1**) que recebe como entrada o conjunto de RAs e retorna a conjunção das RAs que podem ser ativadas simultaneamente para a transação **Verificar Conjunção**.

Em seguida, a transação **Verificar Conjunção** checa se existe alguma inconsistência entre as regras recebidas, retorna o resultado para o sub-processo **Analisar Resultado** (**2**) e RAs é a entrada da transação **Verificar Produto**. Então, a transação **Verificar Produto** recebe como entrada um PD e um conjunto de RCs, checa se a adaptação do produto PD a ser realizada com base em RAs é correta, retorna o resultado (**3**) para o sub-processo **Analisar Resultado** e o fluxo do sub-processo **Verificar PA** finaliza (**4**).

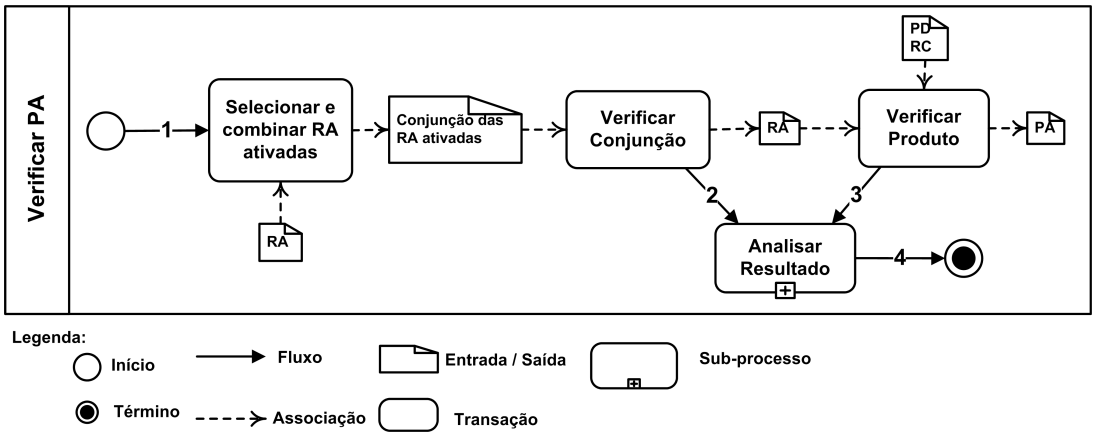

Figura 4.11: Verificação do PA.

Entradas:

1. Regras de adaptação (RAs).
2. Regras de composição (RCs).
3. Produto derivado (PD).

Procedimento:

1. Selecionar e combinar na forma de conjunção as RAs que podem ser ativadas simultaneamente.
2. Verificar inconsistências na conjunção do item anterior.

3. Analisar os resultados.

4. Verificar a consistência do PA com relação às RCs com base na Definição 36 apresentada no Capítulo 6.

5. Analisar os resultados.

6. Verificar a corretude do PA com relação às propriedades de boa formação descritas na Definição 32 do Capítulo 6.

7. Analisar os resultados.

8. Verificar a corretude do PA com relação às propriedades de presença de anomalias descritas na Seção 6.2 do Capítulo 6.

9. Analisar os resultados.

Saídas:

1. Produto adaptado (PA).

4.4 Considerações finais

A Engenharia das LPSs corresponde a uma forma de reutilização estratégica que propõe um conjunto de métodos e técnicas com o objetivo de explorar as similaridades e variabilidades de uma família de produtos de software para prover o desenvolvimento de produtos de software personalizados. Esse paradigma possui uma peculiaridade que refere-se à necessidade de um modelo para documentar as características comuns e variáveis da LPS. O modelo mais popular usado com essa finalidade é o MC.

A análise das LPS e, em particular, dos MCs é uma área de pesquisa que tem atraído a atenção da comunidade científica de LPS. A importância da análise de MCs reside no fato de que essa análise é realizada em um alto nível de abstração, permitindo a detecção de erros na fase inicial do processo de desenvolvimento da LPS. No entanto, as propostas atuais para a análise de MCs apresentam alguns problemas: (i) trabalham com um baixo nível de abstração, no sentido de que focam apenas em MCs específicos; (2) em geral, falta-lhes uma semântica formal, o que pode levar a interpretações equivocadas;

e (3) as técnicas encontradas na literatura não documentam as soluções de análise para MCs estendidos, considerados mais completos e semanticamente mais ricos.

O PRECISE apresentado neste Capítulo é um processo de verificação formal desenvolvido para verificar a corretude e a consistência de MMSCs, considerando os seguintes elementos: MCS, RCs, PD, MCC, RAs e, por fim, PA, formalmente especificados no Capítulo 5. Essa especificação formal define os elementos gráficos da notação a ser usada para expressar os modelos e o significado semântico desses elementos, usando Lógica de Primeira Ordem. Além disso, a especificação elaborada possibilita uma modelagem sem ambiguidades, bem como uma implementação correta baseada em modelos.

A verificação de cada um dos elementos de um MMSC é detalhada a partir das entradas, procedimentos e saídas necessárias e utiliza as propriedades de boa formação, consistência e presença de anomalias formalizadas no Capítulo 6. Essas propriedades definem pontos a serem analisados para minimizar a ocorrência de defeitos e inconsistências nos MMSC.

Capítulo 5

Especificação Formal de um Modelo de Características Móvel e Sensível ao Contexto

Neste Capítulo, os elementos de um MMSC são especificados formalmente usando uma abordagem baseada em Lógica de Primeira Ordem. Os conceitos formalizados nessa especificação são necessários para a implementação do PRECISE, apresentado no Capítulo 4, e para a definição das propriedades dos MMSCs, descritas no Capítulo 6.

Uma visão geral da estratégia adotada para a formalização dos MMSCs é apresentada na Seção 5.1 e, na Seção 5.2, os elementos que compõem um $MMSC$ são formalizados.

5.1 Introdução

A abordagem adotada para documentar formalmente a sintaxe e a semântica dos elementos de um MMSC é baseada na Lógica de Primeira Ordem. A Lógica de Primeira Ordem, também conhecida como Cálculo de Predicados de Primeira Ordem, é um sistema lógico que utiliza predicados e quantificadores, o que lhe confere um poder expressivo suficiente para formalizar praticamente toda a matemática [1]. Portanto, uma especificação formal descrita usando Lógica de Primeira Ordem é genérica e expressiva.

Uma especificação formal é composta por uma sintaxe e uma semântica. A sintaxe especifica a notação usada para expressar os modelos e define os elementos gráficos utilizados. Já a semântica tem o propósito de especificar o significado desses elementos. Do ponto de vista teórico, uma especificação formal é necessária para avaliar com precisão a expressividade de uma linguagem, pois possui uma maior concisão, rigor e adequação para a realização de provas matemáticas. Do ponto de vista prático, uma especificação formal é considerada um pré-requisito para a modelagem sem ambiguidades, bem como

para a implementação correta baseada em modelos [26].

A especificação formal proposta nesta tese de doutorado é o resultado de uma extensa revisão da literatura em que várias técnicas de modelagem de características e modelagem de contexto foram analisadas, a fim de extrair as contribuições de cada técnica e identificar lacunas a serem preenchidas, por exemplo [96], [49], [50], [20], [148], [160], [139], [72], [66], [51], [83], [104], [128], [63], [132], [98]. No entanto, não foram utilizadas técnicas de revisão sistemática.

Esta especificação oferece uma notação na forma de predicados que permite a formalização dos elementos e relacionamentos entre os elementos de um MMSC, bem como o significado desses elementos e relacionamentos. É importante destacar que o conceito de **característica atributo** que adiciona uma semântica extra a um MC ao permitir ao engenheiro de software incluir detalhes específicos de uma característica, não está formalizado precisamente na literatura e é abordado nesta especificação formal.

5.2 Especificação Formal de um MMSC

Conforme definido no Capítulo 1, um MMSC é composto pelo MCS e pelo MCC. Nesta Seção, a notação gráfica e a especificação formal usada para representar estes modelos são apresentadas.

5.2.1 Notação gráfica

No contexto deste trabalho, os **MCs estendidos** [22] são usados como notação padrão para elaboração inicial dos MMSCs. Essa notação é adotada por incorporar uma semântica mais rica, uma vez que inclui conceitos como cardinalidade e característica atributo.

Na Figura 5.1, a qual é descrita em detalhe a seguir, é ilustrado um MC *estendido* que representa uma pequena parte do MCS de um Guia de Visita Móvel e Sensível ao Contexto construído no projeto MobiLine, descrito no Capítulo 2. Um Guia de Visita Móvel e Sensível ao Contexto é uma aplicação criada para guiar os visitantes dentro de um ambiente (e.g., museu, shopping, laboratório, dentre outros). Este aplicativo é executado no dispositivo móvel do usuário e fornece informações do ambiente através de mapas de localização. À medida que o usuário percorre o ambiente, as informações

sobre as salas e as mídias relacionadas são exibidas. O comportamento da aplicação pode ser adaptado de acordo com o contexto atual do usuário, que pode incluir a sua localização *indoor*, seu perfil ou preferências, as características do seu dispositivo móvel e as informações sobre outras pessoas presentes no ambiente.

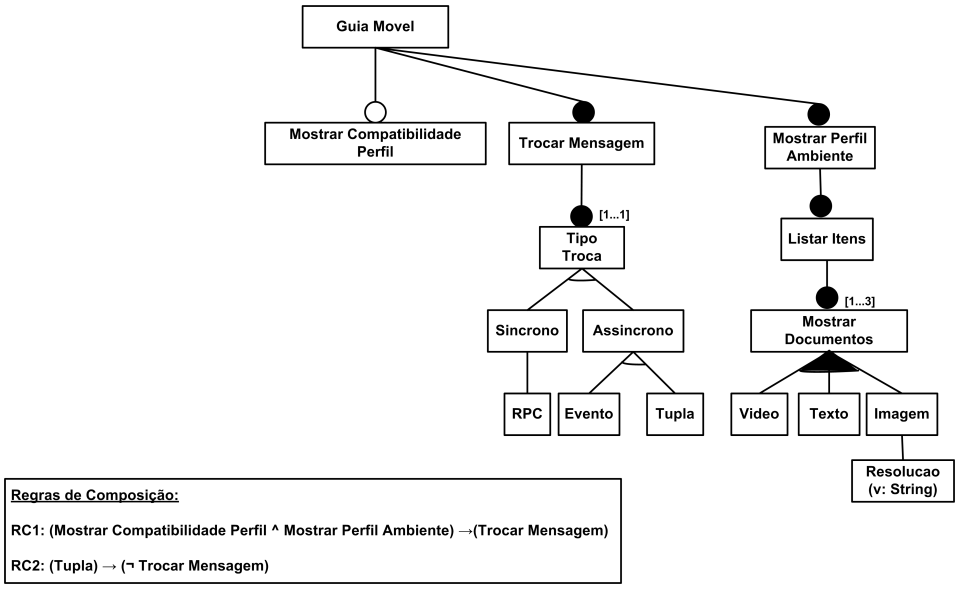

Figura 5.1: Exemplo de parte do MCS do Guia de Visita Móvel e Sensível ao Contexto.

De acordo com a Figura 5.1, as características são representadas por retângulos. Um círculo vazio corresponde a uma característica opcional, enquanto um círculo preenchido representa uma característica mandatória. Graficamente, as características do tipo alternativa (XOR) são ligadas às suas características descendentes por um triângulo não preenchido, enquanto as características do tipo opcional (OR) são ligadas às suas características descendentes por um triângulo preenchido.

O MCS do Guia de Visita Móvel e Sensível ao Contexto, apresentado na Figura 5.1, é composto pela característica opcional $Mostrar\ Compatibilidade\ Perfil$ e pelas características mandatórias $Trocar\ Mensagem$ e $Mostrar\ Perfil\ Ambiente$. A característica $Mostrar\ Compatibilidade\ Perfil$ é do tipo opcional e exibe a compatibilidade do perfil do visitante com relação ao perfil de outras pessoas que se encontrem no mesmo ambiente do visitante. A funcionalidade mandatória $Trocar\ Mensagem$ é do tipo mandatória e executa o serviço de troca de mensagens entre dispositivos. Por fim, a funcionalidade mandatória $Mostrar\ Perfil\ Ambiente$ é composta pelas características mandatórias $Listar\ Itens$ e $Mostrar\ Documentos$. A característica $Mostrar\ Documentos$ é do tipo opcional e exibe informações (video, imagem ou texto) sobre os objetos e pessoas presentes no ambiente do visitante. Uma imagem possui uma carac-

terística atributo *Resolucao*, que define a resolução usada pelo dispositivo móvel para imagens.

As RCs descrevem os relacionamentos e restrições entre as características e são expressas usando um formato textual. As RCs definidas na Figura 5.1 especificam que as características $Mostrar\ Compatibilidade\ Perfil$ e $Mostrar\ Perfil\ Ambiente$ implicam na presença da característica $Trocar\ Mensagem$. Além disso, a característica *Tupla* implica na exclusão da característica *Evento*.

Como um MCS se assemelha graficamente a uma estrutura de árvore, na qual os nós correspondem às características e as arestas descrevem as relações hierárquicas existentes entre essas características, essa estrutura foi escolhida para representar um MCS neste trabalho de pesquisa. A definição formal da estrutura de árvore adotada aqui é apresentada em [144]. Uma árvore de um MCS contém uma única raiz **r** ou conceito que representa o domínio modelado. Os nós restantes são divididos em conjuntos disjuntos não-vazios que são subárvores de **r**, denotadas por S_r.

Por exemplo, a árvore representada graficamente na Figura 5.1 possui três subárvores: $S_{MostrarCompatibilidadePerfil}$, $S_{TrocarMensagem}$ e $S_{MostrarPerfilAmbiente}$.

Se um nó $\mathbf{n}'$ pertence à subárvore do nó **n**, denominada S_n, então $\mathbf{n}'$ é descendente de **n** e **n** é chamado ancestral de $\mathbf{n}'$.

Por exemplo, o nó *Tipo Troca* é descendente do nó *Trocar Mensagem* e o nó *Mostrar Perfil Ambiente* é ancestral do nó *Listar Itens*.

No contexto deste trabalho, um MCC é definido como uma especialização de um MCS e também é representado por uma estrutura de árvore, com a restrição de possuir apenas quatro níveis, devido à estrutura convencional encontrada na literatura [138] para capturar as informações de contexto. No primeiro nível, fica a raiz, representando o contexto modelado. As características do segundo nível da hierarquia da árvore correspondem às entidades de contexto, as características do terceiro nível correspondem às informações de contexto e as folhas (características atributo no quarto nível) representam os atributos de contexto.

A Figura 5.2 representa graficamente parte do MCC correspondente ao MCS para guias de visita móveis e sensíveis ao contexto. De acordo com o MCC ilustrado, as aplicações do guia de visita são influenciadas pelas alterações no contexto da latência da

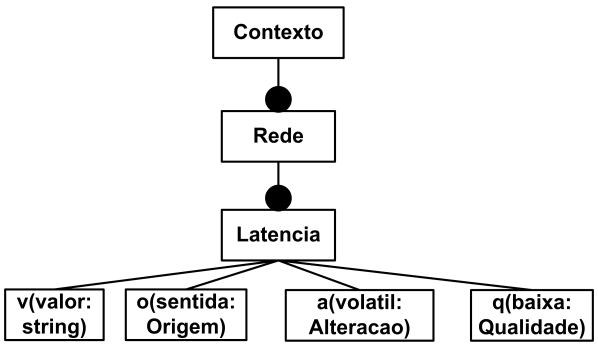

Figura 5.2: Exemplo de parte do MCC do Guia de Visita Móvel e Sensível ao Contexto.

rede, na qual essas aplicações dos dispositivos móveis são executadas. O MCC apresentado na Figura 5.2 é composto pela característica mandatória *Rede* que é composta pela característica mandatória *Latencia*. A característica *Latencia* possui as características atributo v, o, a e q, que definem os atributos de contexto *valor*, *origem*, *alteracao* e *qualidade*, respectivamente.

As RAs definem as situações de contexto que podem disparar alterações no MCS e são expressas usando um formato textual. De acordo com o MCS da Figura 5.1 e o MCC da Figura 5.2, um exemplo que ilustra uma regra de adaptação é: RA = (Rede.Latencia.q = Baixa)) $\rightarrow$ (Wifi). Esse exemplo afirma que, se a qualidade da informação de contexto latência é baixa, implica que a característica *Wifi* deve existir no MCS.

As Figuras 5.1 e 5.2 serão usadas ao longo deste Capítulo para exemplificar as definições apresentadas.

5.2.2 Formalização de um MCS

A notação dos MCs *estendidos* é apenas uma representação gráfica, portanto, é necessário definir a sintaxe e a semântica formal dos elementos contidos nesses modelos, de modo a torná-los propícios para serem submetidos para o processo de verificação formal proposto no Capítulo 4.

Garantir a qualidade dos MCs, tanto na forma como representam o domínio, como na integridade dos próprios modelos no que diz respeito aos elementos utilizados, é um desafio [99]. As linguagens para modelagem de características têm sido foco de estudos há mais de vinte anos [86]. No entanto, apenas recentemente, os pesquisadores têm investido na fundamentação semântica dessas linguagens, por exemplo [126], [51],

[132].

A semântica apresentada na especificação proposta é genérica e independente da tecnologia. Logo, pode ser reutilizada para ser implementada em qualquer linguagem de programação, possibilitando com isso o suporte automático para as atividades de verificação formal dos modelos. Nesse sentido, a ferramenta *Fixture* [45] foi implementada utilizando como base a especificação formal proposta para um MMSC deste trabalho de doutorado. Essa ferramenta foi desenvolvida no contexto de uma dissertação de mestrado do grupo de pesquisa Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat) [69].

As definições apresentadas nesta Seção correspondem à **sintaxe abstrata** e à **semântica** de um **Modelo de Características do Sistema** (MCS). Nessas definições, os elementos gráficos são descritos formalmente através de predicados, conforme [60] e [127]. O Quadro 5.1 apresenta um resumo dos predicados definidos nesta tese de doutorado, os quais estão descritos através da notação BNF a seguir.

<ancestral>::=<nome><nome>[<atributo><cardinalidade><opcional><mandatório><comum>]

<nome>::= <String>

<atributo>::= <ancestral><nome>

<atributo>::= <nome>[<domínio>]

<domínio>::= <String> | <Inteiro> | <Real> | <Enumeração> | <Booleano>

<cardinalidade>::= [<min><max>]

<min>::= <Inteiro>

<max>::= <Inteiro>

<opcional>::= <ancestral><nome>

<mandatário>::= <ancestral><nome>

<comum>::= <nome><valor>

<valor>::= <sim> | <não>

Onde <String>, <Inteiro>, <Real>, <Booleano> e <Enumeração> representam o conjunto de todos os valores string, números inteiros, números reais, valores booleanos e enumerações, respectivamente.

Por exemplo, a subárvore $S_{TipoTroca}$ mostrada na Figura 5.1 é formalizada a seguir.

mandatorio(Trocar Mensagem, Tipo Troca)

ancestral(Tipo Troca, Sincrono)

Quadro 5.1: Predicados para formalizar MMSCs

Predicado	**Descrição**
ancestral	descreve um relacionamento entre uma característica ancestral e sua respectiva característica descendente
opcional	identifica um relacionamento opcional entre uma característica descendente e sua respectiva característica ancestral
mandatorio	identifica um relacionamento mandatório entre uma característica descendente e sua respectiva característica ancestral
comum	identifica uma característica comum a todos os produtos
nome	especifica o nome da característica
min	especifica a cardinalidade mínima de uma característica
max	especifica a cardinalidade máxima de uma característica
atributo	descreve uma característica atributo
dominio	especifica os tipos dos valores das características atributo
presenteMCS	especifica se uma característica está presente no MCS
ec	especifica se uma característica é uma entidade de contexto
ic	especifica se uma característica é uma informação de contexto
ac	especifica se uma característica é um atributo de contexto

ancestral(Tipo Troca, Assincrono)

min(Tipo Troca, 1)

max(Tipo Troca, 1)

ancestral(Sincrono, Tupla) ancestral(Sincrono, Evento)

min(Sincrono, 1)

max(Sincrono, 1)

ancestral(Assincrono, Procedimento Remoto)

Definição 1. *[Característica] Uma característica* ***c*** *corresponde a um elemento da árvore e possui um* ***nome*** *que a identifica: nome(c) = nome, onde* ***nome*** *corresponde ao nome da característica.*

Por exemplo, a característica Mostrar Compatibilidade Perfil, ilustrada na Figura 5.1, representa uma característica na árvore cujo valor do nome é Mostrar Compatibilidade Perfil: nome(Mostrar Compatibilidade Perfil) = Mostrar

Compatibilidade Perfil

Chamamos **conjunto de características**, o conjunto $\mathbf{C} = \{c_1, c_2, \cdots, c_s\}$, onde cada c_i, $1 \leq i \leq |C|$, corresponde a uma característica da árvore.

Por exemplo, C = {Guia Movel, Mostrar Compatibilidade Perfil, Trocar Mensagem, Mostrar Perfil Ambiente $\cdots$, (Resolucao(v:String))} corresponde ao conjunto de características da árvore ilustrada na Figura 5.1.

Definição 2. *[Ancestral, Descendente] Seja c_1, $c_2 \in C$. Se c_1 é antecessor à característica c_2 na estrutura hierárquica da árvore, então dizemos que c_1 é **ancestral** de c_2 e c_2 é **descendente** de c_1, denotado por: ancestral(c_1, c_2).*

Por exemplo, a característica Guia Movel é ancestral da característica Trocar Mensagem na Figura 5.1, logo: ancestral(Guia Movel, Trocar Mensagem).

Se a característica **c** é ancestral do conjunto de características $\mathbf{C} = \{c_1, c_2, \cdots, c_s\}$, denotamos:

ancestral(c, c_1)

...

ancestral(c, c_s)

Por exemplo, a característica Tipo Troca na Figura 5.1 é ancestral das características Sincrono e Assincrono, logo:

ancestral(Tipo Troca, Sincrono)

ancestral(Tipo Troca, Assincrono)

Definição 3. *[Raiz] Seja $\mathbf{c} \in \boldsymbol{C}$, dizemos que a característica **c** é uma **raiz** se: $\forall\ c \in C \mid (\nexists\ c' \in C \mid ancestral(c', c))$.*

A raiz também é chamada de **conceito**, pois identifica o domínio modelado pelo conjunto de características representado na árvore.

Por exemplo, a característica Guia Movel é a raiz ou conceito do modelo ilustrado na Figura 5.1.

Definição 4. *[Característica Atributo] Seja $c_1, c_2 \in \boldsymbol{C}$. Se a característica c_1 é ancestral de c_2, e c_2 é caracterizado por três valores: **n** (nome da característica), **v** (valor do atributo) e **d** (domínio do valor do atributo), **n(v:d)**, dizemos que c_2 é uma **característica atributo** da característica c_1, logo:*

ancestral(c_1, c_2)

atributo(c_1, c_2)

dominio(c_2, d)

Por exemplo, a característica (Resolucao (String)) na Figura 5.1 é uma característica atributo descendente da característica Imagem, onde o domínio do valor da característica atributo Resolucao é String, logo:

ancestral(Imagem, Resolucao)

atributo(Imagem, Resolucao)

dominio(Resolucao, String)

O **domínio** de uma característica atributo é especificado apenas para uma configuração particular. No contexto deste trabalho, os domínios permitidos para os valores das características atributo são:

- `String`: um conjunto de caracteres delimitado por aspas simples;
- `Inteiro`: um número inteiro;
- `Real`: um número real;
- **Enumeração**: um conjunto de valores separados por vírgula; e
- `Booleano`: um valor lógico: verdadeiro (`True`) ou falso (`False`).

É possível especificar uma coleção de características atributo para uma característica **c**. Para isso, é necessário definir várias características atributo como características descendentes da característica **c**.

Chamamos **conjunto de características atributo**, o conjunto $\mathbf{C} = \{c_1(v_1{:}d_1), c_2(v_2{:}d_2), \cdots, c_s(v_s{:}d_s)\}$, onde cada $c_i(v_i{:}d_i)$, $1 \leq \mathrm{i} \leq |\mathrm{C}|$, corresponde a uma característica atributo da árvore.

Por exemplo, C = {(Resolucao (String))} corresponde ao conjunto de características atributo da árvore ilustrada na Figura 5.1.

Definição 5. *[Cardinalidade] Seja* $\boldsymbol{C = \{c_1, c_2, \cdots, c_s\}}$*, tal que* c_1 *é ancestral de* $\boldsymbol{(c_2, \cdots, c_s)}$*. Se* ***no mínimo i*** *e* ***no máximo j*** *das características descendentes de* c_1 *devem ser selecionadas, quando* c_1 *é selecionada, dizemos que a* ***cardinalidade*** *da característica* c_1 *é* ***no mínimo i*** *e* ***no máximo j****, denotada por* $card_s\boldsymbol{[i \cdots j]}$*, onde* $\boldsymbol{i \leq j}$ *e* $\boldsymbol{i, j} \in \mathbb{N}$*:*

ancestral(c_1, c_2)

...

ancestral(c_1, c_s)

min(c_1, i)

max(c_1, j)

Por exemplo, a característica Mostrar Documentos na Figura 5.1, possui cardinalidade $card_3[1 \cdots 3]$, implicando que no mínimo 1 e no máximo 3 características descendentes podem ser selecionadas, quando a característica Mostrar Documentos é selecionada.

ancestral(Mostrar Documentos, Video)

ancestral(Mostrar Documentos, Texto)

ancestral(Mostrar Documentos, Imagem)

min(Mostrar Documentos, 1)

max(Mostrar Documentos, 3)

Definição 6. *[Folha] Seja* $\boldsymbol{c} \in \boldsymbol{C}$*, dizemos que a característica* $\boldsymbol{c}$ *é uma* ***folha*** *de uma árvore se:* $\forall\ c \in C \mid (\nexists\ c' \in C \mid ancestral(c, c'))$.

Por exemplo, Tupla corresponde a uma folha do MCS apresentado graficamente na Figura 5.1, pois não existe nenhuma característica c, tal que a característica Tupla seja ancestral de c.

As folhas são associadas implicitamente a uma cardinalidade $card_0[\mathbf{0} \cdots \mathbf{0}]$.

Denominamos **conjunto de folhas**, o conjunto $\mathbf{C} = \{c_1, c_2, \cdots, c_s\}$, onde cada c_i, $\mathbf{1} \leq \mathbf{i} \leq |\mathbf{C}|$, corresponde a uma folha da árvore.

Por exemplo, $C =$ {Mostrar Compatibilidade Perfil, Tupla, Evento, Procedimento Remoto, Video, Texto, Resolucao (String)} é o conjunto de folhas da árvore ilustrada na Figura 5.1.

Definição 7. *[Característica Mandatória] Seja* c_1, $c_2 \in C$*. Se* c_1 *é ancestral de* c_2*, tal que a seleção de* c_1 *implica na seleção de* c_2*, então dizemos que* c_2 *é uma característica* ***mandatória*** *a partir da característica* c_1*, denotado por:*

ancestral(c_1, c_2)

mandatorio(c_1, c_2)

Por exemplo, a característica Tipo Troca é mandatória na Figura 5.1 a partir da característica Trocar Mensagem, logo:

ancestral(Trocar Mensagem, Tipo Troca)
mandatorio(Trocar Mensagem, Tipo Troca)

Definição 8. *[Característica Opcional] Seja c_1, $c_2 \in C$. Se c_1 é ancestral de c_2, tal que a seleção de c_1 não implica na seleção de c_2, então dizemos que a característica c_2 é* ***opcional*** *a partir da característica c_1, denotado por:*

ancestral(c_1, c_2)
opcional(c_1, c_2)

Por exemplo, a característica Mostrar Compatibilidade Perfil é opcional na Figura 5.1 a partir da característica Guia Movel, logo:

ancestral(Guia Movel, Mostrar Compatibilidade Perfil)
opcional(Guia Movel, Mostrar Compatibilidade Perfil)

Definição 9. *[Característica XOR] Seja $\boldsymbol{C = \{c_1, c_2, \cdots, c_s\}}$, tal que c_1 é ancestral de $\boldsymbol{(c_2, \cdots, c_s)}$. Se a cardinalidade de c_1 é dada por $card_s$ $\boldsymbol{[1 \cdots 1]}$, então dizemos que c_1 é uma* ***característica XOR****, logo:*

ancestral(c_1, c_2)
...
ancestral(c_1, c_s)
min(c_1, 1)
max(c_1, 1)

Por exemplo, a característica Tipo Troca é uma característica XOR na Figura 5.1, logo:

ancestral(Tipo Troca, Sincrono)
ancestral(Tipo Troca, Assincrono)
min(TipoTroca, 1)
max(TipoTroca, 1)

Definição 10. *[Característica OR] Seja $\boldsymbol{C = \{c_1, c_2, \cdots, c_s\}}$, tal que c_1 é ancestral de $\boldsymbol{(c_2, \cdots, c_s)}$. Se a cardinalidade de c_1 é dada por $card_s$ $\boldsymbol{[1 \cdots s]}$, então dizemos que c_1 é uma* ***característica OR****, logo:*

ancestral(c_1, c_2)

...

ancestral(c_1, c_s)

min(n_1, 1)

max(n_1, s)

Por exemplo, a característica Mostrar Documentos é uma característica OR na Figura 5.1, logo:

ancestral(Mostrar Documentos, Video)

ancestral(Mostrar Documentos, Texto)

ancestral(Mostrar Documentos, Imagem)

min(Mostrar Documentos, 1)

max(Mostrar Documentos, 3)

É importante ressaltar que as características descendentes de características **XOR** e características **OR** *não possuem cardinalidade*, de modo a evitar informação redundante ou inconsistente entre as características descendentes e a respectiva característica ancestral.

Definição 11. *[Regra de Composição] Uma* ***regra de composição (RC)*** *é definida por expressões booleanas, onde cada expressão é um conjunto de fórmulas proposicionais sobre o conjunto de características ou características atributos de um MCS. Um regra de composição é essencialmente composta por duas partes: uma expressão antecedente (antecedente) e uma expressão consequente (consequente), conforme definido pela seguinte BNF:*

<CR>::=<antecedente> $\rightarrow$ *<consequente>*

<antecedente>::= <expressao>

<consequente>::= <expressao>

<expressao>::=<expressao> <logico> <expressao> |

<c> | *<*$\neg$*c>* | *<n(v:d)><relacional><dominio>*

<logico>::= $\wedge$ | $\vee$

<relacional>::= $>$ | $<$ | $\geq$ | $\leq$ | $=$ | $\neq$

Onde **c** e **n(v:d)** correspondem a uma característica opcional e uma característica atributo opcional de um MCS, respectivamente, e **<dominio>** corresponde aos valores possíveis que podem ser atribuídos a uma característica atributo.

Por exemplo, na Figura 5.1 são apresentadas duas RC definidas para o MCS MCS_{GuiaM}

RC_1 = *(Mostrar Compatibilidade Perfil* $\wedge$ *Mostrar Perfil Ambiente)* $\rightarrow$ *(Trocar Mensagem)*
RC_2 = *(Tupla)* $\rightarrow$ *(*$\neg$*(Evento))*

A regra RC_1 é composta pela expressão antecedente (Mostrar Compatibilidade Perfil $\wedge$ Mostrar Perfil Ambiente) e pela expressão consequente (Trocar Mensagem) e expressa que a presença das características Mostrar Compatibilidade Perfil e Mostrar Perfil Ambiente no MCS $MCS_{GuiaMovel}$ exige que a característica Trocar Mensagem também esteja presente no $MCS_{GuiaMovel}$.

A regra RC_2 é composta pela expressão antecedente (Tupla) e pela expressão consequente ($\neg$(Evento)) e afirma que a presença da característica Tupla no MCS $MCS_{GuiaMovel}$ exige que a característica Evento não esteja presente no MCS_{GuiaM}

$RC_{GuiaMovel}$ implica no conjunto de RC do $MCS_{GuiaMovel}$. Logo, $RC_{GuiaMovel}$ = {RC_1, RC_2}

Definição 12. *[Modelo de Características do Sistema] Um* ***modelo de características do sistema*** *(**MCS**) é definido pela tupla* **<*r*,** S_r**,** RC_{MCS}**>**, *onde:*

1. ***r*** *é a raiz do* ***MCS****.*
2. S_r *é o conjunto de subárvores da raiz do* ***MCS****.*
3. RC_{MCS} *é o conjunto de regras de composição definidas para o* ***MCS****.*

Por exemplo, na Figura 5.1 é ilustrada uma representação gráfica do $MCS_{GuiaMovel}$ = <Guia Movel, $S_{GuiaMovel}$, $RC_{GuiaMovel}$>, onde:

1. r = Guia Movel
2. $S_{GuiaMovel} = S_{MostrarCompatibilidadePerfil} \wedge S_{TrocarMensagem} \wedge S_{MostrarPerfilAmbiente}$
3. $RC_{GuiaMovel}$ = {RC_1, RC_2}

Onde:

RC_1 = *(Mostrar Compatibilidade Perfil* $\wedge$ *Mostrar Perfil Ambiente)* $\rightarrow$ *(Trocar Mensagem)*
RC_2 = *(Tupla)* $\rightarrow$ *(*$\neg$*(Evento))*

$S_{TrocarMensagem}$:

ancestral(Trocar Mensagem, Tipo Troca)

mandatorio(Trocar Mensagem, Tipo Troca)

ancestral(Tipo Troca, Sincrono)

ancestral(Tipo Troca, Assincrono)

min(Tipo Troca, 1)

max(Tipo Troca, 1)

ancestral(Sincrono, Tupla)

ancestral(Sincrono, Evento)

min(Sincrono, 1)

max(Sincrono, 1)

ancestral(Assincrono, Procedimento Remoto)

$S_{MostrarPerfilAmbiente}$:

ancestral(Mostrar Perfil Ambiente, Listar Itens)

mandatorio(Mostrar Perfil Ambiente, Listar Itens)

ancestral(Listar Itens, Mostrar Documentos)

ancestral(Mostrar Documentos, Imagem)

ancestral(Mostrar Documentos, Texto)

ancestral(Mostrar Documentos, Video)

min(Mostrar Documentos, 1)

max(Mostrar Documentos, 3)

ancestral(Imagem, Resolucao)

atributo(Imagem, Resolucao)

atributo(Resolucao, String)

Definição 13. *[Produto] Um PD é derivado de um **MCS**. O produto corresponde a um subconjunto das características do **MCS**.*

Por exemplo, o produto $P_{GuiaMovel_1}$ (Figura 7.8) representa uma configuração do $MCS_{GuiaMovel}$.

$P_{GuiaMovel_1}$:

1. `r = Guia Movel`
2. $S_{GuiaMovel_1} = S_{TrocarMensagem} \wedge S_{MostrarPerfilAmbiente}$

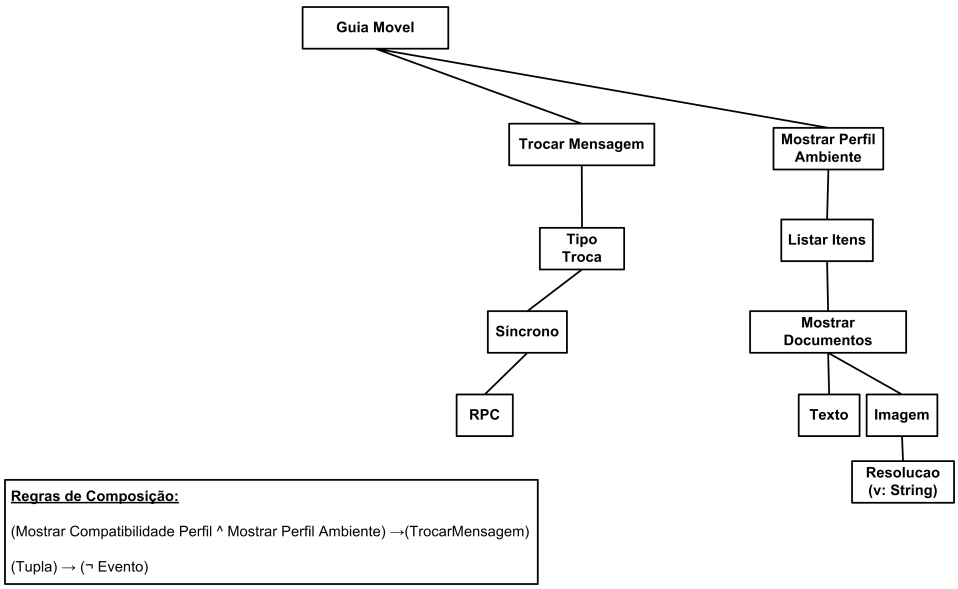

Figura 5.3: Exemplo - MCS do Produto $GuiaMovel_1$.

3. $RC_{GuiaMovel_1}$: `conjunto de regras de composição do` $MCS_{GuiaMovel}$

De acordo com a Figura 7.8, a característica opcional Mostrar Compatibilidade Perfil não foi selecionada para a configuração do produto $P_{GuiaMovel_1}$.Além disso, apenas as características Sincrono e RPC foram selecionadas para a característica XOR Tipo Troca e para a característica OR Mostrar Documentos foram selecionadas as características Texto e Imagem.

É importante observar que todas as variabilidades do $MCS_{GuiaMovel}$ e restrições das $RC_{GuiaMovel}$ foram resolvidas e respeitadas pelo produto $P_{GuiaMovel_1}$.

A Definição 1 até a Definição 13 representam a **sintaxe** dos elementos de um MCS. Uma **Linha de Produtos de Software** representa o **domínio semântico** desse modelo.

Definição 14. *[Linha de Produtos de Software] Uma* ***linha de produtos de software (LPS)*** *corresponde a um conjunto de produtos que podem ser derivados a partir de um MCS.*

`Por exemplo, o produto apresentado na Definição 13 e o produto` $P_{GuiaMovel_2}$`, descrito a seguir, representam produtos pertencentes à` $LPS_{GuiaMovel}$`.`

`O produto` $P_{GuiaMovel_2}$ `(Figura 5.4) representa uma configuração do` $MCS_{GuiaMovel}$`.`

$MCS_{GuiaMovel_2}$:

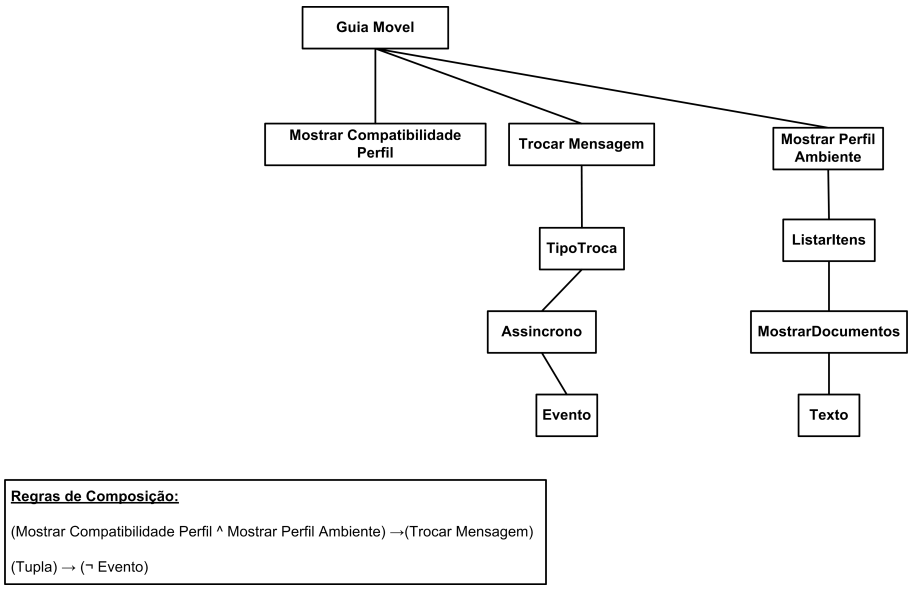

Figura 5.4: Exemplo - MCS do Produto $GuiaMovel_2$.

1. `r = Guia Movel`
2. $S_{GuiaMovel_2} = S_{MostrarCompatibilidadePerfil} \wedge S_{TrocarMensagem} \wedge S_{MostrarPerfilAmbiente}$
3. $RC_{GuiaMovel_2}$: `conjunto de regras de composição do` $MCS_{GuiaMovel}$

De acordo com a Figura 5.4, a característica opcional Mostrar Compatibilidade Perfil foi selecionada para a configuração de $P_{GuiaMovel_2}$. Além disso, as características Assincrono e Evento foram selecionadas para a característica XOR Tipo Troca e, para a característica OR Mostrar Documentos, apenas a característica Texto foi selecionada.

5.2.3 Formalização de um MCC

A sintaxe abstrata descrita nas definições a seguir formaliza os elementos de um MCC.

Definição 15. *[Atributo de Contexto] Seja* $\boldsymbol{c} \in \boldsymbol{C}$*. Se a característica* $\boldsymbol{c}$ *é uma* ***característica atributo****, então dizemos que* $\boldsymbol{c}$ *é um* ***atributo de contexto****.*

Os nomes e tipos permitidos para os atributos de contexto são:

1. **v**: corresponde ao valor do atributo de contexto e deve ter um tipo. Os tipos permitidos para os valores dos atributos de contexto **v** estão descritos conforme a Definição 4.

2. **q**: corresponde à qualidade da informação de contexto. Os valores permitidos para o atributo de contexto **q** devem ser do tipo **Qualidade**:

   - `Alta`: a informação de contexto possui qualidade confiável; e
   - `Baixa`: a informação de contexto não possui qualidade confiável.

3. **o**: corresponde à origem da informação de contexto. Os valores permitidos para o atributo de contexto **o** devem ser do tipo **Origem**:

   - `Usuário`: a informação de contexto é fornecida pelo usuário;
   - `Sentida`: a informação de contexto é obtida a partir de sensores;
   - `Perfil`: a informação de contexto é obtida a partir do perfil ou preferências do usuário; e
   - `Derivada`: a informação de contexto é obtida a partir de outras informações de contexto.

4. **a**: corresponde à frequência de alteração da informação de contexto. Os valores permitidos para o atributo de contexto **a** devem ser do tipo **Alteração**:

   - `Permanente`: o valor da informação de contexto não sofre alteração;
   - `Raramente`: o valor da informação de contexto sofre alterações raramente;
   - `Frequente`: o valor da informação de contexto sofre alterações frequentemente; e
   - `Volátil`: o valor da informação de contexto sofre alterações com uma frequência maior que Frequente.

`Por exemplo, na Figura 5.2, a característica atributo q(baixa:Qualidade) é um atributo de contexto, cujo nome é q, o valor é Baixa e o tipo é Qualidade.`

Definição 16. *[Informação de Contexto] Seja* $\mathbf{c} \in \mathbf{C}$*. Se* ***v(v: Tipo)***, ***q(v: Qualidade)***, ***o(v: Origem)***, ***a(v: Alteracao)*** *são atributos de contexto filhos da característica* **c**, *então dizemos que* **c** *é uma* ***informação de contexto****:*

ancestral(c, v)

ancestral(c, q)

ancestral(c, o)

ancestral(c, a)

atributo(c, v)

dominio(v, Tipo)

atributo(c, q)

dominio(q, Qualidade)

atributo(c, o)

dominio(o, Origem)

atributo(c, a)

dominio(a, Alteracao)

Por exemplo, na Figura 5.2:

atributo(Latencia, v)

dominio(v, Tipo)

...

atributo(Latencia, a)

dominio(a, Alteracao)

Portanto, a característica Latencia corresponde a uma informação de contexto.

Definição 17. *[Entidade de Contexto] Seja* $C=\{c_1, c_2, \cdots, c_s\}$*, tal que* c_i*,* $1 \leq i \leq |C|$*, é uma informação de contexto e é descendente de uma característica* $\mathbf{c}$*. Dizemos que* $\mathbf{c}$ *é uma* ***entidade de contexto****:*

ancestral(c, c_1*)*

...

ancestral(c, c_s*)*

Por exemplo, na Figura 5.2: ancestral(Latencia, Rede). Portanto, a característica Rede é uma entidade de contexto, pois é ancestral da característica Latencia que é uma informação de contexto.

Definição 18. *[Modelo de Características do Contexto] Um* ***modelo de características do contexto (MCC)*** *é um MCS onde:*

1. $\boldsymbol{r}$ *= Contexto.*

2. $S_r = S_{EntidadedeContexto_1} \wedge \cdots \wedge S_{EntidadedeContexto_n}$, *onde cada* $S_{EntidadedeContexto_i}$, *1* $\leq i \leq n$, *corresponde à subárvore de uma entidade de contexto.*

3. $RC_{MCC} = \emptyset$.

Por exemplo, a Figura 5.2 ilustra um exemplo de uma representação gráfica de um MCC, onde:

1. r = Contexto.

2. $S_{Contexto}$ = S_{Rede}.

3. $RC_{MCC} = \emptyset$.

S_{Rede}:

ancestral(Rede, Latencia)

... ancestral(Latencia, v)

...

ancestral(Latencia, q)

atributo(Latencia, q)

dominio(q, String)

...

atributo(Latencia, a)

dominio(a, Alteracao)

5.2.4 Formalização de um MMSC

Um MMSC é composto por um MCS, um MCC, um conjunto de *RCs* e um conjunto de *RAs*. O conjunto de *RAs* define as situações de contexto com base no MCC que, se ativadas, disparam alterações no MCS. Dessa forma, o conjunto de *RAs* é responsável pela composição desses dois modelos.

Definição 19. *[Regra de Adaptação] Uma* ***regra de adaptação (RA)*** *é definida por expressões booleanas onde cada expressão é um conjunto de fórmulas proposicionais sobre o conjunto de características ou características atributos pertencentes a um MCC ou a um MCS. Um regra de adaptação é essencialmente composta por duas partes: uma expressão*

do contexto (contexto) e uma expressão do sistema (sistema), conforme definido pela seguinte BNF:

<RA>::= <contexto> → <sistema>

<contexto>::= <contexto><logico><contexto> |

<cc ∈ EC(MCC)>.<cc ∈ IC(MCC)>.<cc ∈ AC(MCC)><relacional><dominio>

<sistema>::= <sistema>< ∧ ><sistema> |

<c> | <¬c> | <n(v:d)><=><dominio>

<logico>::= ∧ | ∨

<relacional>::= > | < | ≥ | ≤ | = | ≠

Onde **cc** corresponde a uma característica de um MCC e **EC(MCC)**, **IC(MCC)**, e **CA(MCC)** correspondem aos conjuntos Entidades de Contexto, Informação de Contexto e Atributo de Contexto, respectivamente. Além disso, **c**, **n(v:d)** e <**dominio**> possuem a mesma semântica definida para RC.

Chamamos de **conjunto de regras de adaptação**, o conjunto $\mathbf{RA} = \{RA_1, RA_2, \cdots, RA_n\}$, onde cada RA_i, $\mathbf{1} \leq \mathbf{i} \leq |\mathbf{RA}|$ é uma regra de adaptação.

Por exemplo, considerando o $MCS_{GuiaMovel}$ ilustrado na Figura 5.1 e o $MCC_{GuiaMovel}$ ilustrado na Figura 5.2, a seguinte regra é uma regra de adaptação:

RA_1=*((Rede.Latencia.v ≥ 70) ∧ (Rede.Latencia.q = Baixa)) → (Wifi).*

A regra RA_1 afirma que se o valor da latência da rede for maior ou igual a 70 e a qualidade da informação de contexto latência é baixa, implica que a característica Wifi deve existir no $MCS_{GuiaMovel}$. O conjunto RA definido para o exemplo anterior é: RA = $\{RA_1\}$.

Definição 20. *[Modelo de Características Móvel e Sensível ao Contexto] Um* ***modelo de características móvel e sensível ao contexto*** *(**MMSC**) é formado pela tupla* ***MMSC = <MCS, MCC, RC, RA>****, onde:*

1. ***MCS*** *é o modelo de características do sistema que compõe o* ***MMSC****;*
2. ***MCC*** *é o modelo de características do contexto que compõe o* ***MMSC****;*
3. ***RC*** *é o conjunto de regras de composição do* ***MMSC****; e*
4. ***RA*** *é o conjunto de regras de adaptação responsável pela composição do modelo* ***MCS*** *com o modelo* ***MCC****.*

Por exemplo, a composição do $MCS_{GuiaMovel}$, ilustrado no exemplo da Definição 12, com o $MCC_{GuiaMovel}$, ilustrado no exemplo da Definição 18, por meio das regras de adaptação RA, apresentadas no exemplo da Definição 19, corresponde ao $MMSC_{GuiaMovel}$.

Definição 21. *[Produto Adaptado] Um **produto adaptado (PA)** é uma configuração de um **MMSC**. Um **PA** corresponde a um subconjunto das características do **MCS** que compõe o **MMSC**. Um **PA** pode sofrer alterações de acordo com o conjunto de **RAs** definido para o **MMSC**.*

Por exemplo, a composição do produto $P_{GuiaMovel1}$, ilustrado no exemplo da Definição 13, com $MCC_{GuiaMovel}$, ilustrado no exemplo da Definição 18, por meio do conjunto RA, ilustrado como exemplo da Definição 19, representa o $PA_{GuiaMovel_1}$. Esse produto corresponde a uma configuração do $MMSC_{GuiaMovel}$, ilustrado no exemplo da Definição 20.

Definição 22. *[Linha de Produtos de Software Móvel e Sensível ao Contexto] Uma **linha de produtos de software mável e sensível ao contexto (LPMSC)** corresponde a um conjunto de **PA** que podem ser configurados a partir de um **MMSC**.*

5.3 Considerações finais

Neste capítulo, a base da teoria dos métodos formais no processo de desenvolvimento de software foi descrita e a relação existente no uso de métodos formais nas atividades de verificação de software foi apresentada, destacando-se o uso de especificações formais como técnica para verificação formal. Uma verificação sistemática e confiável exige uma maior formalização. Nesta tese de doutorados a Lógica de Primeira Ordem foi utilizada para formalizar a especificação formal dos elementos que compõem um MMSC. Essa especificação formal corresponde à base do PRECISE, descrito no Capítulo 4, em que foram especificadas as operações de verificação de um MMSC e de seus produtos derivados/adaptados.

É importante destacar que as definições contidas na especificação formal disponibilizada em Lógica de Primeira Ordem para $MMCSCs$ podem ser usadas em qualquer processo de verificação de MMSCs.

Capítulo 6

Propriedades de um Modelo de Características Móvel e Sensível ao Contexto

Neste Capítulo, o conjunto de propriedades semânticas para um $MMSC$ correto e consistente é descrito. As propriedades estão definidas em um nível genérico, de modo a permitir a verificação de qualquer $MMSC$.

Na Seção 6.1, as propriedades de boa formação são apresentadas. Na Seção 6.2, as situações que indicam a presença de anomalias em um $MMSC$ são descritas formalmente. Por fim, na Seção 6.3, a consistência entre as regras de um $MMSC$ é discutida.

6.1 Propriedades de Boa formação

As propriedades de boa formação (PBF) apresentadas nesta tese formalizam as regras que devem ser analisadas durante a verificação formal de um $MMSC$. Essas propriedades determinam a corretude dos elementos que compõem um $MMSC$. Nas próximas seções, considere N o conjunto de características em um MCS e N' o conjunto de características em um MCC.

6.1.1 Boa formação de um MCS

Definição 23. *[Modelo de Características do Sistema Bem Formado]*

PBF1 *Duas características mandatórias ou opcionais não podem ter o mesmo nome:* $\forall\ n \in N \Rightarrow \nexists\ n' \in N \mid nome(n) = nome(n')$.

PBF2 *Uma característica não pode ter duas ou mais características atributo com o*

mesmo nome: $\forall\ n \in N \Rightarrow \nexists\ n',\ n'' \in N \mid ancestral(n,\ n') \wedge ancestral(n,\ n'') \wedge atributo(n,\ n') \wedge atributo(n,\ n'') \wedge nome(n') = nome(n'')$.

PBF3 *Uma característica não pode ser descendente de si mesma:* $\nexists\ n \in N \mid ancestral(n,\ n)$.

PBF4 *A relação entre uma característica ancestral e suas características descendentes deve ser acíclica:* $\forall\ n \in N \Rightarrow \nexists\ n' \in N \mid ancestral(n,\ n') \wedge ancestral(n',\ n)$.

PBF5 *Todas as características e características atributo, com exceção da raiz, devem ser descendentes de alguma outra característica que deve ser única:* $\forall\ n,\ n',\ n'' \in N \mid ancestral(n',\ n) \wedge ancestral(n'',\ n) \Rightarrow nome(n') = nome(n'')$.

PBF6 *Uma característica atributo não pode ter características descendentes:* $\forall\ n,\ n' \in N \mid ancestral(n,\ n') \wedge atributo(n,\ n') \Rightarrow \nexists\ n'' \in N \mid ancestral(n',\ n'')$.

PBF7 *Um elemento mandatório deve sempre ser precedido por um elemento mandatório:* $\forall\ n',\ n'' \in N \mid ancestral(n',\ n'') \wedge mandatorio(n',\ n'') \Rightarrow \exists\ n \in N \mid ancestral(n,\ n') \wedge mandatorio(n,\ n')$.

PBF8 *Um elemento descendente de um elemento opcional deve ser opcional:* $\forall\ n,\ n' \in N \mid ancestral(n,\ n') \wedge opcional(n,\ n') \Rightarrow \exists\ n'' \in N \mid ancestral(n',\ n'') \wedge opcional(n',\ n'')$.

6.1.2 Boa formação de uma RC

Definição 24. *[Regra de Composição Bem Formada]*

Nas propriedades a seguir, considere EA uma expressão antecedente e EC uma expressão consequente de uma RC.

PBF9 *Uma característica opcional ou característica atributo em uma expressão antecedente ou em uma expressão consequente devem pertencer a um MCS:* $\forall\ n \in AE \cup CE,\ n \in N$.

PBF10 *As características referenciadas em uma expressão antecedente ou em uma expressão consequente devem ser opcionais ou características atributo:* $\forall\ n,\ n' \in EA \cup EC \mid ancestral(n,\ n') \Rightarrow (opcional(n,\ n') \oplus atributo(n,\ n'))$.

PBF11 *Uma característica opcional ou característica atributo não pode requerer a si própria ou a uma de suas características ancestrais:* $\forall\ n \in EA \cup EC \Rightarrow \nexists\ n' \in EA \cup EC \mid ancestral(n,\ n') \wedge (opcional(n,\ n') \oplus atributo(n,\ n')) \wedge (n' \rightarrow n') \wedge (n' \rightarrow n)$.

PBF12 *Uma característica opcional ou característica atributo não pode excluir a si própria ou a uma de suas características ancestrais:* $\forall\ n \in EA \cup EC \Rightarrow \nexists\ n' \in EA \cup EC \mid ancestral(n,\ n') \wedge (opcional(n,\ n') \oplus atributo(n,\ n')) \wedge (n' \rightarrow (\neg n')) \wedge (n' \rightarrow (\neg n))$.

6.1.3 Boa formação de um PD

Definição 25. *[Produto Bem Formado]*

PBF13 *O* MCS_P *satisfaz às propriedades de boa formação de um* MCS.

PBF14 *Se uma característica descendente é selecionada para um produto PD, então a característica ancestral correspondente também deve ser selecionada para PD:* $\forall\ n,\ n' \in N \mid ancestral(n,\ n') \wedge n' \in PD \Rightarrow n \in PD$.

PBF15 *Se uma característica atributo é selecionada para um produto PD, então a característica ancestral correspondente também deve ser selecionada para PD:* $\forall\ n,\ n' \in N \mid ancestral(n,\ n') \wedge atributo(n,\ n') \wedge n' \in PD \Rightarrow n \in PD$.

PBF16 *Se uma característica ancestral não é selecionada para um produto PD, então nenhuma das suas características descendentes deve ser selecionada para PD:* $\forall\ n,\ n' \in N \mid ancestral(n,\ n') \wedge n \notin PD \Rightarrow n' \notin PD$.

PBF17 *Se uma característica ancestral não é selecionada para um produto PD, então nenhuma das suas características atributo deve ser selecionada para PD:* $\forall\ n,\ n' \in N \mid ancestral(n,\ n') \wedge atributo(n,\ n') \wedge n \notin PD \Rightarrow n' \notin PD$.

PBF18 *Se uma característica ancestral é selecionada para um produto PD, então todas as suas características descendentes mandatórias ou características atributo devem ser selecionadas para PD:* $\forall\ n,\ n' \in N \mid ancestral(n,\ n') \wedge (mandatorio(n,\ n') \oplus atributo(n,\ n')) \wedge n \in PD \Rightarrow n' \in PD$.

PBF19 *Se uma característica ancestral XOR com k características descendentes é selecionada para um produto PD, então uma única característica descendente deve ser selecionada para PD:* $\forall\ n, n_1, \cdots, n_k \in N \mid (ancestral(n, n_1) \wedge \cdots\ ancestral(n, n_k)) \wedge max(n, 1) \wedge min(n, 1) \wedge n \in PD \Rightarrow (n_1 \in PD \Leftrightarrow (n_2, \cdots, n_k \notin PD)) \vee (n_2 \in PD \Leftrightarrow (n_1, \cdots, n_k \notin PD)) \vee \cdots \vee (n_k \in PD \Leftrightarrow (n_1, \cdots, n_{k-1} \notin PD))$.

PBF20 *Se uma característica ancestral OR com k características descendentes é selecionada para um produto PD, então no mínimo 1 e no máximo k características descendentes devem ser selecionadas para PD:* $\forall\ n, n_1, \cdots, n_k \in N \mid (ancestral(n, n_1) \wedge \cdots\ ancestral(n, n_k)) \wedge min(n, 1) \wedge max(n, k) \wedge n \in PD \Rightarrow (n_1 \vee n_2 \vee \cdots \vee n_k) \in PD$.

6.1.4 Boa formação de um MCC

Definição 26. *[Modelo de Características do Contexto Bem Formado]*

PBF21 *As regras de boa formação PBF1 a PBF6 do MCS se aplicam para o MCC.*

PBF22 *As características do segundo nível da árvore representam entidades de contexto:* $\forall\ n, n' \in N \mid ancestral(n, n') \wedge nome(n) = nome(r) \Rightarrow ec(n, n')$

PBF23 *As características do terceiro nível da árvore representam informações de contexto:* $\forall\ n, n', n'' \in N \mid ancestral(n, n') \wedge ancestral(n', n'') \wedge nome(n) = nome(r) \Rightarrow ic(n', n'')$

PBF24 *As características do quarto nível da árvore devem ser folhas que representam os quatro atributos de contexto e possuem nomes e tipos pré-definidos para valor, origem, qualidade e frequência de alteração:* $\forall\ n, n', n'', n''' \in N \mid ancestral(n, n') \wedge ancestral(n', n'') \wedge ancestral(n'', n''') \wedge nome(n) = nome(r) \Rightarrow ac(n'', n''') \wedge (nome(n''') = v \vee nome(n''') = o \vee nome(n''') = q \vee nome(n''') = a)$

6.1.5 Boa formação de uma RA

Definição 27. *[Regra de Adaptação Bem Formada]*

Nas propriedades a seguir, considere EX uma expressão de contexto e ES uma expressão de sistema de uma RA.

PBF25 *As características e as características atributo referenciadas na expressão do contexto de uma regra de adaptação devem pertencer ao MCC:* $\forall\ n \in EX \Rightarrow n \in N'$.

PBF26 *As características opcionais e as características atributo referenciadas na expressão do sistema de uma regra de adaptação devem pertencer ao MCS:* $\forall\ n,\ n' \in ES \mid ancestral(n,\ n') \wedge (opcional(n,\ n') \oplus atributo(n,\ n')) \Rightarrow n,\ n' \in N$.

PBF27 *As características opcionais referenciados na expressão do sistema de uma regra de adaptação não podem ser quantificadas mais de uma vez. Seja f uma função que atribui os valores presente ou ausente para uma característica opcional:* $\forall\ n,\ n' \in ES \mid ancestral(n,\ n') \wedge opcional(n,\ n') \Rightarrow \nexists\ f \mid f{:}n \leftarrow v_1 \wedge f{:}n \leftarrow v_2 \wedge v_1 \neq v_2$.

PBF28 *As características atributo referenciadas na expressão do sistema de uma regra de adaptação não podem ser quantificadas mais de uma vez. Seja f uma função que atribui valores para uma característica atributo:* $\forall\ n,\ n' \in ES \mid ancestral(n,\ n') \wedge atributo(n,\ n') \Rightarrow \nexists\ f \mid f{:}n \leftarrow valor(n,\ v_1) \wedge f{:}n \leftarrow valor(n,\ v_2) \wedge v_1 \neq v_2$.

6.1.6 Boa formação de um $MMSC$

Definição 28. *[Modelo de Características Móvel e Sensível ao Contexto Bem Formado]*

PBF29 *O MCS que compõe o MMSC satisfaz a todas as propriedades de boa formação de um MCS.*

PBF30 *O MCC que compõe o MMSC satisfaz a todas as propriedades de boa formação de um MCC.*

PBF31 *O conjunto RC do MMSC satisfaz a todas as propriedades de boa formação das regras de composição.*

PBF32 *O conjunto RA do MMSC satisfaz a todas as propriedades de boa formação das regras de adaptação.*

6.1.7 Boa formação de um PA

Definição 29. *[Produto Sensível Adaptado Bem Formado]*

PBF33 *Satisfaz às propriedades de boa formação definidas para um produto.*

6.2 Propriedades de presença de anomalias

Uma anomalia corresponde a uma situação em que a configuração de alguns produtos são perdidas devido à existência de contradições no $MMSC$ causadas pelo mau uso das RCs. Isso implica que, apesar de um modelo ser avaliado como correto, este ainda pode conter anomalias, pois, em geral, as anomalias estão relacionadas à problemas semânticos inseridos no modelo [100]. As anomalias de um $MMSC$ incluem: características mortas, característica opcional falsa e cardinalidade errada. ([100],[22]). Cada uma destas anomalias (AN) é descrita nas próximas subseções. De acordo com o nosso conhecimento, as situações que indicam a presença de anomalias em um $MMSC$ não estão documentadas precisamente na literatura.

6.2.1 Característica morta

Uma característica morta é aquela que nunca aparece em nenhum produto de um $MMSC$.

Definição 30. *[Característica Morta]Uma característica é uma característica morta se satisfaz à uma das seguintes situações.*

AN1 *Se um $MMSC$ possui uma RC, tal que, se RC é satisfeita, então qualquer característica descendente de uma característica alternativa (XOR) nunca está presente em nenhum dos PAs gerados a partir do $MMSC$: $\forall\ n, n_1, \cdots, n_k \in N \mid (ancestral(n, n_1) \wedge \cdots \wedge ancestral(n, n_k)) \wedge max(n, 1) \wedge min(n, 1) \Rightarrow \forall\ PA \mid n \in PA \rightarrow (n_1 \vee n_2 \vee \cdots \vee n_k) \notin PA$.*

AN2 *Se um $MMSC$ possui uma regra de composição RC, tal que, se RC é satisfeita, então qualquer característica descendente de uma característica opcional (OR) nunca está presente nos PAs gerados a partir do $MMSC$: $\forall\ n, n_1, \cdots, n_k \in N \mid (ancestral(n, n_1) \wedge \cdots \wedge \wedge(n, n_k)) \wedge min(n, 1) \wedge max(n, k) \Rightarrow \forall\ PA \mid n \in PA \rightarrow (n_1 \vee n_2 \vee \cdots \vee n_k) \notin PA$.*

6.2.2 Característica opcional falsa

Uma característica é opcional falsa se é modelada como opcional, mas é incluída em todos os produtos de um $MMSC$.

Definição 31. *[Característica Opcional Falsa]Uma característica é uma característica opcional falsa se satisfaz à uma das seguintes situações.*

AN3 *Se um $MMSC$ possui uma RC, tal que, se RC é satisfeita, então uma característica descendente de uma característica alternativa (XOR) sempre está presente nos PAs gerados a partir do $MMSC$: $\forall$ n, n_1, $\cdots$, $n_k \in N \mid (ancestral(n, n_1) \wedge \cdots \wedge ancestral(n, n_k)) \wedge max(n, 1) \wedge min(n, 1) \Rightarrow \forall PA \mid n \in PA \rightarrow (n_1 \vee n_2 \vee \cdots \vee n_k) \in PA$.*

AN4 *Se um $MMSC$ possui uma RC, tal que, se RC é satisfeita, então uma característica descendente de uma característica opcional (OR) sempre está presente nos PAs gerados a partir do $MMSC$: $\forall$ n, n_1, $\cdots$, $n_k \in N \mid (ancestral(n, n_1) \wedge \cdots \wedge ancestral(n, n_k)) \wedge max(n, k) \wedge min(n, 1) \Rightarrow \forall PA \mid n \in PA \rightarrow (n_1 \vee n_2 \vee \cdots \vee n_k) \in PA$.*

AN5 *Se um $MMSC$ possui uma RC, tal que, se RC é satisfeita, então uma característica opcional sempre está presente em todos os PAs gerados a partir do $MMSC$: $\forall$ n, $n_1 \in N \mid ancestral(n, n_1) \wedge opcional(n, n_1) \Rightarrow \forall PA \mid n \in PA \rightarrow n_1 \in PA$.*

6.2.3 Cardinalidade errada

Uma cardinalidade está errada quando não pode ser instanciada para nenhum produto. Essa anomalia é causada por inconsistências entre alguma RC do $MMSC$.

Definição 32. *[Cardinalidade Errada]Uma característica possui uma cardinalidade errada se satisfaz à uma das seguintes situações.*

AN6 *Se um $MMSC$ possui uma RC, tal que, se RC é satisfeita, então são gerados PA que contêm mais de uma ou nenhuma característica descendente para uma característica alternativa (XOR): $\forall$ n, n_1, $\cdots$, $n_k \in N \mid (ancestral(n, n_1) \wedge \cdots \wedge ancestral(n, n_k)) \wedge max(n, 1) \wedge min(n, 1) \Rightarrow \forall PA \mid n \in PA \rightarrow ((n_1 \vee n_2 \vee \cdots \vee n_k) \in PA \vee (n_1 \wedge n_2 \wedge \cdots \wedge n_k) \notin PA)$.*

AN7 *Se um $MMSC$ possui uma RC, tal que, se RC é satisfeita, então são gerados PA que não contêm nenhuma característica descendente para uma característica opcional (OR): $\forall$ n, n_1, $\cdots$, $n_k \in N \mid (ancestral(n, n_1) \wedge \cdots \wedge ancestral(n, n_k)) \wedge max(n, k) \wedge min(n, 1) \Rightarrow \forall PA \mid n \in PA \rightarrow (n_1 \wedge n_2 \wedge \cdots \wedge n_k) \notin PA$.*

6.3 Propriedade de consistência

6.3.1 Consistência de um modelo de características do sistema

Definição 33. *Um modelo de características do sistema MCS é consistente se ao menos um produto válido P é configurado a partir de MCS. Para isso, a conjunção do MCS com as RCs correspondentes deve ser consistente.*

6.3.2 Consistência entre regras

Os $MMSC$ possuem RCs e RAs e a presença dessas regras pode alterar o conjunto válido de combinações de características dos produtos configurados ou adaptados. No entanto, apesar da importância de analisar as situações de possíveis inconsistências entre as regras, essa questão não é formalizada na literatura relacionada com a análise de $MMSC$.

A propriedade de consistência entre as regras se refere à consistência inter-regras de um $MMSC$. Para isso, as RC e as RA são combinadas e o resultado é verificado. Uma composição inter-regra é consistente quando não possui informações redundantes ou contraditórias.

Definição 34. *[Regras Consistentes]Um conjunto de regras definido para um $MMSC$ é consistente se satisfaz as seguintes propriedades (PCR).*

PCR1 *As regras de composição definidas para o $MMSC$ são consistentes entre si.*

PCR2 *As regras de adaptação definidas para o $MMSC$ são consistentes entre si.*

PCR3 *As regras de composição e adaptação definidas para o $MMSC$ são consistentes entre si.*

A consistência entre RCs, entre RAs, e entre RCs e RAs é analisada com o objetivo de permitir a verificação da consistência dos modelos que constituem um $MMSC$ e a verificação, em tempo de desenvolvimento, da corretude dos produtos adaptados.

6.3.3 Consistência de um modelo de características sensível ao contexto

Definição 35. *[Modelo de Características Móvel e Sensível ao Contexto Inconsistente]Um modelo de características sensível ao contexto $MMSC$ é inconsistente quando a conjunção de pelo menos uma RA com uma RC é inconsistente.*

Se um $MMSC$ é inconsistente, então existem situações de contexto que causam adaptações de produtos incorretas. Assim, podemos afirmar que, se as inconsistências inter-regras são evitadas, a qualidade do processo de adaptação dos produtos aumenta.

Definição 36. *[Produto Adaptado Bem formado]Um produto adaptado PA é bem formado quando a conjunção das RAs que geram o PA com as RCs do $MMSC$ a partir do qual o PA é configurado é consistente.*

6.4 Considerações finais

Neste Capítulo, são apresentadas as propriedades de boa formação e de consistência e quais operações podem ser executadas sobre os $MMSCs$. Essas propriedades e operações foram obtidas a partir de uma revisão bibliográfica detalhada das boas práticas para modelagem de características e modelagem de contexto. Apesar de não ser possível afirmar que o conjunto de propriedades e operações fornecido garante a corretude, a consistência e a ausência de anomalias de um $MMSC$, é possível afirmar que essas auxiliam na melhoria da qualidade das $LPMSCs$. Além disso, a análise bibliográfica realizada com base em erros comuns encontrados em diferentes tipos de modelos de características, pode ser utilizado como uma contribuição adicional para a área de pesquisa de análise de $LPSs$.

Capítulo 7

Validação do PRECISE

Neste Capítulo, o PRECISE é validado com o objetivo de verificar se os resultados previstos são alcançados.

Na Seção 7.1, uma visão geral do processo de validação é apresentada. Na Seção 7.2, são descritas as atividades realizadas na primeira prova de conceito da validação que utiliza a ferramenta *FixTure*. Na Seção 7.3, é descrita a segunda prova de conceito da validação realizada que usa a linguagem Prolog, e por fim, na Seção 7.4, o PRECISE é comparado com relação a outras ferramentas para checagem da corretude de MCs.

7.1 Introdução

A validação apresentada aqui consiste em mostrar como o PRECISE usa a especificação formal apresentada no Capítulo 5 e as propriedades definidas no Capítulo 6 para identificar defeitos, anomalias e inconsistências existentes em um $MMSC$.

É importante ressaltar que não foram encontradas na literatura exemplos de $LPSs$ sensíveis ao contexto que possam ser usadas como estudo de caso para essa pesquisa de doutorado. Por esse motivo, o $MMSC$ da $LPMSC$ para Guias de Visita Móveis e Sensíveis ao Contexto desenvolvido no projeto MobiLine é utilizado como estudo de caso. Portanto, essa $LPMSC$ foi selecionada para a validação do PRECISE por ser a única $LPMSC$ disponível na literatura e devido à facilidade de acesso aos artefatos desenvolvidos.

Com o objetivo de mostrar a flexibilidade da especificação formal apresentada no Capítulo 5, duas provas de conceito são realizadas na validação do PRECISE. Na primeira, é utilizada a ferramenta *FixTure* [45], desenvolvida no contexto de uma dissertação de mestrado do GREat [69] a partir deste trabalho de doutorado. Tanto a especificação formal e as propriedades aqui propostas para um $MMSC$, quanto o perfil UML apresentado na subseção 7.2.1 são usados na sua implementação. A segunda prova de conceito

complementa a primeira, pois as fases do PRECISE são analisadas individualmente a partir de defeitos, anomalias e inconsistências injetadas nos exemplos utilizados. Na segunda prova de conceito, a especificação formal e as propriedades de um $MMSC$ são mapeadas manualmente para linguagem Prolog [38] e a ferramenta SWI-Prolog [142] é utilizada para checar a presença de problemas em um $MMSC$.

7.2 Prova de conceito usando a ferramenta *FixTure*

O passo a passo da primeira prova de conceito realizada na validação do PRECISE está ilustrado na Figura 7.1. Nessa prova de conceito, duas linguagens de especificação, ilustradas pelas entradas (OCL e Lógica Proposicional) e quatro atividades são utilizadas. Na atividade **Verificação Inicial**, a ferramenta *FixTure* usa um Perfil UML enriquecido com especificações OCL para verificar a boa formação dos modelos (MCS e MCC) e das regras (RCs e RAs). Na atividade **Verificação da Consistência**, é verificada a consistência do MCS e a consistência inter-regras do $MMSC$ (RC-RC, RA-RA e RC-RA). Na atividade **Verificação dos Produtos Derivados**, a *FixTure* checa a corretude e a consistência de um produto inicial derivado pelo usuário. Finalmente, na atividade **Verificação dos Produtos Adaptados**, o processo de simulação proposto na fase 5 do PRECISE é usado para identificar adaptações incorretas dos produtos causadas por mudanças no contexto. Cada uma dessas atividades está detalhada nas seções seguintes.

7.2.1 Verificação inicial

Na verificação inicial, o $MMSC$ construído no projeto MobiLine é mapeado para o metamodelo formal da UML. Esse metamodelo fornece o mecanismo de extensão denominado Perfil UML, que permite a customização da UML para domínios específicos. Portanto, é possível criar uma linguagem de modelagem específica para construir $MMSCs$. Nesse sentido, a especificação formal apresentada no Capítulo 5 é definida usando estereótipos e especificações OCL que são agrupados em um perfil UML, detalhado a seguir.

O perfil UML elaborado possui restrições específicas para construir $MMSCs$ corretos. Portanto, as propriedades descritas no Capítulo 6 são definidas usando restrições OCL, pois a associação de restrições OCL aos estereótipos de um perfil UML reduz a probabilidade de erros de modelagem, uma vez que essas restrições são checadas sempre

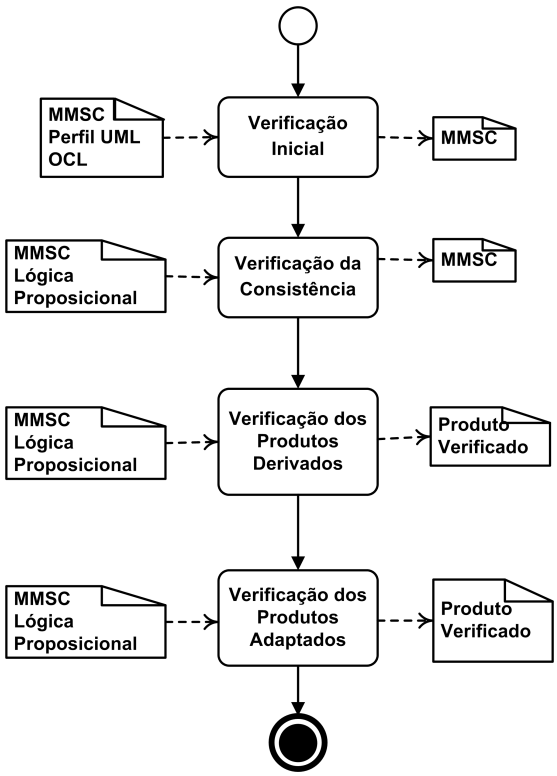

Figura 7.1: Entradas, saídas e atividades da prova de conceito do PRECISE usando a *FixTure*.

que o estereótipo é aplicado em um modelo. Por exemplo, na Listagem 1, é apresentada a especificação OCL correspondente à propriedade de boa formação PBF2, definida no Capítulo 6.

Listagem 1 - Propriedade PBF2 especificada em OCL

```
% PBF2: Duas características mandatórias ou opcionais não podem ter
% o mesmo nome.
inv:
self.extension->exists((o | o.ownedEnd.type = Opcional) or
(m | m.ownedEnd.type = Mandatoria)) implies
o.ownedEnd.name =\= m.ownedEnd.name;
```

O perfil UML criado é usado no desenvolvimento da ferramenta *FixTure* que implementa a verificação formal dos modelos e regras que compõem um $MMSC$. Embora existam diversas ferramentas disponíveis para construir e verificar MCs, em geral, essas ferramentas não fornecem acesso completo aos seus recursos, o que dificulta a sua extensão. Portanto, a ferramenta *FixTure* é desenvolvida com objetivo de implementar o processo de verificação formal PRECISE.

Para iniciar a validação do PRECISE utilizando a ferramenta *FixTure*, o *MMSC* da *LPMSC* para Guias de Visita Móveis e Sensíveis ao Contexto e um produto inicial derivado pelo usuário são especificados usando a interface ilustrada na Figura 7.2. A *FixTure* auxilia na construção dos modelos e das regras, possibilitando a verificação da boa formação dos mesmos.

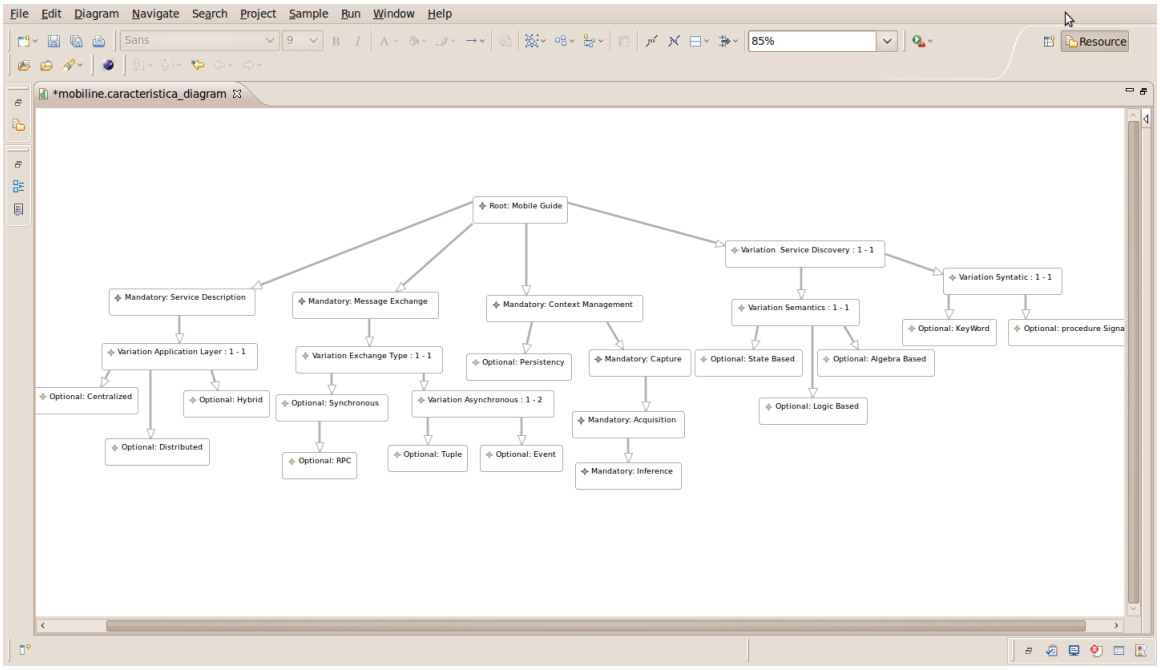

Figura 7.2: Imagem da interface da ferramenta *FixTure*.

Uma vez que a especificação do *MMSC* é finalizada e verificada pelas restrições OCL, a boa formação das regras que compõem o *MMSC* é checada pela ferramenta. Por exemplo, considere as *RCs* ilustradas na Figura 7.3 extraídas do *MMSC* do projeto MobiLine. A RC_1 quebra a *PBF*10 definida no Capítulo 6, que afirma que as características referenciadas em uma expressão antecedente ou em uma expressão consequente de uma *RC* devem ser características opcionais ou características atributo. Entretanto, as características *Mostrar Perfil Ambiente* e *Trocar Mensagem* são mandatórias. Nesse caso, não foi possível incluir a RC_1 na ferramenta *FixTure*, uma vez que a ferramenta não permite a seleção de características mandatórias na criação das *RCs*. Esse problema ocorreu porque quando as regras do MobiLine foram criadas, a PBF10 não foi considerada.

Regras de Composição:

RC1: (Mostrar Compatibilidade Perfil ^ Mostrar Perfil Ambiente) →(Trocar Mensagem)

RC2: (Tupla) → (¬ Trocar Mensagem)

Figura 7.3: Regras de composição do *MMSC* do MobiLine.

A RA_1 ilustrada na Figura 7.4 não satisfaz à BFN definida no Capítulo 5 para regras de adaptação, pois de acordo com essa BNF, o operador lógico ¬ deve ser aplicado a uma característica opcional, porém na RA_1 esse operador está aplicado à característica mandatória *Trocar Mensagem*.

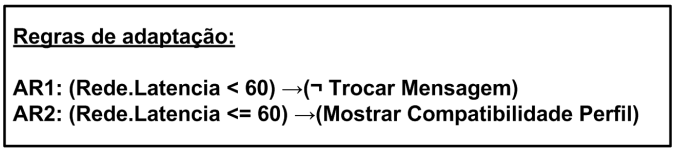

Figura 7.4: Exemplo de regra de adaptação.

Em resumo, na atividade **Verificação Inicial**, a ferramenta *FixTure* auxilia na verificação da boa formação dos modelos e das regras de um $MMSC$. Essa atividade corresponde à **fase 1** e parte da **fase 2** do PRECISE, mostradas na Figura 4.2 do Capítulo 4. A verificação da boa formação do MCS e do MCC correspondem à **fase 1**. A verificação da boa formação das regras corresponde apenas à uma parte da **fase 2**, que também inclui a verificação da consistência das regras, não realizada na atividade **Verificação Inicial**.

Criação do perfil UML para $MMSC$

O perfil UML usado no desenvolvimento da ferramenta *FixTure* segue as orientações para especificação de perfis UML sugeridas por Selic [134]. As extensões descritas no perfil são baseadas na versão 2.4.1 da UML ([111], [112]) e abrangem apenas o diagrama de classes. De acordo com a Figura 7.5, quatro atividades são executadas para construção do perfil UML: *Elicitação dos Requisitos*, *Elaboração do Metamodelo*, *Construção do Perfil* e *Definição dos Estereótipos*.

Inicialmente, a atividade *Elicitação dos Requisitos* é realizada para definir uma notação para modelar características no domínio de aplicações móveis e sensíveis ao contexto. Para isso, técnicas existentes na literatura para modelagem de características e modelagem de contexto, por exemplo, [71], [140], [85], [48], [83], [104], [128], [63] e [98] são analisadas com o objetivo de extrair as contribuições de cada técnica.

Na atividade *Elaboração do Metamodelo*, os requisitos elicitados são capturados por meio de um metamodelo. Em seguida, os mecanismos de extensão da UML são usados na atividade *Construção do Perfil*. Para finalizar, na atividade *Definição dos Estereótipos*,

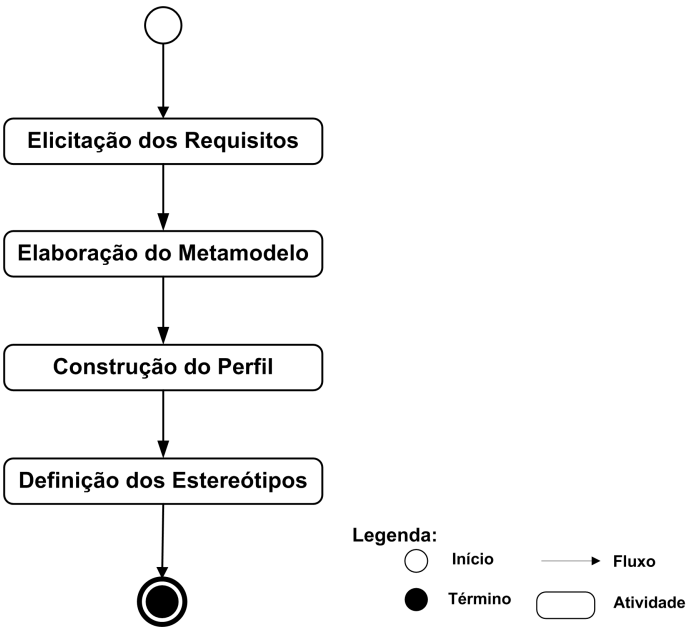

Figura 7.5: Atividades para a criação do perfil UML.

os estereótipos e as restrições necessárias para construir $MMSCs$ corretos e consistentes são definidos.

Elicitação dos requisitos

Conforme já mencionado no Capítulo 2, os $MMSCs$ são compostos por $MCSs$ e $MCCs$. Portanto, na atividade *Elicitação dos Requisitos*, é necessário documentar os requisitos necessários para modelagem desses dois modelos.

Kang *et al.* [86] propuseram o primeiro MC para capturar similaridades e variabilidades das $LPSs$. Entretanto, após anos de pesquisa, ainda não existe um consenso entre os pesquisadores e várias extensões para esse modelo foram propostas, por exemplo, [71], [140], [85], [126], [48].

Por outro lado, um grande número de aplicações móveis e sensíveis ao contexto têm sido desenvolvido nesses últimos anos para uma variedade de domínios de aplicação, utilizando modelos de contexto diversos, por exemplo, [74], [97], [84], [3]. Esses trabalhos influenciaram a definição de um conjunto de requisitos para uma notação que tenha por objetivo expressar $MMCSs$. Os requisitos para $MMSCs$ adotados nesta tese de doutorado são apresentados a seguir.

Elementos mandatórios (invariantes) - O modelo deve expressar as partes comuns a todos os produtos de uma família de aplicações.

Elementos opcionais (invariantes) - O modelo deve expressar as partes opcionais aos produtos de uma família de aplicações, deixando explícitos quais os elementos que podem ser descartados na derivação de um produto específico.

Pontos de variação mandatórios e opcionais - O modelo deve representar explicitamente os elementos (mandatórios e opcionais) que permitem a customização do modelo na derivação de um produto específico.

Elementos variantes mandatórios - O modelo deve representar explicitamente os elementos mandatórios usados para customizar um modelo na derivação de um produto específico.

Relacionamentos entre os elementos - O modelo deve representar explicitamente a relação entre os elementos de um modelo. Entre tais relacionamentos podem ser incluídos o relacionamento existente entre as variantes e os pontos de variação. Além desses, existem várias relações entre os tipos de informações de contexto. Por exemplo, uma relação de dependência ocorre quando as entidades de informação de contexto dependem de outras entidades de informação de contexto.

Restrições de dependência e exclusividade entre elementos - É desejável que o modelo represente as restrições de dependência entre elementos, bem como relações de mútua exclusividade, para quaisquer conjuntos de elementos.

Heterogeneidade - O modelo deve ser capaz de expressar as diferentes origens das informações de contexto, pois os modelos de contexto precisam lidar com uma grande variedade de fontes de informações de contexto que diferem em sua taxa de atualização e nível semântico. Algumas informações de contexto são percebidas ou sentidas. Os sensores podem capturar determinados estados do mundo físico e fornecer acesso rápido e quase em tempo real, oferecendo dados brutos (como uma posição de um GPS), que devem ser interpretados antes de serem utilizados por aplicações. Informação de contexto pode ser fornecida pelo usuário ou obtida a partir do perfil dos usuários. Essa informação, em geral, é atualizada e não precisa de interpretação adicional. Dados de contexto também podem ser derivados de outras informações de contexto existentes. Um contexto de dados obtido a partir de bases de dados ou bibliotecas digitais, em geral, é estático.

Histórico - É desejável que o modelo capture o histórico do contexto. As aplicações

sensíveis ao contexto podem ter acesso a estados passados e estados futuros. A gestão do histórico do contexto pode se tornar complexa, se o número de atualizações for muito alto. Dessa forma, pode não ser viável armazenar todos os valores para o acesso futuro.

Qualidade - O modelo deve incluir a informação da qualidade da informação de contexto. As informações do contexto podem possuir um nível de qualidade variável devido à sua natureza dinâmica e heterogênea. Na verdade, uma informação pode até mesmo ser incorreta. Por exemplo, a maioria dos sensores possui uma imprecisão inerente (e.g., poucos metros de posições em um GPS). Além disso, as informações de contexto podem estar incompletas ou conflitantes.

Atualização - O modelo deve ser capaz de representar a frequência de alteração das informações de contexto. As informações do contexto podem ter uma frequência de alteração variada. Por exemplo, a bateria de um celular possui um valor de contexto que varia constantemente, já as bibliotecas de um dispositivo móvel podem não variar nunca.

Elaboração do metamodelo

Nesta Seção, é apresentado o metamodelo proposto para formalizar os elementos existentes em um $MMSC$. Um metamodelo captura os elementos de um domínio, os relacionamentos entre esses elementos e as restrições que governam sua sintaxe e semântica.

Os requisitos identificados durante a atividade *Elicitação dos Requisitos* facilitam o entendimento dos elementos chaves de uma notação para expressar um MCS e um MCC e, portanto, são essenciais para o desenvolvimento do metamodelo para $MMSCs$.

A Figura 7.6 ilustra as meta-classes e relacionamentos usados para capturar um $MMSC$. Esse metamodelo é representado por um diagrama de classes. Os relacionamentos adotados são os mesmos relacionamentos definidos para os diagramas de classes da UML.

A meta-classe *MMSC* é uma agregação das meta-classes *Caracteristica* e *Grupo de Caracteristicas*. A meta-classe *Grupo de Caracteristicas* representa o conceito ponto de variação e corresponde a uma agregação de meta-classes *Caracteristica* que, por sua vez, é especializada pelas meta-classes *Informacao de Contexto*, *Entidade de Contexto*,

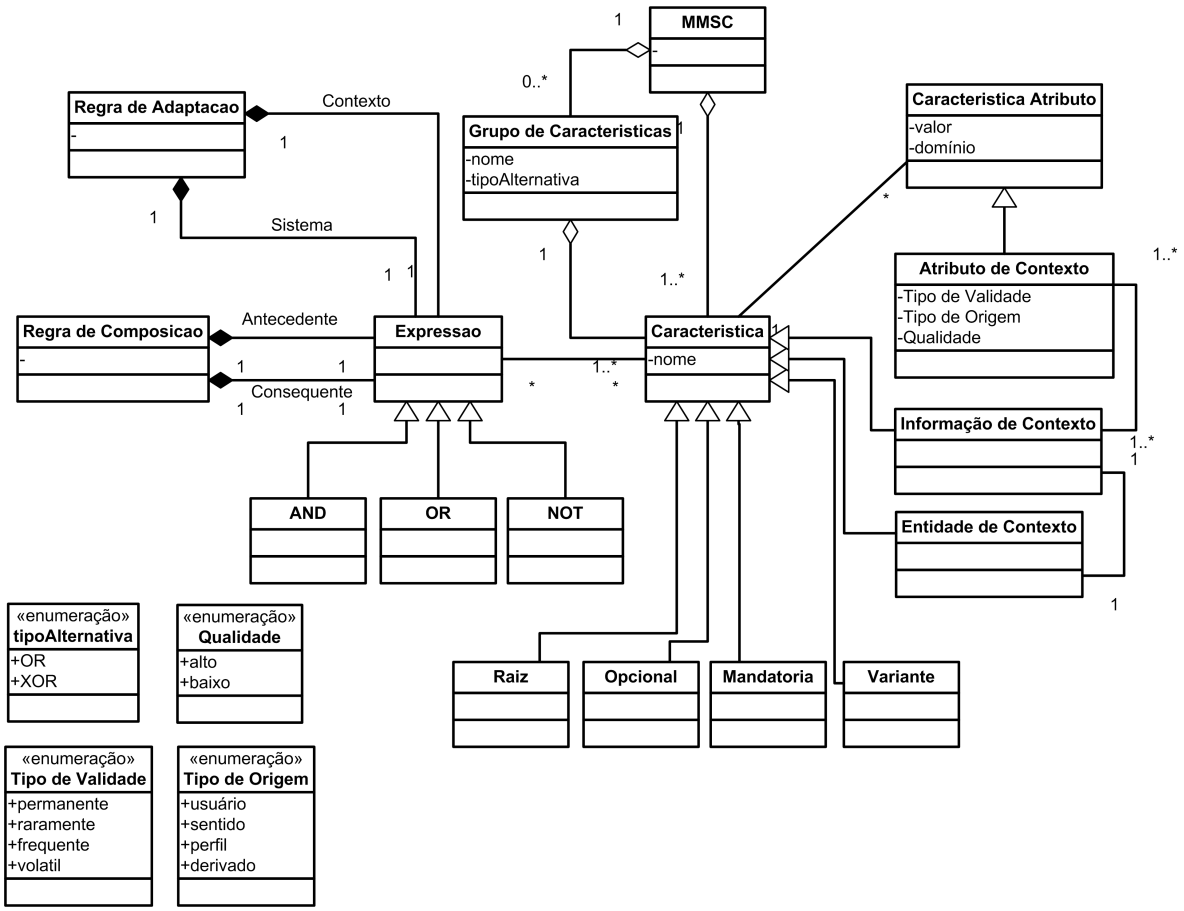

Figura 7.6: Metamodelo do $MMSC$.

Variante, *Mandatoria*, *Opcional* e *Raiz*. A meta-classe *Raiz*, também chamada conceito, representa o domínio modelado. A meta-classe *Opcional* representa as características que podem ou não estar presentes nos produtos derivados. A meta-classe *Mandatoria* representa as características que devem estar presentes em todos os produtos derivados. A meta-classe *Variante* representa as características alternativas da meta-classe *Grupo de Caracteristicas*.

A meta-classe *Entidade de Contexto* representa os elementos de contexto relevantes para o domínio modelado e que podem influenciar na adaptação de um produto de uma $LPMSC$. A meta-classe *Entidade de Contexto* está associada a uma ou mais meta-classes *Informacao de Contexto*, que corresponde a uma ou mais informações relacionada ao elemento de contexto. Portanto, é definida uma associação [1, 1..*] entre as meta-classes *Entidade de Contexto* e *Informacao de Contexto*.

A meta-classe *Informacao de Contexto* está associada à meta-classe *Atributo de Contexto*, implicando que uma informação de contexto pode ter um ou mais atributos que capturam as propriedades de uma informação de contexto. A meta-classe *Atributo de*

Contexto especializa a meta-classe *Caracteristica Atributo* que possui os atributos *valor*, que define o valor da informação de contexto e o atributo que determina o domínio dos valores que esses atributos podem ter.

A meta-classe *Atributo de Contexto* possui três atributos do tipo Enumeração: *Tipo de Validade*, *Tipo de Origem* e *Qualidade*. O atributo *Tipo de Validade* representa a frequência de atualização da informação de contexto e pode assumir os valores: *permanente* (a informação de contexto não sofre atualização), *raramente* (a informação de contexto possui um baixo nível de atualização), *frequente* (a informação de contexto possui um alto nível de atualização, mas em uma frequência menor que volátil) e *volatil* (a informação de contexto possui um alto nível de atualização).

O atributo *Tipo de Origem* representa a origem da informação de contexto e pode assumir os valores: *usuario* (a informação de contexto é informada pelo usuário), *sentida* (a informação de contexto é obtida a partir de um sensor), *perfil* (a informação de contexto é obtida a partir de um perfil) e *derivado* (a informação de contexto é derivada a partir de outra informação de contexto). O atributo *Qualidade* representa o nível de confiabilidade da informação de contexto e pode assumir os valores: *alto* (a informação de contexto possui alto nível de confiabilidade) ou *baixo* (a informação de contexto possui baixo nível de confiabilidade).

A meta-classe *Grupo de Caracteristicas* possui um atributo *tipoAlternativa* que restringe o tipo da seleção das características variantes da meta-classe *Variante*. O atributo *tipoAlternativa* corresponde a uma enumeração e pode assumir os valores $\vee$ ou $\oplus$. O valor $\vee$ implica que uma ou mais meta-classe *Variante* pode ser selecionada para um *Grupo de Características*. O valor $\oplus$ implica que exatamente uma da meta-classe *Variante* deve ser selecionada para um *Grupo de Características*.

As regras de composição [86] definem restrições existentes entre as características e que não são explicitamente expressas no modelo. Uma regra de composição possui uma estrutura do tipo *antecedente + implica + consequente*, onde o *antecedente* e o *consequente* são expressões, que podem ser literais ou booleanas, e denotam uma característica do domínio ou uma combinação entre características. A meta-classe *Regra de Composicao* possui duas composições com a meta-classe *Expressao* que representam a expressão da parte *antecedente* e a expressão da parte *consequente* de uma *RC*. A meta-classe *Expressao*, por sua vez, está associada à meta-classe *Caracteristica*, implicando que as *RCs* são

compostas por expressões que são compostas por características ou atributos de características. As características e atributos das características podem ser combinados por meio dos operadores relacionais ($>$, $<$, $\geq$, $\leq$, $=$ e $\neq$) e dos operadores lógicos ($\neg$, $\vee$ e $\wedge$).

As regras de adaptação capturam as situações que podem disparar uma adaptação no MCS. Essas regras possuem uma estrutura do tipo *evento + condição + ação*, em que o *evento* é representado por uma expressão de *Contexto* e a *ação* é representada por uma expressão de *Sistema*. Uma expressão de *Contexto* corresponde a um valor de uma informação de contexto ou a combinações de valores de informações de contexto que podem ser obtidas a partir dos operadores relacionais ($>$, $<$, $\geq$, $\leq$,$=$ e $\neq$) e operadores lógicos ($\vee$ e $\wedge$). Uma expressão de *Sistema* é formada por uma característica ou uma característica atributo ou uma combinação dessas características de um MCS. Os operadores lógicos utilizados em uma expressão de *Sistema* podem ser do tipo $\wedge$ ou $\neg$. A presença de uma característica ou característica atributo na expressão de *Sistema* indica a sua adição no modelo. No entanto, a presença do operador lógico $\neg$ precedendo uma característica na expressão de *Sistema* indica a remoção da característica do MCS.

A meta-classe *Regra de Adaptacao* possui duas composições com a meta-classe *Expressao* que representam a expressão de *Contexto* e a expressão de *Sistema* de uma *RA*. A meta-classe *Expressao*, por sua vez, está associada à meta-classe *Caracteristica* que é especializada em entidades e informações do contexto.

Construção do perfil

Um perfil UML provê mecanismos de extensão genéricos que têm como objetivo customizar a UML para domínios específicos. Esses mecanismos de extensão permitem refinar a semântica da UML, de modo a manter a semântica da UML padrão, ou seja, os construtores da UML (e.g., classes, associações, dentre outros) podem ser customizados de acordo com a necessidade do usuário, sem, no entanto, deixar de satisfazer as regras e restrições definidas no metamodelo original da UML.

Os perfis são definidos usando *estereótipos*, *tagged values* e *restrições* que são aplicadas a elementos do modelo [111], [112]. Os *estereótipos* permitem estender o vocabulário da UML, de modo a criar novos elementos para o modelo, derivados de elementos existentes, mas que tenham propriedades específicas que sejam adequadas para um domínio de um problema particular ou para um uso específico. Um *tagged value* corresponde a um atributo de um estereótipo. Cada estereótipo possui zero ou mais *tagged values*. Por

fim, as *restrições* representam regras de boa formação para os elementos do metamodelo e sempre que um metamodelo é instanciado, as restrições definidas são verificadas.

Neste trabalho, é apresentado um perfil UML elaborado com base no metamodelo construído para $MMSC$. As meta-classes da UML são identificadas pelo estereótipo ≪metaclasse≫ e os estereótipos específicos do perfil são identificados com o estereótipo ≪ estereótipo ≫. A Figura 7.7 ilustra o perfil proposto para os $MMSCs$.

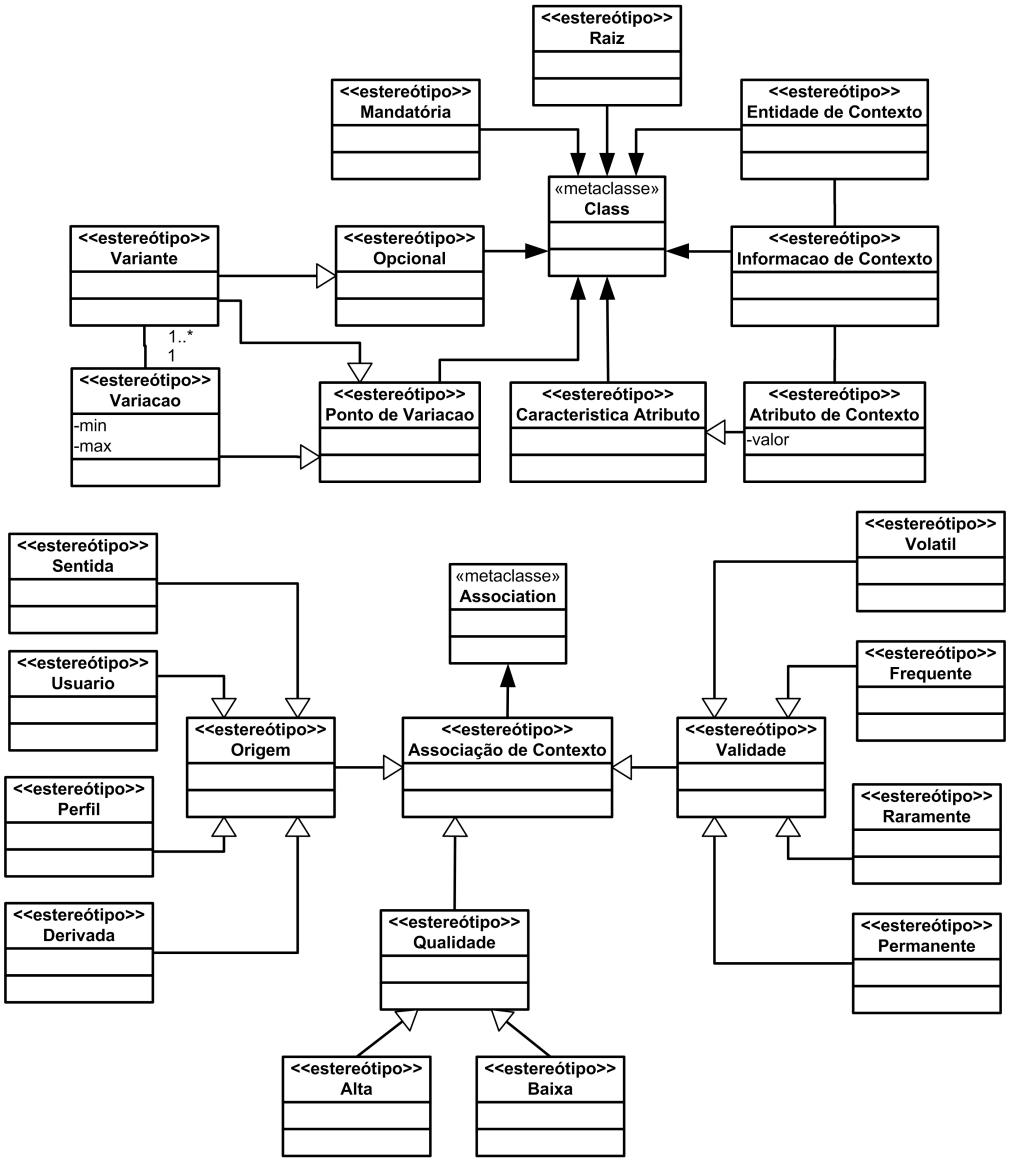

Figura 7.7: Perfil para um $MMSC$.

Os estereótipos ≪Raiz≫, ≪Opcional≫, ≪Mandatoria≫, ≪Ponto de Variacao≫, ≪Caracteristica Atributo≫, ≪Entidade de Contexto≫ e ≪Informacao de Contexto≫ estendem a meta-classe da UML ≪Class≫. O estereótipo ≪Raiz≫ foi definido para representar a raiz do $MMSC$. O estereótipo ≪Opcional≫ foi definido para representar características opcionais e o estereótipo ≪Mandatoria≫ para representar as característi-

cas obrigatórias.

O estereótipo ≪Ponto de Variacao≫ é abstrato e é especializado pelos estereótipos ≪Variante≫ e ≪Variacao≫. Uma ≪Variacao≫ é composta de um ou mais estereótipos do tipo ≪Variante≫ e uma ≪Variante≫ é associada a uma e somente uma ≪Variacao≫. O estereótipo ≪Variante≫ estende o estereótipo ≪Opcional≫, implicando que uma ≪Variante≫ pode ser selecionada ou não durante uma derivação de um produto.

O estereótipo ≪Variacao≫ possui dois *tagged values* (*min* e *max*) que correspondem aos valores mínimo e máximo da cardinalidade associada ao ponto de variação, pois na derivação de um produto, um estereótipo do tipo ≪Variacao≫ pode ser mutuamente exclusivo (uma única ≪Variante≫ deve ser selecionada) ou opcional (uma ou mais ≪Variante≫ podem ser selecionadas). Essas restrições são especificadas no perfil usando restrições OCL.

O estereótipo ≪Entidade de Contexto≫ é usado para modelar elementos de contexto e o estereótipo ≪Informacao de Contexto≫ é usado para capturar as informações de contexto que caracterizam uma entidade de contexto. O estereótipo ≪Atributo de Contexto≫ representa os valores das informações de contexto e possui o *tagged value valor*, que é usado para armazenar o valor do atributo de contexto para cada informação de contexto.

O estereótipo abstrato ≪Associacao de Contexto≫ estende o estereótipo da UML ≪Association≫ e é usado para classificar as associações entre os estereótipos ≪Entidade de Contexto≫ e ≪Informacao de Contexto≫. O estereótipo ≪Associacao de Contexto≫ é especificado pelos estereótipos ≪Origem≫, ≪Qualidade≫, e ≪ Validade ≫. A origem da informação de contexto é representada pelos estereótipos ≪Sentida≫, ≪ Usuario ≫, ≪Perfil≫ e ≪Derivada≫. A validade da informação é representada pelos estereótipos ≪Permanente≫, ≪Raramente≫, ≪Frequente≫ e ≪Volatil≫. A qualidade da informação de contexto é representada pelos estereótipos ≪Alta≫ e ≪Baixa≫, que são especializações do estereótipo ≪Qualidade≫.

Definição dos estereótipos

A OCL (*Object Constraint Language*) [110] é uma linguagem declarativa e tipada, cujos princípios estão baseados na Lógica de Primeira Ordem e na Teoria dos Conjuntos, e pode ser utilizada para diferentes propósitos, dentre os quais destacam-se

[111], [112]:

1. linguagem de consulta;
2. especificação de invariantes em classes e tipos do modelo de classes;
3. especificação de invariantes para estereótipos;
4. Descrição de pré-condições e pós-condições para operações;
5. especificação de restrições para as operações; e
6. especificação de regras de derivação para atributos de qualquer expressão sobre um modelo UML.

A linguagem OCL foi criada para complementar a semântica dos diagramas da UML e definida com o objetivo de ser uma linguagem de especificação baseada em construções semelhantes àquelas encontradas em linguagens orientadas a objetos de modo a facilitar o seu entendimento e uso.

Pelas características 2, 3 e 6 citadas anteriormente, a OCL foi selecionada como uma das linguagens de especificação formal para a validação do PRECISE. Os estereótipos definidos para o perfil UML elaborado foram documentados conforme a estrutura sugerida na especificação da UML [111]. Portanto, para cada estereótipo foi definido um *nome*, uma *classe base* (do metamodelo da UML), uma *descrição* e uma *restrição*. No Quadro 7.1 são apresentados exemplos de estereótipos definidos para as atividades de verificação da boa formação dos $MMSCs$.

Por exemplo, o estereótipo ≪Opcional≫ é definido para a classe base *Class*, logo sempre que esse estereótipo for aplicado a uma classe durante a construção de um modelo, a restrição especificada será verificada. Neste exemplo, a regra de boa formação especificada em OCL restringe que uma classe opcional só pode ser especializada por outra classe opcional. Já no estereótipo ≪Sentida≫, a classe base é *Association*, implicando que esse estereótipo pode ser aplicado somente a construtores do tipo associação. Ao associar o estereótipo ≪Sentida≫ a uma associação, a restrição especificada em OCL verifica se os construtores pertencentes à associação correspondem à uma informação de contexto e um atributo de contexto, respectivamente.

Quadro 7.1: Exemplos de estereótipos para MMSCs

Estereótipo	**Classe Base**	**Descrição**	**Restrição**
Opcional	Class	Um elemento opcional deve ser especializado por um elemento opcional	`inv: self.extension.general → exists (g | g.ownedEnd.type = Opcional) implies self.extension.specific → exists(s | s.ownedEnd.type = Opcional)`
Mandatória	Class	Um elemento mandatório deve ser precedido por um elemento mandatório	`inv: self.extension.specific → exists (s | s.ownedEnd.type = Mandatório) implies self.extension.general → exists(s | s.ownedEnd.type = Mandatório)`
Variação	Class	Um elemento variação deve estar associado a elementos do tipo variante	`inv: self.extension.general → exists (g | g.ownedEnd.type = Variação) implies self.extension.specific → exists(s | s.ownedEnd.type = variante)`
Sentida	Association	Um elemento sentida deve associar informações de contexto e atributos de contexto	`inv: self.extension → exists (g | g.ownedEnd.type = Sentida) implies self.memberEnds → select(m | m.oclIsTypeOf (Informação de Contexto) or select(m | m.oclIsTypeOf (Atributo de Contexto)`

7.2.2 Verificação da consistência

Conforme mencionado no Capítulo 3, estudos na literatura propõem a transformação dos MCs para notações formais, de modo a possibilitar a sua verificação formal utilizando provadores de teorema. Para possibilitar a verificação da consistência do MSC e das regras que o compõem, a especificação formal de um $MMSC$ apresentada no Capítulo 5 é implementada na ferramenta *FixTure* usando Lógica Proposicional. Portanto, antes de apresentar como as verificações são realizadas, é necessário introduzir os conceitos a seguir que são usados nas verificações de consistência realizadas pela *FixTure*.

Um *certificado* C [44] é uma atribuição de valores para as variáveis de uma fórmula proposicional. Uma fórmula proposicional é *satisfazível* sempre que existe um certificado C que atribui um valor *true* para a fórmula proposicional. Se não houver tal certificado, então a fórmula é *não satisfazível*. Nesse contexto, com o objetivo de decidir se uma fórmula proposicional é satisfazível, cinco funções são definidas nesta tese de doutorado: ϕ, σ, γ, γ^{-1} e δ.

A função ϕ transforma qualquer MCS em sua fórmula proposicional correspondente. Essa transformação segue as regras apresentadas em Benavides *et al.* [22]. A função σ trabalha como um provador de teoremas (*Sat Solver*) que tem como entrada uma fórmula proposicional *f*. Se f é satisfazível, a saída da função σ retorna *true*, caso contrário, retorna *false*.

A função γ recebe um conjunto de itens e valores de atribuição, que pode corresponder a uma configuração de um produto ou a uma regra de composição ou a uma expressão de sistema de uma regra de adaptação ou uma regra de composição, como entrada e retorna um certificado C, que corresponde a uma atribuição de valores para uma fórmula proposicional.

A função γ^{-1} é o inverso da função γ. Sua entrada é um certificado C e sua saída é um conjunto de itens que correspondem à configuração de um produto ou a uma expressão de sistema de uma regra de adaptação ou uma regra de composição. Por fim, a função δ recebe como entrada uma fórmula proposicional e um certificado, e retorna *true* se o certificado faz com que a fórmula proposicional seja avaliada como *true* e retorna *false*, caso contrário.

Por exemplo, para um determinado MCS, é possível verificar se um produto derivado PD a partir do MCS é satisfazível usando as funções acima. Primeiramente, o MCS é transformado na sua fórmula proposicional *mcs* usando a função ϕ ($\phi(MCS) = mcs$). Então, um certificado é gerado para PD, que corresponde a uma atribuição de valores para a fórmula proposicional *mcs* usando a função γ ($\gamma(PD) = C$). O certificado C corresponde ao produto derivado PD. Para verificar se PD satisfaz o MCS, a função δ é usada (δ(mcs, C)). Se δ retorna *true*, então PD é satisfazível. Se δ retorna *false*, então PD não é satisfazível.

Consistência do MCS

Para verificar a consistência de um MCS, a ferramenta *FixTure* aplica a função ϕ no MCS e nas RCs que retorna as fórmulas proposicionais correspondentes. Então, a conjunção dessas duas fórmulas é submetida à função σ. Se a função σ retorna *true*, o MCS é avaliado como consistente (deriva ao menos um produto). Caso contrário, o MCS é avaliado como inconsistente (não deriva nenhum produto). Por exemplo, aplicando a função ϕ ao MSC e às RCs ilustrados na Figura 7.8, a seguinte fórmula proposicional f é obtida.

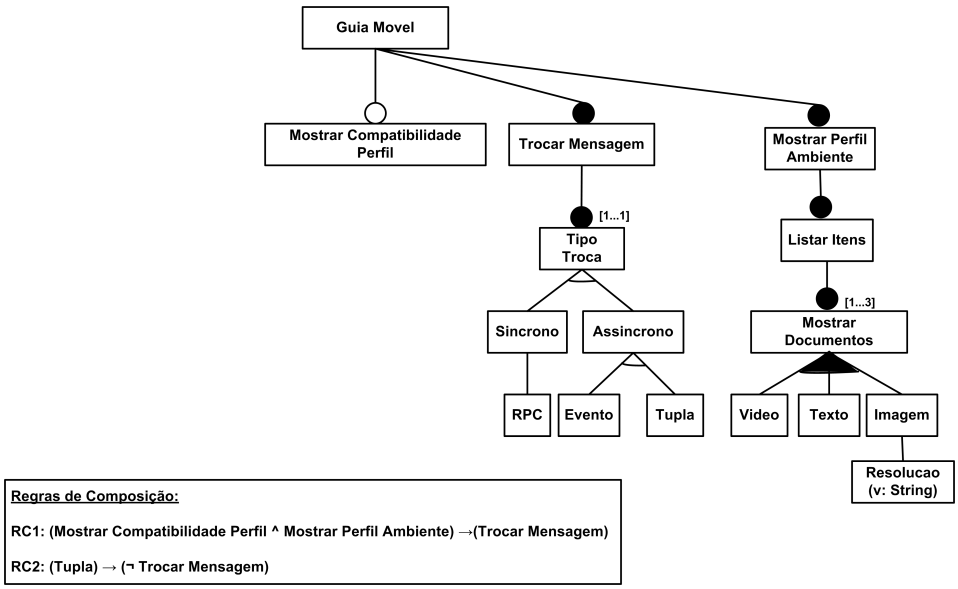

Figura 7.8: Exemplo do MCS do Guia de Visita Móvel e Sensível ao Contexto.

$\phi(MCS) = f$

$f = Guia\ Movel \wedge (Mostrar\ Compatibilidade\ Perfil \rightarrow Guia\ Movel) \wedge$

$(Trocar\ Mensagem \rightarrow Guia\ Movel) \wedge$

$(Mostrar\ Perfil\ Ambiente \rightarrow Guia\ Movel) \wedge$

$(Sincrono \rightarrow Tipo\ Troca) \wedge$

$(Assincrono \rightarrow Tipo\ Troca) \wedge$

$(Mostrar\ Documentos \rightarrow Listar\ Itens) \wedge$

$(Tupla \rightarrow Sincrono) \wedge$

$(Evento \rightarrow Sincrono) \wedge$

$(Procedimento\ Remoto \rightarrow Assincrono) \wedge$

$(Video \rightarrow Mostrar\ Documentos) \wedge$

$(Texto \rightarrow Mostrar\ Documentos) \wedge$

$(Imagem \rightarrow Mostrar\ Documentos) \wedge$

$(Resolucao \rightarrow Imagem) \wedge$

$(Guia\ Movel \rightarrow Trocar\ Mensagem) \wedge$

$(Trocar\ Mensagem \rightarrow Tipo\ Troca) \wedge$

$(Guia\ Movel \rightarrow Mostrar\ Perfil\ Ambiente) \wedge$

$(Mostrar\ Perfil\ Ambiente \rightarrow Listar\ Itens) \wedge$

$(Listar\ Itens \rightarrow Mostrar\ Documentos) \wedge$

$(Tipo\ Troca) \rightarrow (Sincrono \wedge \neg Assincrono \vee \neg Sincrono \wedge Assincrono) \wedge$

$(Sincrono) \rightarrow (Tupla \wedge \neg Evento \vee \neg Tupla \wedge Evento) \wedge$

(Mostrar Documento) $\rightarrow$ *(Video* $\vee$ *Texto* $\vee$ *Imagem)* $\wedge$
(Mostrar Compatibilidade Perfil $\wedge$ *Mostrar Perfil Ambiente)* $\rightarrow$ *(Trocar Mensagem)* $\wedge$
(Tupla) $\rightarrow$ *(*$\neg$*Trocar Mensagem)*

Aplicando a função σ à fórmula f, $\sigma(f) = true$, que implica que o MCS é consistente. Em seguida, a consistência inter-regras é verificada.

Consistência entre RCs

Para verificar a consistência entre as RCs, a ferramenta *FixTure* utiliza a seguinte técnica. Sejam RC_1, RC_2, $\cdots$, RC_n regras de composição. As RCs são consistentes entre si quando a conjunção $RC_1 \bullet RC_2 \bullet \cdots \bullet RC_n$ é consistente, ou seja, $\sigma(RC_1 \wedge RC_2 \wedge \cdots \wedge RC_n) = true$.

Por exemplo, considere as RCs mostradas na Figura 7.3 que correspondem às regras de composição da Figura 7.8. Ao aplicar a função σ à conjunção de RC_1 e RC_2, então σ(((Mostrar Compatibilidade Perfil $\wedge$ Mostrar Perfil Ambiente) $\rightarrow$ (Trocar Mensagem)) $\wedge$ ((Tupla) $\rightarrow$ ($\neg$Trocar Mensagem))) = $true$, portanto, a conjunção é satisfazível e as RCs RC_1 e RC_2 são consistentes entre si.

Consistência entre RAs

Para verificar a consistência entre as RAs, a ferramenta utiliza a seguinte ideia. Sejam RA_1, RA_2, $\cdots$, RA_n regras de adaptação. A conjunção $RA_1 \bullet RA_2 \bullet \cdots \bullet RA_n$ é consistente quando as expressões de contexto são satisfeitas simultaneamente e as expressões de sistema correspondentes são consistentes entre si.

Dessa forma, seja RAs o conjunto de regras de adaptação e ra o $\mathcal{P}_{>1}(RAs)$ (o conjunto dos subconjuntos de RAs com cardinalidade maior que um). Para cada subconjunto ra_i em ra, seja x_i a conjunção das expressões de contexto das regras de adaptação em ra_i. Se $\sigma(x_i)$ retorna $true$, então a conjunção é satisfazível, logo suas expressões de contexto podem ser ativadas simultaneamente. Por exemplo, considere as regras de adaptação do projeto MobiLine ilustradas na Figura 7.4. Nesse exemplo, ra_1 = ((Rede.Latencia $<$ 60 $\rightarrow$ $\neg$Trocar Mensagem) $\wedge$ (Rede.Latencia $<=$ 60 $\rightarrow$ Mostrar Compatibilidade Perfil)) e x_1 = ((Rede.Latencia $<$ 60) $\wedge$ (Rede.Latencia $<=$ 60)). Aplicando a função σ à x_1 ($\sigma(x_1)$), x_1 é avaliada como $true$, logo a conjunção das expressões de contexto é consistente. Portanto, as regras de adaptação RA_1 e RA_2 podem ser ativadas

simultaneamente.

Seja y_i as expressões de sistema relativas às regras de adaptação ativadas em ra_i, logo $y_1 = ((\neg$Trocar Mensagem$) \wedge$ (Mostrar Compatibilidade Perfil)). Aplicando a função σ à y_1 ($\sigma(y_1)$), y_1 é avaliada como *true*, logo a conjunção das expressões de sistema é consistente. Então, as regras de adaptação RA_1 e RA_2 são consistentes entre si. Essa verificação é executada para cada subconjunto de ra.

Consistência entre RCs e RAs

As inconsistências entre as RAs e as RCs sugerem a presença de problemas na composição dos modelos MCS e MCC e, consequentemente, defeitos nos produtos adaptados. Logo, é necessário verificar a consistência entre essas regras. Para isso, as RAs são combinadas com cada RC individualmente. Em outras palavras, seja RAs um conjunto de regras de adaptação e RC uma regra de composição. A conjunção (RAs • RC) é consistente sempre que a conjunção das expressões de sistema das regras de adaptação com cada RC é avaliada como *true*.

Por exemplo, considerando as RAs ilustradas na Figura 7.4 e as RCs da Figura 7.3, então:

$\sigma((\neg$Trocar Mensagem $\wedge$ Mostrar Compatibilidade Perfil$) \wedge ($(Mostrar Compatibilidade Perfil $\wedge$ Mostrar Perfil Ambiente$) \rightarrow$ (Trocar Mensagem$))) = true$

$\sigma((\neg$Trocar Mensagem $\wedge$ Mostrar Compatibilidade Perfil$) \wedge$ (Tupla $\rightarrow \neg$Trocar Mensagem$)) = true$

Portanto, no exemplo considerado, não existem inconsistências entre as RAs e as RCs.

Na atividade **Verificação da Consistência**, a ferramenta *FixTure* verifica a consistência entre as regras que compõem um $MMSC$ e a consistência do MSC. Ao verificar a consistência entre as regras, parte da **fase 2** do PRECISE é validada e, ao verificar a consistência do MSC, a **fase 3** do PRECISE é validada. As fases 2 e 3 estão ilustradas na Figura 4.2 do Capítulo 4.

7.2.3 Verificação dos produtos derivados

Um produto derivado a partir de um $MCSC$ corresponde a uma atribuição de valores *true* ou *false* para as variáveis de cada característica. A configuração do produto é válida ou satisfazível, se e somente se, a sua fórmula é avaliada como *true*. Por exemplo, considere o produto PD ilustrado na Figura 7.9. Aplicando a função γ ao produto PD, é obtido o certificado C a seguir.

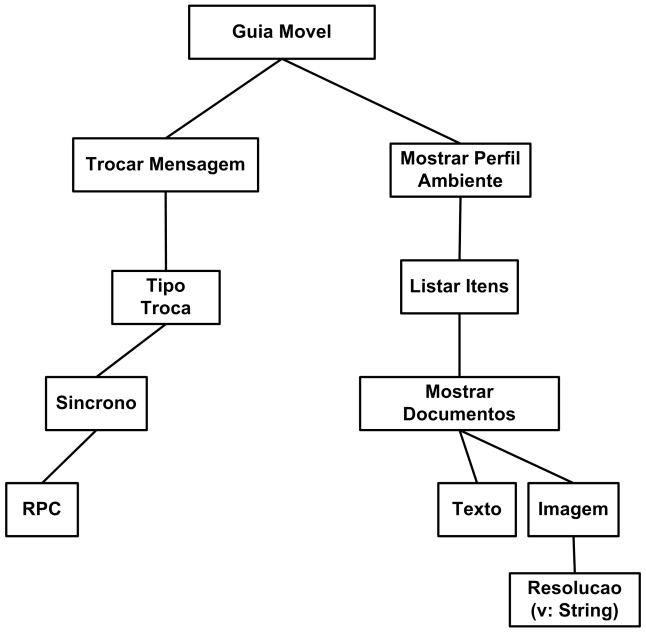

Figura 7.9: Exemplo de um produto derivado do $MMSC$ da Figura 5.1 do Capítulo 5.

$\gamma(PD) = C$

$C = 1 \wedge (0 \rightarrow 1) \wedge (1 \rightarrow 1) \wedge \ (1 \rightarrow 1) \wedge (1 \rightarrow 1) \wedge (0 \rightarrow 1) \wedge (1 \rightarrow 1) \wedge (1 \rightarrow 1) \wedge (0 \rightarrow 1) \wedge (0 \rightarrow 0) \wedge (0 \rightarrow 1) \wedge (1 \rightarrow 1) \wedge (1 \rightarrow 1) \wedge (1 \rightarrow 1)$

O certificado C corresponde à atribuição de valores da fórmula correspondente ao produto derivado PD obtido a partir do $MCSC$ da Figura 7.8. Para verificar se PD satisfaz ao $MCSC$, a função δ é usada. Ao aplicar ($\delta(f, C)$) o valor *true* é obtido, então PD é satisfaz ao MCS da Figura 7.8.

Portanto, na atividade **Verificação dos Produtos Derivados**, a ferramenta *FixTure* verifica a boa formação dos produtos derivados pelo usuário a partir de um $MMSC$, correspondendo à **fase 4** do PRECISE mostrada na Figura 4.2 do Capítulo 4.

7.2.4 Verificação dos produtos adaptados

Para verificar possíveis adaptações dos produtos de um $MMSC$, um processo de simulação foi proposto na **fase 5** do PRECISE, mostrada na Figura 4.2 do Capítulo 4. O principal objetivo desse processo de simulação é antecipar situações que podem levar à adaptações incorretas ou adaptações inseguras dos produtos. Esse processo usa as RAs definidas em tempo de desenvolvimento que estabelecem previamente os cenários de mudança de contexto que podem ocorrer em tempo de execução do produto.

Gerar valores aleatórios para as regras de adaptação e verificar quais regras são ativadas é a solução mais trivial, porém não garante que os valores gerados cobrem o maior escopo possível para as regras de adaptação. Assim, o processo de simulação proposto e implementado na *FixTure* funciona da seguinte forma:

1. É criado o conjunto ra que corresponde ao conjunto dos subconjuntos de RAs com cardinalidade maior que um ($\mathcal{P}_{>1}(RAs)$);
2. Para cada subconjunto ra_i em ra, x_i representa a conjunção das expressões de contexto das regras de adaptação em ra_i;
3. São gerados valores para a conjunção das expressões de contexto x_i;
4. Se $\sigma(x_i) = true$, então a conjunção é satisfazível, o que implica que as expressões de contexto em x_i podem ser ativadas simultaneamente;
5. Para as regras de adaptação relativas a x_i, são extraídas as expressões de sistema (y_i);
6. É verificado se existe alguma inconsistência entre as fórmulas que compõem as expressões de sistema em y_i ($\sigma(y_i)$);
7. Se $\sigma(y_i) = true$, então a conjunção das expressões de sistema é consistente e é criado um certificado C' para y_i usando a função γ ($\gamma((y_i))$);
8. É construída uma junção dos certificados C (que corresponde ao MCS e às RCs do $MMSC$) e C' que consiste da união entre C' e a diferença entre C e C', denominado C''; e
9. A função δ é aplicada à fórmula f do MCS e ao novo certificado C'' ($\delta(f, C'')$). Se o retorno de δ for *true*, o certificado C'' corresponde a uma adaptação segura.

A simulação finaliza quando o número limite de simulações estabelecido previamente pelo Engenheiro de Software é atingido. O principal benefício da simulação está em permitir checar as adaptações dos produtos em tempo de desenvolvimento. A abordagem proposta consegue identificar defeitos associados à cenários predefinidos em tempo de desenvolvimento e não possui a habilidade de criar novos cenários em tempo de execução.

A Figura 7.10 apresenta uma simulação realizada utilizando a ferramenta *FixTure*. Essa simulação foi executada com base no $MMSC$ da $LPMSC$ para Guias de Visita Móveis e Sensíveis ao Contexto do projeto MobiLine.

O processo de simulação inicia com o produto P_0 e devido à mudanças no contexto a adaptação de P_0 para P_1 leva a um produto incorreto. Nesse caso, a simulação retorna para o último produto correto (P_0). Outra adaptação é disparada pela ferramenta de P_0 para P_2. Como essa adaptação é segura, P_2 é selecionado como o último produto correto.

Uma nova adaptação é disparada de P_2 para ele mesmo, que é uma adaptação segura. Prosseguindo com outra adaptação, o processo de simulação chega ao produto P_5 que é uma adaptação segura. Então, uma nova adaptação de P_5 para P_3 é realizada. Essa adaptação leva a um produto incorreto (adaptação insegura), assim o processo retorna para o produto P_5. Quando outra adaptação insegura é disparada de P_5 para P_4, o limite máximo de simulações é atingido e o processo finaliza.

Na atividade **Verificação dos Produtos Adaptados**, a ferramenta *FixTure* verifica a boa formação dos produtos adaptados, correspondendo à **fase 5** do PRECISE mostrada na Figura 4.2 do Capítulo 4.

7.3 Prova de conceito usando Prolog

Nesta seção, a segunda prova de conceito do PRECISE é apresentada. Nessa prova de conceito, todas as fases do PRECISE, com exceção da fase 5, são validadas usando a linguagem Prolog e a ferramenta SWI-Prolog. Para isso, são usados exemplos que correspondem à modificações do $MMSC$ da $LPMSC$ para Guias de Visita Móveis e Sensíveis ao contexto desenvolvido no projeto MobiLine. Nos exemplos criados, defeitos, inconsistências e anomalias foram inseridos em cada fase do PRECISE para que o mesmo fosse

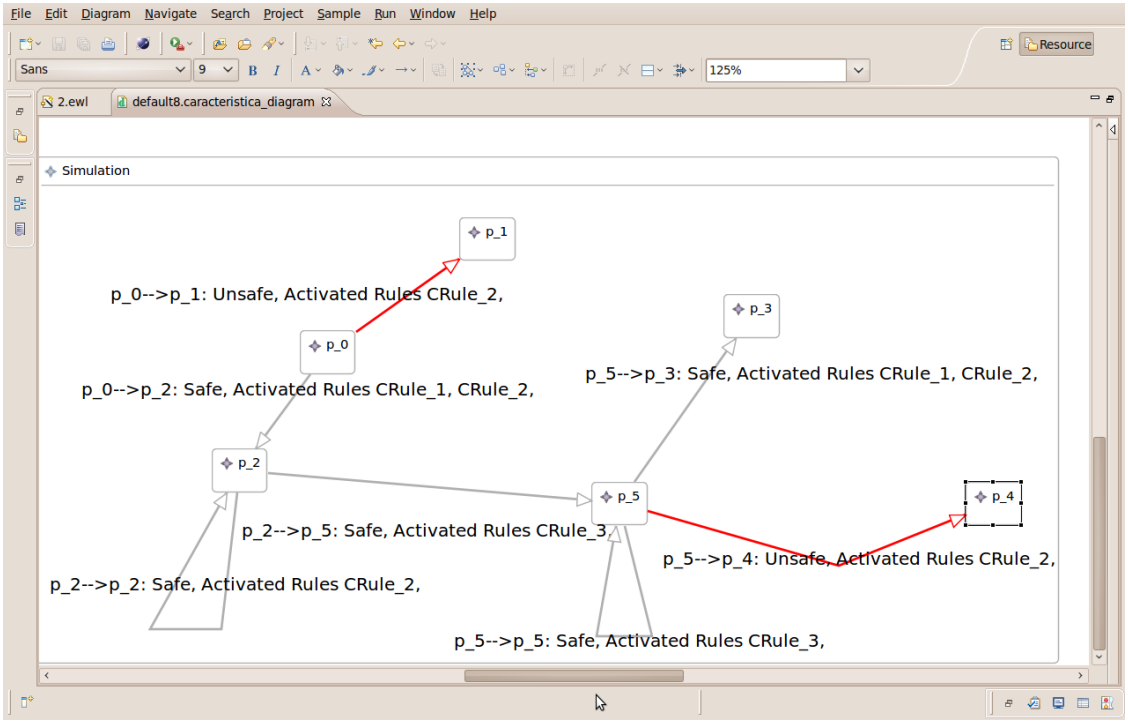

Figura 7.10: Simulação usando a ferramenta *FixTure*.

usado com o objetivo de mostrar que o processo proposto consegue identificar os problemas existentes. A fase 5 não pode ser validada por meio da ferramenta SWI-Prolog porque utiliza o mecanismo de simulação que está implementado apenas na ferramenta *FixTure*.

Nesta tese de doutorado, a linguagem Prolog [38] é selecionada, uma vez que a mesma usa Lógica de Predicados para a inferência e análise dos dados, o que facilita o mapeamento da especificação formal elaborada no Capítulo 5 para a linguagem usada pelo Prolog.

Os predicados correspondentes à especificação de um $MMSC$ são representados por fatos e por relações usadas pelo Prolog e constituem a base de dados que será utilizada durante a verificação. As restrições (e.g., mandatoriedade, cardinalidade, regras de composição, dentre outras) e as propriedades apresentadas no Capítulo 6 são convertidas em consultas sobre essa base de dados. Portanto, a linguagem Prolog verifica se as restrições e propriedades são satisfeitas a partir das consultas elaboradas.

Para testar o uso da linguagem Prolog na verificação da corretude e da consistência dos produtos, foi utilizada a ferramenta SWI-Prolog [142]. A SWI-Prolog oferece um ambiente livre para linguagem Prolog, licenciado sob a Licença Pública *GNU Lesser*. O seu desenvolvimento começou em 1987 e desde então tem sido amplamente utilizado

em pesquisas, bem como em aplicações comerciais.

Devido à extensão da especificação, na Listagem 2, apenas parte da formalização do MCS correto (i.e. já verificado) ilustrado na Figura 7.8 é apresentada, usando a notação de predicados adotada nesta tese de doutorado.

Listagem 2 - MCS da Figura 7.8

```
********** INÍCIO *********
presenteMCS(guiamovel).
presenteMCS(mostrarcompatibilidadeperfil).
presenteMCS(trocarmensagem).
...
presenteMCS(mostrardocumentos).
presenteMCS(texto).
presenteMCS(imagem).

ancestral(guiamovel,mostrarcompatibilidadeperfil).
ancestral(guiamovel,trocarmensagem).
ancestral(guiamovel,mostrarperfilambiente).
...
ancestral(gerenciarcontexto, acesso).
ancestral(gerenciarcontexto, aquisicao).
ancestral(captura, viasensor).
...
ancestral(listaritens, mostrardocumentos).
ancestral(mostrardocumentos, texto).
ancestral(mostrardocumentos, imagem).
ancestral(imagem, resolucao).
atributo(imagem,resolucao).

% Restrições de mandatoriedade
mandatorio(guiamovel, mostrarmapa).
mandatorio(guiamovel, gerenciarcontexto).
mandatorio(guiamovel, trocarmensagem).
```

```
mandatorio(guiamovel, mostrarperfilambiente).
...

% Restrições de cardinalidade
% Alternativo
(presenteMCS(viasensorexterno),(not(presenteMCS(viasensor))));
(presenteMCS(viasensor),(not(presenteMCS(viasensorexterno)))).

(presenteMCS(sincrono),(not(presenteMCS(assincrono))));
(presenteMCS(assincrono),(not(presenteMCS(sincrono)))).

(presenteMCS(tupla),(not(presenteMCS(evento))));
(presenteMCS(evento),(not(presenteMCS(tupla)))).

% Opcional
(presenteMCS(video);presenteMCS(texto);presenteMCS(imagem)).

% Regras de composição
RC1 = presenteMCS(mostrarmapa),presenteMCS(mostrarcompatibilidadeperfil),
presenteMCS(mostrarperfilambiente),presenteMCS(gerenciarcontexto).

RC2 = presenteMCS(mostrarmapa),presenteMCS(mostrarcompatibilidadeperfil),
presenteMCS(mostrarperfilambiente),presenteMCS(trocarmensagem).

RC3 = (presenteMCS(viasensorexterno),(not(presenteMCS(viasensor))));
(presenteMCS(viasensor),(not(presenteMCS(viasensorexterno)))).

% Regras de adaptação
RA1 = presenteMCS(listaritens):-((usuario.localizacao.v =:= ambiente),
                                 (usuario.perfil.v <= 0,6 * perfil)).
RA2 = presenteMCS(Sintatico):-(Usuario.Localizacao.v =:= Ambiente).
********** FIM *********
```

Na especificação mostrada na Listagem 2, por exemplo, temos o fato *presenteMCS*

e a relação *ancestral*. Além disso, são mostradas as regras que definem as restrições de cardinalidade (*xor*, *or*), as regras de composição *rc* e as regras de adaptação *ra*.

7.3.1 Fase 1 - Verificar os modelos

Na **fase 1**, o PRECISE verifica a corretude do *MCS* e a corretude do *MCC*. Considere os exemplos com defeitos inseridos ilustrados nas Figuras 7.11 e 7.12. Esses exemplos estão formalizados na Listagem 3 e na Listagem 4, respectivamente.

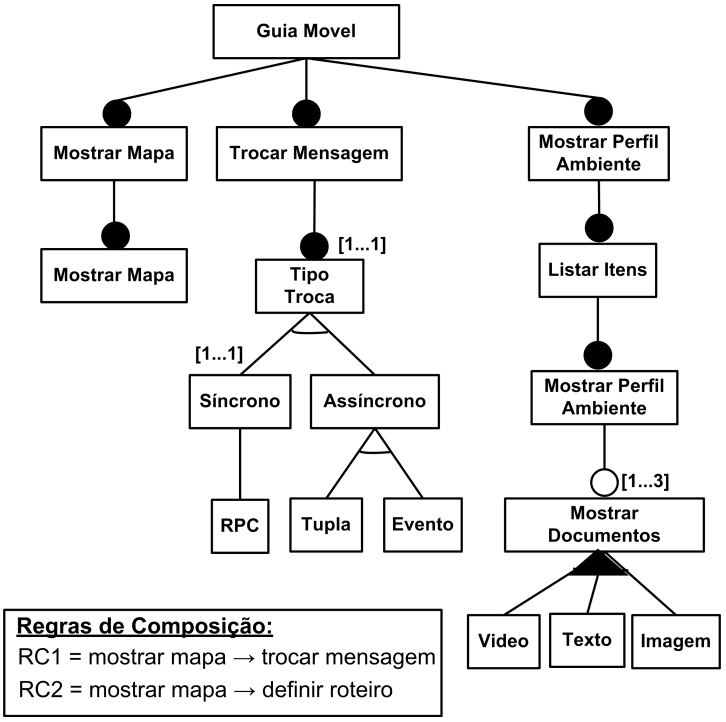

Figura 7.11: Detecção de defeitos de um *MCS* na Fase 1 do PRECISE.

Listagem 3 - Defeitos em um MCS - Fase 1 do PRECISE

```
********** INÍCIO *********
presenteMCS(guiamovel).
presenteMCS(mostrarmapa).
presenteMCS(trocarmensagem).
presenteMCS(mostrarperfilambiente).
presenteMCS(mostrarmapa).
presenteMCS(tipotroca).
presenteMCS(sincrono).
presenteMCS(assincrono).
```

```
presenteMCS(tupla).
presenteMCS(evento).
presenteMCS(procedimentoremoto).
presenteMCS(listaritens).
presenteMCS(mostrarperfilambiente).
presenteMCS(mostrardocumentos).
presenteMCS(video).
presenteMCS(texto).
presenteMCS(imagem).

ancestral(guiamovel,mostrarmapa).
ancestral(guiamovel,trocarmensagem).
ancestral(guiamovel,mostrarperfilambiente).
ancestral(trocarmensagem,tipotroca).
ancestral(tipotroca,sincrono).
ancestral(tipotroca,asssincrono).
ancestral(sincrono,tupla).
ancestral(sincrono,evento).
ancestral(assincrono,procedimentoremoto).
ancestral(mostrarperfilambiente,listaritens).
ancestral(listaritens,mostrarperfilambiente).
ancestral(mostrarperfilambiente,mostrardocumentos).
ancestral(mostrardocumentos,video).
ancestral(mostrardocumentos,texto).
ancestral(mostrardocumentos,imagem).
********** FIM *********

********** INÍCIO - PROPRIEDADES *********
%O produto satisfaz às propriedades de boa formação do MCS?
%PBF1: Duas características mandatórias ou opcionais não podem ter o mesmo nome:
pbf1:- bagof(Y, presenteMCS(Y), Z),
       setof(Y, presenteMCS(Y), R),
       tam(Z,T1), tam(R,T2), \+ T1==T2.
```

```
tam([],0).
tam([_|L1],N):- tam(L1,N1), N is N1 + 1.

%PBF3: Uma característica não pode ser descendente de si mesma:
pbf3(X,Y):- ancestral(X,Y),(not(diferente(X,Y))).
********** FIM - PROPRIEDADES *********
```

Ao executar as consultas $pbf1$ e $pbf3$ sobre a base de dados correspondente ao MCS apresentado na Figura 7.11, os resultados obtidos mostram que:

- a propriedade PBF1 não é satisfeita. No caso da PBF1, significa que o modelo possui características mandatórias ou opcionais com o mesmo nome; e
- a propriedade PBF3 não é satisfeita, pois a consulta associada (pbf3) exibe a característica (*mostrar mapa*), implicando que existe um relacionamento em que uma característica foi modelada como descendente de si mesma.

```
********** INÍCIO CONSULTAS *********
?- pbf1.
true.
?- pbf3(X,Y).
X = Y, Y = mostrarmapa;
true.

********** FIM CONSULTAS *********
```

Listagem 4 - Defeitos em um MCC - Fase 1 do PRECISE

```
********** INÍCIO *********
ec(dispositivo).
ic(memoria).
ac(vm).
ac(oM).
ac(aM).
```

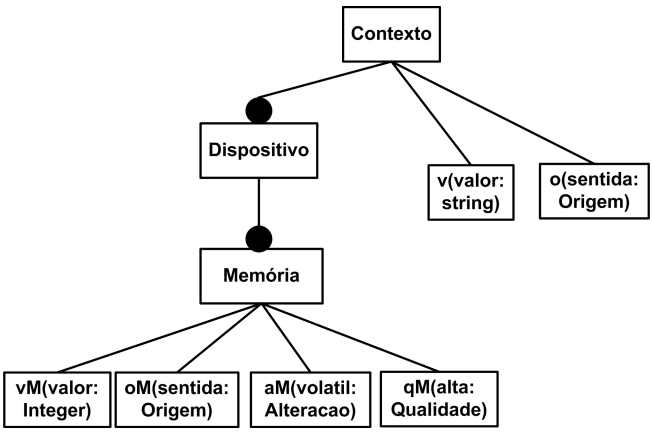

Figura 7.12: Detecção de defeitos de um MCC na Fase 1 do PRECISE.

```
ac(qM).
ac(v).
ac(o).

ancestral(contexto,dispositivo).
ancestral(contexto,v).
ancestral(contexto,o).
ancestral(dispositivo,memoria)
ancestral(memoria,vm).
ancestral(memoria,om).
ancestral(memoria,am).
ancestral(memoria,qm).
********** FIM *********

********** INÍCIO - PROPRIEDADES *********
%O produto satisfaz às propriedades de boa formação do MCC?
%PBF24: As características do segundo nível da árvore representam
%entidades de contexto:
pbf24:- bagof(Y, ec(Y), Z),
        bagof(Y, ancestral(contexto, ec(Y)), R),
        tam(Z,T1), tam(R,T2), \+ T1==T2.
tam([],0).
tam([_|L1],N) :- tam(L1,N1), N is N1 + 1.
```

```
%PBF26:As características do quarto nível da árvore representam
%os atributos de contexto e possuem tipos pré-definidos para valor,
%origem, qualidade e frequência de alteração:
pbf26:- bagof(Y, ac(Y), Z),
        bagof(Y, ancestral(ic(Y),X), R),
        tam(Z,T1), tam(R,T2), \+ T1==T2.
********** FIM - PROPRIEDADES *********
```

Ao executar as consultas $pbf24$ e $pbf26$ sobre a base de dados correspondente ao MCC apresentado na Figura 7.12, os resultados obtidos mostram que:

- a consulta associada à propriedade PBF24 verifica se o número de características no segundo nível é diferente do número de entidades de contexto. Como a consulta é avaliada como *true*, implica que a propriedade PBF24 não é satisfeita, pois existem características no segundo nível que não são entidades de contexto; e
- a consulta associada à propriedade PBF26 verifica se o número de características no quarto nível é diferente do número de atributos de contexto. Como a consulta é avaliada como *true*, implica que a propriedade PBF26 não é satisfeita, pois existem características no quarto nível que não são atributos de contexto.

```
********** INÍCIO CONSULTAS *********

?- pbf24.
true.
?- pbf26.
true.

********** FIM CONSULTAS *********
```

7.3.2 Fase 2 - Verificar regras

Na **fase 2**, o PRECISE checa a corretude e a consistência das regras de composição e das regras de adaptação. Considere o MCS com defeitos inseridos mostrado na Figura

7.11 e sua formalização na Listagem 3. A listagem 5 apresenta a formalização das *RCs* ilustradas na Figura 7.11.

Listagem 5 - Defeitos nas RCs - Fase 2 do PRECISE

```
********** INÍCIO *********
antecedente(mostrarmapa).
consequente(definirroteiro).
consequente(trocarmensagem).
********** FIM *********

********** INÍCIO - PROPRIEDADES *********
tam([],0).
tam([_|L1],N) :- tam(L1,N1), N is N1 + 1.
membro(X,[X|_]).
membro(X,[_|C]):- membro(X,C).
intersec([X | Y], L, [X |Z]):- membro(X, L), intersec(Y, L, Z).
intersec([_ |X], L, Y):- intersec(X, L, Y).
intersec(_, _, []).
%PBF9: uma características opcional ou característica
%atributo em uma expressão antecedente
%ou em uma expressão consequente devem pertencer a um MCS:
pbf9:- setof(P,presenteMCS(P),Z),setof(A,antecedente(A),R),
       intersec(R,Z,L),tam(R,TR),
       tam(L,TL), TR=\=TL,
       setof(C,consequente(C),Y),
       intersec(Y,Z,F),tam(Y,TY),
       tam(F,TF), TY=\=TF.
********** FIM - PROPRIEDADES *********
```

Ao executar a consulta $pbf9$ sobre a base de dados correspondente ao MCS apresentado na Figura 7.11 e as suas respectivas RCs, o resultado obtido mostra que:

- a propriedade PBF9 não é satisfeita, pois ao calcular a interseção entre o conjunto de características na expressão antecedente com o conjunto de características do MCS,

verifica-se que os conjuntos são diferentes, logo alguma característica da expressão antecedente da RCs (no exemplo, *definir roteiro*) não está presente no MCS.

```
********** INÍCIO - CONSULTA *********

?- pbf9.
true.

********** FIM - CONSULTA *********
```

7.3.3 Fase 3 - Verificar a consistência do MCS

Na **fase 3**, o PRECISE checa a consistência do MCS que implica em verificar se algum produto pode ser derivado a partir do MCS. Considere o MCS e as RCs mostrados na Figura 7.11. A Listagem 6 apresenta a formalização da consulta utilizada para mostrar que o PRECISE identifica a inconsistência inserida pela regra de composição RC2.

Listagem 6 - Inconsistência em um MCS - Fase 3 do PRECISE

```
********** INÍCIO - PROPRIEDADE *********
%O MCS é consistente?
consistente:- presenteMCS(guiamovel),
presenteMCS(mostrarmapa),
presenteMCS(trocarmensagem),
presenteMCS(mostrarperfilambiente),
presenteMCS(mostrarmapa),
presenteMCS(tipotroca),
presenteMCS(sincrono),
presenteMCS(assincrono),
presenteMCS(tupla),
presenteMCS(evento),
presenteMCS(procedimentoremoto),
presenteMCS(listaritens),
```

```
presenteMCS(mostrarperfilambiente),
presenteMCS(mostrardocumentos),
presenteMCS(video),
presenteMCS(texto),
presenteMCS(imagem),
(not(presenteMCS(mostrarmapa)); presenteMCS(trocarmensagem)),
((not(presenteMCS(mostrarmapa));presenteMCS(definirroteiro))).
********** FIM - PROPRIEDADE *********
```

Ao executar a consulta *consistente* apresentada na Listagem 6 sobre a base de dados correspondente ao MCS e suas respectivas RCs na Figura 7.11, o resultado obtido mostra que:

- o MCS não é consistente, pois ao checar a conjunção das características do modelo com as RCs o resultado avaliado é *false*, implicando que alguma regra de composição está inconsistente com o modelo.

```
********** INÍCIO - CONSULTA *********

?- consistente.
false.

********** FIM - CONSULTA *********
```

7.3.4 Fase 4 - Verificar o produto derivado

Na **fase 4**, o PRECISE checa a corretude de um produto derivado PD a partir de um $MMSC$. O PD ilustrado na Figura 7.13, gerado a partir do MCS mostrado na Figura 7.8 e formalizado na Listagem 7, possui defitos e inconsistências com relação às RCs e restrições de mandatoriedade e cardinalidade do MCS mostrado na Figura 7.8.

Listagem 7 - Defeitos em um PD - Fase 4 do PRECISE

```
********** INÍCIO *********
```

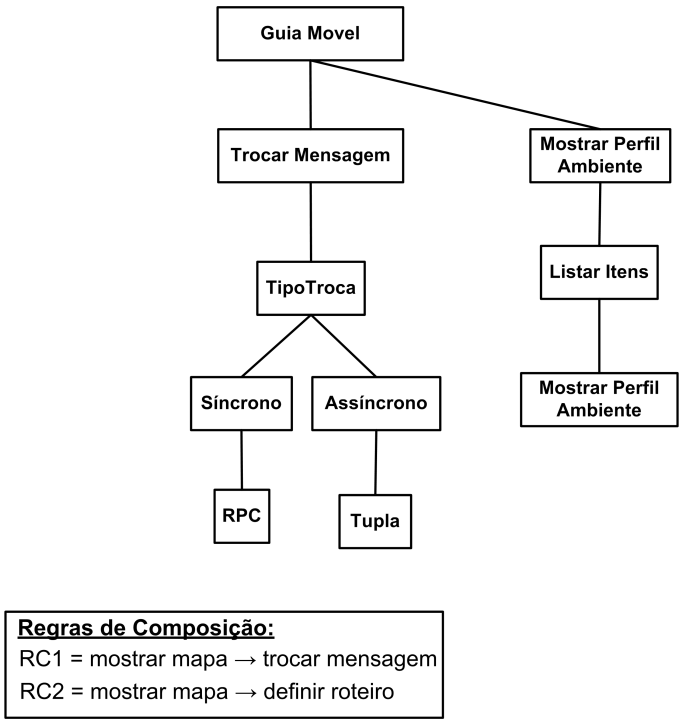

Figura 7.13: Detecção de defeitos da Fase 4 do PRECISE.

```
%Características selecionadas para o PD:
presenteMCS(guiamovel).
presenteMCS(trocarmensagem).
presenteMCS(mostrarperfilambiente).
presenteMCS(tipotroca).
presenteMCS(sincrono).
presenteMCS(assincrono).
presenteMCS(tupla).
presenteMCS(procedimentoremoto).
presenteMCS(listaritens).
presenteMCS(mostrarperfilambiente).

%Relacionamentos:
ancestral(guiamovel,trocarmensagem).
ancestral(guiamovel,mostrarperfilambiente).
ancestral(trocarmensagem,tipotroca).
ancestral(tipotroca,sincrono).
ancestral(tipotroca,asssincrono).
ancestral(sincrono,tupla).
```

```
ancestral(assincrono,procedimentoremoto).
ancestral(mostrarperfilambiente,listaritens).
ancestral(listaritens,mostrarperfilambiente).
********** FIM *********
```

Ao executar as consultas apresentadas na Listagem 7 sobre a base de dados correspondente ao PD da Figura 7.13, o resultado obtido mostra que:

- o PD não satisfaz às restrições de mandatoriedade, pois existem consultas avaliadas como *false*); e
- o PD não satisfaz a primeira restrição de cardinalidade, pois a consulta é avaliada como *false*.

O produto satisfaz as restrições de mandatoriedade?

```
********** INÍCIO - CONSULTAS *********
mandatorio(X,Y):- presenteMCS(X), presenteMCS(Y).
?- mandatorio(guiamovel,mostrarmapa).
false.
?- mandatorio(guiamovel,trocarmensagem).
true.
?- mandatorio(guiamovel,mostrarperfilambiente).
true.
?-mandatorio(mostrarmapa,mostrarmapa).
false.
?-mandatorio(trocarmensagem,tipotroca).
true.
?-mandatorio(mostrarperfilambiente,listaritens).
true.
?-mandatorio(listaritens,mostrarperfilambiente).
true.
********** FIM - CONSULTAS *********
```

O produto satisfaz as restrições de cardinalidade alternativas?

```
********** INÍCIO - CONSULTAS *********
?-(presenteMCS(sincrono),(not(presenteMCS(assincrono)))),
  (presenteMCS(assincrono),(not(presenteMCS(sincrono)))).
false.
?-(presenteMCS(tupla),(not(presenteMCS(evento)))),
  (presenteMCS(evento),(not(presenteMCS(tupla)))).
true.
********** FIM - CONSULTAS *********
```

O produto satisfaz as restrições de cardinalidade opcionais?

```
********** INÍCIO - CONSULTA *********
?-(presenteMCS(video);presenteMCS(texto);presenteMCS(imagem)).
true.
********** FIM - CONSULTA *********
```

O produto satisfaz as regras de composição?

```
********** INÍCIO - CONSULTAS *********
?-(not(presenteMCS(mostrarmapa)); presenteMCS(trocarmensagem)).
true.
?-((not(presenteMCS(mostrarmapa));presenteMCS(definirroteiro))).
true.
********** FIM - CONSULTAS *********
```

7.4 Considerações finais

Na validação do PRECISE, são usadas a *FixTure* e a SWI-Prolog que implementam as fases do PRECISE utilizando como base a especificação formal, as propriedades definidas nesta tese de doutorado. Para ilustrar o aspecto genérico da especificação formal proposta, linguagens de especificação distintas foram utilizadas durante a validação. Na *FixTure*, a especificação formal e as propriedades são mapeadas para a notação formal do Metamodelo da UML, para restrições OCL e para Lógica Proposicional. Já na ferramenta SWI-Prolog, a especificação formal e as propriedades são mapeadas para a linguagem Prolog.

Quadro 7.2: Ferramentas para construção e análise dos modelos de características

Ferramenta	**Notação**	**Especificação Formal**	**Sensibilidade ao Contexto**
RequiLine	FODA		
pure::variants	FODA	Prolog	
AmiEddi	Czarnecki		
CaptainFeature	FODA		
FeaturePlugin	Czarnecki	OCL	
XFeature	Czarnecki		
FeatureIDE	FODA	Lógica proposicional	
FeatureMapper	Czarnecki	OCL	
UbiFEX	Odyssey-FEX		Modelo de Contexto
PRECISE	**Czarnecki**	**Lógica de Primeira Ordem**	**MCC**

Na literatura, existem diversas ferramentas desenvolvidas para dar suporte para a construção de modelos de características corretos e consistentes, e para a configuração correta desses modelos. Por exemplo, [100], [27], [6], [43], [89], [77] e [64]. Essas ferramentas foram usadas como base para o desenvolvimento da pesquisa desta tese de doutorado. O Quadro 7.2 apresenta as características de cada ferramenta.

De acordo com o Quadro 7.2, é possível observar que as ferramentas analisadas utilizam a notação FODA [86] ou a notação baseada em cardinalidades proposta por Czarnecki *et al.* [48], com exceção da UbiFEX que adota uma notação própria denominada Odyssey-FEX. Outro ponto de destaque nas ferramentas apresentadas é que a maioria dos critérios de verificação implementados não são especificados formalmente. Além disso, o Quadro 7.2 mostra que uma área de pesquisa pouco explorada é a de modelos de características sensíveis ao contexto, uma vez que garantir a corretude e a consistência de um $MMSC$ é uma tarefa complexa, ressaltando a principal contribuição e inovação desta tese de doutorado.

Capítulo 8

Conclusão

Nesta tese de doutorado, foi proposto um processo de verificação baseado em definições rigorosas e com alto nível de abstração para fornecer suporte para a análise de LPMSCs, mais especificamente de MMSCs.

Neste último Capítulo, é apresentada uma visão geral da tese, na Seção 8.1, em seguida, na Seção 8.2, a hipótese deste trabalho de pesquisa é analisada. Na Seção 8.3, são destacadas as principais contribuições desta tese de doutorado e os resultados alcançados. Além disso, na Seção 8.4, alguns trabalhos futuros identificados são discutidos.

8.1 Visão geral do trabalho

No decorrer do desenvolvimento desta tese de doutorado foi essencial o entendimento do conceito de software móvel e sensível ao contexto. Conforme ressaltado no Capítulo 1, o desenvolvimento desse tipo de software ainda apresenta desafios para a Engenharia de Software devido a requisitos como dinamicidade, mobilidade, sensibilidade ao contexto e adaptabilidade. Além disso, as limitações dos dispositivos móveis (e.g., capacidade de processamento, bateria, dentre outros) também precisam ser consideradas. Nesse cenário, o reúso sistemático propiciado pelas LPSs mostra-se uma solução eficiente para o desenvolvimento dessas aplicações. No entanto, o domínio móvel e sensível ao contexto apresenta requisitos não cobertos pelas LPSs tradicionais. Por esse motivo, neste trabalho de pesquisa, foram utilizadas as LPMSCs que implicam no uso do paradigma das LPSs para construir aplicações móveis e sensíveis ao contexto. Os produtos de uma LPMSC podem sofrer adaptações quando mudanças ocorrem no ambiente em que o produto está inserido.

É importante ressaltar que a criação de um modelo para as LPMSC exigiu incluir a modelagem do contexto e do impacto desse contexto nos produtos derivados. Dessa forma, foram descritas técnicas de modelagem específicas para as LPMSCs, dentre

as quais destacou-se o MC.

Na criação das etapas, atividades, entradas e saídas do PRECISE também foi essencial analisar as operações de análise e abordagens para verificação da corretude dos MCs disponíveis na literatura. Os trabalhos analisados foram a base para identificar as atividades e operações necessárias para o desenvolvimento do processo para checagem da corretude e consistência dos MMSCs.

Com o objetivo de formalizar os elementos do MMSC, as operações e as propriedades utilizadas pelo PRECISE, de modo a evitar ambiguidades no seu uso, percebeu-se a importância da elaboração de uma especificação formal a partir de uma revisão bibliográfica que incluiu várias técnicas de modelagem de características e de modelagem do contexto.

Em seguida, foi necessário validar como a especificação formal e as propriedades definidas para identificar defeitos, anomalias e inconsistências existentes nos MMSCs foram usadas no PRECISE. Durante a validação, foi possível mostrar a flexibilidade da especificação formal elaborada, pois foram realizadas duas provas de conceito, utilizando linguagens de especificação e ferramentas distintas. Na primeira prova de conceito, foi utilizada a ferramenta *FixTure* implementada no contexto de uma dissertação de mestrado do grupo de pesquisa GREat. Essa ferramenta automatizou as atividades de verificação dos MMSCs e utilizou a especificação formal apresentada nesta tese de doutorado no seu desenvolvimento, de modo que a ferramenta tornou-se um reflexo da especificação formal proposta. A ferramenta *FixTure* utilizou as linguagens de especificação formal OCL e Lógica Proposicional.

Na segunda prova de conceito, decidiu-se utilizar a linguagem Prolog em conjunto com a ferramenta SWI-Prolog. Nessa segunda prova de conceito, as bases de dados e consultas usadas para validar o PRECISE foram construídas manualmente e submetidas para a ferramenta SWI-Prolog que executava as consultas e retornava se o modelo verificado estava correto.

Para exemplificar os conceitos apresentados no decorrer desta tese, foi utilizada uma LPMSC para Guias de Visita Móveis e Sensíveis ao Contexto desenvolvida pelo GREat do Departamento de Computação da Universidade Federal do Ceará, em parceria com o Grupo de Reutilização de Software do Programa de Engenharia de Sistemas e Computação da Universidade Federal do Rio de Janeiro, no contexto do projeto MobiLine.

8.2 Análise da hipótese de pesquisa

Retomando a hipótese levantada no Capítulo 1, é possível compreender a importância em garantir a qualidade dos modelos de características móveis e sensíveis ao contexto frente a proposta apresentada nesta tese. Considerando as análises realizadas na execução da validação, a seguir são apresentadas evidências que comprovam essa hipótese.

Hipótese: A criação e utilização de um processo de verificação formal para modelos de características móvel e sensível ao contexto aumenta a qualidade dos elementos que compõem esses modelos em termos de corretude e consistência dos mesmos.

Resultado: Aceita. Foi constatado a partir das provas de conceito realizadas durante a validação que o processo de verificação formal PRECISE consegue identificar defeitos e inconsistências existentes nos modelos de características móveis e sensíveis ao contexto, assegurando, assim um aumento na qualidade da construção desses modelos e, consequentemente, das linhas de produtos de software móveis e sensíveis ao contexto.

8.3 Contribuições e resultados alcançados

O modelo de variabilidades é um dos principais artefatos da Engenharia de uma LPS e, portanto, a análise desse modelo nos primeiros estágios do processo de desenvolvimento é essencial para o sucesso da LPS. Sendo assim, a principal contribuição desta tese de doutorado consiste na criação do processo de verificação formal PRECISE que identifica defeitos existentes nos MMSCs, promovendo uma melhoria na corretude das LPMSCs e dos PDs gerados a partir delas.

Em resumo, o PRECISE pode ser visto como uma solução que satisfaz os seguintes requisitos:

- corretude dos modelos e regras que compõem um MMSC por meio das propriedades de boa formação;
- consistência entre as regras de um MMSC por meio das propriedades de consistência;
- consistência de um MMSC, verificando a possibilidade de configurar ao menos um

produto a partir desse modelo;

- corretude dos produtos derivados a partir de um MMSC por meio das propriedades de boa formação; e
- minimizar a ocorrência de defeitos nos produtos adaptados devido a mudanças no contexto, usando o processo de simulação proposto.

Além do processo propriamente dito, um outro resultado mais específico desta tese de doutorado está na especificação formal usada pelo processo de verificação que foi construída usando Lógica de Primeira Ordem. Esse fato trouxe benefícios em termos de rigor e clareza para o processo, uma vez que o mesmo utiliza a notação matemática proposta que aborda explicitamente os aspectos semânticos dos MMSCs. Vale a pena ressaltar que essa especificação formal incorporou conceitos ainda não formalizados na literatura relativa aos MCs, tais como cardinalidade e característica atributo, bem como conceitos relacionados com a modelagem do contexto, por exemplo, entidades de contexto, informações de contexto e atributos de contexto.

Correlacionado com a especificação formal, outro resultado importante obtido nesta tese está na formalização das propriedades para MMSC. Essas propriedades abordam critérios de boa formação dos elementos que compõem um MMSC e regras de consistência entre esses elementos. Além disso, é importante destacar que as situações que sugerem a presença de anomalias nos MMSCs também foram formalizadas.

O processo de simulação proposto foi validado e conseguiu detectar e prevenir defeitos nos produtos adaptados devido à mudanças no contexto em que o produto estava inserido. O fato desse processo de simulação garantir a corretude de uma variedade de situações que podem levar à reconfigurações dos produtos, também, pode ser considerado um resultado importante desta pesquisa de doutorado.

Alguns resultados apresentados nesta tese de doutorado só foram obtidos devido à cooperação com o grupo de pesquisa GREat, mais especificamente durante a realização do projeto MobiLine que possibilitou o entendimento e fixação dos conceitos e do processo de desenvolvimento de uma LPMSC.

Os resultados obtidos no decorrer do desenvolvimento desta tese foram publicados na forma de artigos científicos em conferências nacionais e internacionais, periódicos e relatórios técnicos. O Quadro 8.1 apresenta as publicações produzidas, incluindo arti-

gos publicados e artigos aceitos que ainda se encontram em revisão. O primeiro artigo apresenta as ideias iniciais desta tese de doutorado e foi apresentado em um simpósio de doutorado. Esse artigo propõe o uso de perfis UML enriquecidos com restrições especificadas em OCL para verificar MMSC.

Os resultados descritos no segundo, terceiro e quarto artigos não fazem parte das contribuições específicas dessa tese de doutorado, mas descrevem resultados concretos do desenvolvimento do projeto MobiLine. O segundo artigo apresenta o processo de desenvolvimento usado para criar uma LPMSC e configurar um produto a partir dessa linha. O terceiro artigo discute o modelo da arquitetura criado para possibilitar o desenvolvimento da LPMSC do MobiLine. O quarto artigo corresponde a uma extensão do segundo artigo, uma vez que este foi selecionado entre os melhores artigos da conferência (*14th International Software Product Line Conference*) e convidado para ser submetido para o periódico *Science of Computer Programming*.

O quinto artigo descreve a versão inicial do processo de verificação proposto nesta tese de doutorado e, por fim, o sexto artigo apresenta uma das técnicas adotadas para prevenir defeitos na adaptação dos produtos por meio da combinação das regras de adaptação e regras de composição de um MMSC.

Além dessas publicações, um relatório técnico resultou de uma das fases do processo de qualificação realizado no decorrer deste doutorado. Esse relatório descreve e avalia o uso de diversas técnicas e métodos formais na Engenharia de Software e está sendo revisado para ser disponibilizado na biblioteca da Universidade Federal do Ceará.

8.4 Trabalhos futuros

Nesta pesquisa de doutorado foi constatado que, atualmente, o PRECISE cobre uma parte das operações identificadas na literatura, sendo necessário aumentar o escopo das operações tratadas. Outro aspecto a ser melhorado diz respeito à descoberta e sugestão de correção dos defeitos identificados. Desse forma, o PRECISE deve melhorar o retorno das explicações, origem e sugestões de correção dos problemas identificados durante a execução do processo.

Um ponto a ser analisado diz respeito ao estudo de caso. Na validação do PRECISE, foi utilizada uma única LPMSC como estudo de caso. Com o objetivo de

analisar melhor os benefícios do processo criado nesta tese de doutorado, um dos trabalhos futuros inclui aplicar o PRECISE em outras linhas. Infelizmente, não foram encontradas outras LPSs sensíveis ao contexto disponíveis na literatura.

Outro possível trabalho futuro identificado diz respeito à implementação da especificação formal. Como a especificação formal foi construída usando Lógica de Primeira Ordem, sendo, portanto, genérica, pode-se desenvolver implementações baseadas em outras linguagens de especificação formal, tais como ontologias ou linguagens restritivas e fazer comparações entre os resultados obtidos, de modo a realizar uma análise mais aprofundada de qual linguagem formal pode ser considerada a mais adequada.

Além disso, a verificação do processo de simulação é executada em tempo de desenvolvimento, portanto as situações de contexto simuladas são pré-definidas pelo usuário, limitando o escopo dos defeitos identificados, uma vez que é possível que situações não previstas em tempo de desenvolvimento ocorram em tempo de execução dos produtos. Dessa forma, um trabalho futuro a ser considerado inclui a identificação de mudanças de contexto e verificação da adaptação do produto em tempo de execução.

Além dos trabalhos que exigem continuidade citados anteriormente, esta tese de doutorado não aborda temas de pesquisa de interesse crescente para a comunidade de LPSs, tais como camadas abstratas, *binding time*, auto-adaptação, manutenção e evolução das LPSs, sendo necessário portanto, analisar a viabilidade de abordar esses temas em trabalhos futuros. É possível que a combinação do PRECISE com abordagens auto-adaptativas possa oferecer um suporte significativo para assegurar a qualidade da adaptação dos produtos. Assim, esta tese de doutorado pode potencialmente contribuir para as pesquisas em curso sobre esses temas.

Quadro 8.1: Artigos científicos produzidos

No.	Referência	Tipo	Status
1	**MARINHO, F. G.**, A proposal for consistency checking in dynamic software product line models using OCL. In: 32nd International Conference on Software Engineering - Doctoral Symposium, 2010, Cape Town. 32nd ACM/IEEE International Conference on Software Engineering - Doctoral Symposium. New York : ACM, 2010. v. 2. p. 333-334.	Simpósio de doutorado	Publicado
2	**MARINHO, F. G.**; LIMA, F. F. P.; FERREIRA FILHO, J. B.; ROCHA, L. S.; MAIA, M. E. F.; AGUIAR, S. B.; 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: 14th International Software Product Line Conference, 2010, Jeju Island, South Korea. 14th International Software Product Line Conference, SPLC'10. Germany : Springer, 2010. v. 6287. p. 346-360.	Conferência	Publicado
3	**MARINHO, F. G.**; BORGES NETO, J. B.; COSTA, A. L.; DANTAS, V. L. L.; FERREIRA FILHO, J. B.; LIMA, F. F. P.; ROCHA, L. S; ANDRADE, R. M. C.; TEIXEIRA, E.; WERNER, C. An Architecture Proposal for Nested Software Product Lines in the Domain of Mobile and Context-Aware Applications. In: I Congresso Brasileiro de Software: Teoria e Prática - IV SBCARS, 2010, Salvador- Bahia. IV Simpósio Brasileiro de Componentes, Arquitetura e Reutilização de Software. Washington: IEEE Computer Society, 2010. p. 51-60.	Conferência	Publicado
4	**MARINHO, F. G.**; ANDRADE, R. M. C.; WERNER, C.; VIANA, W.; MAIA, E. F.; ROCHA, L. S.; TEIXEIRA, E.; FERREIRA FILHO, J. B.; DANTAS V .L. L.; LIMA, F. F. P.; AGUIAR, S. B. MobiLine: A Nested Software Product Line for the Domain of Mobile and Context-Aware Applications. Science of Computer Programming (Print), 2012.	Periódico	Publicado
5	**MARINHO, F. G.**; ANDRADE, R. M. C.; WERNER, C. A Verification Mechanism of Feature Models for Mobile and Context-Aware Software Product Lines. In: II Congresso Brasileiro de Software: Teoria e Prática - V SBCARS, 2011, São Paulo - SP. SBCARS 2011 V Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software. Washington: IEEE Computer Society, 2011. v. 3. p. 1-10.	Conferência	Publicado
6	**MARINHO, F. G.**; ANDRADE, R. M. C.; COSTA, P.A; MAIA P.H.; VIDAL, V.; WERNER, C. Safe Adaptation in Context-Aware Feature Models. In: Fourth International Workshop on Feature-Oriented Software Development (FOSD), 2012, Dresden, Germany. Generative Programming and Component Engineering (GPCE'12)	Workshop	Aceito
7	**MARINHO, F. G.**; ANDRADE, R. M. C.; COSTA, P. A; ALMEIDA NETO, B. J., Uso de Métodos Formais na Engenharia de Software	Relatório Técnico	Finalizando

Bibliografia

[1] ABE J. Introducao a Logica Para a Ciencia Da Computacao. Arte & Ciência, 1987. ISBN 9788574730455.

[2] ABOWD, G. D. et al. Towards a better understanding of context and context-awareness. In: *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. London, UK, UK: Springer-Verlag, 1999. (HUC '99), p. 304–307. ISBN 3-540-66550-1.

[3] ACHER, M. et al. Modeling Context and Dynamic Adaptations with Feature Models. In: *Proceedings of the 4th International Workshop Models@run.time*. United States: [s.n.], 2009. p. 10.

[4] ACHER, M. et al. Comparing approaches to implement feature model composition. In: KÜHNE, T. et al. (Ed.). *ECMFA*. [S.l.]: Springer, 2010. (Lecture Notes in Computer Science, v. 6138), p. 3–19. ISBN 978-3-642-13594-1.

[5] ACHER, M. et al. Slicing feature models. In: ALEXANDER, P.; PASAREANU, C. S.; HOSKING, J. G. (Ed.). *ASE*. [S.l.]: IEEE, 2011. p. 424–427. ISBN 978-1-4577-1638-6.

[6] ANTKIEWICZ, M.; CZARNECKI, K. Featureplugin: feature modeling plug-in for eclipse. In: *Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*. New York, NY, USA: ACM, 2004. (eclipse '04), p. 67–72.

[7] APEL, S. et al. An Algebra for Feature-Oriented Software Development. 2007.

[8] ARBOLEDA, H. et al. Product Derivation in a Model-Driven Software Product Line using Decision Models. In: Proceedings of the 12th Iberoamerican Conference on Requirements Engineering and Software Environments (IDEAS'09). Medellin, Colombia: [s.n.], 2009. p. 59–72. ISBN 978-958-44-5028-9.

[9] ATKINSON, C. et al. Modeling components and component-based systems in kobra. In: RAUSCH, A. et al. (Ed.). The Common Component Modeling Example:

Comparing Software Component Models [result from the Dagstuhl research seminar for CoCoME, August 1-3, 2007]. [S.l.]: Springer, 2007. (Lecture Notes in Computer Science, v. 5153), p. 54–84. ISBN 978-3-540-85288-9.

[10] AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. Dependable and Secure Computing, IEEE Transactions on, v. 1, n. 1, p. 11–33, jan.-march 2004. ISSN 1545-5971.

[11] BALDAN, P. et al. Compositional semantics for open petri nets based on deterministic processes. *Mathematical. Structures in Comp. Sci.*, Cambridge University Press, New York, NY, USA, v. 15, n. 1, p. 1–35, fev. 2005. ISSN 0960-1295.

[12] BALDAUF, M.; DUSTDAR, S.; ROSENBERG, F. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, Inderscience Publishers, Inderscience Publishers, Geneva, SWITZERLAND, v. 2, n. 4, p. 263–277, jun. 2007. ISSN 1743-8225.

[13] BALLARD, B. Designing the Mobile User Experience. [S.l.]: John Wiley Sons, 2007. ISBN 0470033614, 9780470033616.

[14] BARBOSA, E. F. et al. *Introdução ao Teste de Software*. João Pessoa, PB: [s.n.], October 2000. 330–378 p. Minicurso apresentado no XIV Simpósio Brasileiro de Engenharia de Software (SBES 2000).

[15] BASTIDA, L.; NIETO, F. J.; TOLA, R. Context-aware service composition: a methodology and a case study. In: *Proceedings of the 2nd international workshop on Systems development in SOA environments*. New York, NY, USA: ACM, 2008. (SDSOA '08), p. 19–24. ISBN 978-1-60558-029-6.

[16] BATORY, D. Feature models, grammars, and propositional formulas. In: OBBINK, H.; POHL, K. (Ed.). *Software Product Lines*. Springer Berlin / Heidelberg, 2005, (Lecture Notes in Computer Science, v. 3714). p. 7–20. ISBN 978-3-540-28936-4. 10.1007/115548443.

[17] BATORY, D.; BENAVIDES, D.; RUIZ-CORTES A. Automated analysis of feature models: challenges ahead. *Commun. ACM*, ACM, New York, NY, USA, v. 49, n. 12, p. 45–47, dez. 2006. ISSN 0001-0782.

[18] BATORY, D.; SARVELA, J. N.; RAUSCHMAYER, A. Scaling step-wise refinement. In: *Proceedings of the 25th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2003. (ICSE '03), p. 187–197. ISBN 0-7695-1877-X.

[19] BECKER, M. Towards a General Model of Variability in Product Families. In: *Proceedings of the 1st Workshop on Software Variability Management*. Groningen, Netherlands: [s.n.], 2003.

[20] BENAVIDES, D. On the Automated Analysis of Software Product Lines Using Feature Models. A framework for developing automated tool support. Tese (Doutorado) — University of Seville, 2007.

[21] BENAVIDES, D.; MARTIN-ARROYO, P. T.; CORTES, A. R. Automated reasoning on feature models. In: PASTOR, O.; CUNHA, J. F. e (Ed.). *CAiSE*. [S.l.]: Springer, 2005. (Lecture Notes in Computer Science, v. 3520), p. 491–503. ISBN 3-540-26095-1.

[22] BENAVIDES, D.; SEGURA, S.; RUIZ-CORTES, A. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, v. 35, n. 6, p. 615–636, 2010. ISSN 0306-4379.

[23] BENAVIDES, D. et al. A first step towards a framework for the automated analysis of feature models. In: *Managing Variability for Software Product Lines: Working With Variability Mechanisms*. [s.n.], 2006.

[24] BENCOMO, N. et al. Reflective component-based technologies to support dynamic variability. In: HEYMANS, P. et al. (Ed.). *VaMoS*. [S.l.: s.n.], 2008. (ICB Research Report), p. 141–150.

[25] BENCOMO, N. et al. Dynamically adaptive systems are product lines too: Using model-driven techniques to capture dynamic variability of adaptive systems. In: THIEL, S.; POHL, K. (Ed.). *SPLC (2)*. [S.l.]: Lero Int. Science Centre, University of Limerick, Ireland, 2008. p. 23–32. ISBN 978-1-905952-06-9.

[26] BERKENKOTTER, K. Reliable uml models and profiles. *Electron. Notes Theor. Comput. Sci.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 217, p. 203–220, jul. 2008. ISSN 1571-0661.

[27] BEUCHE, D. Modeling and building software product lines with pure::variants. In: *Proceedings of the 15th International Software Product Line Conference, Volume 2*. New York, NY, USA: ACM, 2011. (SPLC '11), p. 46:1–46:1. ISBN 978-1-4503-0789-5.

[28] BLANC, X. et al. Incremental detection of model inconsistencies based on model operations. In: *Proceedings of the 21st International Conference on Advanced Information Systems Engineering*. Berlin, Heidelberg: Springer-Verlag, 2009. (CAiSE '09), p. 32–46. ISBN 978-3-642-02143-5.

[29] BODIRSKY, M.; PINSKER, M. Schaefer's theorem for graphs. *CoRR*, abs/1011.2894, 2010.

[30] BODIRSKY, M.; PINSKER, M. Schaefer's theorem for graphs. In: *Proceedings of the 43rd annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 2011. (STOC '11), p. 655–664. ISBN 978-1-4503-0691-1.

[31] BOSCH, J. *Design and use of software architectures: adopting and evolving a product-line approach*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000. ISBN 0-201-67494-7.

[32] BOSCH, J. Software variability: process and management. *Software Process: Improvement and Practice*, v. 10, n. 1, p. 3–5, 2005.

[33] BOUCHER, Q. et al. Introducing TVL, a text-based feature modelling language. In: *Proceedings of the Fourth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'10), Linz, Austria, January 27-29*. University of Duisburg-Essen, 2010. p. 159–162.

[34] BPMN - Business Process Model and Notation. January 2011. Http://www.bpmn.org/.

[35] BROWN, P. J.; BOVEY, J. D.; CHEN, X. Context-aware applications: from the laboratory to the marketplace. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, v. 4, n. 5, p. 58–64, 1997.

[36] BRYANT, R. E. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, IEEE Computer Society, Washington, DC, USA, v. 35, n. 8, p. 677–691, ago. 1986. ISSN 0018-9340.

[37] BRYANT, R. E. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 24, n. 3, p. 293–318, set. 1992. ISSN 0360-0300.

[38] CASANOVA, A. *Programação em lógica e a linguagem Prolog.* Edgard Blucher, 1987.

[39] CECHTICKY, V. et al. Xml-based feature modelling. In: *Software Reuse: Methods, Techniques and Tools: 8th International Conference, ICSR 2004.* [S.l.]: Springer-Verlag, 2004. p. 5–9.

[40] CETINA, C. et al. An architectural discussion on dspl. In: THIEL, S.; POHL, K. (Ed.). *SPLC (2).* [S.l.]: Lero Int. Science Centre, University of Limerick, Ireland, 2008. p. 59–68. ISBN 978-1-905952-06-9.

[41] CHEN, K. et al. An approach to constructing feature models based on requirements clustering. In: *Proceedings of the 13th IEEE International Conference on Requirements Engineering.* Washington, DC, USA: IEEE Computer Society, 2005. (RE '05), p. 31–40. ISBN 0-7695-2425-7.

[42] CLEMENTS, P.; NORTHROP, L. *Software product lines: practices and patterns.* Addison-Wesley, 2002. (The SEI series in software engineering). ISBN 9780201703320.

[43] CONCEPT FOR THE XFEATURE TOOL. June 2005. Http://pnp-software.com/XFeature/pdf/XFeatureToolConcept.pdf.

[44] CORMEN, T. H. et al. *Introduction to Algorithms.* 2nd. ed. [S.l.]: McGraw-Hill Higher Education, 2001. ISBN 0070131511.

[45] COSTA, P. A.; ANDRADE, R. Uma ferramenta para verificação de consistência e integridade do modelo de características de linhas de produtos de software sensíveis ao contexto baseado em perfis da uml. In: *CBSoft 2011 - WTDSoft ().* Sao Paulo: [s.n.], 2011.

[46] CZARNECKI, K. et al. Generative programming for embedded software: An industrial experience report. In: *Proceedings of the 1st ACM SIGPLAN/SIGSOFT conference on Generative Programming and Component Engineering.* London, UK: Springer-Verlag, 2002. (GPCE '02), p. 156–172. ISBN 3-540-44284-7.

[47] CZARNECKI, K.; EISENECKER, U. W. *Generative programming: methods, tools, and applications*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000. ISBN 0-201-30977-7.

[48] CZARNECKI, K.; HELSEN, S.; EISENECKER, U. Formalizing cardinality-based feature models and their specialization. In: *Software Process: Improvement and Practice*. [S.l.: s.n.], 2005. p. 2005.

[49] CZARNECKI, K.; HELSEN, S.; EISENECKER, U. W. Staged configuration using feature models. In: NORD, R. L. (Ed.). *SPLC*. [S.l.]: Springer, 2004. (Lecture Notes in Computer Science, v. 3154), p. 266–283. ISBN 3-540-22918-3.

[50] CZARNECKI, K.; KIM, C. H. P.; KALLEBERG, K. T. Feature models are views on ontologies. In: *SPLC*. [S.l.]: IEEE Computer Society, 2006. p. 41–51. ISBN 0-7695-2599-7.

[51] CZARNECKI, K.; PIETROSZEK, K. Verifying feature-based model templates against well-formedness ocl constraints. In: *Proceedings of the 5th international conference on Generative programming and component engineering*. New York, NY, USA: ACM, 2006. (GPCE '06), p. 211–220. ISBN 1-59593-237-2.

[52] CZARNECKI, K.; WASOWSKI, A. Feature diagrams and logics: There and back again. In: *Proceedings of the 11th International Software Product Line Conference*. Washington, DC, USA: IEEE Computer Society, 2007. p. 23–34. ISBN 0-7695-2888-0.

[53] DAVIS, M.; PUTNAM, H. A computing procedure for quantification theory. *J. ACM*, ACM, New York, NY, USA, v. 7, n. 3, p. 201–215, jul. 1960. ISSN 0004-5411.

[54] De Moura, L.; BJØRNER, N. Z3: an efficient smt solver. In: *Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems*. Berlin, Heidelberg: Springer-Verlag, 2008. (TACAS'08/ETAPS'08), p. 337–340. ISBN 3-540-78799-2 , 978-3-540-78799-0.

[55] DEAN, M.; SCHREIBER, G. *OWL Ontology Language Reference. [S.l.], fev. 2004.*

[56] DERRICK, J.; WEHRHEIM, H. Model transformations across views. *Sci. Comput. Program.*, v. 75, n. 3, p. 192–210, 2010.

[57] DEY, A. K. Understanding and using context. *Personal Ubiquitous Comput.*, Springer-Verlag, London, UK, v. 5, n. 1, p. 4–7, 2001. ISSN 1617-4909.

[58] DEY, A. K.; ABOWD, G. D. The context toolkit: Aiding the development of context-aware applications. In: . [S.l.]: ACM Press, 1999. p. 434–441.

[59] DINKELAKER, T. et al. A dynamic software product line approach using aspect models at runtime. *9th IEEE International Conference on Computer and Information Technology*, CEUR-WS.org, p. 11–18, 2010.

[60] ELFAKI, A. O.; PHON-AMNUAISUK, S.; HO, C. K. Knowledge based method to validate feature models. In: THIEL, S.; POHL, K. (Ed.). *SPLC (2)*. [S.l.]: Lero Int. Science Centre, University of Limerick, Ireland, 2008. p. 217–225. ISBN 978-1-905952-06-9.

[61] ERIKSSON, M.; BÖRSTLER, J.; BORG, K. The pluss approach - domain modeling with features, use cases and use case realizations. In: OBBINK, J. H.; POHL, K. (Ed.). *SPLC*. [S.l.]: Springer, 2005. (Lecture Notes in Computer Science, v. 3714), p. 33–44. ISBN 3-540-28936-4.

[62] FAN, S.; ZHANG, N. Feature model based on description logics. In: *Proceedings of the 10th international conference on Knowledge-Based Intelligent Information and Engineering Systems - Volume Part II*. Berlin, Heidelberg: Springer-Verlag, 2006. (KES'06), p. 1144–1151. ISBN 3-540-46537-5 , 978-3-540-46537-9.

[63] FERNANDES, P.; WERNER, C.; MURTA, L. G. P. Feature modeling for context-aware software product lines. In: *SEKE*. [S.l.]: Knowledge Systems Institute Graduate School, 2008. p. 758–763. ISBN 1-891706-22-5.

[64] FERNANDES, P.; WERNER, C. M. L. Ubifex: Modeling context-aware software product lines. In: THIEL, S.; POHL, K. (Ed.). *SPLC (2)*. [S.l.]: Lero Int. Science Centre, University of Limerick, Ireland, 2008. p. 3–8. ISBN 978-1-905952-06-9.

[65] GALLOPOULOS, E.; HOUSTIS, E.; RICE, J. R. Computer as thinker/doer: Problem-solving environments for computational science. *IEEE Comput. Sci. Eng.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 1, n. 2, p. 11–23, jun. 1994. ISSN 1070-9924.

[66] GHEYI, R.; MASSONI, T.; BORBA, P. A theory for feature models in alloy. In: *Proceedings of the ACM SIGSOFY First Alloy Workshop*. Portland, United States: [s.n.], 2006. p. 71–80.

[67] GHEYI, R.; MASSONI, T.; BORBA, P. Automatically checking feature model refactorings. *J. UCS*, v. 17, n. 5, p. 684–711, 2011.

[68] GOMAA, H.; SHIN, M. E. A multiple-view meta-modeling approach for variability management in software product lines. In: *ICSR*. [S.l.]: Springer, 2004. (Lecture Notes in Computer Science, v. 3107), p. 274–285. ISBN 3-540-22335-5.

[69] GREAT - Grupo de Redes de Computadores, Engenharia de Software e Sistemas. Julho 2012. Http://www.great.ufc.br.

[70] GRISS, M. L. Implementing product-line features with component reuse. In: FRAKES, W. B. (Ed.). *ICSR*. [S.l.]: Springer, 2000. (Lecture Notes in Computer Science, v. 1844), p. 137–152. ISBN 3-540-67696-1.

[71] GRISS, M. L.; FAVARO, J.; ALESSANDRO, M. d. Integrating feature modeling with the rseb. In: *Proceedings of the 5th International Conference on Software Reuse*. Washington, DC, USA: IEEE Computer Society, 1998. (ICSR '98), p. 76–. ISBN 0-8186-8377-5.

[72] 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.

[73] HALLSTEINSEN, S. et al. Dynamic software product lines. *Computer*, IEEE Computer Society, Los Alamitos, CA, USA, v. 41, p. 93–95, 2008. ISSN 0018-9162.

[74] HALLSTEINSEN, S. et al. Using product line techniques to build adaptive systems. In: *Proceedings of the 10th International on Software Product Line Conference*. Washington, DC, USA: IEEE Computer Society, 2006. (SPLC '06), p. 141–150. ISBN 0-7695-2599-7.

[75] HARTMANN, H.; TREW, T. Using feature diagrams with context variability to model multiple product lines for software supply chains. In: *Proceedings of the 2008*

12th International Software Product Line Conference. Washington, DC, USA: IEEE Computer Society, 2008. (SPLC '08), p. 12–21. ISBN 978-0-7695-3303-2.

[76] HEIDENREICH, F. Towards systematic ensuring well-formedness of software product lines. In: *Proceedings of the First International Workshop on Feature-Oriented Software Development*. New York, NY, USA: ACM, 2009. (FOSD '09), p. 69–74. ISBN 978-1-60558-567-3.

[77] HEIDENREICH, F.; KOPCSEK, J.; WENDE, C. Featuremapper: mapping features to models. In: *Companion of the 30th international conference on Software engineering*. New York, NY, USA: ACM, 2008. (ICSE Companion '08), p. 943–944. ISBN 978-1-60558-079-1.

[78] HENRICKSEN, K.; INDULSKA, J. A software engineering framework for context-aware pervasive computing. In: *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*. Washington, DC, USA: IEEE Computer Society, 2004. (PERCOM '04), p. 77–. ISBN 0-7695-2090-1.

[79] HEYMANS, P. et al. (Ed.). *Second International Workshop on Variability Modelling of Software-Intensive Systems, Universität Duisburg-Essen, Germany, January 16-18, 2008, Proceedings*, (ICB Research Report). [S.l.: s.n.], 2008.

[80] HUBAUX, A. et al. Towards multi-view feature-based configuration. In: WIERINGA, R.; PERSSON, A. (Ed.). *REFSQ*. [S.l.]: Springer, 2010. (Lecture Notes in Computer Science, v. 6182), p. 106–112. ISBN 978-3-642-14191-1.

[81] IEC, I. Iso/iec 15504-1 information technology - process assessment - part 1: Concepts and vocabulary. *Software Process: Improvement and Practice*, International Organization for Standardization, v. 2, n. 1, 2004.

[82] JACKSON, D. *Software Abstractions: Logic, Language, and Analysis*. [S.l.]: The MIT Press, 2006. ISBN 0262101149.

[83] JANOTA, M.; KINIRY, J. Reasoning about feature models in higher-order logic. In: *Software Product Line Conference, 2007. SPLC 2007. 11th International*. [S.l.: s.n.], 2007. p. 13–22.

[84] JAROUCHEH, Z.; LIU, X.; SMITH, S. Candel: product line based dynamic context management for pervasive applications. In: *CICIS'10 Proceedings*. Krakow Poland: IEEE Computer Society Press, 2010. p. 209–216.

[85] KANG, K. et al. Form: A feature-oriented reuse method with domain-;specific reference architectures. *Annals of Software Engineering*, Springer Netherlands, v. 5, p. 143–168, 1998. ISSN 1022-7091. 10.1023/A:1018980625587.

[86] KANG, K. C. et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. [S.l.], November 1990.

[87] BUREGIO, V. A.; MEIRA, S. R. L; ALMEIDA,E. S. *Characterizing Dynamic Software Product Lines - A Preliminary Mapping Study - SPLC Workshops*. 2010. 53-60 p.

[88] KANG, K. C.; LEE, J. L. J.; DONOHOE, P. *Feature-oriented product line engineering*. IEEE, 2002. 58–65 p.

[89] KASTNER, C. et al. Featureide: A tool framework for feature-oriented software development. In: *Proceedings of the 31st International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2009. (ICSE '09), p. 611–614. ISBN 978-1-4244-3453-4.

[90] LAUENROTH, K.; POHL, K. Towards automated consistency checks of product line requirements specifications. In: *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. New York, NY, USA: ACM, 2007. (ASE '07), p. 373–376. ISBN 978-1-59593-882-4.

[91] LEE, J.; KANG, K. A feature-oriented approach to developing dynamically reconfigurable products in product line engineering. In: *Software Product Line Conference, 2006 10th International*. [S.l.: s.n.], 2006. p. 10 pp.–140.

[92] LEE, V.; SCHNEIDER, H.; SCHELL, R. *Aplicações Móveis: Arquitetura, Projeto e Desenvolvimento*. [S.l.]: Editora Pearson Education do Brasil e Makron Books, 2005.

[93] LINDEN, F. J. v. d.; SCHMID, K.; ROMMES, E. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007. ISBN 3540714367.

[94] LOPEZ-HERREJON, R. E.; EGYED, A. C2mv2: Consistency and composition for managing variability in multi-view systems. In: MENS, T.; KANELLOPOULOS, Y.; WINTER, A. (Ed.). *CSMR*. [S.l.]: IEEE Computer Society, 2011. p. 347–350. ISBN 978-0-7695-4343-7.

[95] MAIA, M. E.; ROCHA, L. S.; ANDRADE, R. M. Requirements and challenges for building service-oriented pervasive middleware. In: *Proceedings of the 2009 international conference on Pervasive services*. New York, NY, USA: ACM, 2009. (ICPS '09), p. 93–102. ISBN 978-1-60558-644-1.

[96] MANNION, M. Using first-order logic for product line model validation. In: *Proceedings of the Second International Conference on Software Product Lines*. London, UK, UK: Springer-Verlag, 2002. (SPLC 2), p. 176–187. ISBN 3-540-43985-4.

[97] MARINHO, F. et al. A software product line for the mobile and context-aware applications domain. In: BOSCH, J.; LEE, J. (Ed.). *Software Product Lines: Going Beyond*. Springer Berlin / Heidelberg, 2010, (Lecture Notes in Computer Science, v. 6287). p. 346–360. ISBN 978-3-642-15578-9. 10.1007/978-3-642-15579-624.

[98] MARINHO, F. G.; ANDRADE, R. M.; WERNER, C. A verification mechanism of feature models for mobile and context-aware software product lines. *Software Components, Architectures and Reuse, Brazilian Symposium on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 1–10, 2011.

[99] MASSEN, T. von der; LICHTER, H. Modellierung von variabilität mit uml use cases. *Softwaretechnik-Trends*, v. 23, n. 1, 2003.

[100] MASSEN, T. von der; LICHTER, H. Requiline: A requirements engineering tool for software product lines. In: LINDEN, F. van der (Ed.). *PFE*. [S.l.]: Springer, 2003. (Lecture Notes in Computer Science, v. 3014), p. 168–180. ISBN 3-540-21941-2.

[101] MCKINLEY, P. K. et al. *A Taxonomy of Compositional Adaptation*. [S.l.], 2004.

[102] MENDONCA, M.; WASOWSKI, A.; CZARNECKI, K. Sat-based analysis of feature models is easy. In: MUTHIG, D.; MCGREGOR, J. D. (Ed.). *SPLC*. [S.l.]: ACM, 2009. (ACM International Conference Proceeding Series, v. 446), p. 231–240.

[103] MENDONCA, M. et al. Efficient compilation techniques for large scale feature models. In: SMARAGDAKIS, Y.; SIEK, J. G. (Ed.). *GPCE*. [S.l.]: ACM, 2008. p. 13–22. ISBN 978-1-60558-267-2.

[104] METZGER, A. et al. Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In: *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*. [S.l.: s.n.], 2007. p. 243–253. ISSN 1090-705X.

[105] MORIN, B.; BARAIS, O.; JéZéQUEL, J. marc. K@rt: An aspect-oriented and model-oriented framework for dynamic software product lines. In: *in Proceedings of the 3rd International Workshop on Models@Runtime, at MoDELS 08*. [S.l.: s.n.], 2008.

[106] MORIN, B. et al. An aspect-oriented and model-driven approach for managing dynamic variability. In: *IN: ACM/IEEE MODELS 08*. [S.l.: s.n.], 2008.

[107] NETO, A. C. D.; TRAVASSOS, G. H. A picture from the model-based testing area: Concepts, techniques, and challenges. *Advances in Computers*, v. 80, p. 45–120, 2010.

[108] NETO, F.; NETO, P. *Designing Solutions-Based Ubiquitous and Pervasive Computing: New Issues and Trends*. IGI Global, 2010. ISBN 9781615208449.

[109] NUSEIBEH, B.; KRAMER, J.; FINKELSTEIN, A. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 20, p. 760–773, October 1994. ISSN 0098-5589.

[110] OCL - Object Constraint Language. Janeiro 2012. Http://www.omg.org/spec/OCL/2.3.1/.

[111] OMG - UML Infrastructure. Agosto 2011. Http://www.omg.org/spec/UML/2.4.1/Infrastructure/.

[112] OMG - UML Superstructure. Agosto 2011. Http://www.omg.org/spec/UML/2.4.1/Superstructure/.

[113] OMG Model Driven Architecture. Junho 2003. Http://www.omg.org/mda/.

[114] PAIGE, R. F.; BROOKE, P. J.; OSTROFF, J. S. Metamodel-based model conformance and multiview consistency checking. *ACM Trans. Softw. Eng. Methodol.*, ACM, New York, NY, USA, v. 16, July 2007. ISSN 1049-331X.

[115] PARRA, C.; BLANC, X.; DUCHIEN, L. Context awareness for dynamic service-oriented product lines. In: *Proceedings of the 13th International Software Product Line Conference*. Pittsburgh, PA, USA: Carnegie Mellon University, 2009. (SPLC '09), p. 131–140. Disponível em: <http://dl.acm.org/citation.cfm?id=1753235.1753254>.

[116] PELLET. Marco 2010. Http://pellet.owldl.com/.

[117] PERROUIN, G. et al. Reconciling automation and flexibility in product derivation. In: *Proceedings of the 2008 12th International Software Product Line Conference*. Washington, DC, USA: IEEE Computer Society, 2008. (SPLC '08), p. 339–348. ISBN 978-0-7695-3303-2.

[118] PETRENKO, A.; SIMÃO, A. da S.; MALDONADO, J. C. Model-based testing of software and systems: recent advances and challenges. *STTT*, v. 14, n. 4, p. 383–386, 2012.

[119] POHL, K.; BÖCKLE, G.; LINDEN, F. J. van der. *Software Product Line Engineering: Foundations, Principles and Techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005. ISBN 3540243720.

[120] PRESSMAN, R. *Software engineering: a practitioner's approach*. McGraw-Hill Higher Education, 2010. ISBN 9780073375977.

[121] PROTéGé-OWL. Abril 2012. Http://protege.stanford.edu.

[122] RAATIKAINEN, K. E. E. Middleware for mobile aplications beyond 3g. In: *Proceedings of the IFIP TC6 WG6.7 Seventh International Conference on Intelligence in Networks: Smart Networks*. Deventer, The Netherlands, The Netherlands: Kluwer, B.V., 2002. (SMARTNET '02), p. 3–18. ISBN 1-4020-7008-X.

[123] RACER. Marco 2012.Http://www.racer-systems.com/.

[124] RAKITIN, S. R. *Software Verification and Validation for Practitioners and Managers, Second Edition*. 2nd. ed. Norwood, MA, USA: Artech House, Inc., 2001. ISBN 1580532969.

[125] RAMOS, A. C. et al. Knowledge management for adapted information retrieval in ubiquitous environments. In: CORDEIRO, J. A. M. et al. (Ed.). *WEBIST (1)*. [S.l.]: INSTICC Press, 2006. p. 21–29. ISBN 978-972-8865-46-7.

[126] RIEBISCH, M. Towards a more precise definition of feature models. In: *Modelling Variability for Object-Oriented Product Lines*. [S.l.]: BookOnDemand Publ. Co, 2003. p. 64–76.

[127] SALINESI, C.; MAZO, R.; DIAZ, D. Criteria for the verification of feature models. In: *INFORSID*. [S.l.: s.n.], 2010. p. 293–308.

[128] SALINESI, C. et al. Looking for product line feature models defects: Towards a systematic classification of verification criteria. In: *Requirements Engineering Conference, 2009. RE '09. 17th IEEE International*. [S.l.: s.n.], 2009. p. 385–386. ISSN 1090-705X.

[129] SANCHEZ, P. et al. Software language engineering. In: GAVSEVIC, D.; LAMMEL, R.; WYK, E. (Ed.). Berlin, Heidelberg: Springer-Verlag, 2009. cap. Engineering Languages for Specifying Product-Derivation Processes in Software Product Lines, p. 188–207. ISBN 978-3-642-00433-9.

[130] SCHAEFER, I. Variability modelling for model-driven development of software product lines. In: *Fourth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2010)*. [S.l.: s.n.], 2010.

[131] SCHILIT, B. N.; THEIMER, M. M. Disseminating active map information to mobile hosts. *Network, IEEE*, v. 8, n. 5, p. 22–32, 1994.

[132] SCHOBBENS, P.-Y. et al. Generic semantics of feature diagrams. *Computer Networks*, v. 51, n. 2, p. 456 – 479, 2007. ISSN 1389-1286.

[133] SEGURA, S. et al. Automated merging of feature models using graph transformations. In: LÄMMEL, R.; VISSER, J.; SARAIVA, J. (Ed.). *GTTSE*. [S.l.]: Springer, 2007. (Lecture Notes in Computer Science, v. 5235), p. 489–505. ISBN 978-3-540-88642-6.

[134] SELIC, B. A systematic approach to domain-specific language design using uml. *10th IEEE International Symposium on Object and ComponentOriented RealTime Distributed Computing ISORC07*, Ieee, p. 2–9, 2007.

[135] SIMONS, C. Cmp: A uml context modeling profile for mobile distributed systems. In: *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*. Washington, DC, USA: IEEE Computer Society, 2007. (HICSS '07), p. 289b–. ISBN 0-7695-2755-8.

[136] SOMMERVILLE, I. *Software Engineering*. Addison-Wesley, 2007. (International Computer Science Series). ISBN 9780321313799.

[137] SPIVEY, J. M. *The Z notation: a reference manual*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989. ISBN 0-13-983768-X.

[138] STRANG, T.; LINNHOFF-POPIEN, C. A context modeling survey. *Graphical Models*, Citeseer, Workshop o, n. 4, p.1, 2004.

[139] SUN, J. et al. Formal semantics and verification for feature modeling. In: *ICECCS*. [S.l.]: IEEE Computer Society, 2005. p. 303–312. ISBN 0-7695-2284-X.

[140] SVAHNBERG, M.; GURP, J. van; BOSCH, J. A taxonomy of variability realization techniques: Research articles. *Softw. Pract. Exper.*, John Wiley & Sons, Inc., New York, NY, USA, v. 35, p. 705–754, July 2005. ISSN 0038-0644.

[141] SVM System. March 2012. Http://www.cs.cmu.edu/modelcheck/smv.html.

[142] SWI-PROLOG. March 2012. Http://www.swi-prolog.org/.

[143] SWRL. January 2005. Http://www.w3.org/Submission/2005/01/.

[144] SZWARCFITER, J. *Grafos e algoritimos computacionais*. Campus, 1988.

[145] TAWHID, R.; PETRIU, D. C. Towards automatic derivation of a product performance model from a uml software product line model. In: *Proceedings of the 7th international workshop on Software and performance*. New York, NY, USA: ACM, 2008. (WOSP '08), p. 91–102. ISBN 978-1-59593-873-2.

[146] THIEL, S.; POHL, K. (Ed.). *Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8-12, 2008, Proceedings. Second Volume (Workshops)*. [S.l.]: Lero Int. Science Centre, University of Limerick, Ireland, 2008. ISBN 978-1-905952-06-9.

[147] TRINIDAD, P.; BENAVIDES, D.; CORTES, A. R. Isolated features detection in feature models. In: BOUDJLIDA, N.; CHENG, D.; GUELFI, N. (Ed.). *CAiSE Forum*. [S.l.]: CEUR-WS.org, 2006. (CEUR Workshop Proceedings, v. 231).

[148] TRINIDAD, P. et al. Automated error analysis for the agilization of feature modeling. *Journal of Systems and Software*, Elsevier Science Inc., New York, NY, v. 81, n. 6, p. 883–896, Jun 2008. ISSN 0164-1212.

[149] TRINIDAD, P. et al. Automated error analysis for the agilization of feature modeling. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 81, p. 883–896, June 2008. ISSN 0164-1212.

[150] TRINIDAD, P. et al. Mapping feature models onto component models to build dynamic software product lines. In: *SPLC (2)*. [S.l.]: Kindai Kagaku Sha Co. Ltd., Tokyo, Japan, 2007. p. 51–56. ISBN 978-4-7649-0342-5.

[151] UBAYASHI, N.; NAKAJIMA, S.; HIRAYAMA, M. Context-dependent product line practice for constructing reliable embedded systems. In: BOSCH, J.; LEE, J. (Ed.). *Software Product Lines: Going Beyond*. Springer Berlin / Heidelberg, 2010, (Lecture Notes in Computer Science, v. 6287). p. 1–15. ISBN 978-3-642-15578-9. 10.1007/978-3-642-15579-61.

[152] VIANA, W.; ANDRADE, R. M. C. Xmobile: A mb-uid environment for semi-automatic generation of adaptive applications for mobile devices. *Journal of Systems and Software*, v. 81, n. 3, p. 382–394, 2008.

[153] VIANA, W. et al. Towards the semantic and context-aware management of mobile multimedia. *Multimedia Tools Appl.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 53, n. 2, p. 391–429, jun. 2011. ISSN 1380-7501.

[154] VIEIRA, V. *CEManTIKA: A Domain-Independent Framework for Designing Context- Sensitive System*. 187 p. Tese (Doutorado), 2008.

[155] WANG, H. et al. A semantic web approach to feature modeling and verification. In: *In Workshop on Semantic Web Enabled Software Engineering (SWESE 05*. [S.l.: s.n.], 2005.

[156] WANG, H. H. et al. Verifying feature models using owl. *Web Semant.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v.5, p. 117–129, June 2007. ISSN 1570-8268.

[157] WEISS, D. M.; CHI. Software Product-Line Engineering: A Family-Based Software Development Process. [S.l.]: Addison-Wesley Professional; Har/Cdr edition, 1999.

[158] ZAID, L. A.; KLEINERMANN, F.; De Troyer, O. Applying semantic web technology to feature modeling. In: Proceedings of the 2009 ACM symposium on Applied Computing. New York, NY, USA: ACM, 2009. (SAC '09), p. 1252–1256. ISBN 978-1-60558-166-8.

[159] ZHANG, D.; ADIPAT, B. Challenges, Methodologies, and Issues in the Usability Testing of Mobile Applications. International Journal of Human-Computer Interaction, Taylor & Francis, v. 18, n. 3, p. 293–308, jul. 2005.

[160] ZHANG, W.; ZHAO, H.; MEI, H. A propositional logic-based method for verification of feature models. In: DAVIES, J.; SCHULTE, W.; BARNETT, M. (Ed.). *ICFEM*. [S.l.]: Springer, 2004. (Lecture Notes in Computer Science, v.3308), p. 115–130. ISBN 3-540-23841-7.