Universidade Federal de Goiás
Instituto de Informática

Flayson Potenciano e Silva

Abordagem baseada em Metamodelos para a Representação e Modelagem de Características em Linhas de Produto de Software Dinâmicas

Goiânia
2016

sistema de bibliotecas ufg

UFG

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR AS TESES E DISSERTAÇÕES ELETRÔNICAS NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

1. Identificação do material bibliográfico: **[X] Dissertação** **[ ] Tese**

2. Identificação da Tese ou Dissertação

1. Nome completo do autor: Flayson Potenciano e Silva

2. Título do trabalho: Abordagem baseada em Metamodelos para a Representação e Modelagem de Características em Linhas de Produto de Software Dinâmicas

5. Informações de acesso ao documento:

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF da tese ou dissertação.

Flayson Potenciano e Silva
Assinatura do(a) autor(a)

Data: 13 / 09 / 2016

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.

Flayson Potenciano e Silva

Abordagem baseada em Metamodelos para a Representação e Modelagem de Características em Linhas de Produto de Software Dinâmicas

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. Sérgio Teixeira de Carvalho.

Goiânia
2016

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Potenciano e Silva, Flayson
Abordagem baseada em Metamodelos para a Representação e Modelagem de Características em Linhas de Produto de Software Dinâmicas [manuscrito] / Flayson Potenciano e Silva. - 2016.
CXIII, 113 f.: il.

Orientador: Prof. Dr. Sérgio Teixeira de Carvalho.
Dissertação (Mestrado) - Universidade Federal de Goiás, Instituto de Informática (INF), Programa de Pós-Graduação em Ciência da Computação, Goiânia, 2016.
Bibliografia. Apêndice.
Inclui siglas, tabelas, lista de figuras, lista de tabelas.

1. Linha de Produto de Software Dinâmica (LPSD). 2. Modelo de Características. 3. Modelo de Configuração de Características. 4. Requisitos em tempo de execução. 5. Metamodelo. I. Teixeira de Carvalho, Sérgio, orient. II. Título.

CDU 004

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UFG

ATA Nº 17/2016

ATA DA SESSÃO DE JULGAMENTO DA DISSERTAÇÃO DE MESTRADO DE FLAYSON POTENCIANO E SILVA

Aos seis dias do mês de setembro de dois mil e dezesseis, às quinze horas, na sala 150 do Instituto de Informática da Universidade Federal de Goiás, Campus Samambaia, reuniu-se a banca examinadora designada na forma regimental pela Coordenação do Curso para julgar a dissertação de mestrado intitulada **"Abordagem baseada em Metamodelos para a Representação e Modelagem de Características em Linhas de Produto de Software Dinâmicas"**, apresentada pelo aluno Flayson Potenciano e Silva como parte dos requisitos necessários à obtenção do grau de Mestre em Ciência da Computação, área de concentração Ciência da Computação. A banca examinadora foi presidida pelo orientador do trabalho de dissertação, Professor Doutor Sérgio Teixeira de Carvalho (INF/UFG), tendo como membros o Professor Doutor Alan Keller Gomes (IFG – Campus Inhumas) e a Professora Doutora Juliana Pereira de Souza-Zinader (INF/UFG). Aberta a sessão, o candidato expôs seu trabalho. Em seguida, o aluno foi arguido pelos membros da banca e:

(X) tendo demonstrado suficiência de conhecimento e capacidade de sistematização do tema de sua dissertação, a banca concluiu pela **aprovação** do candidato, sem restrições.

( ) tendo demonstrado suficiência de conhecimento e capacidade de sistematização do tema de sua dissertação, a banca concluiu pela **aprovação** do candidato, condicionado a satisfazer as exigências listadas na Folha de Modificação de Dissertação de Mestrado anexa à presente ata, no prazo máximo de 60 dias, a contar da presente data, ficando o professor orientador responsável por atestar o cumprimento dessas exigências.

( ) não tendo demonstrado suficiência de conhecimento e capacidade de sistematização do tema de sua dissertação, a banca concluiu pela **reprovação** do candidato.

Os trabalhos foram encerrados às dezoito horas. Nos termos do Regulamento Geral dos Cursos de Pós-Graduação desta Universidade, lavrou-se a presente ata que, lida e julgada conforme, segue assinada pelos membros da banca examinadora.

Prof. Dr. Sérgio Teixeira de Carvalho _____

Prof. Dr. Alan Keller Gomes _____

Profa. Dra. Juliana Pereira de Souza-Zinader _____

Flayson Potenciano e Silva

Graduou-se em Bacharelado em Informática pelo IFG - Instituto Federal de Goiás / Câmpus Inhumas. Durante sua graduação, foi monitor de Cálculo I e bolsista do CNPq em um trabalho de iniciação científica intitulado “Modelagem Matemática para Seleção de Áreas Prioritárias para Conservação na Região do Município de Inhumas e na Região Metropolitana de Goiânia: Diagnóstico, Mapeamento e Caracterização Biofísica”. Realizou estágio na Fábrica de Software na mesma instituição onde graduou-se e colaborou em pesquisas para propor uma metodologia de desenvolvimento de software para projetos do IFG / Câmpus Inhumas. Durante o Mestrado, na UFG - Universidade Federal de Goiás, foi bolsista da CAPES. Atualmente é professor substituto de Informática no IFG / Câmpus Inhumas.

Dedico este trabalho à minha família, ao INF/UFG e a todos aqueles que têm o interesse de aprofundar em representação de características para Linha de Produtos de Software Dinâmicas.

Agradecimentos

A Deus, por meio da minha fé e perseverança para a realização deste grande passo na minha vida acadêmica.

Ao meu orientador, Sérgio, pela sua paciência, dedicação, persistência, sinceridade e principalmente pela sua humildade.

Ao INF pelo acolhimento e estrutura para a minha pesquisa, em especial a dedicação da secretária Mirian que sempre se dispôs em atender a todos os alunos de pós-graduação.

À CAPES pelo auxílio financeiro.

E por todos os ensinamentos e motivações que a minha família me ofereceu.

Na vida, não existe nada a temer, mas a entender.

Marie Curie,
Tempus Fugit.

Resumo

Potenciano e Silva, Flayson. **Abordagem baseada em Metamodelos para a Representação e Modelagem de Características em Linhas de Produto de Software Dinâmicas**. Goiânia, 2016. 113p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Esta dissertação apresenta uma abordagem de representação de requisitos para Linhas de Produto de Software Dinâmicas (LPSD). LPSDs são voltadas para a produção de aplicações adaptativas e cada requisito é representado como uma característica. Tradicionalmente, características são representadas em uma Linha de Produto de Software (LPS) por meio de um Modelo de Características (MC). Tal modelo, no entanto, não possui, originalmente, suporte para a representação de características dinâmicas. Esta dissertação propõe uma extensão ao MC, incorporando uma representação para as características dinâmicas, de forma que o modelo tenha maior expressividade quanto às condições de mudanças de contexto e da própria aplicação. Para isso, um metamodelo baseado no meta-metamodelo Ecore foi desenvolvido, para possibilitar a definição tanto de Modelos de Características Dinâmicas (extensão do MC proposta) quanto também de Modelos de Configuração de Características Dinâmicas (MCC-D), estes utilizados para descrever as possíveis configurações dos produtos em tempo de execução. Além de uma representação para características dinâmicas e do metamodelo, essa dissertação traz como contribuição uma ferramenta que interpreta o metamodelo proposto e permite a construção de Modelos de Características Dinâmicas. Simulações envolvendo mudanças de estado das configurações de características dinâmicas foram realizadas, considerando cenários de uma aplicação ubíqua de monitoramento de pacientes domiciliares.

Palavras–chave

Linha de Produto de Software Dinâmica (LPSD), Modelo de Características, Modelo de Configuração de Características, Requisitos em tempo de execução, Metamodelo.

Abstract

Potenciano e Silva, Flayson. **Metamodel Based Approaches for Representation and Features Modeling in Dynamic Software Product Lines**. Goiânia, 2016. 113p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

This dissertation presents a requirement representation approach for Dynamic Software Product Lines (DSPLs). DSPLs are oriented towards the designing of adaptive applications and each requirement is represented as a feature. Traditionally, features are represented in a Software Product Line (SPL) by a Feature Model (FM). Nonetheless, such a model does not originally support dynamic features representation. This dissertation proposes an extension to FM by adding a representation for dynamic feature to it so that the model can have a higher expressivity regarding the context change conditions and the application itself. Therefore, a metamodel based on Ecore metametamodel has been developed to enable the definition of both Dynamic Feature Models (proposed extension to FM) and Dynamic Feature Configurations (DFC), the latter used to describe the possible configuration of products at-runtime. In addition to a representation for dynamic features and the metamodel, this dissertation provides a tool that interprets the proposed model and allows Dynamic Feature Models design. Simulations involving dynamic feature state changes have been carried out, considering scenarios of a ubiquitous monitoring application for homecare patients.

Keywords

Dynamic Software Produc Line, Feature Model, Feature Model Configuration, Requeriments at-runtime, Metamodel.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

D-FC	Dynamic Feature Configuration
D-FM	Dynamic Feature Model
DSL	Domain Specific Language
DSPL	Dynamic Software Product Line
EMF	Eclipse Modeling Framework
FC	Feature Configuration
FM	Feature Model
FODA	Feature-Oriented Domain Analysis
FOSD	Feature-Oriented Software Development
GMF	Graphical Modeling Framework
IDE	Integrated Development Environment
LDS	Linguagem de Domínio Específico
LPS	Linha de Produto de Software
LPSD	Linha de Produto de Software Dinâmica
MC	Modelo de Características
MC-D	Modelo de Características Dinâmicas
MCC	Modelo de Configuração de Características
MCC-D	Modelo de Configuração de Características Dinâmicas
MER	Modelo Entidade Relacionamento
MOF	Meta-Object Facility
MVC	Model View Controller
OMG	Object Management Group
S.P.L.O.T	Software Product Lines Online Tools
SPL	Software Product Line
UML	Unified Modeling Language
XML	eXtensible Markup Language

CAPÍTULO 1

Introdução

Esta dissertação utiliza a Linha de Produto de Software Dinâmica (LPSD), do inglês, *Dynamic Software Product Line* (DSPL) [24], [26], como uma abordagem no sentido de se modelar aplicações ubíquas, em especial no que se refere à modelagem de características (requisitos), de aplicações autoadaptativas, que podem mudar em tempo de execução.

Mudanças realizadas pela própria aplicação, caracteriza-se como uma aplicação autoadaptativa (do inglês, *self-adaptive*). De acordo com [34], uma aplicação autoadaptativa modifica o seu próprio comportamento de acordo com o ambiente, realizando mudanças em tempo de execução [34] [18], com o objetivo de se adequar a requisitos externos. Por exemplo, o fato de um recurso computacional ou mesmo uma pessoa, estar presente ou não em um lugar por um determinado período, pode provocar mudanças no ambiente, e, por conseguinte, adaptações na aplicação. Nesse cenário, uma aplicação autoadaptativa, deve reconhecer tal condição e se adaptar diante das mudanças do ambiente relacionadas ao recurso e/ou à pessoa. Uma Linha de Produto de Software (LPS), do inglês, *Software Product Line* (SPL) com a capacidade de derivar produtos de software autoadaptativos é denominada de LPSD.

A LPS ou LPSD contém um conjunto de funcionalidades comuns e outro conjunto de funcionalidades que podem ser selecionadas no desenvolvimento de um produto. Tais funcionalidades são chamadas de características, do inglês, *features*, e se referem aos requisitos dos produtos da linha. A LPS, por padrão, tem seus requisitos representados na forma de Modelos de Características (MC), do inglês *Feature Model*, (FM) [28]. O MC representa todas as características possíveis para a seleção que a linha pode gerar. O produto gerado a partir de um MC é representado em um Modelo de Configuração de Características (MCC), do inglês, *Feature Configuration* (FC). O MCC, por sua vez, representa as características selecionadas, por meio do MC, em tempo de implantação.

1.1 Cenário e Motivação

Esta dissertação considera um cenário, para uma melhor contextualização, de uma aplicação ubíqua para monitoramento de saúde em casa (*homecare*) [29].

A aplicação é customizada a partir de requisitos levantados junto aos seus usuários. De acordo com [29], os requisitos de usuários variam conforme o tipo do usuário. Por exemplo, no cenário de *homecare*, podem envolver usuários da área de saúde (médico, enfermeiro, cuidador), paciente(s) e seus familiares. Um profissional de saúde pode exigir requisitos essenciais para um melhor monitoramento e de ações de saúde do paciente. Já o paciente pode exigir requisitos de acordo com a sua saúde e que não interferem no monitoramento da sua saúde, como, por exemplo, requisitos para o conforto em sua casa. Os familiares do paciente podem ter requisitos para acompanhar o estado do paciente, porém, não podem interferir no monitoramento de saúde do paciente. Portanto, os requisitos correspondem as funcionalidades da aplicação, representados como características, e são selecionados para a implantação na casa.

Um paciente idoso é consultado por um profissional de saúde e após realizar consultas e exames, o profissional de saúde oferece a alternativa de se instalar uma aplicação domiciliar de monitoramento da sua saúde.

Durante a execução, a aplicação realiza coletas de dados fisiológicos, temperatura e principalmente de saúde do paciente via sensores. Por exemplo, sensores para monitorar a pressão, os batimentos cardíacos, as atividades físicas, entre outros [13]. Além disso, tal aplicação pode também realizar monitoramento em relação às condições do ambiente, como a temperatura e a umidade. Os dados coletados pela aplicação podem ser utilizados por profissionais de saúde, para auxiliar na definição do plano de cuidados do paciente de acordo com a evolução do seu estado de saúde. O profissional responsável pelo paciente pode alterar a prescrição medicamentosa e realizar consultas. Os familiares podem ainda acompanhar o paciente pela aplicação.

Caso o paciente necessite de alguma emergência, ele poderá acionar um botão ou algum outro dispositivo para notificar os profissionais de saúde. Se os dados coletados descreverem alguma tendência de agravamento de saúde, a central terá o conhecimento desta situação no sentido de tomar as providências cabíveis para o paciente.

Os usuários e a própria aplicação podem ativar novas funcionalidades ou ainda desativar as funcionalidades implantadas de acordo com as suas ne-

cessidades. Quando o processo de ativação e desativação de funcionalidades é realizada durante a sua execução, a aplicação realiza adaptações em tempo de execução.

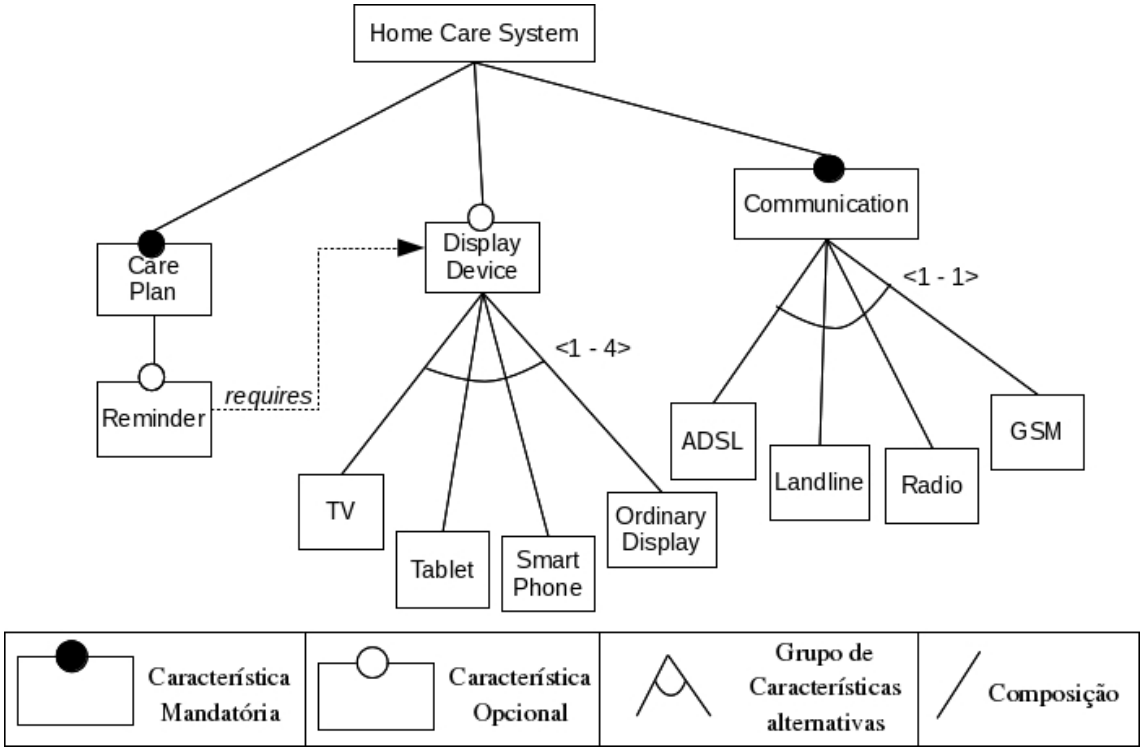


Figura 1.1: *Exemplo de Modelo de Características de uma LPS, adaptado de [12].*

Tais funcionalidades podem ser visualizadas pela TV, computadores, *smart phones* ou *tablets*. Por outro lado, a aplicação pode intervir em relação a componentes críticos. Por exemplo, caso a comunicação com a Internet via ADSL falhe, a aplicação deve escolher uma outra forma de comunicação até que a ADSL se normalize novamente, veja o exemplo de MC na Figura 1.1. As interfaces possibilitam a compreensão dos usuários, por meio da visualização das funcionalidades selecionadas ou contratadas, além do status da situação e do custo financeiro das mesmas.

1.2 Identificação do Problema

Ainda considerando o cenário, aplicações de *homecare* têm sido construídas por meio de técnicas de computação ubíqua [13], [29] e [12]. Conforme apresentado na seção anterior, uma LPSs não possui originalmente suporte para computação ubíqua, essencial para lidar com informações de contexto, como, por exemplo, disponibilidade de recursos e localização do paciente, e

ainda com informações que possam identificar a evolução do estado de saúde do paciente. Informações de contexto são importantes no sentido de se permitir mudanças das funcionalidades da aplicação em tempo de execução.

A modelagem de características em aplicações ubíquas pode ser feita de acordo com as necessidades de cada paciente [29], definindo-se personalizações na forma de variabilidades estáticas modeladas em tempo de desenvolvimento [12]. No entanto, isso não é suficiente quando se demanda suporte para que as personalizações também possam ser definidas ou modificadas em tempo de execução, ou, em outras palavras, na forma de variabilidades dinâmicas. Com variabilidades dinâmicas, LPSDs podem ser modeladas de forma que permitam a criação de diferentes configurações de características (Modelos de Configuração de Características) em tempo de execução para suportar mudanças de requisitos do usuário.

O MC convencional não têm suporte para a concepção de aplicações ou sistemas autoadaptativos [25], [26] e ubíquos [12], fazendo-se necessário incrementar algumas funcionalidades no MC para suportar a adaptação. Por exemplo, não é possível representar a característica dinâmica *Communication* (MC da Figura 1.1) para uma *homecare*.

Alguns autores propõem modelagens de LPSD e/ou MCs [6], [11], [8], e [20], utilizando diferentes técnicas de representação de requisitos em tempo de execução, como em [10], [27], [36], [39] e [45]. No entanto, não apresentam em conjunto a representação de características dinâmicas em relação às características. Além disto, as propostas dos autores não são formalizadas para a modelagem. A abordagem por meio de metamodelos [33], é uma solução para formalizar propostas de modelos.

1.3 Objetivos

O objetivo geral desta dissertação é propor uma abordagem para representar características dinâmicas a partir do MC e do MCC convencionais. A representação de características dinâmicas e/ou do grupo de características dinâmicas devem diferenciar-se das características do modelo original do MC e do MCC.

As propostas de extensão do MC-D e do MCC-D não há uma formalização para estes modelos e, portanto, um metamodelo pode ser útil para este fim. Por sua vez, o metamodelo definir as diretrizes ou regras para a modelagem dos modelos propostos (MC-D e MCC-D), como, por exemplo, regras

de composição, cardinalidade do grupo de características para características estáticas ou dinâmicas e entre outras regras presentes no metamodelo.

O desenvolvimento de uma ferramenta, permite a validação dos modelos estendidos (MC-D e o MCC-D) e do metamodelo. A ferramenta deve permitir a interpretação do metamodelo proposto, a modelagem de MC-D e de MCC-D e realizar simulações no MCC-D. O cenário de *homecare* pode-se gerar produtos ou aplicações representados, como características selecionadas, em um MCC-D.

Os objetivos específicos podem ser divididos a seguir:

- Propor uma abordagem de representação, uma extensão do MC para MC-D e do MCC para MCC-D, que reflita na natureza dinâmica das LPSDs;
- Propor um metamodelo Ecore que permita, definir Modelo de Características Dinâmicas (MC-D) e Modelo de Configuração de Características Dinâmicas (MCC-D);
- Desenvolver uma ferramenta *web* que permita criar estes modelos tendo como base o metamodelo proposto. A modelagem, por meio da ferramenta, deve seguir as regras definidas no metamodelo e da própria ferramenta. O MC-D e MCC-D são persistidos em um banco de dados, derivar produtos conforme as características definidas no MC-D;
- A ferramenta deverá permitir a realização de simulações de derivação, envolvendo mudanças de estados de características estáticas ou dinâmicas no MCC-D;
- Validar a proposta usando o cenário de *homecare* como exemplo, representado em um MC-D, para derivar produtos representados no MCC-D.

1.4 Etapas Realizadas na Pesquisa

As etapas da pesquisa estão detalhas na Tabela 1.1.

Tabela 1.1: *Etapas da pesquisa*

Fases	Atividades
1º Fase	Pesquisas bibliográficas a certa de requisitos em tempo de execução
2º Fase	Proposta de extensão do MC e do MCC
3º Fase	Desenvolvimento de um metamodelo para o MC-D e para o MCC-D
4º Fase	Desenvolvimento da proposta de ferramenta para modelagem de MC-D e MCC-D
5º Fase	Ampliação da proposta de ferramenta para simular mudanças de características em MCC-D
6º Fase	Conclusão da proposta

Na primeira fase da pesquisa foram realizadas pesquisas bibliográficas sobre as principais propostas de requisitos em tempo execução e de representação de características dinâmicas para uma LPSD.

Na segunda fase, propôs-se a abordagem de representação para características dinâmicas (MC-D e MCC-D) estendendo os modelos tradicionais de características e de configuração de características.

Já na terceira fase, iniciou-se o desenvolvimento do metamodelo para modelagem, enquanto que na quarta fase, iniciou-se o desenvolvimento da ferramenta de modelagem.

Na quinta fase, a ferramenta foi ampliada para suportar simulações de mudanças de características estáticas e dinâmicas por meio de uma linguagem de domínio específico. Durante o desenvolvimento da ferramenta às propostas do metamodelo e da representação de características dinâmicas, as quais, foram aperfeiçoadas ao longo da implementação.

Por fim, a sexta fase se refere à conclusão, envolvendo a análise dos resultados obtidos, identificação das suas limitações e de trabalhos futuros. Foi possível concluir que a proposta se mostrou adequada para representar características dinâmicas em uma LPSD.

1.5 Contribuições

A contribuição dessa dissertação está na representação de características dinâmicas em modelos de características de uma LPSD. De forma mais específica, as principais contribuições são:

- O metamodelo para definir modelos de características e modelos de configuração de características dinâmicas em uma LPSD;
- Uma ferramenta que permite, a partir do conhecimento do metamodelo proposto, a criação de MC-D e de MCC-D, e a simulação da derivação de produtos.

1.6 Estrutura da Dissertação

Esta dissertação é composta pelos seguintes capítulos:

- Capítulo 2 - Apresenta os fundamentos a respeito de requisitos em tempo de execução, LPSD, MC, MCC, aplicações ubíquas, metamodelos e modelagem Ecore.
- Capítulo 3 - Apresenta o metamodelo para representação de Modelo de Características Dinâmicas (MC-D) e Modelo de Configuração de Características Dinâmicas (MCC-D). Além disso, são apresentados exemplos de derivações de produtos, para uma LPSD. Na última seção é feita uma comparação de trabalhos relacionados com a proposta deste capítulo.
- Capítulo 4 - Ferramenta e Cenários de Uso. Apresenta uma ferramenta capaz de interpretar metamodelos e, criar modelos MC-D e MCC-D. Além disso, são apresentadas simulações envolvendo cenários de *homecare*. Por fim, são realizadas comparações com outras propostas de ferramentas correlatas.
- Capítulo 5 - Apresenta, além das conclusões da pesquisa, as suas limitações e as possibilidades de trabalhos futuros.

CAPÍTULO 2

Fundamentação Teórica

Este capítulo contém a fundamentação teórica para o desenvolvimento da pesquisa abordando requisitos em tempo de execução, Linha de Produto de Software Dinâmica, modelagem e configuração de características, aplicações sensíveis ao contexto, metamodelo, modelagem Ecore EMF e, por fim, os trabalhos relacionados.

2.1 Requisitos em Tempo de Execução

Em geral, os requisitos definidos na literatura de Engenharia de Requisitos (RE) ([38] e [35]) são utilizados para aplicações estáticas. No entanto, há aplicações que possuem a capacidade de se adaptarem dinamicamente sem a interferência humana. Esta capacidade garante à aplicação a possibilidade de realizar mudanças internas de acordo com requisitos internos, ou, ainda, de acordo com as mudanças de contexto.

Para acompanhar, alterar e representar as mudanças de uma aplicação, [36] defende que os requisitos de aplicações adaptativas devem ser modificados em tempo de execução. Em [16], no entanto, os autores afirmam que é impossível prever quais requisitos de uma aplicação são adaptados em tempo de execução, pois o contexto é imprevisível e a aplicação está sempre realizando mudanças para maximizar suas tarefas para o usuário final. Há uma grande lacuna entre os requisitos dinâmicos e os métodos de estruturação da arquitetura existente [16]. Também a aplicação deve coletar informações de contexto e do próprio comportamento do sistema para oferecer melhor serviço ao usuário final. Por sua vez, os requisitos relacionados ao contexto, dependendo da aplicação e da situação, podem sofrer mudanças inesperadas [36].

Por outro lado, [16] coloca que é possível prever um conjunto de mudanças no sentido de se definir previamente certos requisitos da aplicação. Estes requisitos, na maior parte, são estáticos (invariáveis) e são coletados,

por exemplo, durante a elicitação, análise, desenvolvimento e implantação da aplicação.

Alguns autores, [45], [39] e [27] afirmam que não há representações desenvolvidas para suportar tais requisitos dinâmicos, e propõem mudanças em técnicas correntes de representação de requisitos, no sentido de prover formas de representar requisitos em tempo de execução.

Em [27], os autores afirmam que há uma lacuna entre o documento de requisitos escrito e o modelo de requisitos dinâmicos em tempo de execução do sistema. Este problema é agravado pelo fato de que documentos de requisitos são em sua maioria informais e descritos em linguagem natural, enquanto o modelo de requisitos dinâmico deve ser formal. Os mesmos autores propõem utilizar o modelo i*, baseado em *Goal Model*, acrescentando algumas variáveis para suportar a representação em tempo de execução e as tomadas de decisão de acordo com o contexto. As tomadas de decisão são conhecidas como assertivas o que equivalem, segundo os autores, a requisitos. Caso uma assertiva seja quebrada, um analisador de impacto faz um novo cálculo para atender o requisito.

De acordo com [16], as linguagens atuais não são adequadas para lidar com a incerteza em relação às adaptações em tempo de execução. Em modelos do tipo *Goal*, como, por exemplo, o KAOS [27] utilizam a notação para requisitos em tempo de execução a linguagem natural, conhecida também como linguagem informal.

A pesquisa de [45] aborda as modelagens mais utilizadas como o *goal-oriented* (KAOS, i* e Tropos [10]). Outra técnica é conhecida como lógica temporal, do inglês, *temporal logic*, usada para a especificação de linguagens, como, por exemplo, *Linear Temporal Logic* (LTL) e *Fuzzy Branching Temporal logic* (FBTL). Há também a notação Z para especificar o comportamento de sistemas. Em relação a estados do sistema, e as transições de estado, neste grupo se destacam: cadeia de Markov, Rede de Petri e *Dynamic Decision Network* (DDN) [7]. A UML também pode ser usada para modelar o comportamento de um sistema.

O Modelo de Características (MC) é a forma de representação das características de uma Linha de Produto de Software (LPS), as quais podem ser consideradas como requisitos [45] e [12]. Cada característica representa uma funcionalidade da aplicação, podendo corresponder a um ou mais de seus componentes. O MC originalmente não permite a representação das características em tempo de execução. Há propostas de autores [11], [12], [6] e [21] no sentido de apresentar formas de se representar estas características

em tempo de execução, considerando a abordagem de Linha de Produto de Software Dinâmica (LPSD). As próximas seções apresentam algumas dessas propostas.

2.2 Linha de Produto de Software Dinâmica

Linha de Produto de Software Dinâmica, do inglês, *Dynamic Software Product Line* (DSPL) é uma proposta baseada na LPS. Uma LPS é composta por um conjunto de características de software que são compartilhadas e combinadas para a criação de aplicações. Em uma aplicação de *homecare*, por exemplo, o paciente pode escolher quais características devem ser selecionadas, tais como, sensores de pressão, de temperatura, entre outras, para serem implantadas em sua casa. A abordagem de LPS pode garantir menores custos, melhor qualidade de seus produtos e menos tempo de desenvolvimento [24], [32] e [17].

Segundo [24], a LPS lida com vários tipos de negócios, clientes e projetos. Devido à heterogeneidade dos usuários, a LPS deve ter um gerenciamento para as suas variabilidades, isto é, de todas as suas características. O gerenciamento destas características é realizado durante a montagem (ou derivação) dos produtos, portanto, antes que o produto seja colocado em execução. Durante a montagem das aplicações (ou produtos) pode-se ter três combinações para as características: características comuns a todos os produtos, características comuns a alguns produtos e características específicas (individuais). Uma linha de produto segue, portanto, um modelo de variabilidade, de acordo com o ciclo de vida da linha, contendo regras de seleção das características. Uma forma de modelar variabilidades é por meio de Modelos de Características (veja na seção 2.3).

No desenvolvimento de componentes de uma LPS, há dois tipos de ciclos de vida [24]: i) desenvolver componentes de software para reutilização e combinação em sistemas diferentes; e ii) desenvolvimento de novos softwares reutilizando partes definidas no primeiro ciclo. A estrutura da LPS, por sua vez, é dividida em duas partes: *Domain Engineering* e *Application Engineering*. A primeira refere-se à análise da linha de produto como um todo, enquanto que a segunda é responsável por criar partes específicas de produtos e pela integração destas partes específicas aos produtos individuais. Para essa proposta de dissertação, *Domain Engineering* refere-se à modelagem da linha de produto, e *Aplication Engineering* se refere à modelagem da configuração dos produtos derivados da linha.

LPSs são limitadas em relação a requisitos que mudam constantemente. Tais limitações podem ser tratadas por meio de adaptações em tempo de execução, providas pelas LPSs Dinâmicas (LPSD) [12], [14], [11] e [6]. A capacidade de uma LPSD em lidar com as adaptações está nas derivações de diferentes produtos em tempo de execução. Estas derivações são realizadas por meio de configurações dinâmicas do produto, ou em outras palavras, o produto de software se adapta seguindo regras preestabelecidas na linha.

Como colocado na seção 2.1, não há um modelo de requisitos consolidado para representar os requisitos em tempo de execução em LPSs. Originalmente a LPS representa seus requisitos por meio do Modelo de Características (MC).

2.3 Modelo de Características

As características da LPS equivalem aos requisitos de uma aplicação, sendo representadas por um Modelo de Características (MC), parte integrante da proposta de Kang *et al.* [28] denominada *Feature-Oriented Domain Analysis* (FODA) [28]. O MC é uma forma de representar, em um nível alto de abstração, um domínio ou sistema, inter-relacionando as características dos produtos da LPS. As características são propriedades do sistema consideradas relevantes aos usuários, sendo usadas para capturar os pontos em comum e as variabilidades entre os produtos da LPS [12] e [21].

O MC é representado como uma árvore, conforme apresentado na Figura 2.1. Este modelo é composto por uma raiz, conhecida como *root feature* (*Home Care System*, na Figura 2.1) e por três tipos de características: mandatórias, opcionais e alternativas.

As características mandatórias, representadas pelo *Care Plan* (Plano de Cuidados) e pelo *Communication* (Comunicação), indicam que a escolha deve ser obrigatória, ou seja, que devem estar presentes em todos os produtos a serem derivados da linha, pois são essenciais para a funcionalidade dos mesmos. As características opcionais, representadas por *Reminder* e *Display Device*, são opções de escolhas realizadas pelo usuário. Assim, durante a derivação de um determinado produto, estas características podem ou não estar presentes no produto.

Já as características alternativas são um grupo de alternativas representadas por um arco. Cada grupo contém regras de seleção definidas por uma cardinalidade. Na figura 2.1 há dois grupos de características, *Display Device* e *Communication*. *Display Device* contém alternativas para o paciente

visualizar informações do Plano de Cuidados, em que a cardinalidade <1 - 4> significa que se pode escolher no mínimo um tipo de visor ou no máximo todos os visores disponíveis. Entre os visores encontram-se as alternativas TV, *tablet*, *smart phone* e *Ordinary Display*. Outro exemplo de cardinalidade está na comunicação. De acordo com a cardinalidade <1 - 1>, apenas pode-se selecionar uma forma de comunicação entre as alternativas ADSL, *Landline*, *radio* e *GSM*.

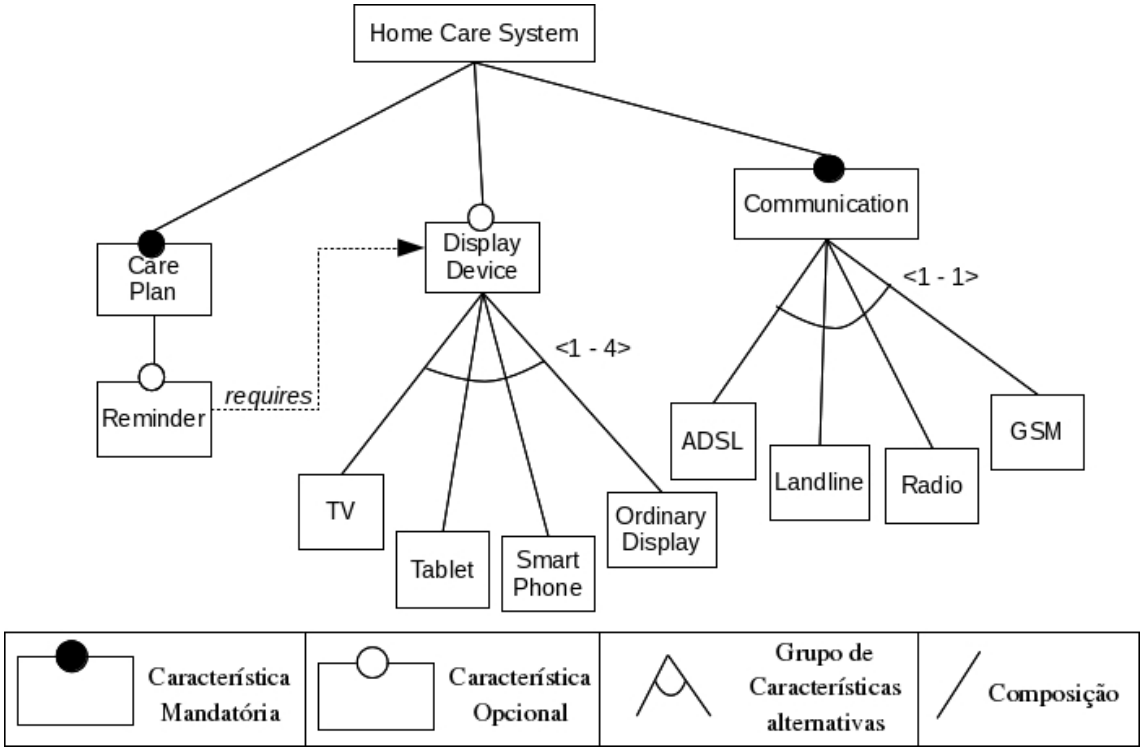


Figura 2.1: *Exemplo de Modelo de Características de uma LPS, adaptado de [12].*

Além das regras de grupos de alternativas, uma característica opcional também pode ter regras para a sua seleção, conhecidas como regras de composição. Há dois tipos de regras de composição: *requires*, quando a característica depende de uma ou mais outras características, e *excludes*, quando a característica exige que uma ou mais outras características sejam excluídas. Na Figura 2.1 há um exemplo de *excludes*, em que os recados podem apenas ser selecionados caso o visor também seja selecionado. Ao se modelar a linha deve-se deixar claras as regras de composição durante a criação do MC para que não ocorram falhas de dependência ou conflitos entre as características.

Como citado anteriormente, nessa dissertação, o Modelo Características (MC) equivale ao *Domain Engineering* em LPSs, enquanto que a configuração de características de um produto (por exemplo, aplicação a ser implantada

na casa de um determinado paciente) equivale à *Aplication Engineering*. Esta configuração é definida por meio do Modelo de Configuração de Características (MCC).

2.4 Modelo de Configuração de Características

O MC mantém todas as características a serem selecionadas para se derivar um produto. O Modelo de Configuração de Características (MCC), por sua vez, especifica as características selecionadas a partir do MC e que compõem a configuração de um determinado produto. Esse processo de seleção de características é denominado derivação de produto, ou ainda, derivação do MC.

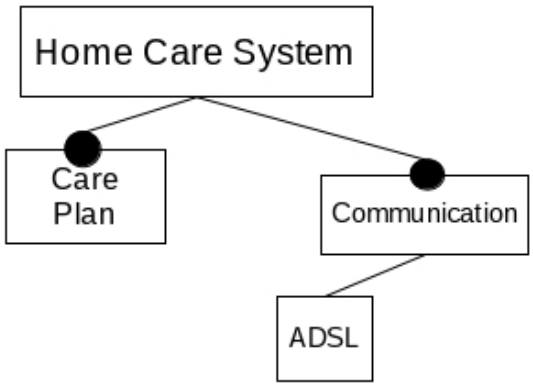


Figura 2.2: *Derivação da LPS (configuração mínima considerando o MC da Figura 2.1).*

De acordo com [24], o MCC derivado de uma LPS é estático, ou seja, depois de configurado o produto de software não pode ser alterado em tempo de execução, somente em tempo de derivação ou implantação.

A representação do MCC é diferente do MC, sendo mais simples em relação às variabilidades, pois o MC representa todos os produtos possíveis de serem derivados. As Figuras 2.2, 2.3 e 2.4 mostram três exemplos de derivações estáticas de acordo com o MC da Figura 2.1.

Ao representar como o produto já está configurado para a execução, não há necessidade de se informar as dependências das características que devem ser obedecidas ou o grupo de características. Apenas a representação da característica mandatória é apresentada em relação à característica opcional. Na Figura 2.2 está a configuração mínima que a MC pode oferecer ao usuário final. O paciente terá, neste caso, o plano de cuidados e a comunicação por meio da ADSL instalados na casa.

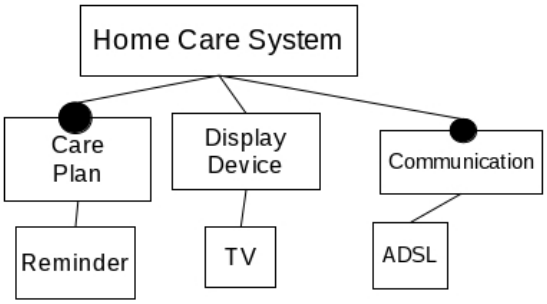


Figura 2.3: *MCC - Derivação da LPS (TV).*

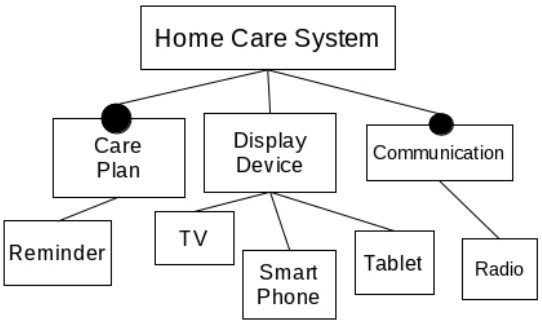


Figura 2.4: *MCC - Derivação da LPS (TV, smart phone e tablet).*

Se o paciente quiser que o Plano de Cuidados seja informado na forma de notificações utilizando alguns dos visores (características *Reminder* e *Display Device*), a aplicação deve passar por uma atualização *off-line* [4], isto é, pela parada do sistema e a implantação de uma nova versão da aplicação que inclua novas características. Na Figura 2.3 há outro exemplo de derivação com suporte apenas para a TV, além da comunicação via ADSL, e na Figura 2.4, há além da TV, o *smart phone* e o *tablet* e a comunicação por *radio*.

O MCC informa todas as características selecionadas de forma mais simplificada, comparando-se com o MC. Cada aplicação contém um MCC e representa uma derivação do MC. A desvantagem desta abordagem em LPS é a necessidade de realizar as derivações somente em tempo de implantação, portanto, antes da execução do produto. Se o paciente necessitar de novas características (requisitos), a sua aplicação deve ser parada para se implantar as novas funcionalidades exigidas pelo usuário e ser, então, iniciada novamente.

2.5 Aplicações Ubíquas

De acordo com [44], a computação ou a aplicação ubíqua tem o propósito de colaborar na execução de tarefas computacionais, sem a percepção do

usuário. A aplicação ubíqua tem utilizado conceitos de computação sensível ao contexto para perceber e executar mudanças de contexto [37]. Um contexto, de acordo com A Bowd *et al.* [1] e Dey *et al.* [19], é uma informação para caracterizar a situação de entidades, como, por exemplo, pessoas, objetos, lugares, funcionalidades da aplicação, etc.

Vieira *et al.* [43] afirma que um Sistema Sensível ao Contexto (SSC), do inglês, *Context-Sensitive System* (CSS), coleta informação de domínio, por meio de gerenciamento de contextos presentes, para ajudar a aplicação na execução de tarefas. A informação de contexto pode originar-se em decorrência do próprio usuário da aplicação (entrada explícita), pela base de dados de conhecimentos contextuais (entrada inferida) e pelo monitoramento do ambiente (entrada percebida) (Figura 2.5).

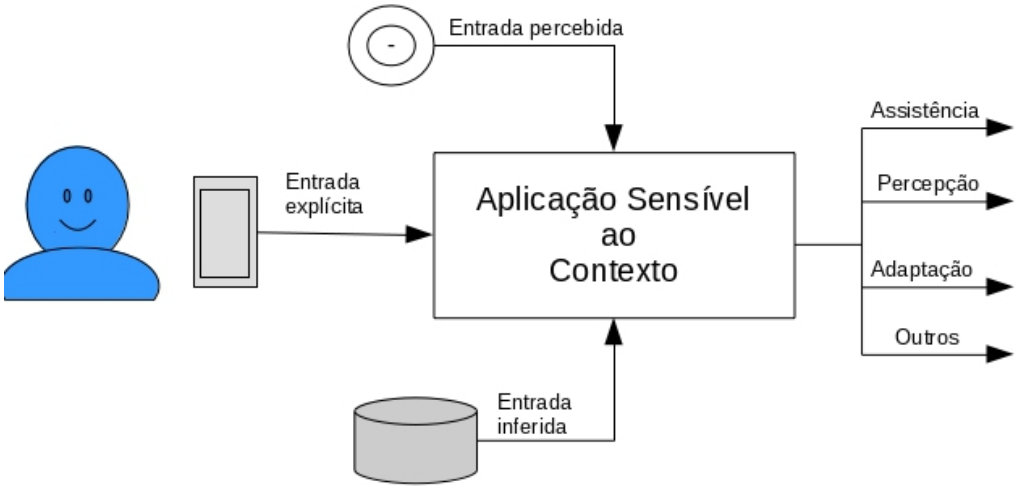


Figura 2.5: *Estrutura de uma aplicação sensível ao contexto, adaptado de [43]*

Uma aplicação de *homecare* ubíqua pode conter entradas do tipo inferida, explícita e percebida. As entradas percebidas podem ser representadas por sensores, como, por exemplo, sensores de umidade, de temperatura, de presença, entre outros. Entradas explícitas são mudanças decorrentes do usuário, isto é, ele pode interferir na decisão preestabelecida da aplicação, como, por exemplo, a mudança de temperatura do ar-condicionado pelo controle remoto. Já as entradas inferidas contêm regras de contexto preestabelecidas para acionar determinadas ações na aplicação de *homecare*. O controle de temperatura é um exemplo de entrada inferida, em que a aplicação deve ter uma configuração preestabelecida para acionar determinados dispositivos de acordo com a mudança de temperatura. Por exemplo, se o sensor de temperatura perceber que a temperatura está acima de 36 graus, a aplicação deve

acionar um atuador (*e.g.*, ventilador, ar-condicionado) para alcançar a temperatura desejada.

A adaptação em tempo de execução, por parte de uma aplicação ubíqua, permite mudanças de contexto para maximizar ou acrescentar serviços conforme o cenário da aplicação. Na seção 4.3 são discutidas simulações de mudanças de contexto, voltados para uma *homecare*.

2.6 Metamodelo

Os níveis de modelagem utilizados nesta dissertação são baseados na versão 1.4 da *Object Management Group* (OMG) [33].

O nível padrão de meta-metamodelo da OMG é o MOF (*Meta-Object Facility*). O MOF oferece uma estrutura para suportar a definição de outros modelos, seguindo a estruturação clássica de quatro camadas [5]. Os quatro níveis da arquitetura, conforme a tabela 2.1, são: M0, M1, M2 e M3.

Tabela 2.1: *Níveis arquiteturais, adaptado de [33]*

Níveis	Exemplo
M3 - Meta-metamodelo	MOF, Ecore.
M2 - Metamodelo	UML, Metamodelo proposto nesta dissertação.
M1 - Modelos	Diagrama de Classes da UML, MC, MCC.
M0 - Instâncias	Artefatos de software, módulos, códigos-fonte.

Os níveis inferiores dependem dos superiores, isto é, são instâncias do nível superior. O nível mais alto, de acordo com a OMG, é o M3, conhecido como meta-metamodelo. Nesta camada, além do MOF, há o meta-metamodelo Ecore. O Ecore surgiu como uma implementação do MOF, e evoluiu por conta da experiência resultante da construção de ferramentas [40], [12].

No nível M2, metamodelos podem ser definidos, seguindo-se as regras, classes, relacionamentos e outras restrições determinadas pelo nível M3. Um exemplo de metamodelo é a especificação UML, desenvolvida a partir do meta-metamodelo MOF. A proposta dessa dissertação de um metamodelo está no nível M2, sendo empregado para o seu desenvolvimento o meta-metamodelo Ecore.

Já o nível M1 contém os modelos definidos a partir do nível M2. São os próprios modelos utilizados na engenharia de um software, incluindo o MC e o MCC de LPSs e LPSDs.

Por fim, o nível M0 se refere aos artefatos, módulos ou códigos do sistema. Trata-se da definição de nível mais baixo, pois representa o produto concretizado para a implantação.

2.7 Modelagem Ecore

A modelagem por meio do meta-metamodelo Ecore pode ser desenvolvida em anotações Java, UML, XML Schema ou outras formas de código [40]. A Figura 2.6 ilustra a hierarquia deste modelo.

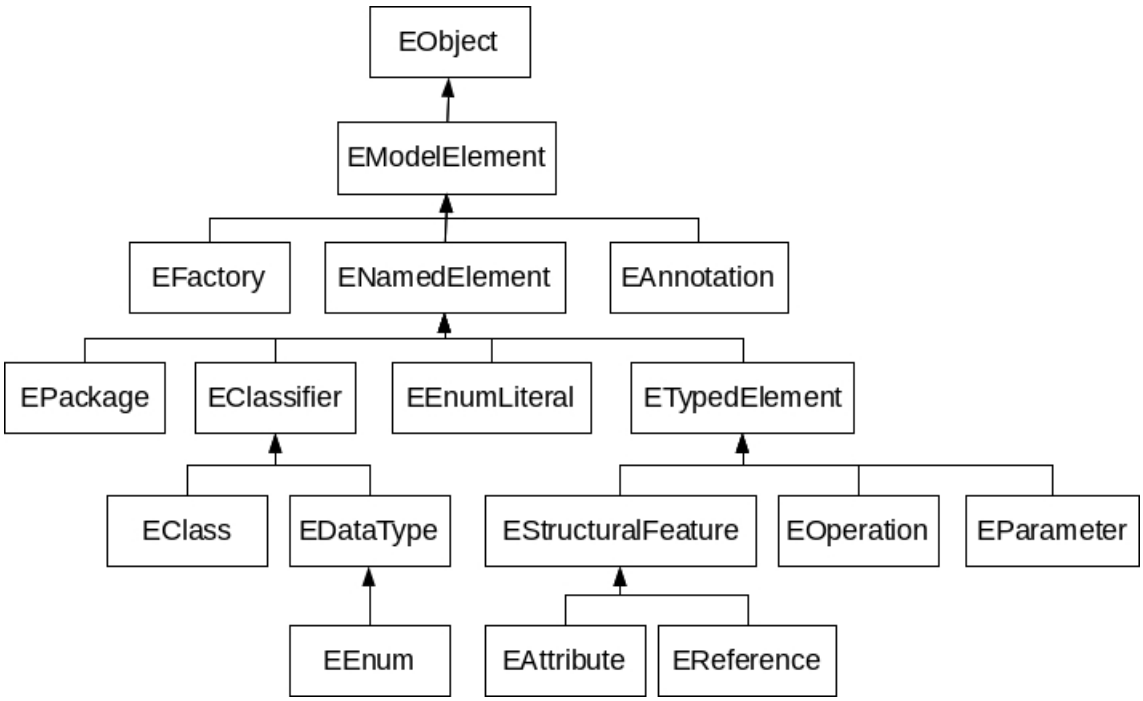


Figura 2.6: *Hierarquia de classes do meta-metamodelo Ecore. Adaptado de [23]*

A Figura 2.6 contém as seguintes classes: `EObject`, `EModelElement`, `EFactory`, `ENamedElement`, `EAnnotation`, `EPackage`, `EClassifier`, `EEnumLiteral`, `ETypedElement`, `EClass`, `EDataType`, `EStructuralFeature`, `EOperation`, `EParameter`, `EEnum`, `EAttribute` e a `EReference` [40]. As descrições destas classes são abordadas a seguir.

- `EObject`: raiz de todas as classes. Permite que todos os objetos do modelo Ecore comecem com a letra “E” maiúscula para distinguir os objetos das instâncias do metamodelo.
- `EModelElement`: superclasse da classe `EAnnotation`. Composto de `EAnnotations`, permite a todas classes no Ecore incluírem a referência `Eannotations`.

- `EFactory`: responsável por criar instâncias de classes no modelo Ecore.
- `ENamedElement`: derivado de outras superclasses. Define um atributo nome para as classes no Ecore.
- `EAnnotation`: informa se o objeto pode ser ligado a um objeto de um modelo Ecore. Esta classe contém um atributo de origem para armazenar o URI para representar o tipo de anotação. Por exemplo, a anotação URI (*namespace*) no Ecore, utilizada para modelar os metamodelos desta dissertação, é configurada como: "http://www.eclipse.org/emf/2002/Ecore".
- `EPackage`: representa os pacotes para `EClassifiers`.
- `EClassifier`: classe base. Tem o objetivo comum de referência de `ETypedElements` e permite definir estruturas, operações e parâmetros para especificar os tipos de classes ou tipos de dados.
- `EEnumLiteral`: especificado pelo `EENum` e herda de um nome de `EnamedElement`.
- `ETypedElement`: define atributos de multiplicidade. São especificados pelos atributos `lowerBound` (mínimo) e `upperBound` (máximo) para o relacionamento no Ecore.
- `EStructuralFeature`: classe do tipo `EtypedElement`. É herdado pelas classes `EReference` e `EAttribute`. Os atributos `lowerBound` e `upperBound` de `EtypedElement` representam o relacionamento do elemento.
- `EAttribute`: representa um atributo. Cada atributo contém um nome e um tipo.
- `EReference`: representam associações entre as `EClass`. As referências possíveis são as seguintes: referência de direção única, bidirecional, `containment` (contenção) ou `composition` (composição) e `SuperType` (herança). A direção única deve ter a referência da `EClass`. A bidirecional deve ter referência da `EClass` oposta, representada por uma reta de duas setas opostas. O `containment` serve para representar uma forte associação entre as `EClass`. O `EReference` contém duas referências e três atributos.
- `EOperation`: define referência adicional, exceções. Pode ter zero ou mais parâmetros ou `EClassifiers`.
- `EParameter`: um tipo de `EtypeElement`. Pode acessar operações da classe `Eoperation`.
- `EClass`: representa o modelo da própria classe. Contém um nome e pode ter atributos e referências. Um relacionamento `eSuperTypes` (herança) permite uma superclasse ter de zero a várias classes herdeiras, sendo

este relacionamento exclusivo para herança de uma outra `Eclass`.

- `EDataType`: representa tipos de variáveis que não são modelados em classes. Pode ser associado a um objeto primitivo da linguagem Java. Por exemplo, a classe `EInteger` é uma instância da classe `Integer` Java. Além disso, o `EDataType` é identificado por nome e tipo de atributo.
- `EEnum`: um tipo especial de dado que define explicitamente uma lista de valores literais.

2.8 Considerações Finais

A LPS representa as características na forma de modelos (MC e MCC), as quais podem ser consideradas como requisitos. Porém, a LPS não é capaz de realizar adaptações em tempo de execução.

Aplicações autoadaptativas podem ser desenvolvidas por meio de uma LPSD. O desafio, no entanto, é a representação de características dinâmicas para MC e MCC. A formalização destes modelos por meio de metamodelos é uma forma de se definir em alto nível como as características podem ser representadas, sendo que o conceito de meta-metamodelo Ecore é a base da implementação da ferramenta proposta para interpretação de metamodelos.

O próximo capítulo apresenta a proposta de representação de características dinâmicas para MC e MCC formalizada em metamodelos, além de exemplos de derivações em tempo de implantação e em tempo de execução.

CAPÍTULO 3

Metamodelo e Representação de Características Dinâmicas

Neste capítulo é apresentada a proposta de um metamodelo que permite estender Mcs e MCCs, representando características dinâmicas nestes modelos. Neste contexto, os modelos que podem ser criados a partir do metamodelo proposto são denominados por MC-D (Modelo de Características Dinâmicas) e MCC-D (Modelo de Configuração de Características Dinâmicas). O capítulo apresenta ainda exemplos de modelos criados a partir do metamodelo, incluindo derivações de produtos.

3.1 O Metamodelo

A partir do metamodelo proposto na Figura 3.1, pode-se modelar MC-Ds e MCC-Ds (Seção 2.6). MC-D e MCC-D são extensões aos modelos MC e MCC, respectivamente, acrescidos com a possibilidade de se modelar e representar características dinâmicas.

Durante a criação de um MC-D, deve-se seguir as diretrizes do metamodelo correspondente, ou seja, respeitar as restrições impostas na sua criação, como o tipo de característica, as regras de composição, o grupo de características, e assim por diante. Uma vez definido o MC-D, pode-se então derivar configurações de características, selecionando as características para o produto derivado. Esta etapa de seleção de características resulta em um ou mais MCC-Ds, representando produtos estáticos (aqueles com apenas características estáticas) e/ou produtos dinâmicos (aqueles com características estáticas e dinâmicas).

De acordo com esta proposta de metamodelo, a adaptação dinâmica pode ser definida durante a modelagem da linha, por meio do MC-D, e/ou na fase derivação do produto, por meio do MCC-D.

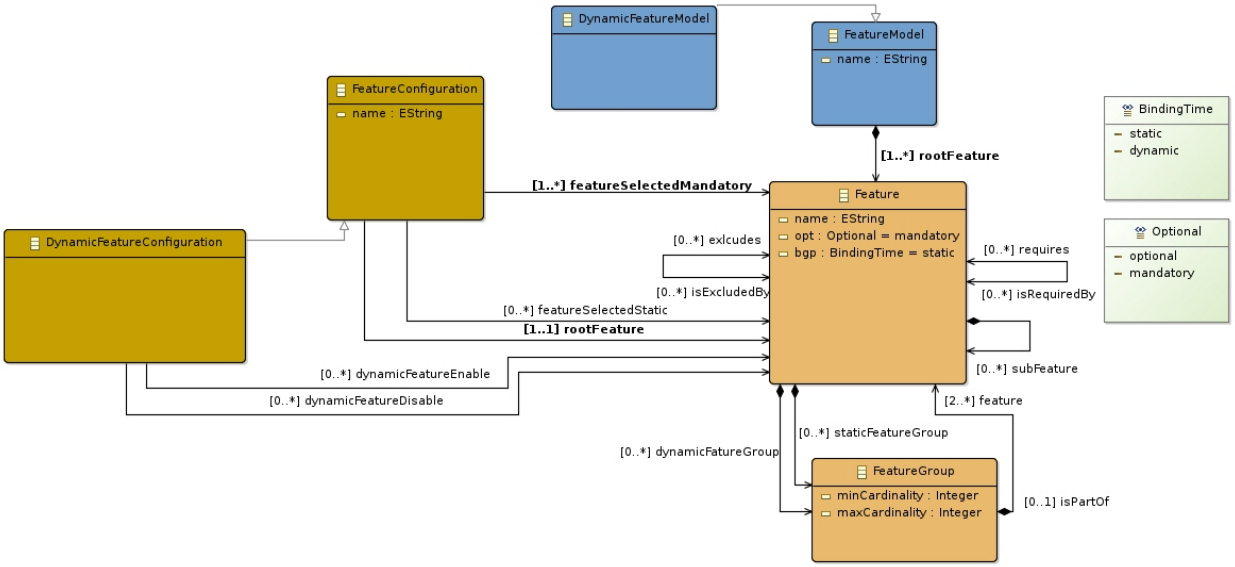


Figura 3.1: *Metamodelo para representação de características e configuração de características dinâmicas.*

Para uma melhor apresentação da proposta, o metamodelo está divido em dois metamodelos, de acordo com as Figuras (3.2 e 3.3), sendo o primeiro voltado para a modelagem do domínio (MC-D) e o segundo para a modelagem de aplicação (MCC-D). As próximas duas Subseções apresentam os metamodelos.

3.1.1 Representação de Características Dinâmicas

O metamodelo para representação de características dinâmicas (Figura 3.2) contém as seguintes metaclasses (EClass): `DynamicFeatureModel`, `FeatureModel`, `Feature` e `FeatureGroup`, além de dois tipos de enumeração (`Enums`), `Optional` e `BindingTime`.

O MC é representado pela superclasse `FeatureModel`, enquanto que o MC-D pela superclasse `DynamicFeatureModel`. Uma `DynamicFeatureModel` é uma `FeatureModel`, portanto, a LPSD contém todas as características da LPS tradicional. Durante a modelagem, deve-se escolher qual o ponto de início da modelagem. A princípio, pode-se escolher qualquer superclasse do metamodelo proposto, porém, o correto é modelar a partir da `FeatureModel` ou a partir de `DynamicFeatureModel`. Com isso, há clareza quanto ao tipo de linha se está modelando.

As superclasses `BindingTime` e `Optional` são do tipo EEnum. `BindingTime` determina se a característica é estática (*static*) ou dinâmica (*dy-*

namic), enquanto que `Optional` tem dois tipos para escolha, a opcional (*optional*) ou a mandatória (*mandatory*).

A superclasse `Feature` representa uma característica, seja no desenvolvimento de um MC-D, seja no desenvolvimento de um MCC-D. Ela contém os seguintes atributos (EAttributes): *name*, para o nome da característica, opt, do tipo `Optional`, e bgp do tipo `BindingTime`. Por exemplo, uma característica Mandatória o bgp deve ser definido como *static* e o opt como *mandatory*, enquanto que a característica Opcional o bgp deve ser definido como *static* e o opt como *optional*, e, por fim, a característica Dinâmica, o bgp deve ser definido como *dynamic* e o opt como *optional*. A `rootFeature` também é considerada como uma `Feature` convencional, porém, é mandatória e pode ter várias superclasses `Feature`. A `rootFeature` é representada em um papel e conforme o metamodelo, deve haver somente uma `rootFeature`. Uma `SubFeature` é uma `Feature`, tratando-se, portanto, de uma composição, permitindo que uma característica tenha outras características filhas.

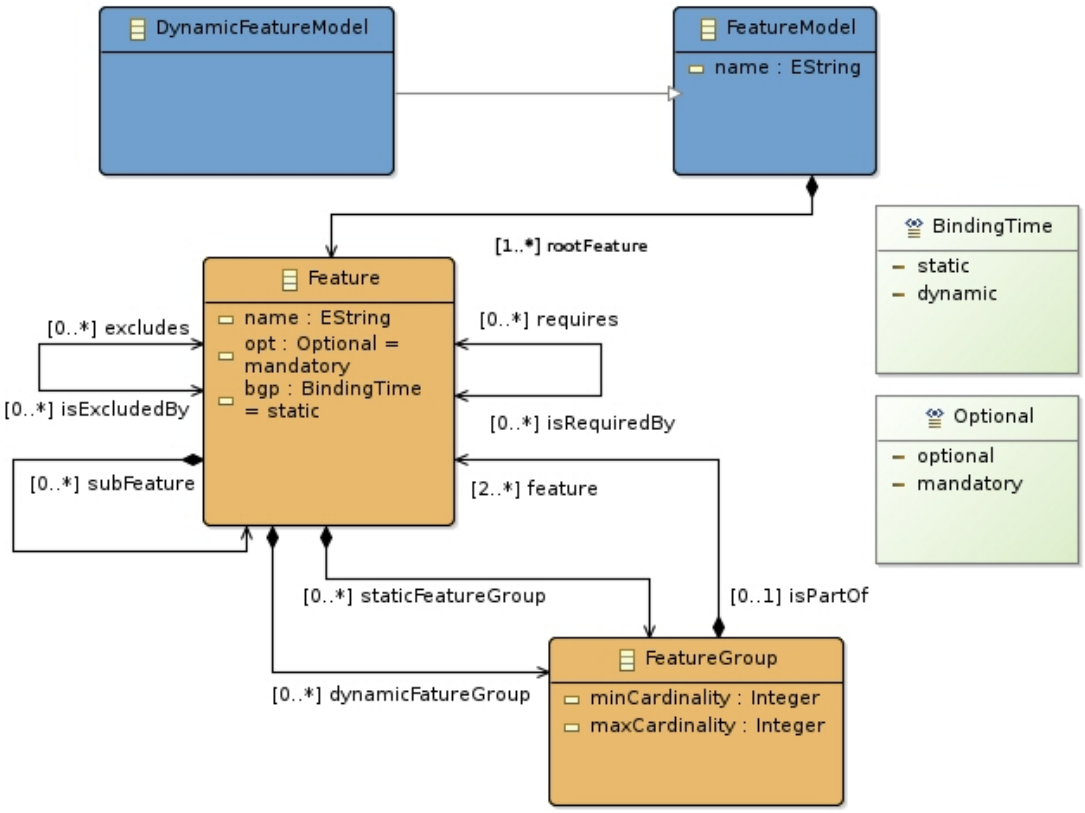


Figura 3.2: *Metamodelo representação de características dinâmicas.*

As regras de composição são definidas pelos papéis *"requires"* (corresponde ao *"requires"* do MC tradicional) e *"excludes"* (correspondente ao

"excludes" do MC convencional). Além destes papéis, há uma relação inversa, formando uma relação bidirecional, que se se refere às características pelas quais se está exigindo a sua seleção `isRequiredBy` ou desativação `isExcludedBy`. De acordo com estas regras, é possível que uma característica possa requerer ou excluir nenhuma ou várias características.

O grupo de características pode ser representado por um arco, semelhante do MC convencional (papel `staticFeatureGroup`), quando as características do grupo são todas estáticas. Quando as características do grupo são todas dinâmicas e a característica responsável pelo grupo não é dinâmica o arco deve ser duplo - veja na Subseção 3.4 - (papel `dynamicFeatureGroup`). O grupo de características é descrito na superclasse `FeatureGroup` e contém os atributos (`EAttributes`) `minCardinality` (quantidade de características obrigatórias para seleção) e `maxCardinality` (limite máximo de características possíveis de seleção) para definir a cardinalidade. Por exemplo, quando o grupo exige a escolha de apenas uma característica, o valor de `minCardinality` é 1 e o valor do `maxCardinality` é 1. Outro exemplo é a possibilidade de escolher várias características em um grupo de características. Nesse caso, o `EAttributes` de `minCardinality` deve ter valor 1 e o `maxCardinality` deve ser maior que 1. A `Feature` pode ter uma ou várias `FeatureGroups`, que é garantido pela composição `featureGroup` e de acordo com o papel `isPartOf`. O conjunto de características de `FeatureGroup` representa a `Feature` raiz, ou seja, é um tipo da `Feature` que ela pertence. Por sua vez, o papel `alternativeFeature` exige que o grupo tenha pelo menos duas características, pois não faz sentido um grupo de característica ter somente uma característica.

A capacidade da aplicação realizar adaptações, quando isso é caracterizado dentro da linha, é estabelecida por `BindingTime`, um tipo `Enumeration` do Ecore, com duas opções: *static* ou *dynamic*. Quando marcada como *static*, a característica é reconhecida como estática. Diferentemente, a opção *dynamic* define a característica como dinâmica, ou seja, determinando que ela pode ser ativada ou desativada em tempo de execução. A ativação e a desativação dinâmica podem ser realizadas pelo próprio usuário ou pelo sistema.

Este metamodelo é exclusivo para definir as diretrizes de modelagem de um MC ou de um MC-D. Na próxima Subseção, é abordado o metamodelo para a modelagem da aplicação (MCC-D).

3.1.2 Representação das Configurações de Características Dinâmicas

Esta proposta considera a possibilidade de se derivar produtos estáticos ou produtos dinâmicos, portanto, a aplicação pode ser totalmente estática ou híbrida (com características estáticas e/ou com características dinâmicas). A segunda parte do metamodelo que permite a criação de MCC-Ds (Figura 3.3) é representada pelas superclasses `FeatureConfiguration` e `DynamicFeatureConfiguration`.

A representação da característica opcional estática é semelhante à do modelo original (círculo com uma borda) - veja na Seção 3.4. No metamodelo, o atributo da superclasse `Feature` opt deve ser igual a *optional* e o atributo bgp deve ser igual a *static*. A EClass `Feature` representa características que compõem uma configuração de características em `FeatureConfiguration` (MCC convencional). Esta superclasse contém o atributo *name* para nomear o MC-D e os papéis `rootFeature`, `featureSelectedMandatory` e `featureSelectedOptional` que representam as seleções de características. A seleção destas características é durante a sua derivação estática, isto é, na implantação, e a aplicação não pode ativar ou desativar as características em tempo de execução. O `rootFeature` é a seleção da característica raiz da aplicação e segue a mesma regra do MCC-D, ou seja, é uma característica obrigatória para a seleção. Já em relação à `selectedMandatoryFeature`, é a seleção de característica mandatória que, por definição, deve selecionar todas as características mandatórias do MC-D. E por fim, o papel `selectedOptionalFeature` refere-se à seleção de características opcionais, isto é, não obrigatórias.

Por outro lado, uma aplicação com características dinâmicas (MCC-D) é representada pela superclasse `DynamicFeatureConfiguration`. Esta superclasse herda de `FeatureConfiguration` e, portanto, garante que o produto seja estático, ou dinâmico ou híbrido (estático e dinâmico). O MCC-D pode ter nenhuma ou várias características dinâmicas ativadas (papel *dynamicFeatureEnable*) e também pode ter nenhuma ou várias características dinâmicas desativadas (papel *dynamicFeatureDisable*).

A superclasse `Feature` é semelhante à do metamodelo anterior (Figura reffig:domainmodel), isto é, contém os `EAttributes` `opt` (`Optional`) e o `bgp` (`BindingTime`) para representarem características estáticas e dinâmicas de acordo com as enumerações `BindingTime` e `Optional`. Não há necessidade de informar as regras de composição `requires` e `excludes`, pois estas regras são para a modelagem de MCs ou MC-Ds.

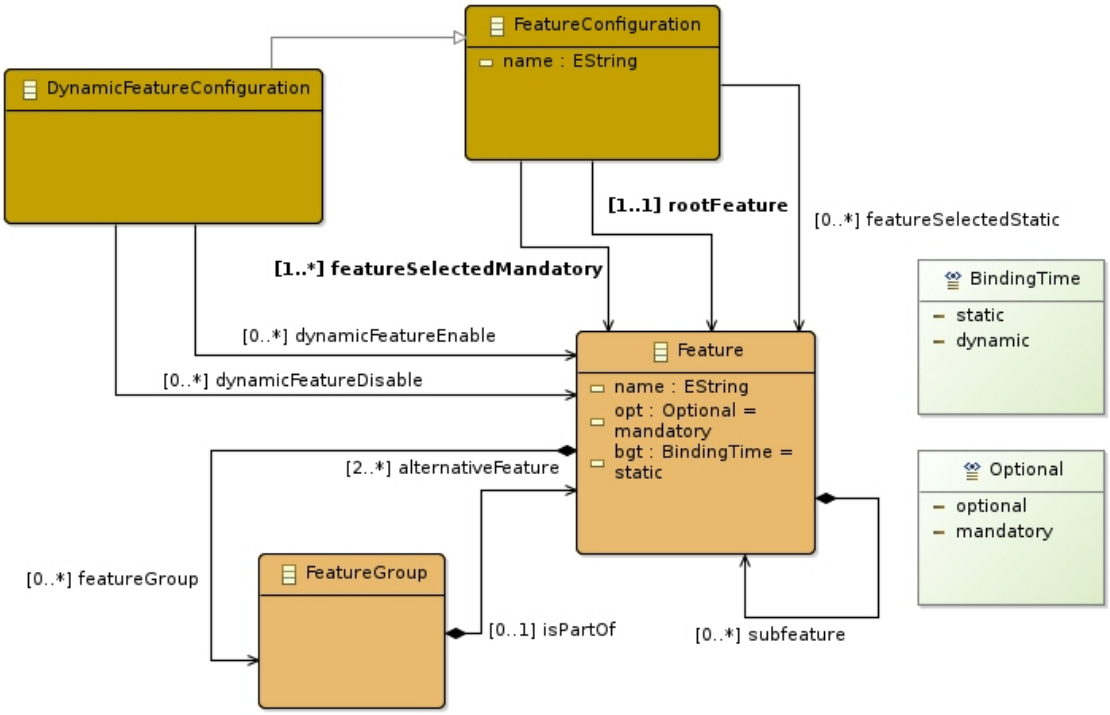


Figura 3.3: *Metamodelo para Representação de Características Dinâmicas*

Outra superclasse semelhante, porém com algumas alterações, é a `FeatureGroup`. `Feature` pode ter grupos de características (papel `FeatureGroup`) conforme o metamodelo anterior (Figura reffig:domainmodel). O grupo deve ter no mínimo duas características (papel `alternativeFeature`) e cada característica deve pertencer exclusivamente ao grupo (papel `isPartOf`). A cardinalidade está ausente e para acrescentar ou remover uma característica no grupo, deve-se seguir as regras do MC ou MC-D correspondente. A representação de características em MC e em MCC convencionais foram estendidas para a representação de MC-D e MCC-D, conforme a proposta desta dissertação.

3.2 Modelo de Características Dinâmicas

A partir do Metamodelo de Características Dinâmicas (Figura 3.2) pode-se instanciar um MC ou um MC-D. A Figura 3.4 apresenta um MC-D, já estendido com um novo símbolo, um círculo de borda dupla, para representar uma característica dinâmica. Esta característica pode ser ativada ou desativada em tempo de execução.

Em relação ao metamodelo proposto, apresentado na Figura 3.2, as

características estáticas opcionais e as mandatórias são semelhantes às do MC convencional representado pela superclasse `FeatureModel`. Por exemplo, a característica estática deve ter o atributo `opt` com valor *optional* e o `bgp` com valor *static*; e a característica mandatória deve ter o atributo `opt` com valor *optional* e o `bgp` com valor *mandatory*. Já a característica dinâmica pertence ao MC-D representado pela superclasse `DynamicFeatureModel`. Este tipo de característica deve ter o atributo `opt` com valor *optional* e o atributo `bgp` de valor *dynamic*; a característica mandatória deve ter o atributo `opt` com valor *optional* e o `bgp` com valor *dynamic*.

O MC-D da Figura 3.4 é um exemplo de modelagem de uma aplicação no contexto do cenário de *homecare*. Este MC-D contém um Plano de Cuidado (*Care Plan*) e por ser essencial para a *homecare* é considerado como uma característica mandatória. A característica *Reminder* é opcional, isto é, selecionada somente em tempo de implantação. Esta característica, quando selecionada, permite ao usuário receber recados, o que também exige a seleção da característica *Display Device*, por conta da regra de composição do metamodelo proposto.

A característica *Display Device* é dinâmica e deve ter um grupo de características também dinâmicas. A cardinalidade deste grupo exige a escolha de no mínimo uma característica e no máximo quatro características (TV, *tablet*, *smart phone* e *ordinary display*). O grupo de características deve ser homogêneo, isto é, todas as características devem ter o mesmo valor para os atributos `opt` e `bgp`, presentes na superclasse `Feature` (veja o metamodelo da Figura 3.1).

Caso haja um grupo cujas características sejam dinâmicas e a característica deste grupo não seja dinâmica, o arco é representado como um arco duplo. As características do grupo permanecem semelhante às da forma convencional de MCs de LPSs. A Figura 3.4 ilustra esta situação. A característica *Communication* é essencial para uma *homecare*, por isso ela está como mandatória. Se um meio de comunicação falhar, outra característica de maior prioridade deve ser ativada em tempo de execução. A prioridade de escolha das características do grupo pode ser formalizada em regras preestabelecidas em tempo de execução ou, ainda, na implantação da aplicação.

Se uma característica não for dinâmica e fizer parte de um grupo que é dinâmico, significa que ela não poderá ser desativada, por ser estática. Por outro lado, se a característica é dinâmica ela pode ser desativada e, neste caso, todas as características filhas serão desativadas também. Enfim, caso uma característica raiz seja dinâmica, as demais características devem ser

dinâmicas.

A ativação e desativação de uma característica dinâmica se dá por meio do MC-D, alterando o MCC-D, pois este representa o produto configurado. A modelagem do MCC-D é detalhado na próxima Seção.

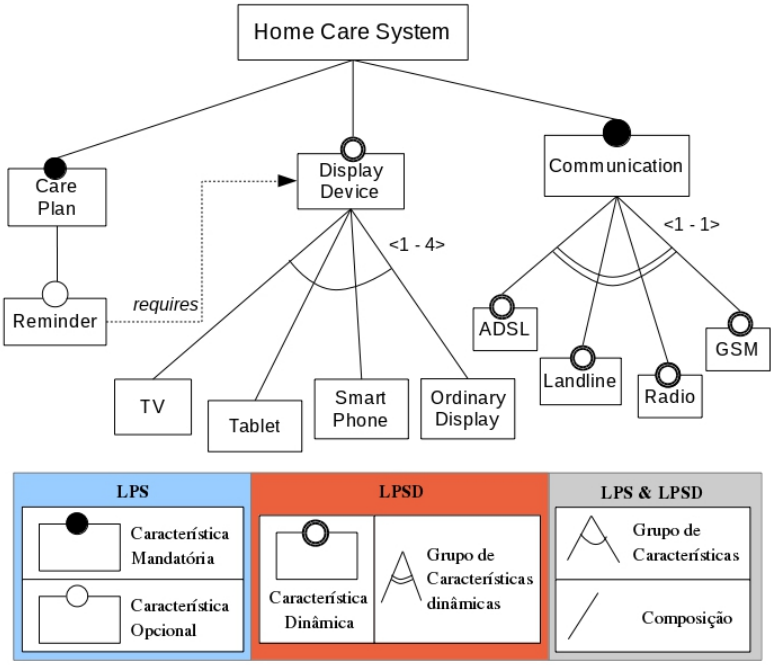


Figura 3.4: *MC Dinâmico com um grupo de característica dinâmicos.*

3.3 Modelo de Configuração de Características Dinâmicas

O MCC-D para LPSD corresponde à configuração do produto por meio de seleções de características, conforme o metamodelo ilustrado na Figura 3.3. As seleções podem ocorrer na implantação e também durante a execução da aplicação. A representação gráfica da característica dinâmica é retratada por um círculo de dupla borda (Figura 3.5). Na Figura 3.6 ilustra um exemplo de seleção de características de um MC-D para gerar um novo MCC-D.

Quando a aplicação está em execução, a seleção é feita através da própria interação do usuário (entrada explícita) ou, quando necessário, pela aplicação (entrada inferida ou percebida).

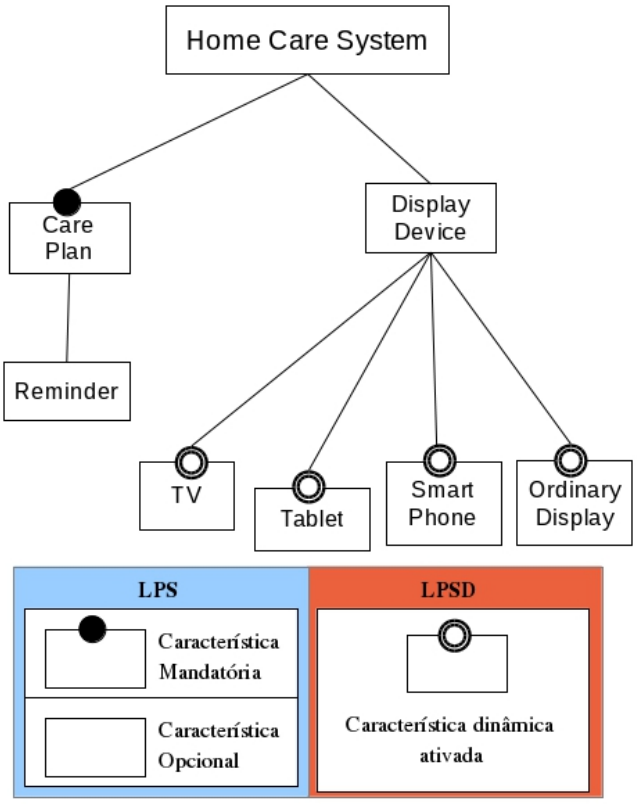


Figura 3.5: *Exemplo de um Modelo de Configuração de Características Dinâmicas (MCC-D)*

Esta dissertação considera duas formas de representar o MCC-D: a primeira como uma cópia do MC-D, e a segunda como a representação apenas das características selecionadas. Um MCC-D como uma cópia de todo o MC-D pode ser vantajoso para que o usuário conheça outras características que podem ser selecionadas, porém, pode tornar o MCC-D complexo, pois se trata de uma réplica exata da linha. A Figura 3.7 ilustra esse caso. Já a segunda abordagem pode ser útil no sentido de simplificar e informar o usuário apenas as características que estão ativas no produto, veja na Figura 3.8.

Em ambas as Figuras (3.7 e 3.8), o usuário ou sistema escolheram a TV e o *smart phone* para receber as notificações. Quando o usuário tem a permissão de selecionar uma característica no MCC-D, a aplicação deve respeitar a escolha do usuário, também conhecida como entrada explícita. Portanto, a preferência do usuário é de maior relevância em relação ao sistema. Salvo em situações críticas de funcionalidade do sistema, o usuário não possui a permissão de mudar as escolhas realizadas pelo sistema.

O MCC-D não somente representa as características selecionadas, mas também serve como um painel de componentes ativados para a aplicação.

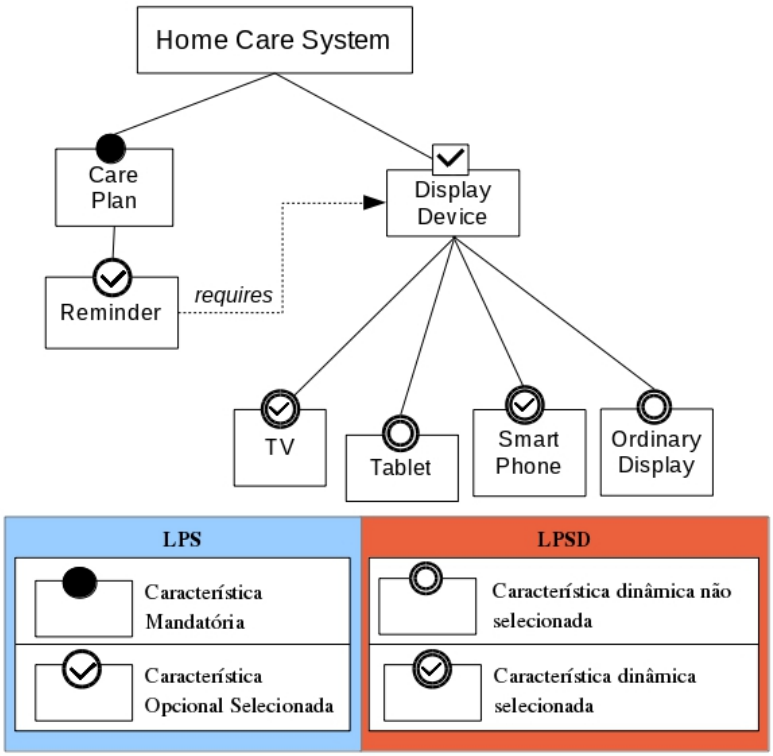


Figura 3.6: *Seleção de características de um MC-D.*

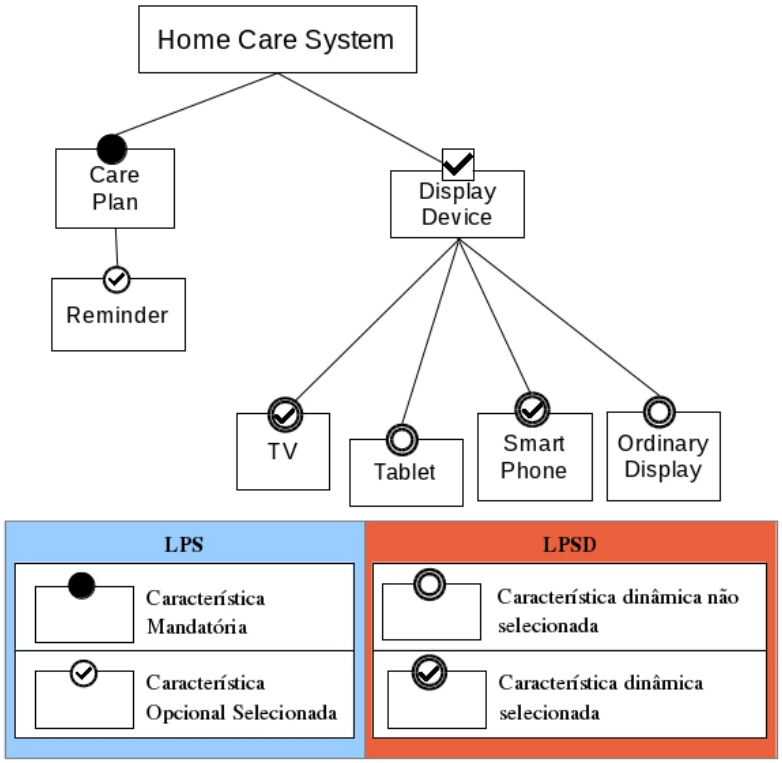


Figura 3.7: *Exemplo de cópia de um MC-D para um MCC-D.*

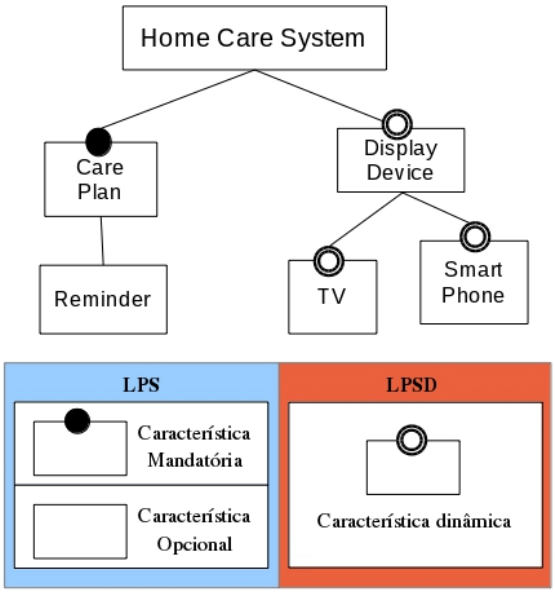


Figura 3.8: *Exemplo de um Modelo de Configuração de Características Dinâmicas*

Na próxima Seção, é abordado um comparativo entre MC-D e MCC-D para grupos de características dinâmicas.

3.4 Representação de Grupo de Características Dinâmicas ou Estáticas

Nesta Seção são detalhadas as possibilidades de se representar grupos de características dinâmicas e/ou estáticas. A Figura 3.9 exemplifica as possibilidades de grupos de características em MC-D e MCC-D. As representações de grupos de características para MC e MCC convencionais estão preservadas nesta proposta, por outro lado, se o grupo possui características dinâmicas, prevalece a proposta de representação desta dissertação.

De forma geral, as características dinâmicas não têm alteradas suas representações no MC-D e no MCC-D. Por outro lado, conforme apresentado na Seção 3.1.1, quando o grupo possui características dinâmicas e a característica responsável pelo grupo não é dinâmica, representa-se o grupo no MC-D como um arco duplo. Todas as características estáticas ou dinâmicas pertencentes ao grupo são representadas de forma semelhante à proposta original de MC.

A próxima Seção apresenta um exemplo de derivação de produtos de uma LPSD.

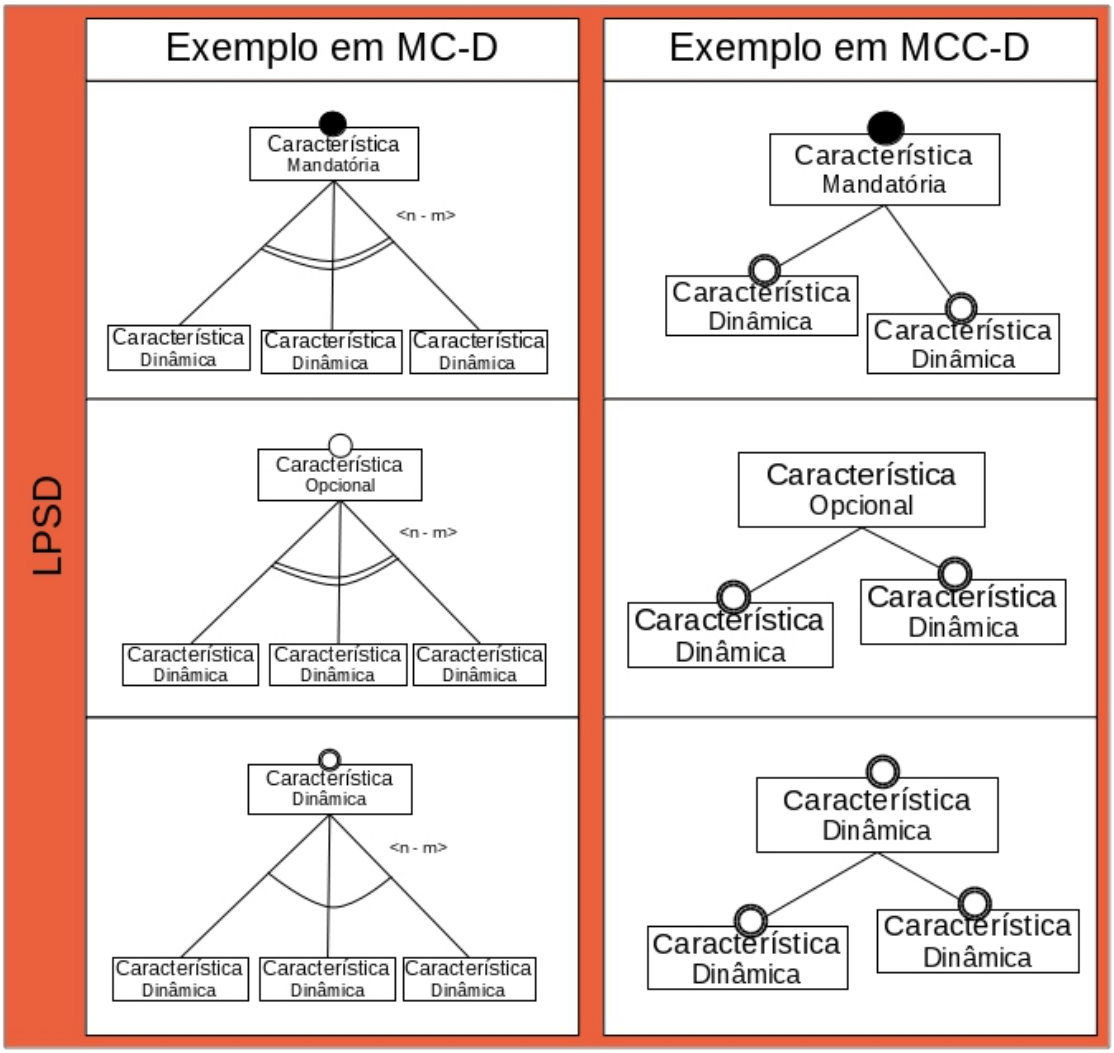


Figura 3.9: *Proposta de representação de grupo de características dinâmicas*

3.5 Derivação de Produtos Estáticos e Dinâmicos

De acordo com o cenário de *homecare* apresentado nesta dissertação, o paciente - ou outro usuário - escolhe as características desejadas a partir do MC-D da Figura 3.10. Este modelo contém as seguintes características estáticas: Plano de Cuidados (*Care Plan*), mandatória; Visor (*Display Device*), opcional; Comunicação (*Communication*), mandatória. As características dinâmicas são: *Display Device* composto por um grupo (TV, *tablet*, *smart phone* e *ordinary display*) e *Communication* (ADSL, *Land line*, *Radio* e o GSM). Por fim, há uma regra de composição *requires* em *Reminder* exigindo a seleção da característica *Display Device*. A partir deste MC-D, são gerados três exemplos

de derivações de produtos, considerando cada derivação como uma evolução do produto anterior. Os exemplos de derivação são: o primeiro com a configuração mínima, o segundo com suporte a recados e o terceiro com exemplos de mudanças dinâmicas nos dois grupos de características.

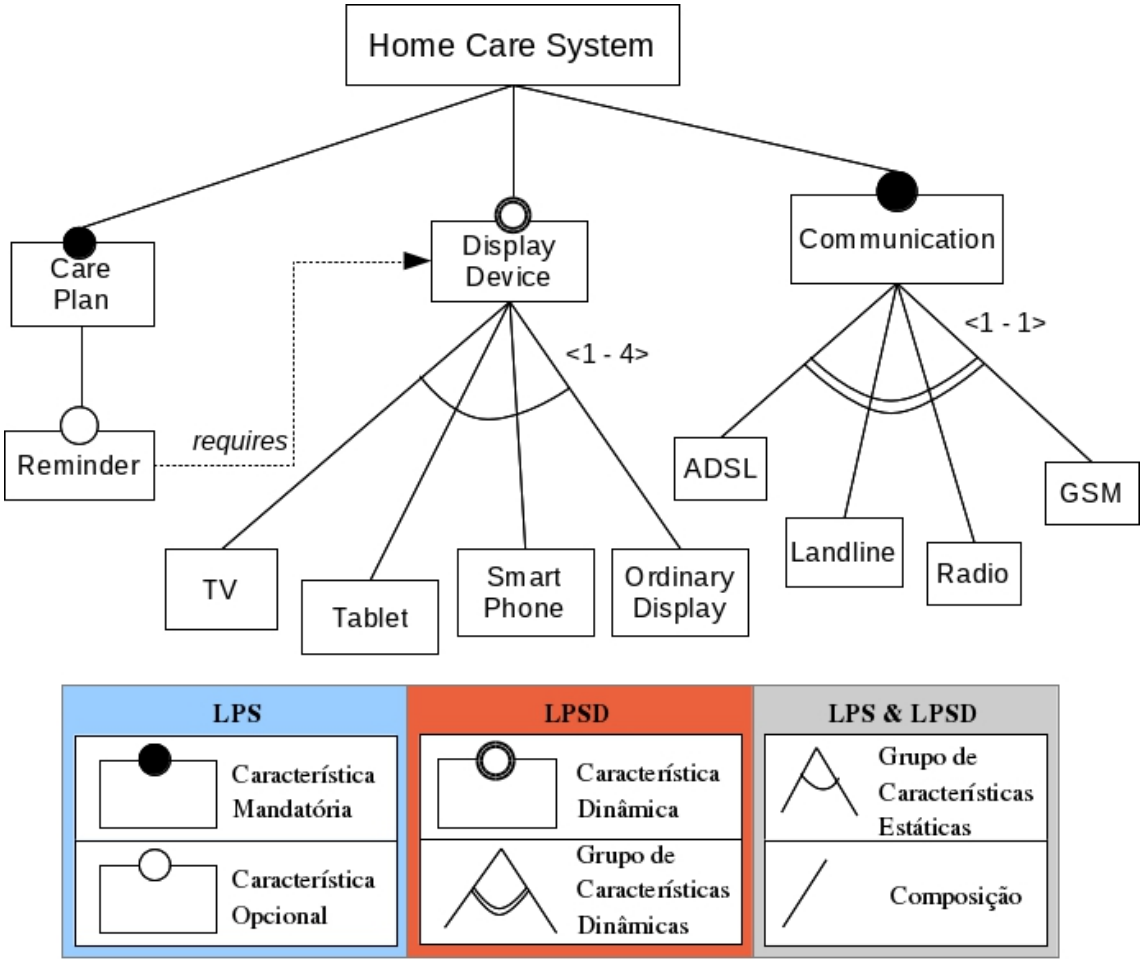


Figura 3.10: *Exemplo de um MC-D para o cenário de homecare*

3.5.1 Produto com Configuração Mínima

O paciente decide optar por uma configuração mínima, isto é, contendo apenas as características mandatórias. Ele tem somente o *Care Plan* e a *Communication*, além de `rootFeature` (*Home Care System*). A característica responsável pela comunicação contém um grupo de características e exige a escolha de uma característica. Esta escolha se dá pela melhor forma de comunicação de acordo com o local e custo do serviço de Internet. A Figura 3.11 apresenta o MCC-D do produto derivado.

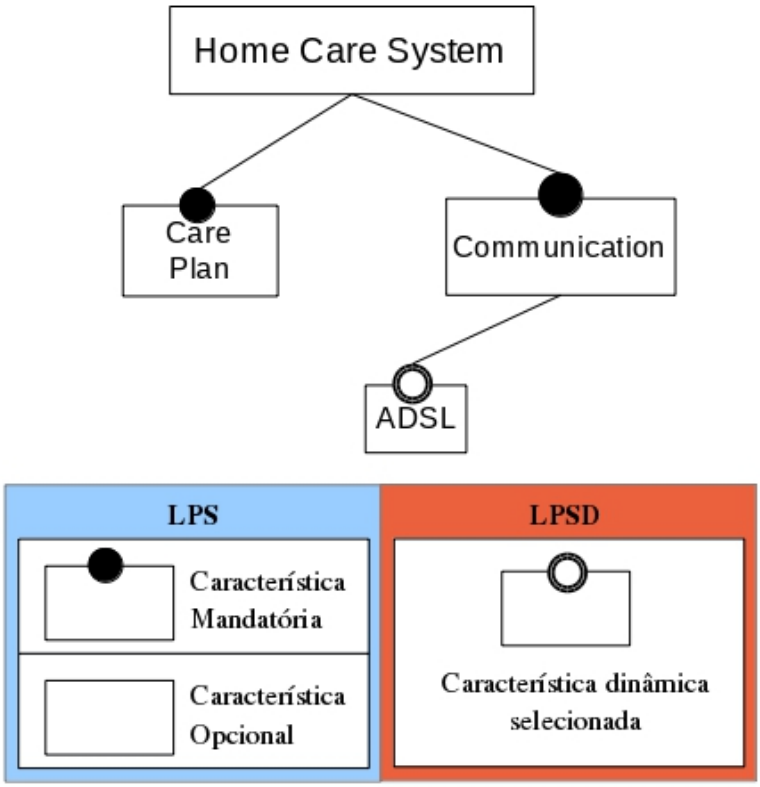


Figura 3.11: *Primeiro exemplo de derivação de produto para o cenário de homecare*

3.5.2 Produto com Suporte para as Características Reminder, TV e Smart Phone

Outra opção de derivação do MC-D está na Figura 3.12. Devido às condições de saúde do paciente, ele pode se esquecer de seguir as prescrições médicas, como, por exemplo, tomar remédios em um determinado horário. A característica estática e opcional *Reminder* auxilia o paciente a seguir a prescrição médica por meio de notificações. Porém, no MC-D, a característica *Reminder* requer a escolha da característica dinâmica *Display Device*, que por sua vez, contém um grupo de características que exige a escolha de, pelo menos, uma característica dinâmica.

O paciente decide escolher (ativar) as características dinâmicas TV e *smart phone*. A escolha da característica dinâmica *smart phone* é por questões de conforto, podendo ser desativada ou ativada novamente a qualquer instante de acordo com a necessidade do paciente ou da própria aplicação. Já a característica *Communication* permanece a mesma (ADSL) do primeiro exemplo de produto ilustrado na Figura 3.11.

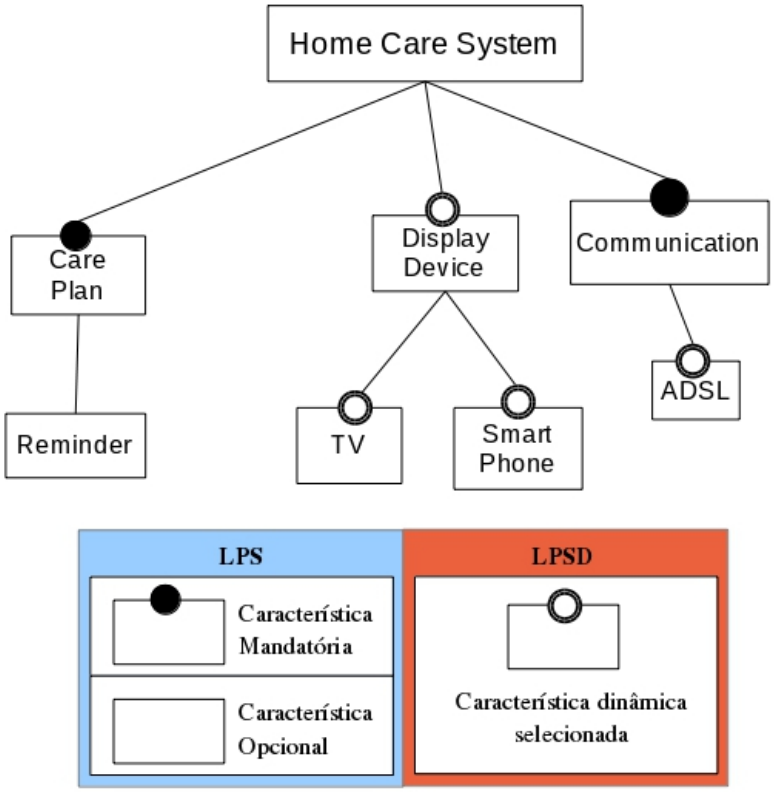


Figura 3.12: *Exemplo MCC Dinâmico com suporte ao Reminder*

3.5.3 Produto com Suporte a Recados e com Mudanças Dinâmicas em Grupos de Características

O terceiro exemplo de derivação está na Figura 3.13. Com o decorrer da utilização da aplicação, o paciente decide contratar suporte para o *tablet*, apenas selecionando no MC-D a característica dinâmica *tablet* em tempo de execução. Após a seleção o MCC-D deve representar a nova característica como marcada.

Outra situação é a falha da ADSL. A aplicação percebe esta falha e seleciona a característica GSM, sendo a segunda alternativa de maior prioridade do grupo *Communication* (Figura 3.13). Conforme regras de prioridades para a seleção de características do grupo, a ADSL tem maior prioridade em relação às outras características. Caso este meio de comunicação se normalize, a característica ADSL deve ser ativada novamente pela aplicação.

Esta proposta de MCC-D possibilita a visualização da configuração da aplicação e auxilia no gerenciamento de suas funcionalidades/requisitos. O MCC-D auxilia também na visualização de mudanças de estados das características dinâmicas decorrentes de suas adaptações em tempo de execução.

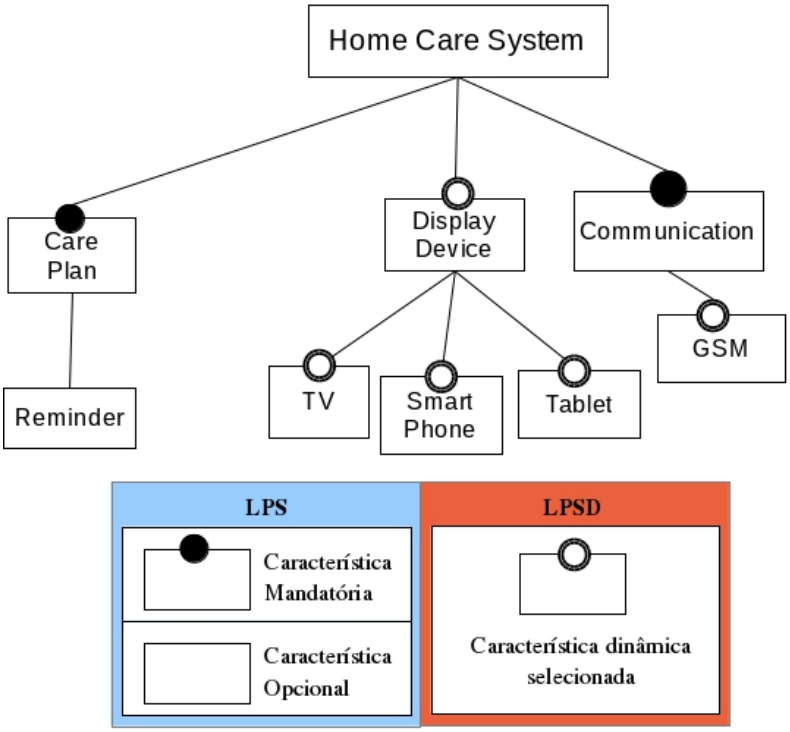


Figura 3.13: *Terceiro exemplo de derivação de produto para o cenário de homecare.*

3.6 Trabalhos Relacionados

Nesta Seção, são apresentados alguns trabalhos relacionados à proposta de representação de características dinâmicas e a formulação em metamodelos. Por fim, é feito um comparativo geral das propostas apresentadas em relação à proposta desta dissertação.

Capilla *et al.* [11] propõem representar o Modelo de Características para uma LPSD voltada para computação ubíqua e autoadaptação, e para o gerenciamento de contexto. O autor não deixa explícito como é o Modelo de Configuração de Características. Além disso, ele apenas propõem alteração no Modelo de Características e não oferece um metamodelo para definir a proposta. A proposta desta dissertação, no cenário de uma aplicação ubíqua, pode auxiliar no gerenciamento de contexto.

Em Carvalho [12], é apresentada a proposta de um metamodelo de configuração de variabilidades para uma LPS voltada para aplicações ubíquas. O Modelo de Configuração de Características é denominado por *Configuration Knowledge* (CK). O CK associa as características e os elementos arquiteturais da aplicação por meio de Contratos para descrever, por exemplo, variabilidades estáticas e dinâmicas, e para associar as características aos elementos

da arquitetura. A adaptação da aplicação é possível por meio de Contratos, definidos por serviços que realizam as operações de adaptação por regras definidas de acordo com o contexto e por negociação, que descrevem a ordem de execução de cada serviço. O autor, no entanto, não propõe uma representação de Modelo de Características e do Modelo de Configuração de Características para uma LPSD. A proposta desta dissertação complementa o trabalho de Carvalho [12] no sentido de suprir a ausência de representação para as características dinâmicas.

Já no trabalho de Baresi *et al.* [6], é uma proposta de uma arquitetura para diferenciar a evolução dinâmica da sincronização para uma LPSD. Os autores estendem o Modelo de Características com cardinalidade e atributos para definir, se necessário, maiores detalhes das características. Em relação ao Modelo de Configuração de Características, o autor utiliza o mesmo modelo de configuração de uma LPS. O Modelo de Características proposto pode ser muito complexo em alguns casos, dependendo da característica, e também não oferece um metamodelo para a proposta.

O trabalho de Fernandes *et al.* [20] e [21] não oferece um metamodelo para o Modelo de Características, produzindo apenas um Modelo de Características voltado para se representar aplicações sensíveis ao contexto. A modelagem é baseada em uma notação de características de contexto denominada UbiFEX, própria para as regras de adaptação e para a representação de informações de contexto no Modelo de Características. As representações dos modelos de característica e de configuração diferem consideravelmente do tradicionalmente empregado em LPS. A ferramenta UbiFEX contém um mecanismo UbiFEX-Simulation para verificar regras de composição (inclusivas e exclusivas), cardinalidade e a seleção de características opcionais na LPS. Apesar de a pesquisa abordar somente LPS, uma LPSD pode garantir a execução de determinadas características essenciais que poderiam mudar de estado em tempo de execução, como, por exemplo, a comunicação com a Internet no cenário de uma *homecare* e de aplicações ubíquas.

Em Trinidad *et al.* [41], os autores propõem um Modelo de Características para modelar estados de um produto. O Modelo de Características mapeia a arquitetura e por meio do componente chamado *Configurator* realiza as mudanças em tempo de execução do produto. Não há uma proposta para representar o Modelo de Características e o Modelo de Configuração de Características dinâmicas. As propostas desta dissertação complementa a pesquisa de Trinidad *et al.* [41], no sentido de oferecer essa representação.

Almeida *et al.* (2014) [3] oferecem uma proposta de adaptação dinâ-

mica baseada em Mape-K voltada para aplicações em *Cloud Computing*. Os autores trabalham no conceito de LPS convencional e envolvem duas fases: modelagem de variabilidades de serviços *cloud* e configuração do make-K. Apesar do foco do artigo ser a prática na construção de uma LPS com base no *framework* de componente FraSCAti, esta dissertação complementa a representação de características do cenário de adaptação de *cloud* proposto pelos autores. A proposta dos autores não traz a possibilidade de se diferenciar características estáticas de dinâmicas.

No trabalho de Cetina *et al.* [15] a representação de MC e MCC estão em um mesmo modelo, semelhante à proposta de [28]. Os autores têm o foco de avaliar a proposta a partir de um protótipo para um *Smart Hotel* com dispositivos reais e a participação de usuários. O trabalha cita falhas no sistema devido à falta de experiência dos usuários, quando o mesmo tem controle no sistema e propõe como trabalho futuro um treinamento para os usuários se capacitarem na aplicação e avaliarem os efeitos das reconfigurações. Como os participantes não compreendem certos efeitos que a aplicação pode ter, a representação de características dinâmicas e estáticas, proposta desta dissertação, poderia auxiliar estes participantes na compreensão de quais características estão selecionadas em tempo de execução. A proposta de ferramenta desta dissertação oferece também a possibilidade de simular mudanças de estados no MCC, servindo para orientar os usuários sobre as possibilidades de cenários que podem ocorrer na aplicação.

Em Murguzur *et al.* [31], os autores afirmam que não há uma formalização para modelar variabilidades no contexto de LPSDs. Os autores propõem uma modelagem, por meio de um metamodelo, de contexto de variabilidade e utilizam como caso de uso um parque eólico. A proposta tem suporte para diferentes variabilidades de contextos e modelagem de domínios, incluindo um *framework* (em construção) que autoconfigura serviços baseados em LPSD e em contexto. A proposta de representação de MC ou MCC pode auxiliar na modelagem de variabilidade de contexto seja por interferência humana, pela aplicação ou por uma base dados presente no *framework* proposto pelos autores. O metamodelo é para o processo de variabilidade voltado para o contexto, ou seja, não há regras claras para a modelagem de MC pelo *framework*. O processamento de dados de contexto é feito por ontologias e os dados do modelo de contexto de ontologia são armazenados em um banco de dados e representados em JSON *Schema*, semelhante à abordagem desta dissertação. Por fim, as mudanças de contexto também podem ser representadas na simulação presente na ferramenta proposta desta dissertação para auxiliar nas mudanças

de estados das características.

A maior parte dos autores não utiliza metamodelos para formalizar suas propostas de modelagem de características e de configuração do produto. Há poucas propostas de representação de Modelo de Características e de Configuração para Características para LPSD, por conta da preocupação com a sincronização, principalmente, em relação às características e à arquitetura do produto de software.

A Tabela 4.1 contém uma lista dos autores discutidos nesta Seção. As colunas foram abreviadas da seguinte forma:

- **C1**: Proposta de metamodelo para a representação de características para modelos de LPS ou LPSD.
- **C2**: Representação de características dinâmicas para o MC.
- **C3**: Representação de características dinâmicas para o MCC.
- **C4**: Proposta para LPSD.

Tabela 3.1: *Comparação de propostas de trabalhos relacionados*

Autor	**C1**	**C2**	**C3**	**C4**
Capilla *et al.* (2014) [11]		**X**		**X**
Carvalho (2013) [12]	**X**			**X**
Baresi *et al.* (2015) [6]		**X**		**X**
Fernandes *et al.* (2011) [21]				**X**
Trinidad *et al.* (2007) [41]				**X**
Almeida *et al.* (2014) [3]				
Cetina *et al.* (2013) [15]				**X**
Murguzur *et al.* (2014) [31]	**X**			**X**
Proposta desta dissertação	**X**	**X**	**X**	**X**

Conforme a Tabela 4.1, a maior parte dos autores não utilizam uma representação para características dinâmicas e uma formulação de modelos por meio de metamodelos. A proposta desta dissertação colabora no sentido de suprir estas ausências de representações dinâmicas e de metamodelos. As representações de MC-D e MCC-D simplificam a compreensão para usuários e auxiliam na percepção das mudanças em tempo de execução, em especial para aplicações ubíquas, como, por exemplo, para o cenário de *homecare*.

3.7 Considerações Finais

As propostas de representação de características dinâmicas em modelos MC-D e MCC-D permitem suprir a lacuna de representação de caracte-

rísticas dinâmicas para uma LPSD. Os metamodelos servem como diretrizes para a compreensão e para a construção destes modelos. Foi necessário diferenciar as características estáticas das características dinâmicas por meio de símbolos simples de se compreender. Os grupos de características devem ter características uniformes, representando características estáticas e dinâmicas conforme o arco ou conforme a característica a que o grupo pertence.

Na Seção 3.5, foram apresentados exemplos de produtos estáticos (características estáticas) e híbridos (características dinâmicas e estáticas) por meio de MCC-Ds. Apresentou-se também a abordagem correta para a seleção de características estáticas mandatórias, estáticas opcionais e dinâmicas opcionais.

O próximo capítulo apresenta uma ferramenta para a modelagem de MC-D, MCC-D, e ainda para a simulação de mudanças de estados de características em tempo de execução e em tempo de implantação.

CAPÍTULO 4

Ferramenta e Cenários de Uso

Neste capítulo é apresenta a ferramenta de modelagem de MC-Ds e MCC-Ds, a partir do metamodelo proposto. Esta ferramenta permite também a simulação da derivação de produtos modelados usando o MCC-D. Além da ferramenta, o capítulo apresenta exemplos com cenários envolvendo aplicação de *homecare*.

4.1 Arquitetura da Ferramenta

A ferramenta desenvolvida nessa dissertação permite a modelagem de MC-D, MCC-D e também a simulação de cenários de uma LPSD. A ferramenta é baseada na *web* e foi desenvolvida em PHP com suporte de conexões Ajax. A Figura 4.1 apresenta a arquitetura da ferramenta, a qual emprega em sua estruturação o padrão arquitetural *Model View Controller* (MVC) [22].

O Controller controla toda a ferramenta e é composto pelos módulos EcoreController, FCInterpreter e FValidator. Ele realiza a leitura de qualquer metamodelo desenvolvido em formato Ecore. O EcoreController faz persistências em um banco de dados, representado na Figura 4.1 por Models (M1), de instâncias de modelos, como, por exemplo, MC-Ds e MCC-Ds. O Apêndice D contém o MER (Modelo Entidade-Relacionamento) para a persistência de modelos realizado pelo módulo EcoreController. Ele também interpreta metamodelos (Metamodels Ecore (M2) na Figura 4.1) e permite a leitura de *scripts* para o módulo FCInterpreter. O FCInterpreter, por sua vez, interpreta uma linguagem específica desenvolvida para adicionar, remover, desabilitar ou habilitar características de um MC-D. Por fim, o FValidator valida os modelos MC-D e MCC-D.

A camada Model, por sua vez, faz a representação de troca de informações entre o Controller e a camada View.

Por fim, a camada View contém o ModelPanel (painel para a modelagem) e o Simulator (Simulador). Além disso, o ModelPanel é composto pelos

painéis de modelagens PanelFM (para modelagem de MC-D) e o PanelFC (para modelagem de MCC-D).

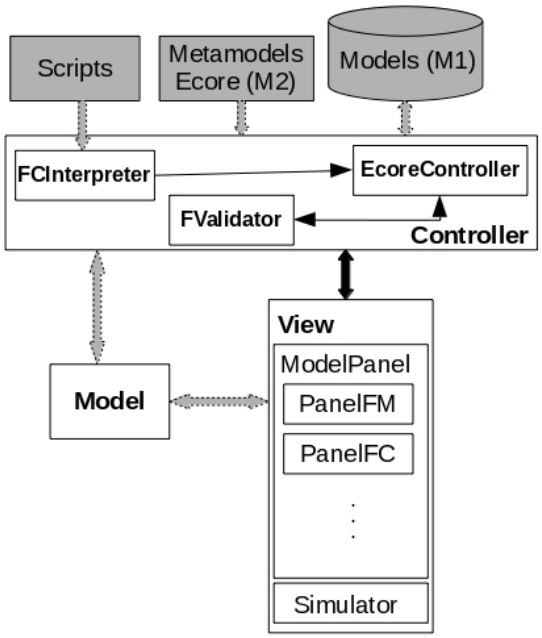


Figura 4.1: *Arquitetura da Ferramenta*

Nas próximas subseções detalham os módulos da ferramenta.

4.1.1 EcoreController

A criação de modelos exige a seleção de um metamodelo construído a partir do meta-metamodelo Ecore. O processo de criação de modelos está presente no diagrama da Figura 4.2.

Durante a criação de um novo modelo, o EcoreController exige o ponto de início da modelagem, neste caso. O ponto de início é denominado por *StartPoint* (Figura 4.2). É possível criar um modelo a partir de outro modelo. Por exemplo, para se criar um MCC-D é necessário definir antes um MC-D.

No término da criação do modelo, o EcoreController persiste as configurações em um banco de dados. Após a criação do modelo é possível seguir o processo de modelagem, descrito na Figura 4.3. A camada View requisita ao EcoreController que informa a EClass que pode ser criada no estado em que se encontra a modelagem. Se for o primeiro `EClassifier`, o *StartPoint* será o `EClassifier` inicial para a modelagem. Caso haja outros `EClassifier` presentes no banco de dados, são processadas pelo EcoreController. Cada processamento de `EClassifier` deve verificar se o mesmo herda de outro `EClassifier`, no metamodelo, por meio do atributo `eSuperType`.

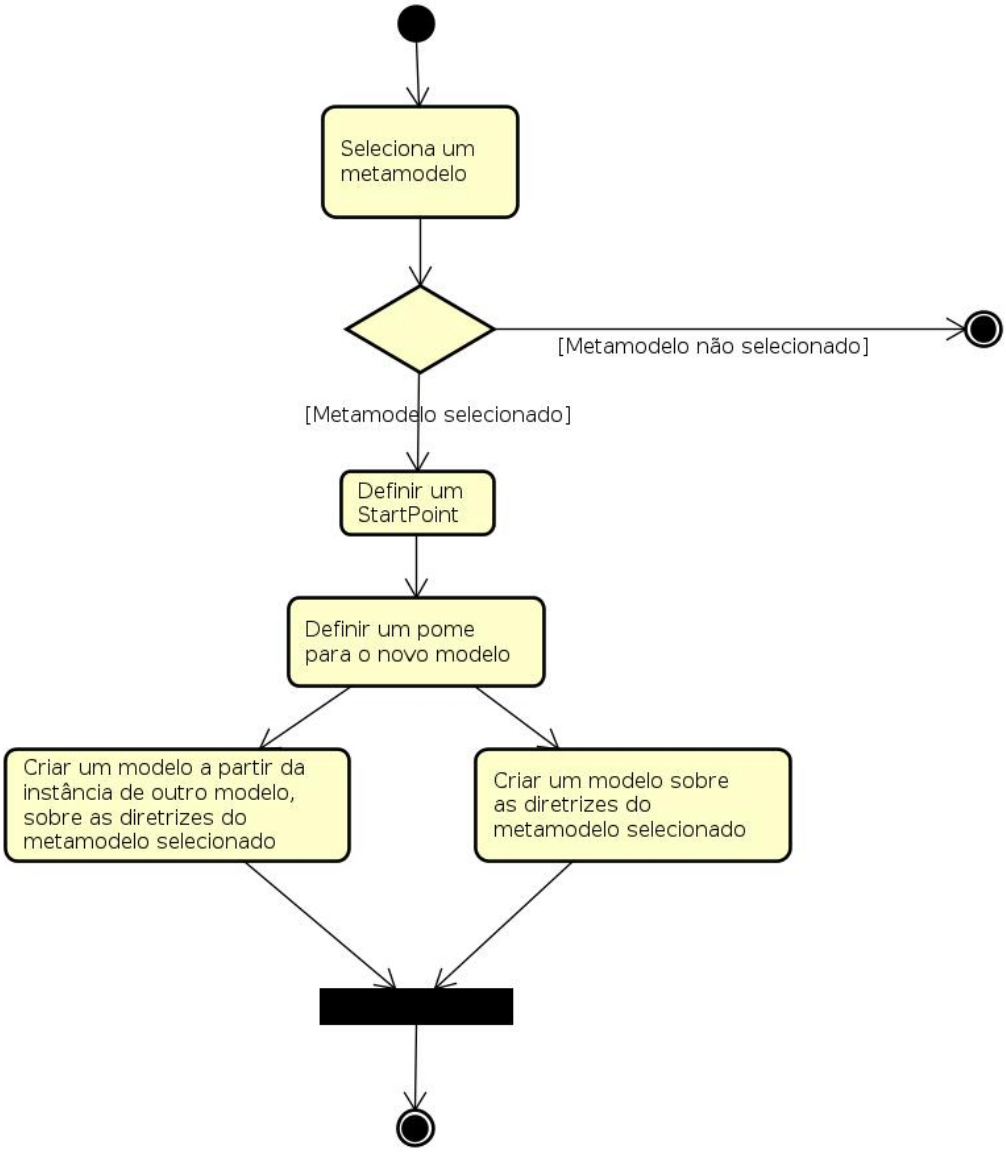


Figura 4.2: *Processo de criação de modelos usando a Ferramenta.*

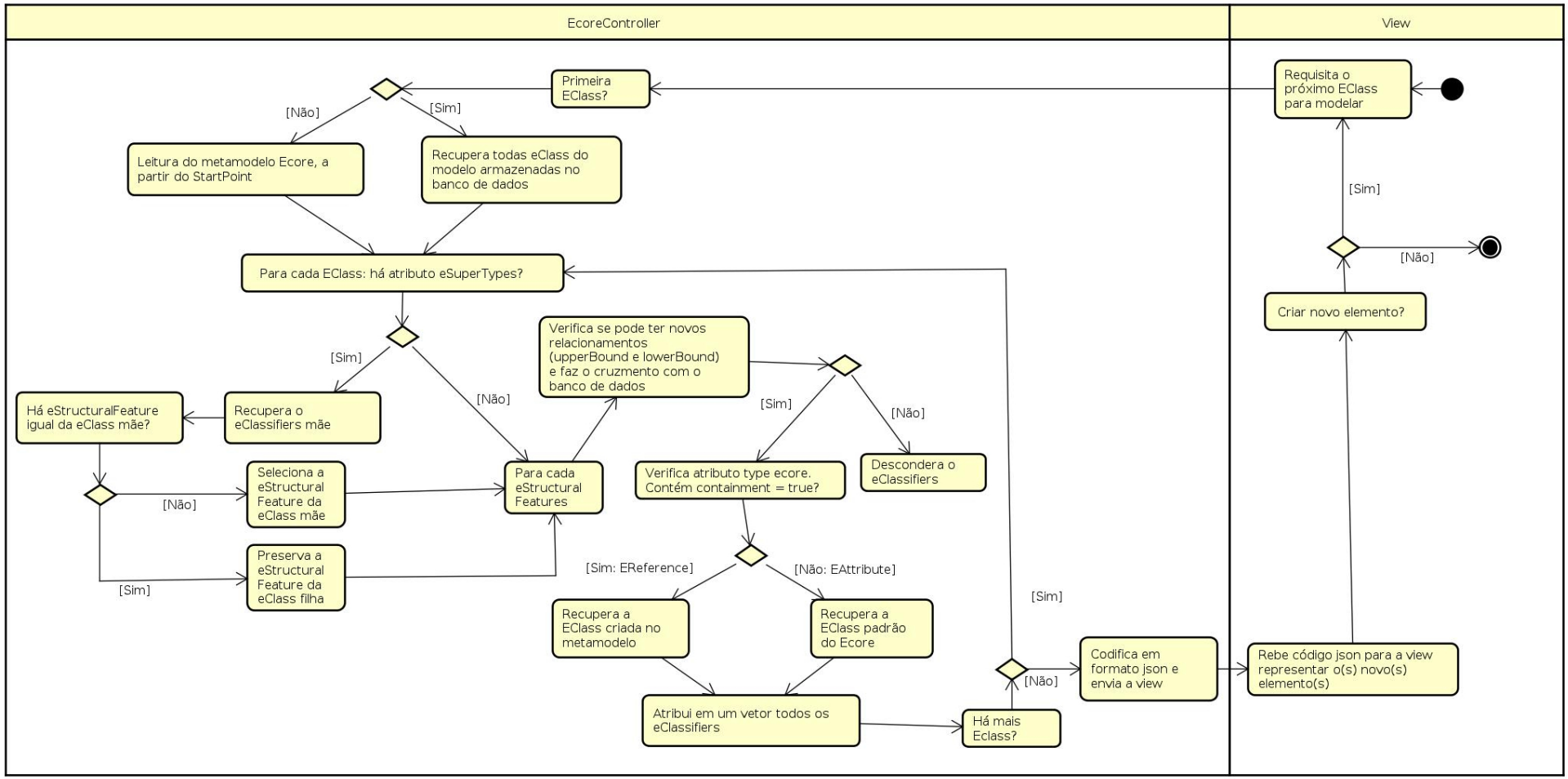


Figura 4.3: *Ferramenta para modelagem e para simulação de uma LPSD.*

Caso um EClassifier herde de outro EClassifier, este deve recuperar todas as EStructuralFeature e realizar uma união entre o EClassifier mãe com o EClassifier filho. Se o EStructuralFeature de um EClassifier mãe for igual ao EstructuralFeature do EClassifier filho, deve prevalecer a EStructuralFeature do filho.

Após o EcoreController reunir todos os EStructuralFeature de um EClassifier, o mesmo confere se cada estrutura pode ter relacionamentos por meio dos atributos upperBound e lowerBound. Além de conferir estes atributos, o controlador confere também a quantidade de *child* cadastradas. Se a quantidade no banco de dados não violar as regras de upperBound e de lowerBound, o EStructuralFeature é selecionado para compor um grupo de EStructuralFeature válidos para modelagem.

Outra verificação para o EStructuralFeature refere-se ao atributo booleano containment. Se este atributo for verdadeiro, o EStructuralFeatures é uma referência de outra EClassifier do metamodelo, caso contrário, trata-se de uma EClassifier padrão do Ecore.

O EcoreController armazena todas as configurações válidas em um vetor. Caso não haja nenhuma configuração válida, será retornado falso. Por fim, o vetor é convertido para o formato JSON e enviado de volta à camada View (Figura 4.3).

A View pode requisitar qual EStructuralFeature gostaria de criar para o EcoreController ou simplesmente enviar um componente para o EcoreController processar e realizar persistências. Na Figura 4.4, a View foi configurada para o usuário criar a rootFeature, porém, a camada View pode requisitar as regras para modelagem e quais elementos podem ser enviados. Por exemplo, no início da modelagem, o EcoreController deve iniciar a modelagem a partir do StartPoint, a EClassifier DynamicFeatureModel. A View recebe uma resposta em formato JSON, representado pelo módulo Model, conforme o exemplo do Código 4.1. Este código contém informações do estado atual da modelagem, isto é, de que o próximo elemento que o MC-D (DynamicFeatureModel) deve ter é uma característica raiz (*rootFeature*) do tipo Feature.

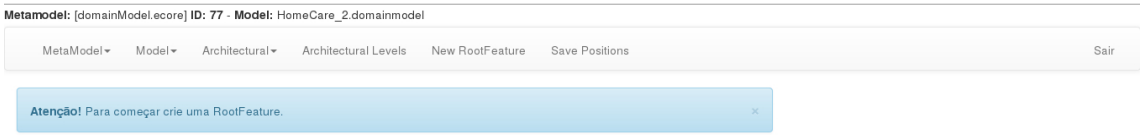


Figura 4.4: *Camada View recebe informação da camada Model para iniciar a modelagem de um MC-D.*

```
{
        "name": {
                "0": "DynamicFeatureModel"
        },
        "structural": [{
                "name": {
                        "0": "rootFeature"
                },
                "type": {
                        "type": "Feature"
                }
        }]
}
```

Código 4.1: *Interpretação do metamodelo pelo EcoreController para a criação de um `rootFeature`.*

4.1.2 FValidator

O módulo Fvalidator utiliza as regras de composição dos metamodelos (MC-D e MCC-D) e também contém regras que não estão presentes nos mesmos. As regras são específicas para LPS ou LPSD e o EcoreController consulta este módulo durante a modelagem.

As regras que não estão presentes nos metamodelos são escritas na própria linguagem PHP. Caso a View envie uma configuração ou requisição errônea o FValidator, por intermédio do EcoreController, retorna uma mensagem em JSON como erro da operação que a View tentou realizar. A mensagem de erro é vista na própria View para alertar ao usuário de que não é possível realizar a ação.

As regras definidas pela ferramenta e nos metamodelos são:

- Os modelos devem conter apenas uma característica *rootFeature*;
- Durante o processo de derivação, deve-se garantir que todas as características mandatórias representadas no MC-D sejam selecionadas para compor o MCC-D;
- A seleção de um grupo de características deve respeitar a cardinalidade da mesma;
- O grupo de características deve ser uniforme, isto é, todas as características devem ser estáticas ou dinâmicas, menos mandatória;
- Devem ser respeitadas a seleção das características do grupo de características (de um MCC-D) e as regras *requires* e *excludes* do MC-D;
- Uma característica não pode excluir (*excludes*) e requerer (*requires*) a mesma característica.

4.1.3 FCInterpreter

O FCInterpreter interpreta uma linguagem para a simulação de tarefas em nível de MCC-D. Trata-se de uma Linguagem de Domínio Específico, do inglês, *Domain Specific Language* (DSL). Uma DSL é uma linguagem de programação com uma sintaxe reduzida e simples para um uso específico [42]. A DSL facilita a automatização de ações no sentido de derivar produtos e realizar simulações.

O interpretador realiza análise léxica e sintática da linguagem. O analisador léxico faz a verificação *top-down* do *script* e transforma as palavras do código em uma sequência de *tokens*. O interpretador pode encontrar palavras reservadas e identificadores [2].

Sintaxe da DSL para MCC-D

Cada característica ou grupo de características é considerado como um *token* do tipo identificador. O *token* identifica como característica aquela cujo nome se inicia com a letra ‘f’, e como um grupo de características aquele cujo nome inicia com ‘fg’. Ambos os *tokens* são seguidos por um identificador (chave primária da tabela *child*) onde se encontra a característica no banco de dados. Por exemplo, f1 é uma característica com o identificador igual 1, fg2 é um grupo de características com o identificador igual a 2. O analisador léxico do módulo FCInterpreter verifica erros existentes ou comandos inválidos no *script*. Este módulo informa os erros no código e desconsidera os comandos com erros.

As palavras reservadas para a DSL desenvolvida são:

- **add**: adiciona uma característica estática;
- **remove**: remove uma característica estática;
- **fn.enable**: ativa uma característica dinâmica em tempo de execução, em que n é o identificador do grupo de características.
- **fn.disable**: desativa uma característica dinâmica em tempo de execução, em que n é o identificador da característica.

Comentários são identificados com duas barras invertidas à esquerda (\\). É possível ainda criar anotações para serem visualizadas na simulação, por meio do símbolo de cerquilha (#). As anotações são úteis para a visualização de informações adicionais, como, por exemplo, para descrever situações de mudança de contexto ou da estrutura do MCC-D.

A ferramenta não permite remover ou desativar completamente um grupo de características de uma característica, apenas permite a remoção ou desativação de características do grupo. Para se remover um grupo, deve-se remover todas as características pertencentes ao grupo.

Esta sintaxe para a DSL desenvolvida permite simular situações para cenários que envolvam uma LPS ou LPSD.

4.1.4 PanelFM e PanelFC

O módulo PanelFM é um painel para se modelar MC-D. A Figura 4.5 ilustra este painel de modelagem.

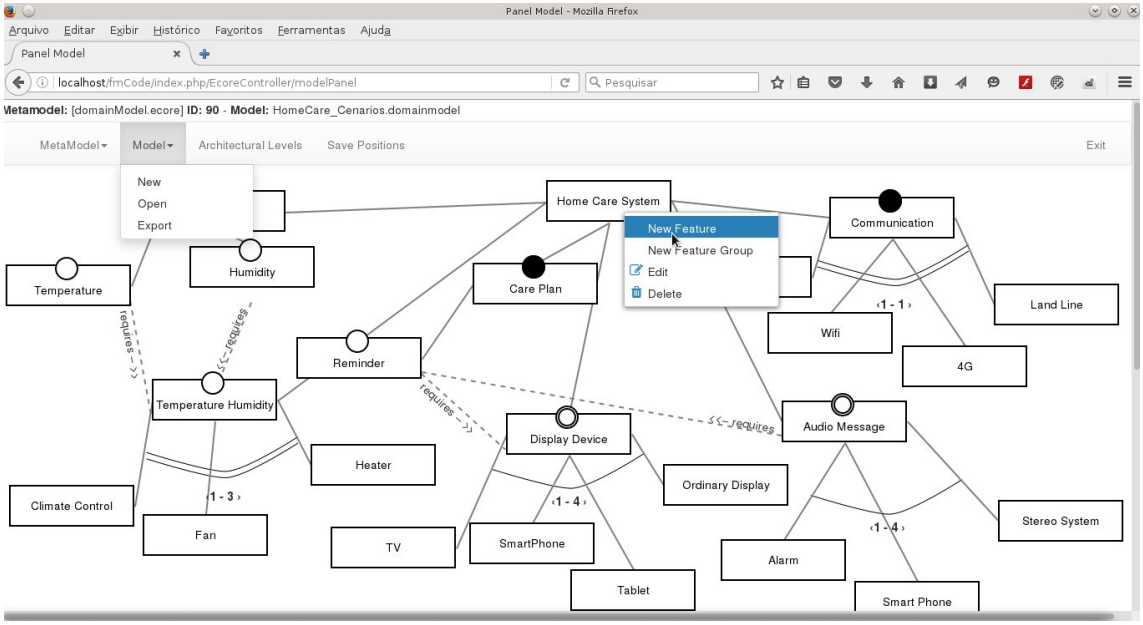


Figura 4.5: *Painel da camada View da Ferramenta para a modelagem de MC-D.*

Após a modelagem de um MC-D, a derivação de produtos é realizada pelo PanelFC. Por sua vez, o PanelFC é responsável por gerar modelos MCC-Ds, com base em um modelo presente no banco de dados. A Figura 4.6 ilustra este módulo. É possível derivar um MCC-D ou criar um produto, além de verificar qual metamodelo é instanciado pelo EcoreController.

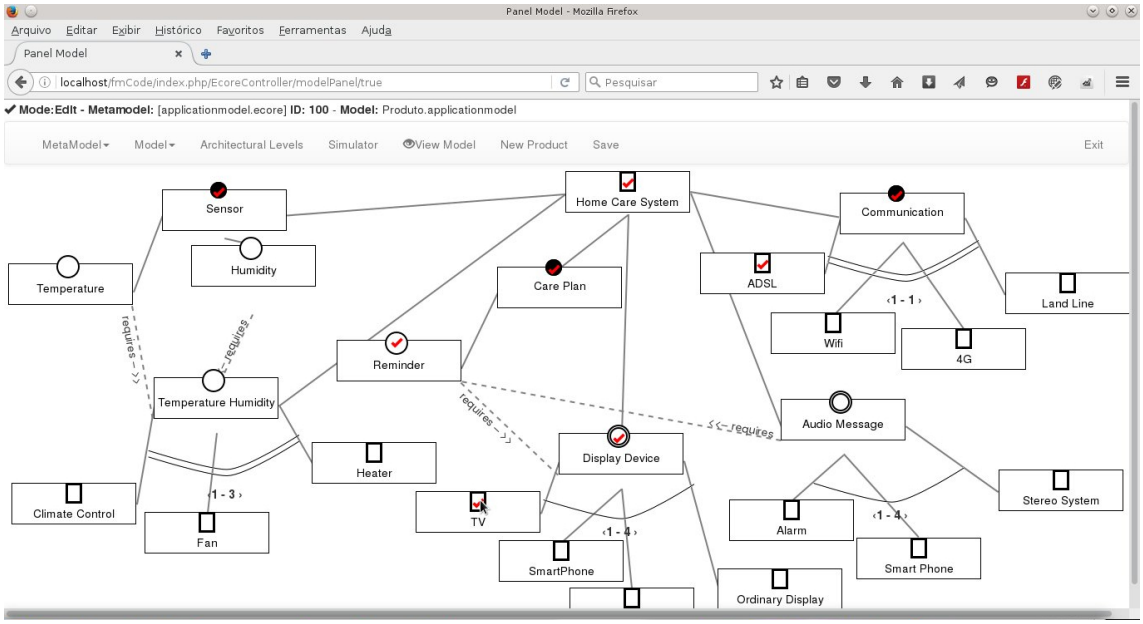


Figura 4.6: *Painel da camada View para modelagem de MCC-D.*

4.1.5 Simulator

Parte integrante da camada View, o Simulator realiza simulações de ativação e desativação de características baseando-se em cenários de LPS ou LPSD (Figura 4.7). O EcoreController recebe do Simulator o número da linha do *script* escrito na DSL e requisita ao FCInterpreter que interpreta o comando. O FCInterpreter faz uma análise léxica do código e consulta o EcoreController para que seja verificada a ação solicitada, se pode ou não ser realizada. Por exemplo, quando a característica ADSL é desativada e a característica GSM é ativada (situação codificada na forma de um *script* usando a DSL desenvolvida), o EcoreController verifica se estas características existem e também se podem ser ativadas ou desativadas em tempo de execução, ou ainda em tempo de implantação. Caso ocorra algum impedimento, o EcoreController retorna *false* em formato JSON ao Simulator, que, por sua vez, não realiza nenhuma ação na camada View. Este processo é realizado sempre que o Simulador requisita a interpretação de uma linha do *script*.

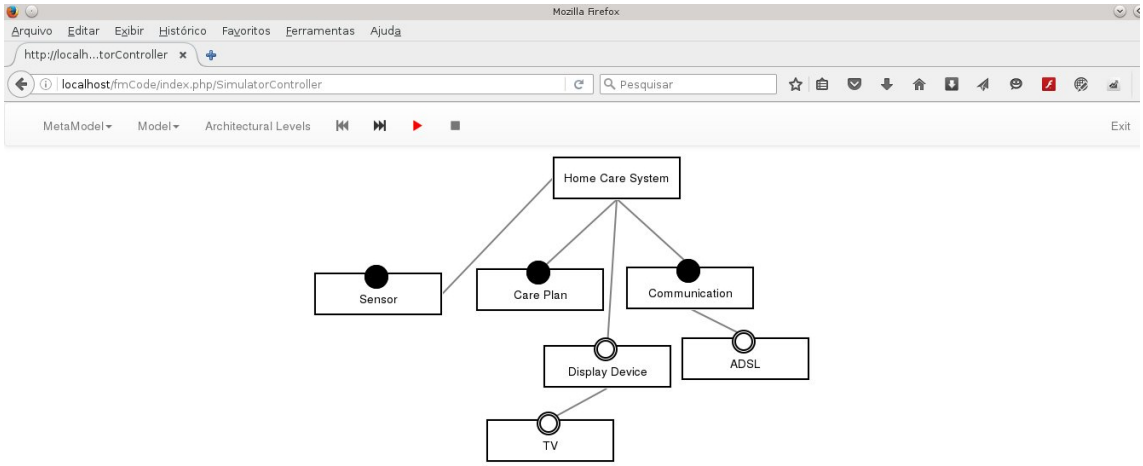


Figura 4.7: *Simulação de mudanças de estados de características estáticas ou dinâmicas*

O Simulator apresenta uma animação representando as mudanças de estado das características. A animação pode ser definidas por um determinado tempo ou manualmente. Cada mudança de estado no produto equivale a um comando no *script*. Na próxima Seção são apresentados exemplos com cenários envolvendo a aplicação de *homecare*. Estes cenários incluem simulações de derivações de produtos.

4.2 Ferramentas Correlatas

Nesta Seção são discutidas algumas ferramentas semelhantes à ferramenta proposta. São abordadas as ferramentas EMF, GMF, featureIDE e a Splot.

4.2.1 Modelagem em EMF

O EMF[1] (*Eclipse Modeling Framework*) é uma ferramenta (*framework*) para modelagem de metamodelos conforme foi apresentado na Seção 2.7. Nesta ferramenta é possível gerar representações de modelos a partir de um metamodelo.

A representação de modelos no EMF segue um mesmo padrão, portanto, não se assemelha aos modelos tradicionais de LPS (MC e MCC) e à proposta de representação de modelos para a LPSD (MC-D e MCC-D). Não é possível representar com clareza os grupos de características, as regras *requires* e *excludes*, as características estáticas ou dinâmicas, a característica raiz, entre outras. O exemplo de modelagem EMF ilustrado na Figura 4.8 contém todas as características apresentadas na proposta de representação de características dinâmicas em um MC-D na Seção 3.2.

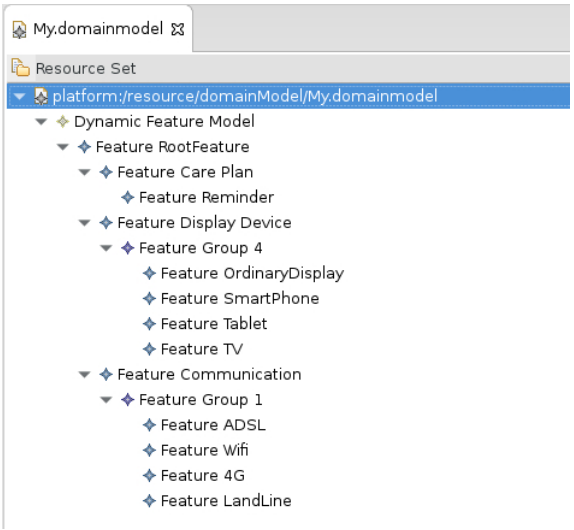


Figura 4.8: *MC desenvolvido em EMF e baseado do MC-D da Seção 3.2*

Este *framework* gera automaticamente classes Java de todo o metamodelo para a implementação de cada característica presente no metamodelo. A

[1] https://eclipse.org/modeling/emf

geração de código favorece a implementação de características no nível M0 proposto pela OMG. A ferramenta é útil para auxiliar na modelagem de metamodelos, tendo inclusive, apoiado bastante nos testes relativos às validações iniciais do metamodelo proposto.

Para enriquecer a representação no EMF, desenvolveu-se o GMF (*Graphical Modeling Framework*), um *framework* para possibilitar melhores representações gráficas de modelos a partir de um metamodelo. A modelagem em GMF é abordada na próxima Seção.

4.2.2 Modelagem em GMF

O GMF[2] (*Graphical Modeling Framework*) é uma extensão do EMF, um *framework* que utiliza o IDE Eclipse para desenvolver ferramentas para modelagens gráficas de modelos.

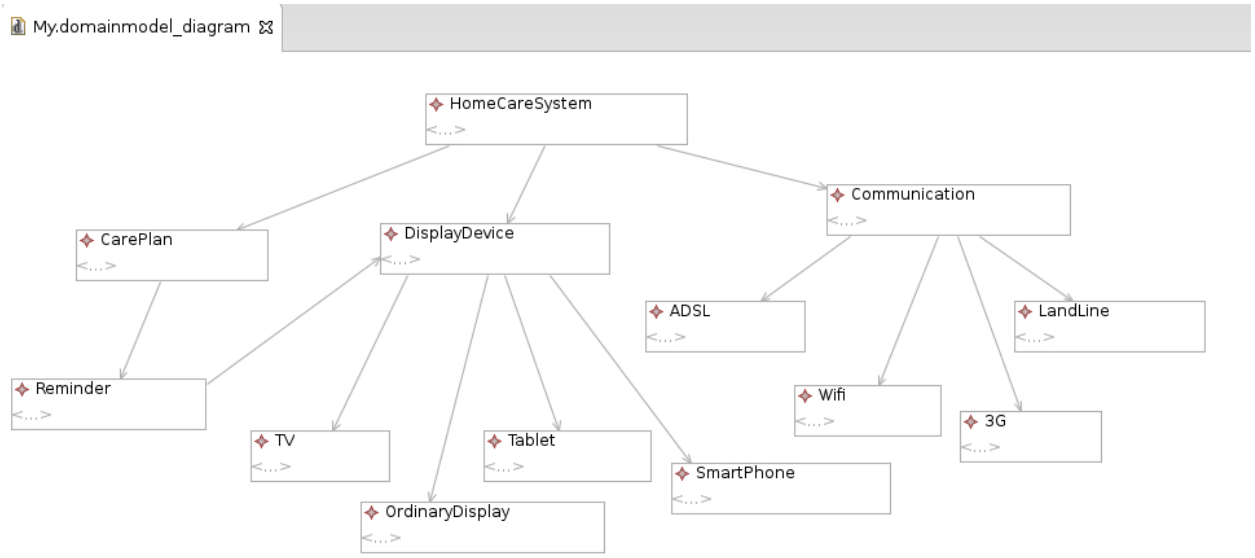


Figura 4.9: *MC desenvolvido em GMF e baseado do MC-D da Seção 3.2*

A Figura 4.9 ilustra um exemplo de modelagem equivalente ao MC-D da Seção 3.2. É possível criar representações específicas para os modelos no GMF, porém ele possui limitações quanto à representação gráfica de modelos e não permite realizar simulações.

[2]http://eclipse.org/gmf-tooling

4.2.3 Modelagem em featureIDE

O featureIDE é um *framework* de código fonte aberto para o desenvolvimento de software orientado a características, do inglês *feature-oriented software development* (FOSD). Ela utiliza a plataforma de desenvolvimento Eclipse IDE e abrange as técnicas de desenvolvimento de programação envolvendo características, aspectos, delta e preprocessadores.

A programação orientada a características pode utilizar objetos ou artefatos do sistema (não é orientado a objetos) de acordo do princípio da uniformidade, isto é, todos artefatos devem ser compostos por documentação, especificação ou modelos.

Já a programação orientada a delta é composta por um módulo central, desenvolvido em uma linguagem de programação do *host*, por exemplo, em Java e um conjunto de módulos delta. Cada módulo delta pode adicionar, remover campos métodos e classes, enfim realizar mudanças na estrutura do código no *host*. As mudanças são realizadas por fórmulas proposicionais para definir regras que permitem mudanças nas rotinas do código. Esta técnica tem suporte na análise de domínio e de requisitos, implementação do domínio e geração de software.

Outra técnica de programação oferecida pelo featureIDE é a programação orientada a aspectos. É uma meta linguagem para possibilitar mudanças nos objetos do programa. Cada característica corresponde a um aspecto na ferramenta. Também acrescentaram métodos para a linguagem orientada a aspectos como o `after()` para realizar chamada (`call()`) de outro código, por exemplo, um método.

Por fim, os preprocessadores são inseridos como comentários seguidos pelo símbolo cerquilha dentro da linguagem de programação do *host*. Por exemplo, na featureIDE é possível utilizar a condicional “se então” como `#if` e `#endif` para mudar as sequências das rotinas de códigos durante a geração do produto.

A featureIDE suporta os processos da técnica FOSD nas fases de: análise, domínio, implementação de domínio, análise de requisitos e geração de software.

Para criar as regras de composição, a ferramenta utiliza *propositional constraint*, do português, restrição proposicional. Por exemplo, no MC da Figura 4.10 a característica estática *Reminder* exige a seleção da característica *Display Device* representado pelo nome requires. A restrição proposicional para a regra *requires* é `Reminder implies DisplayDevice` (veja na Figura 4.10).

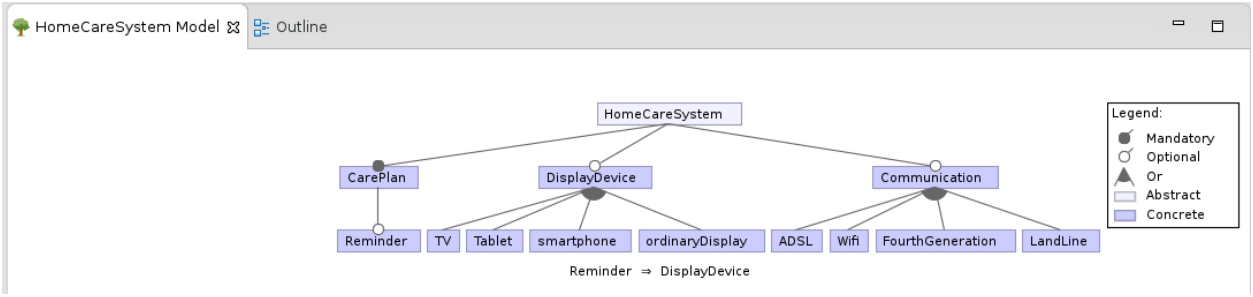


Figura 4.10: *MC desenvolvido em featureIDE e baseado do MC-D da Seção 3.2*

Nativamente, esta ferramenta não tem suporte a aplicação *web-based*. O desenvolvimento da ferramenta depende do IDE Eclipse e, caso haja necessidade de migração, esta ferramenta para a *web* deve mudar a arquitetura e, principalmente, a interface de visão.

Em relação à representação, a modelagem é simples e fácil, porém não é possível mover as características para ajustar conforme o necessário. Esta ferramenta utiliza um diagrama de colaboração para representar os produtos derivados. A ferramenta limita-se à criação de MC para LPS tradicional, portanto, não é possível modelar características dinâmicas para uma LPSD.

A proposta da ferramenta de modelagem complementa a ferramenta featureIDE na representação de MC-D e MCC-D para LPSD, conforme apresentado no Capítulo 3. A featureIDE se limita ao ambiente *Desktop* e não permite realizar simulações de mudanças de estados nas derivações de produtos.

4.2.4 Modelagem em Ferramenta Splot

O S.P.L.O.T [3] (*Software Product Lines Online Tools*) é uma ferramenta desenvolvida em Java para *web* com interfaces gráficas em HTML e Ajax para modelar MC e MCC, exclusivo de uma LPS. Esta ferramenta por ser baseada na *web* se assemelha à proposta de ferramenta desta dissertação.

Esta ferramenta oferece representações em MC e MCC somente para LPS [30]. A Figura 4.11 é um exemplo de modelagem em Splot baseada na modelagem de MC-D da Seção 3.2. A Figura 4.12 é um exemplo de produto derivado da Figura 4.11.

A proposta desta dissertação complementa as representações de MC e MCC da ferramenta Splot. Além disso, a proposta de ferramenta para simulação, pode auxiliar aos usuários na modelagem no Splot.

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

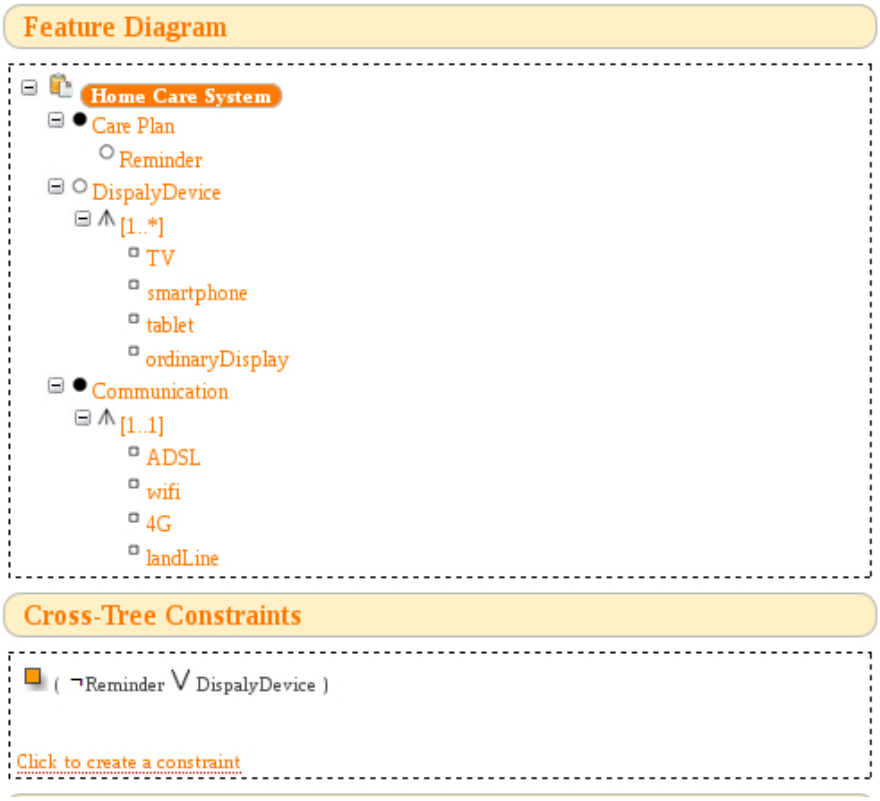


Figura 4.11: *MC desenvolvido em Splot e baseada do MC-D da Seção 3.2*

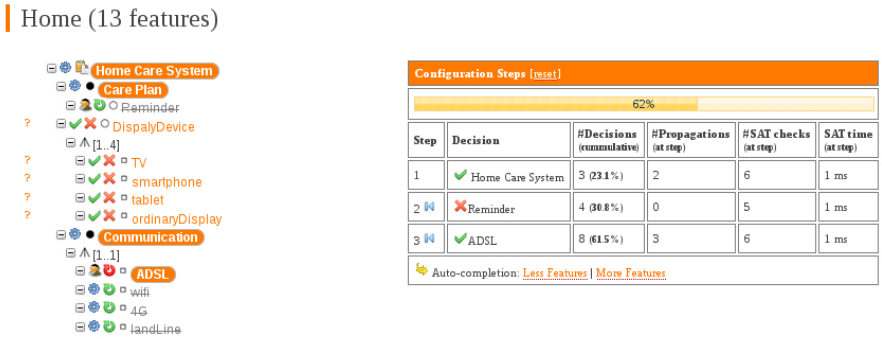


Figura 4.12: *Representação de uma configuração Splot, baseado no modelo da Figura 4.11*

4.2.5 Comparativo

A Tabela 4.1 apresenta uma comparação da proposta de ferramenta desta dissertação com as ferramentas relacionadas n Seção anterior. Os significados das siglas utilizadas para identificar as colunas da Tabela 4.1 são as seguintes:

- **C1**: Permite a interpretação de Metamodelo;
- **C2**: Modelagem para LPS;

- **C3**: Modelagem para LPSD;
- **C4**: Suporte para ambiente *web*;

Tabela 4.1: *Comparação das ferramentas relacionadas*

Nome	**C1**	**C2**	**C3**	**C4**
EMF	**X**	**X**		
GMF	**X**	**X**		
featureIDE		**X**		
Splot		**X**		**X**
Proposta de ferramenta	**X**	**X**	**X**	**X**

As ferramentas listadas na Tabela 4.1 permitem modelar MC e/ou representar derivações de produtos para uma LPS tradicional. Por outro lado, nenhuma das ferramentas propõe representar características dinâmicas em suas modelagens e consequentemente não tem suporte para LPSD. Além disto, a maior parte destas ferramentas são voltadas para ambiente Desktop. As únicas ferramentas que interpretam metamodelos, neste caso, baseados no meta-metamodelo Ecore, são a EMF e o GMF. Também as ferramentas EMF e GMF são executadas por meio do IDE Eclipse.

4.3 Cenários de Uso

Essa Seção apresenta cenários desenvolvidos usando-se o metamodelo e a ferramenta proposta. Os cenários têm como base uma aplicação de *homecare* presentes na Figura 4.13, para LPS, e na Figura 4.14, para LPSD. O cenário para LPS visa ressaltar a importância de aplicações autoadaptativas, ubíquas e na representação de características dinâmicas para uma *homecare*.

A Figura 4.15 ilustra um exemplo de planta de uma residência no contexto de uma aplicação *homecare*, com sensores e atuadores. Este exemplo de planta pode ter novos símbolos de acordo com a seleção de características do MC-D.

A Figura 4.16(a) ilustra as características do grupo *Display Device* (*TV, tablet, smart phone e ordinary display*). A Figura 4.16(b) ilustra o grupo de características *Audio Message* (*Stereo system, Smart phone* e *Alarm*), e na Figura 4.16(c) encontra-se um grupo de características de controle de temperatura (*Temperature and Humidity Control*) que contém as características *Climate Control* (controle de temperatura, fan (ventilador) e *heater* (aquecedor). As características *Display Device*, *Audio Message* e *Temperature and Humidity Control* e seus grupos são dinâmicas, no sentido de ativar ou desativar determinadas características do grupo. A característica *Communication* é mandatória, porém, pode mudar o seu estado de comunicação em tempo de execução, conforme o seu grupo.

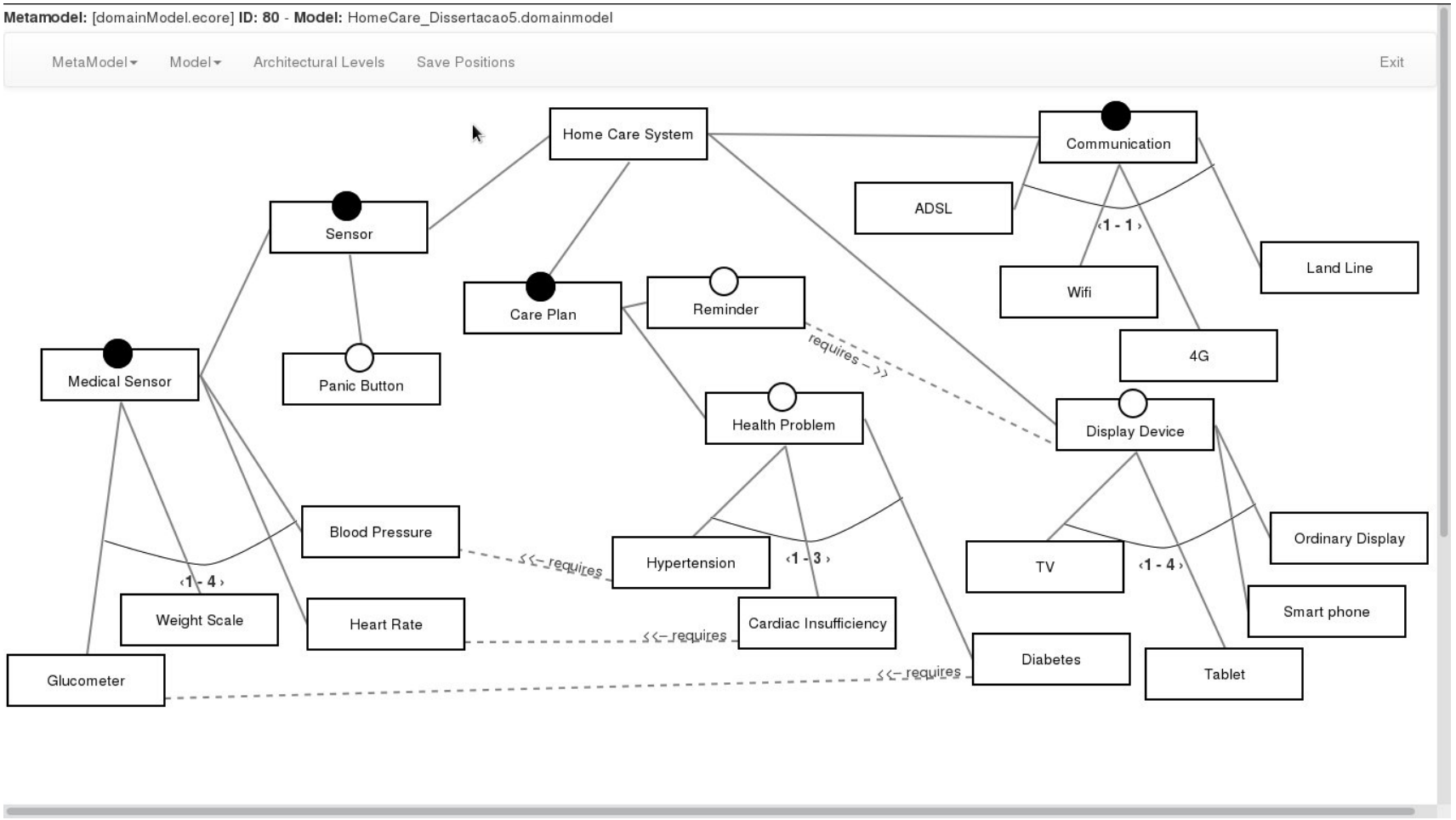


Figura 4.13: *Modelo de Características desenvolvido usando a ferramenta de modelagem*

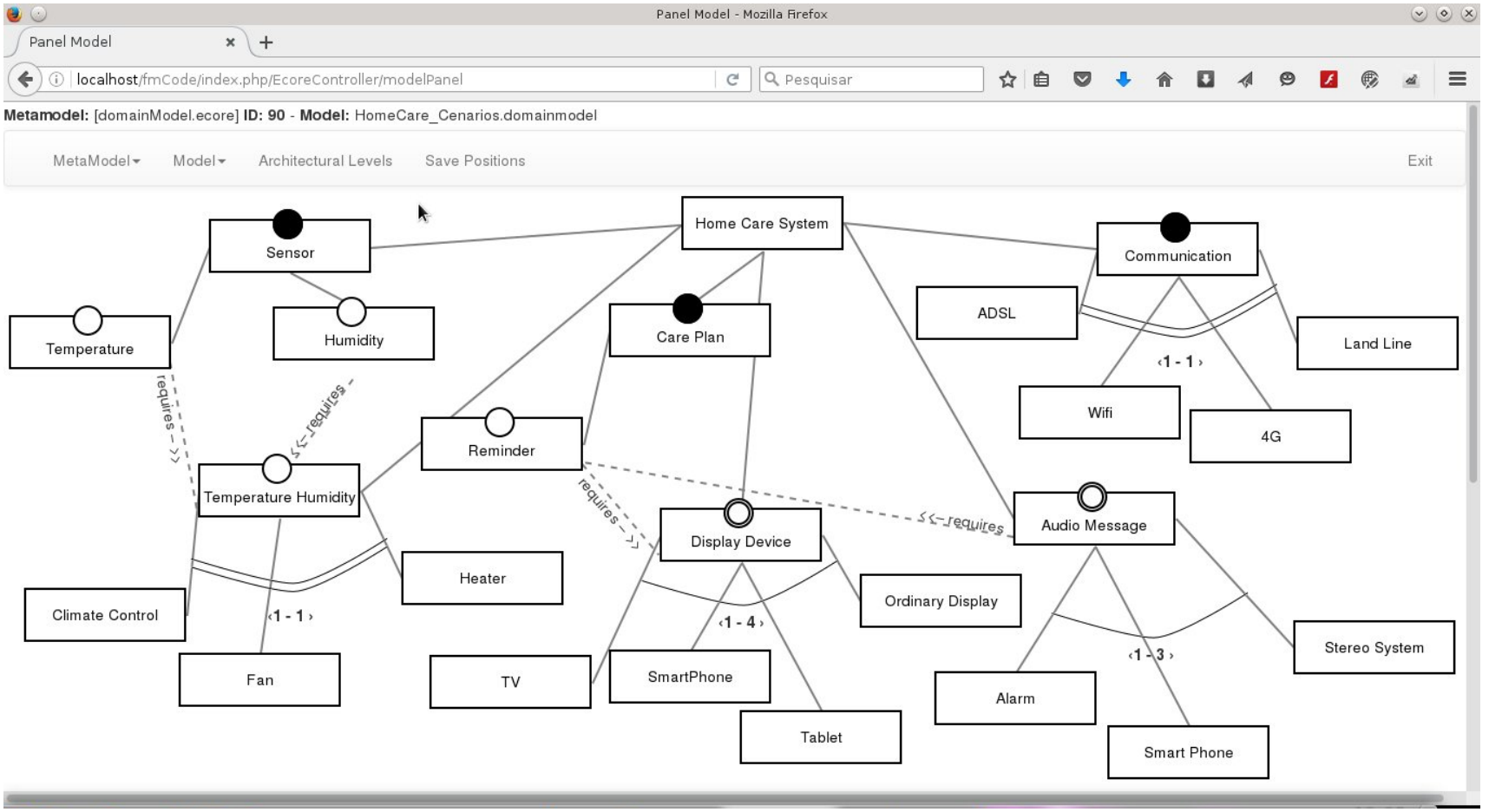


Figura 4.14: *Modelo de Características Dinâmicas desenvolvido usando a ferramenta de modelagem*

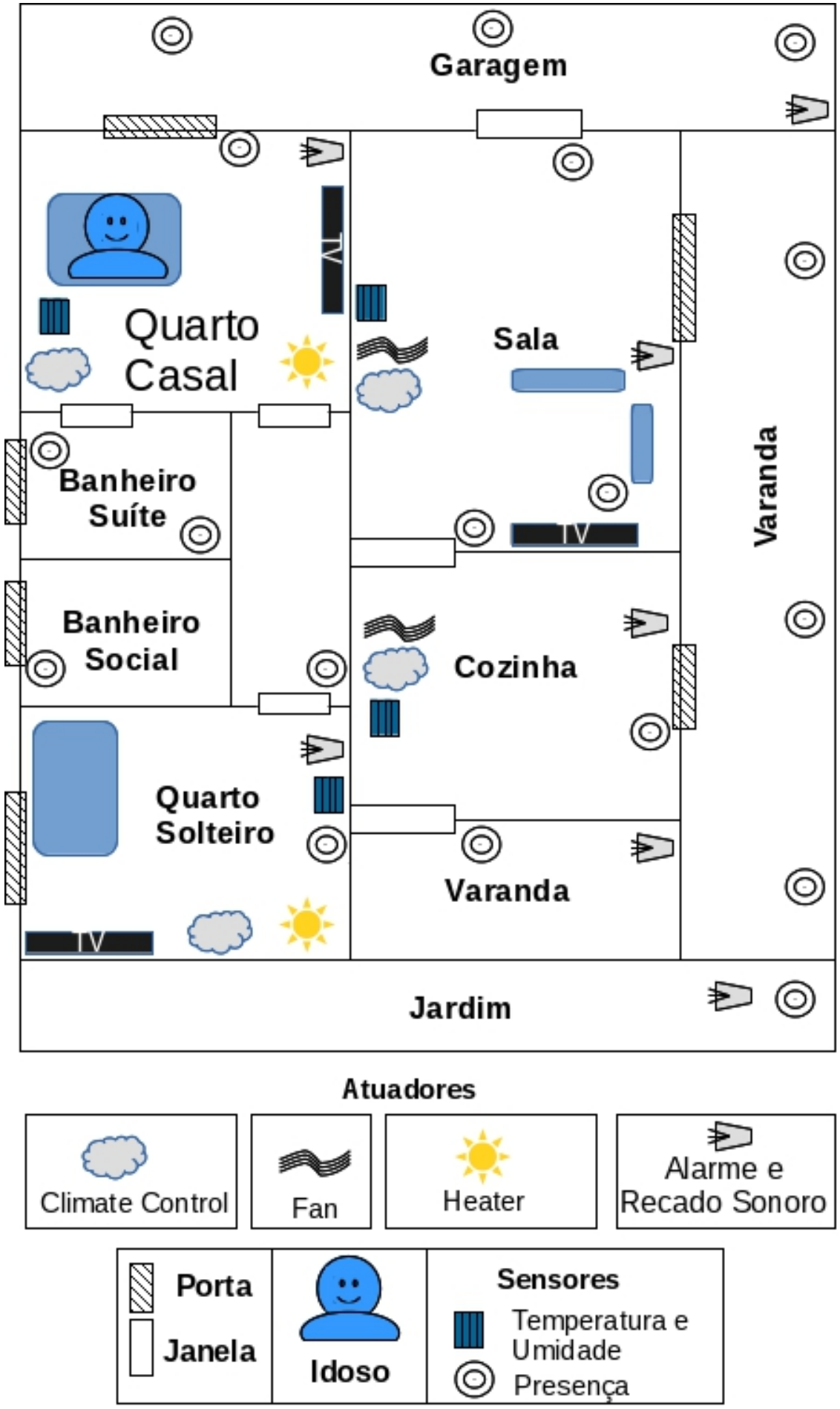
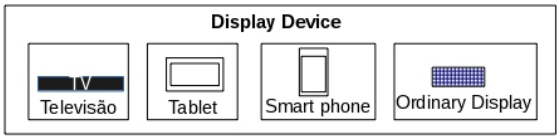
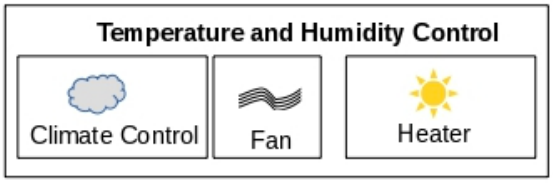


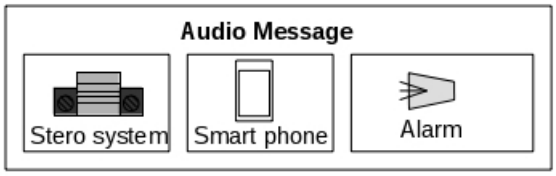
Figura 4.15: *Exemplo de planta para uma residência no cenário de homecare*



(a) *Grupo de características Display Device*



(b) *Grupo de características Temperature Control*



(c) *Grupo de características Audio Message*

Figura 4.16: *Símbolos para representarem grupos de características na planta de exemplo de homecare*

O cenário para LPS (subseção 4.3.1) envolve a comunicação com a Internet, sensor para medir a pressão, hipertensão e o botão de pânico. Já os cenários abordados para LPSD são os seguintes: comunicação dinâmica via Internet (subseção 4.3.2); visualização de recados e de informações provida pelo grupo de características *Display Device* (subseção 4.3.3); notificação em áudio, representada pelo grupo de características *Audio Message* (subseção 4.3.4), e, por fim, controle de temperatura representado pelo grupo de características dinâmicas *Temperature and Humidity* (subseção 4.3.5).

4.3.1 Cenário para uma LPS

Este cenário visa simular implantações de produtos estáticos para uma LPS por meio dos modelos MC e MCC convencionais. A instalação das características de uma LPS exige que o responsável pela aplicação instale na casa do paciente os hardwares necessários e implante a aplicação na casa. As características são baseadas no MC de [12] conforme a Figura 4.13. Os requisitos mínimos para a implantação da aplicação da *homecare* são: *Care Plan* (plano de cuidados); *Sensor* (Sensor), que, por sua vez, exige a seleção de sensor de medicamento (*Medical Sensor*) e a escolha de no mínimo uma

característica do seu grupo de características; *Communication* (comunicação com a Internet por ADSL), além da *rootFeature* (*Home Care System*), que também é considerada como uma característica obrigatória em todos os produtos.

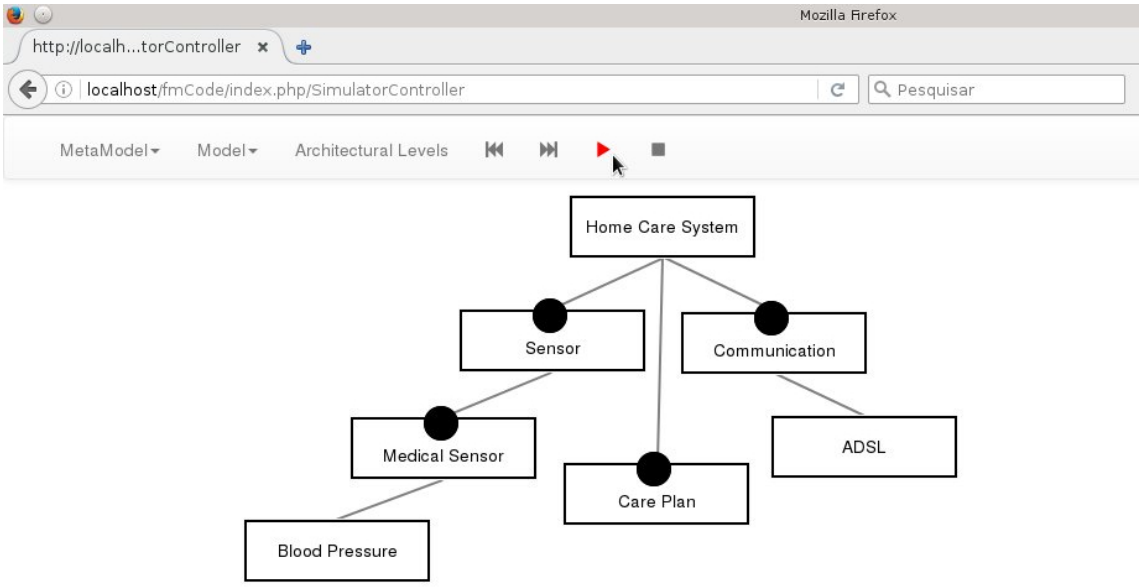


Figura 4.17: *MCC com a configuração mínima, baseado no MC da Figura 4.13*

Para um profissional de saúde acompanhar a pressão do paciente, o mesmo decide escolher um sensor para medir a pressão (*Blood Pressure*) conforme a configuração mínima no MCC, ilustrada na Figura 4.17. Portanto, o paciente deverá medir a pressão para a aplicação enviar os dados a um profissional de saúde. Esta é a primeira configuração da aplicação, porém, pode ser customizada com novas características opcionais, de acordo com as necessidades e da saúde do paciente.

Através de exames e do histórico de saúde coletados no leitor *Blood Pressure*, o profissional de saúde percebeu que o idoso tem hipertensão. É necessário, gerar outra derivação da aplicação para acrescentar as seguintes características: *Health Problem* e *Hypertension*. Assim, o profissional de saúde poderá definir os planos de cuidados para a hipertensão (Figura 4.18), por intermédio da comunicação com a Internet (característica ADSL presente no grupo *Communication*). Este plano de saúde deve ser seguido manualmente pelo paciente. Os dados de saúde coletados formam um registro da pressão do paciente e o profissional de saúde poderá acompanhar a prescrição médica conforme a evolução da saúde do paciente ou até mesmo tomar outras providências necessárias.

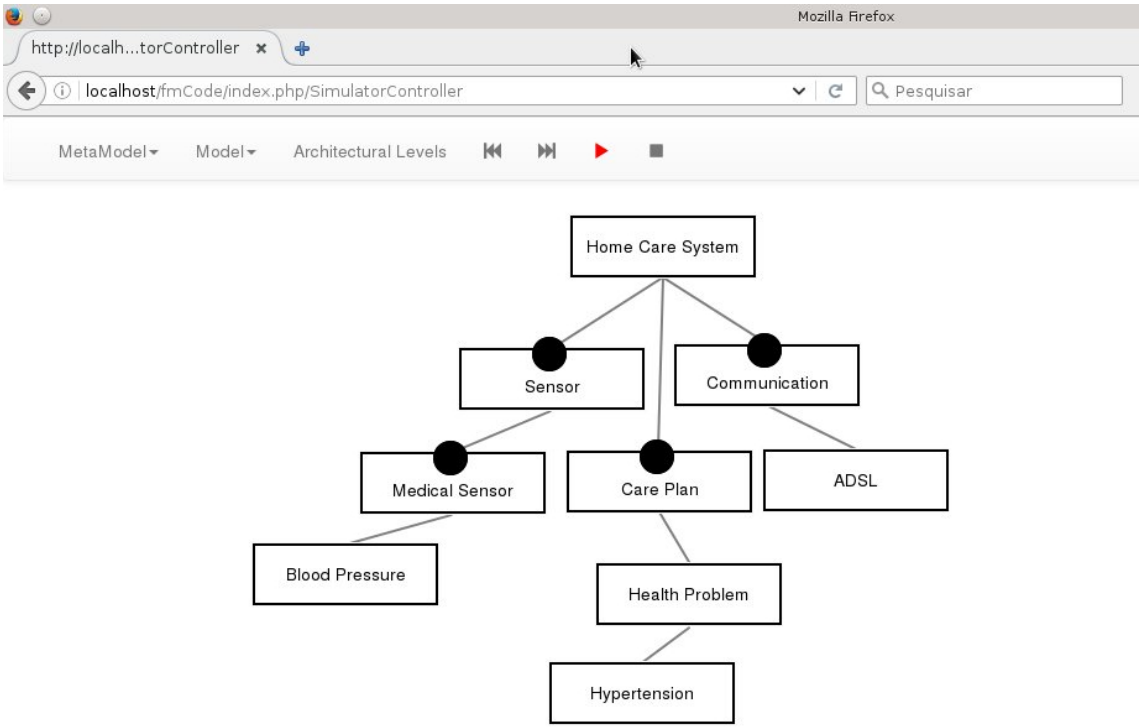


Figura 4.18: *Nova derivação do MCC anterior (Figura 4.13) para pessoas com hipertensão*

O paciente, por sua vez, solicitou um requisito para acionar a um profissional de saúde, em situações de emergência de sua saúde, na *homecare*. A característica *Panic Button* satisfaz este requisito do usuário e a mesma é instalada novamente em tempo de implantação. O MCC é atualizado e poderá ser vista a nova característica opcional no modelo (Figura 4.19).

Se algum dispositivo, representado por uma característica no MCC, não funcionar corretamente, devido, por exemplo, a um defeito no hardware ou de um erro nível de software, o profissional responsável pela aplicação deve ir à *homecare* para corrigir estes problemas na aplicação. Por exemplo, se a conexão ADSL falhar, a aplicação não funcionará, enquanto este dispositivo não voltar a funcionar ou gerar um novo produto com suporte a outra comunicação.

Este cenário de características estáticas não permite mudanças de estados das características em tempo de execução (veja na Seção 2.2). A ausência de adaptações em tempo de execução pode prejudicar o monitoramento da saúde do paciente, devido a algumas características essenciais, como, por exemplo, a característica *Communication*. Portanto, os exemplos de produtos derivados nesta Seção não permitem explorar sensibilidade do contexto ou implantar aplicações ubíquas. Na próxima Seção, são abordadas simulações de

cenários para uma LPSD, desenvolvidos na ferramenta com o foco em aplicações sensíveis ao contexto.

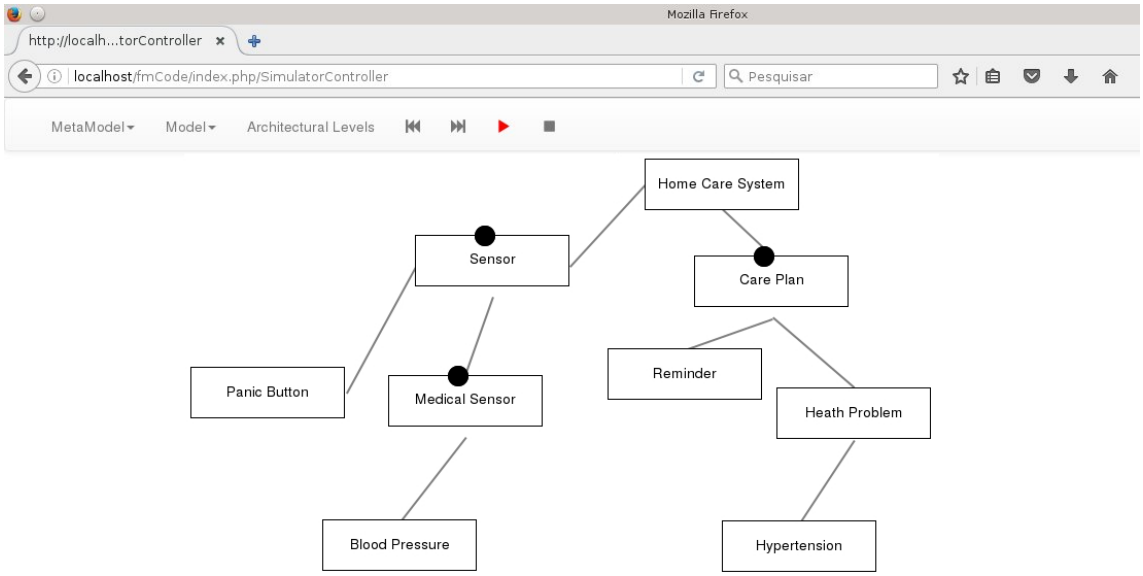


Figura 4.19: *Requisito do usuário: botão de pânico, presente no MC (Figura 4.13)*

4.3.2 Comunicação Dinâmica Via Internet

A comunicação, via Internet, do paciente com o profissional de saúde é um requisito essencial. Esta característica é útil, principalmente, no envio de informações do estado de saúde do paciente.

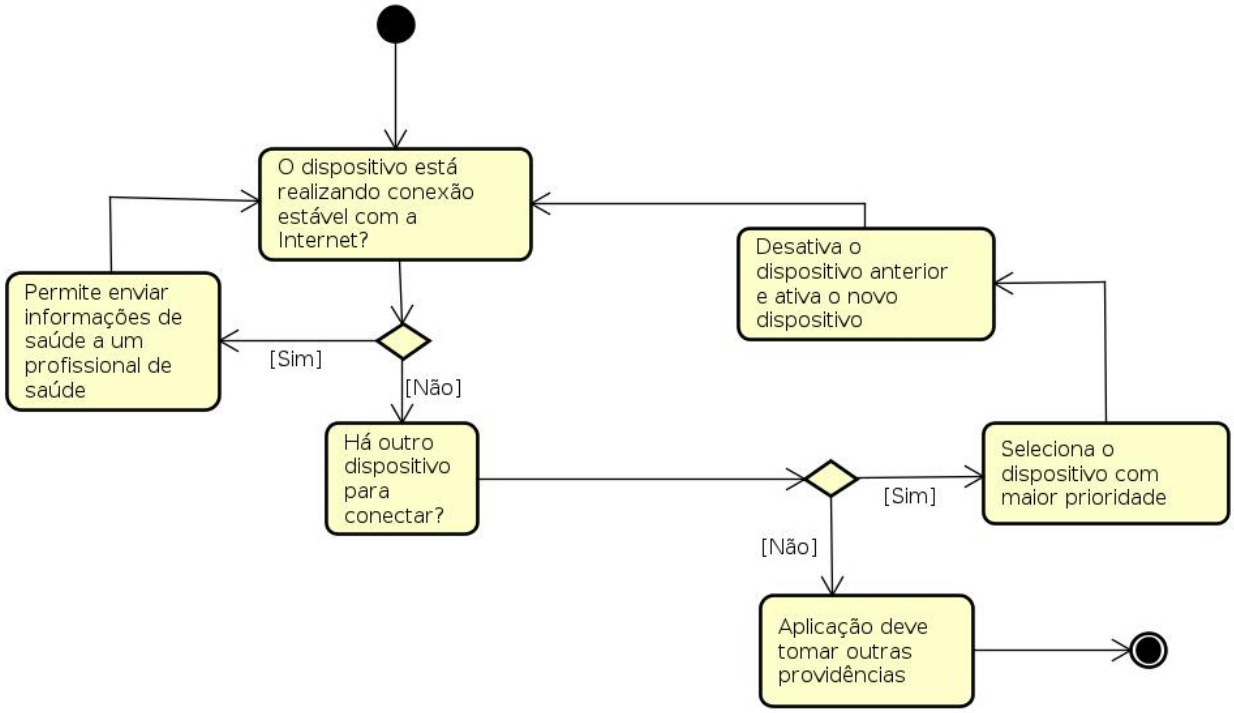


Figura 4.20: *Diagrama de atividades para o cenário de comunicação via Internet*

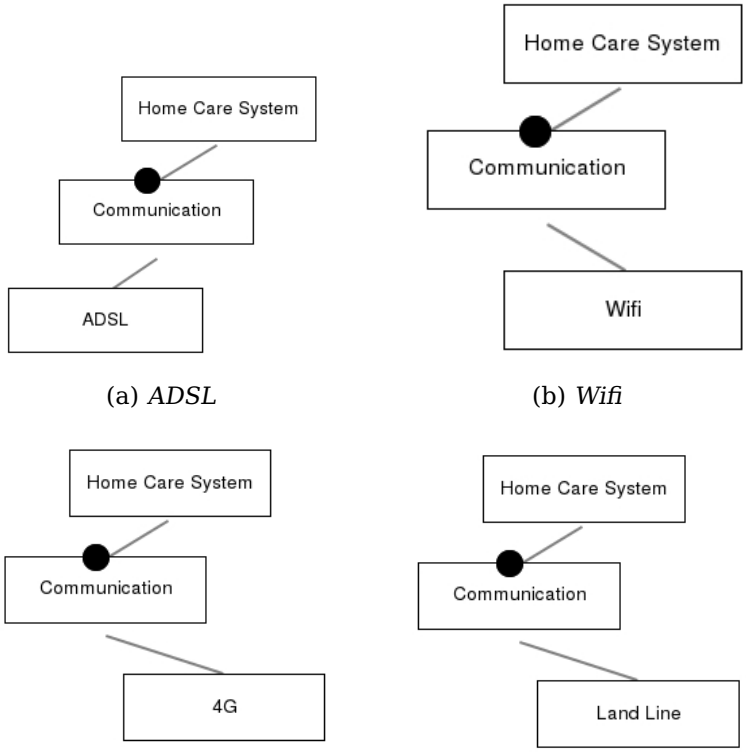
De acordo com o MC-D da Figura 4.14, a característica *Communication* é mandatória. Portanto, exige a seleção de uma característica dinâmica do grupo de características. Conforme o Diagrama de Atividades da Figura 4.20, a forma de conexão é alterada, caso ocorra alguma anormalidade na comunicação. A prioridade de seleção de características do grupo *Communication* pode ser definida da seguinte forma: em primeiro lugar, ADSL; em segundo, conexão Wifi, em terceiro, conexão 4G; e por último *Land line*. A simulação deste cenário é detalhada no *script* 4.2 escrito na DSL.

Código 4.2: *Script para simular exemplos de comunicação via Internet*

```
1  // ******************************************************
2  // Simulação: Comunicação Dinâmica via Internet
3  //
4  // Cenário para mudanças de estados de acordo com
5  // a prioridade da comunicação com a Internet.
6  //
7  // Nesta simulação a ADSL é definida como um
8  // meio de comunicação de maior peso no grupo.
9  // A ordem é: 1º ADSL, 2º wifi, 3º 4G, 5º Land Line
10 // ******************************************************
11
12 // Derivação em tempo de implantação:
```

```
13
14 add f50;     // Communication - mandatória
15 add f51;     // Grupo de Communication - grupo dinâmico
16 f52.enable; // Define como ativo a característica ADSL
17
18 // Exemplo de derivações em tempo de execução:
19
20 //1º ADSL instável
21 f52.disable; // desativa a ADSL
22 f53.enable;  // ativa o Wifi
23
24 //2º Wifi instável
25 f53.disable;  // desativa o Wifi
26 f54.enable;   // ativa o 4G
27
28 //3º 4G instável
29 f54.disable;  // desativa o 4G
30 f55.enable;   // ativa o Land line
31
32 //4º Land line instável
33 f55.disable; // desativa o Land line
34
35 // Caso todas as comunicações estejam estáveis,
36 // a aplicação ficará sem a comunicação com a Internet.
37 // A comunicação só voltará a comunicar se algum dispositivo
38 // voltar a conectar, respeitando a ordem de prioridades.
```

Diante de uma situação de falha em que a aplicação percebe que a ADSL não permite a conexão com a Internet, esta deve ser desativada e a próxima característica (*Wifi*) deve ser ativada. Se o *Wifi* não funcionar, esta conexão deve ser desativada e a 4G ativada, e, por fim, se a 4G não funcionar a última alternativa (*Land line*) deve ser ativada. A Figura 4.21 mostra as derivações do grupo de características. Caso nenhum estado de cada característica atenda o esperado, a aplicação ficará sem a conexão com Internet. Por outro lado, se uma característica com maior prioridade voltar a funcionar, por exemplo, a ADSL, e se houver outra característica ativa, ela deverá ser desativada para ativar a ADSL.



(a) *ADSL* (b) *Wifi*

(c) *4G* (d) *Land Line*

Figura 4.21: *Mudanças de estados de acordo com a prioridade da comunicação com a Internet*

4.3.3 Notificação Sensível ao Contexto

Neste cenário, a característica *Reminder* tem a funcionalidade de receber recados, notificações, entre outras informações. Para visualizar estas informações, é necessário ativar o grupo de características dinâmicas pertencente ao *Display Device*. O *Display Device* serve também para visualizar todas as informações que a aplicação pode oferecer, por meio de um MC-D e as características ativas em um MCC-D. Estas características estão no MC-D ilustrado na Figura 4.14.

O diagrama de atividades da Figura 4.22 mostra o processo de notificação ao paciente. Por meio de sensores de presença, a aplicação verifica se há um dispositivo próximo ao paciente. Se no local onde o paciente está, não há um visor, a aplicação aguarda o paciente mudar de ambiente na casa. A partir do momento que o paciente se encontrar próximo a um ou vários visores, a aplicação seleciona a característica de maior peso para lançar o evento de um novo recado. Por fim, o paciente deve confirmar a ciência de que há um recado para ser visualizado.

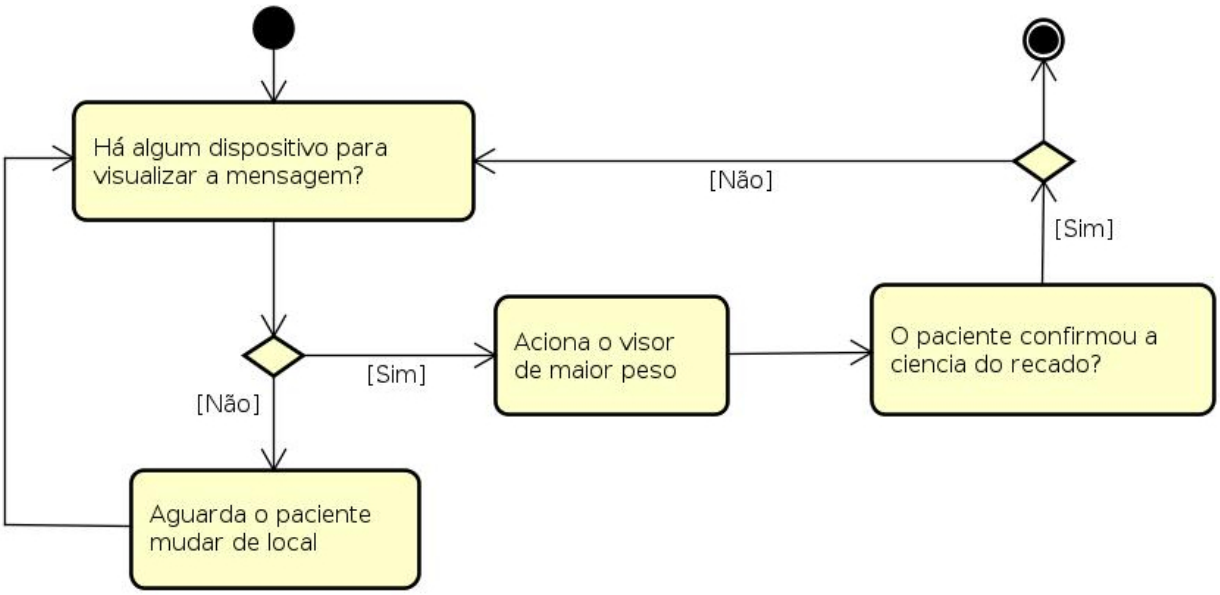
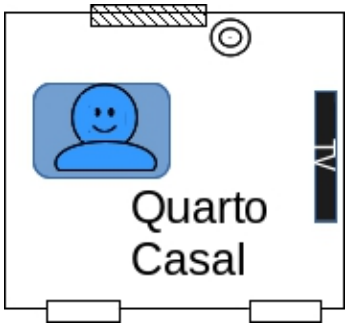


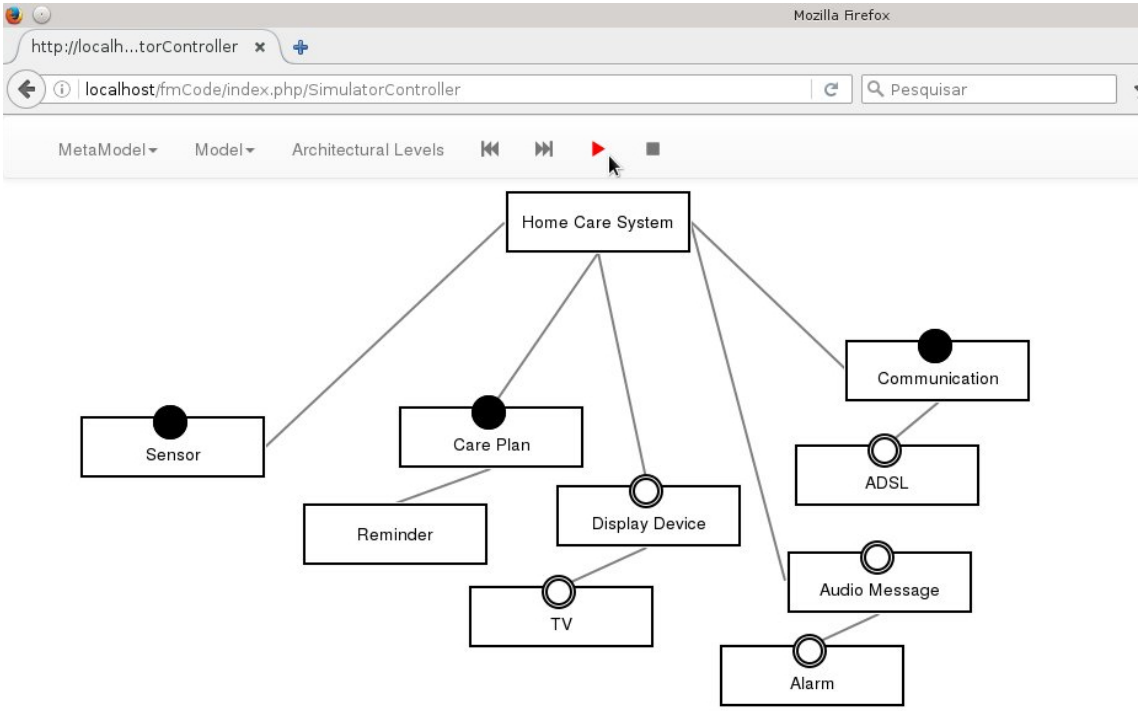
Figura 4.22: *Diagrama de atividades para o cenário de notificação.*

Por exemplo, paciente, em especial, idosos, podem se confundir ou esquecer de quais remédios devem tomar nos horários especificados. O *Reminder*, em conjunto com o grupo de características *Display Device*, auxilia o paciente no uso da medicação. A característica *Reminder* é composta por eventos, e cada evento contém uma condição e uma ação para ser acionado. Por exemplo, o recado para tomar remédios no horário e na quantidade corretos é um evento. A condição é o horário estabelecido pelo profissional de saúde, e a ação é a informação do recado em um dispositivo mais próximo e/ou de maior prioridade para que o paciente possa visualizar a mesma. Por fim, o paciente deve confirmar que já seguiu as recomendações.

Em outra situação, a aplicação informa ao paciente que deve tomar seu remédio pela manhã. A aplicação, por meio de entrada inferida, utiliza a característica *Reminder* para informar um recado. Por sua vez, a característica *Reminder* necessita de um visor para lançar a informação. Por padrão, a característica TV já está ativada no grupo de características, porém, a aplicação procura por outros dispositivos de *Display Device* próximos ao idoso e encontra somente a TV do quarto. Portanto, a característica dinâmica TV é a ideal para este propósito (Figuras 4.23(a) e 4.23(b)), fazendo com que a condição seja satisfeita e a mensagem de notificação lançada na tela do televisor.



(a) *Paciente idoso recebe informações no quarto pela TV sobre a prescrição medicamentosa*

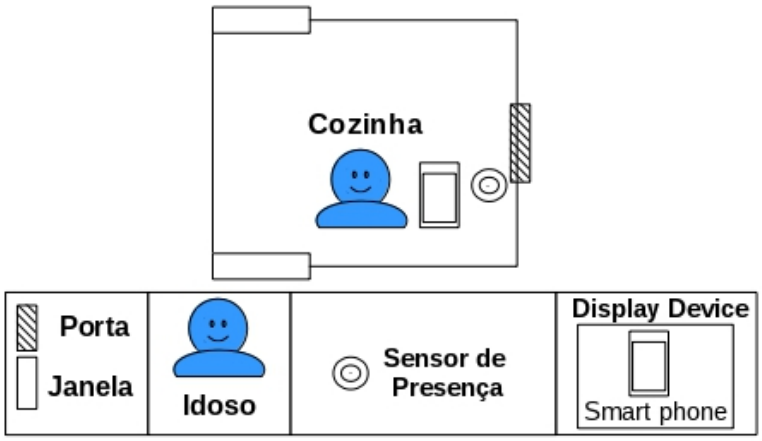


(b) *Estado do MCC-D quando a TV é ativada*

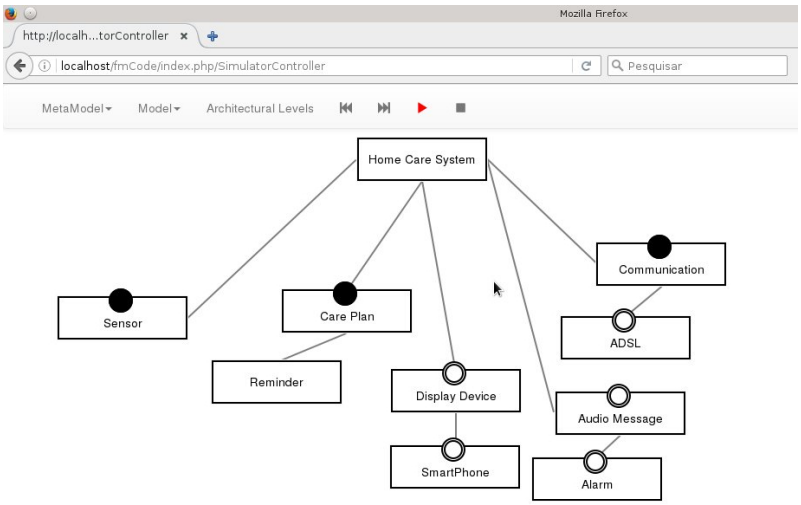
Figura 4.23: *Mudanças de estados de acordo com o local onde o paciente se encontra*

Quando o paciente se dirige à cozinha, os sensores de presença da aplicação percebem que o mesmo mudou de local. Na cozinha, a aplicação reconhece que há um dispositivo mais próximo ao paciente, um *smart phone*. Devido a esta mudança de contexto, a mensagem para o *smart phone* é ativada e o evento de mensagem para a TV do quarto é desativado (Figuras 4.24(a) e 4.24(b)). O paciente ao ver a mensagem, notifica no *smart phone* que já seguiu as recomendações (entrada explícita) e o *Reminder* não mais notificará esta mensagem. Este evento de tomar o(s) remédio(s) pela manhã pode se repetir, caso a prescrição médica assim exija. O código 4.3 mostra o *script*

para simular esse cenário de notificação sensível ao contexto pela ferramenta de modelagem.



(a) *Paciente idoso confirma que já tomou o remédio*



(b) *A característica TV é desativada e o smart phone é ativado*

Figura 4.24: *Mudanças do MCC-D sensível ao contexto baseado na Fgiura 4.15*

Código 4.3: *Script para simular exemplos de notificação sensível ao contexto*

```
1 //*******************************************************
2 // Simulação: Notificação sensível ao contexto
3 //
4 // As características estáticas (mandatórias e opcionais)
5 // são ativadas em tempo de implantação. Quando a aplicação
6 // é executada, as características dinâmicas no script serão
7 // ativadas.
8 //*******************************************************
9
```

```
10 //
11 // Derivação de um produto em tempo de implantação.
12 //
13
14 add f28; // Home Care System - característica raiz
15 add f29; // Sensor - característica mandatória
16 add f30; // Care Plan - característica mandatória
17
18 add f38;     // Display Device - característica dinâmica
19 add fg39;    // Grupo de Display Device - grupo dinâmico
20 f38.enable; // define como ativo a característica Display Device
21 f40.enable; // TV - característica dinâmica
22 add f44;     // Adiciona o Reminder. A regra de composição requires
23              // exige que o Display Device esteja selecionado e ativo
                   primeiro.
24
25 add f50;     // Communication - mandatória
26 add f51;     // Grupo de Communication - grupo dinâmico
27 f52.enable; // Define como ativo a característica ADSL
28
29 //
30 // Derivação em tempo de execução
31 //
32
33 // Como a TV já está ativada, ela continua ativada quando
34 // o idoso está no quarto
35
36 f40.disable; // A TV é desativada, pois não está presente na cozinha
37 f42.enable;  // A aplicação ativa um visor mais próximo
38              // ao paciente: smart phone
```

4.3.4 Notificação por Mensagens de Áudio Sensível ao Contexto

A notificação em formato de áudio pode ser útil para situações em que não é possível visualizar uma informação por meio do grupo de características *Dispaly Device*. As mensagens em áudio podem ser úteis também para os deficientes visuais. Este tipo de característica pode ser instalada em um dispositivo de som ou simplesmente em um alarme para notificar o paciente de que ele deve se conduzir até um outro dispositivo para ouvir ou visualizar o recado.

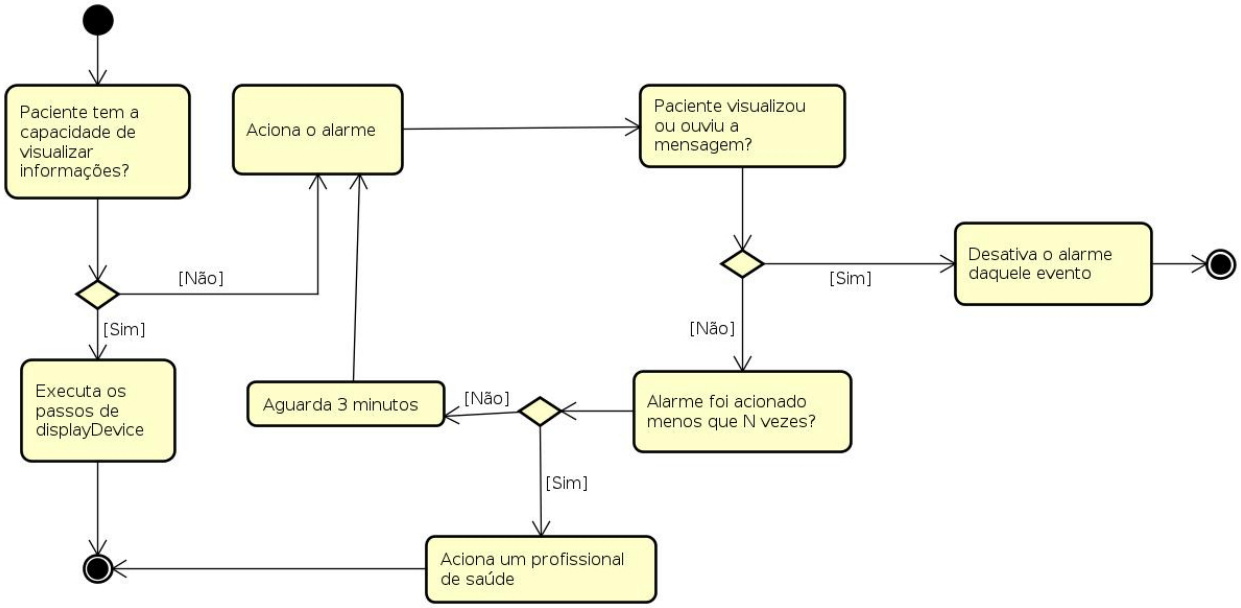


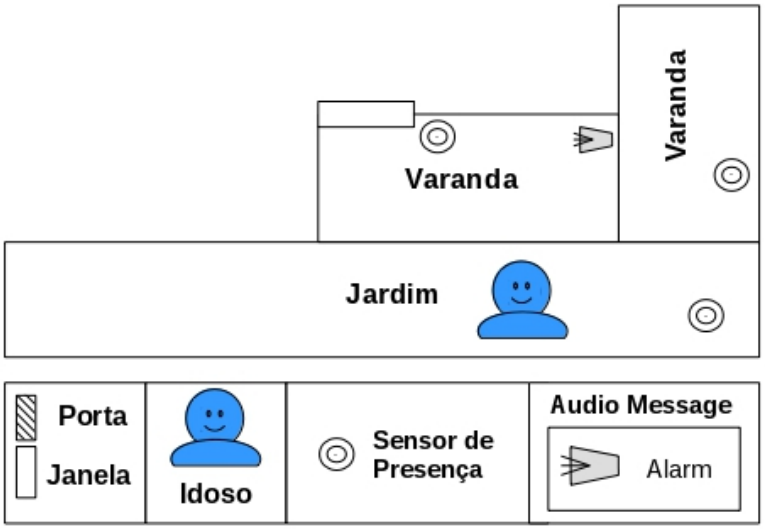
Figura 4.25: *Diagrama de atividades de mensagem de áudio*

A Figura 4.25 representa os estados de notificação via áudio para o paciente de acordo com o MC-D na Figura 4.14. A aplicação aciona o alarme quando percebe que não há outro dispositivo próximo ao paciente. O alarme é acionado por uma certa quantidade e se o paciente não responder a aplicação aciona um profissional médico para tomar as providências necessárias. No MC-D há um grupo de características de cardinalidade com no mínimo uma característica e com no máximo três características para a escolha. Conforme a figura, as características são as seguintes: *Stereo system* (aparelho de som estéreo), som do *Smart phone* e alarme sonoro (*Alarm*).

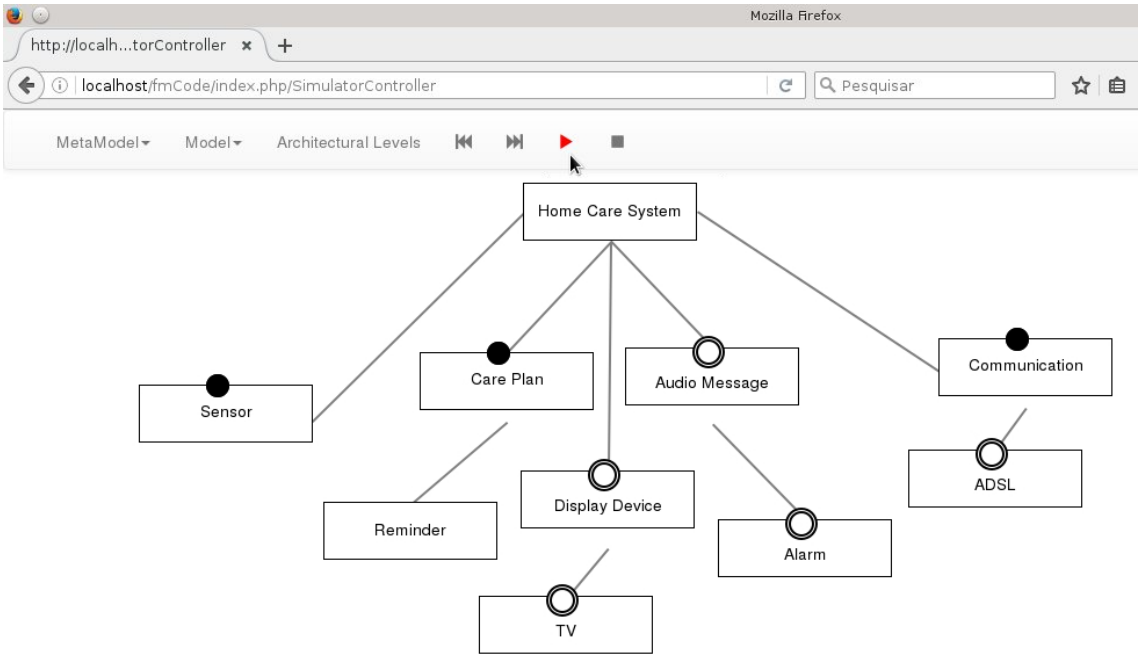
Quando a mensagem em formato de áudio é acionada por meio de um dispositivo de áudio, de maior prioridade, a mesma também pode ser vista mediante um visor, caso esteja disponível em formato visual. A aplicação faz uma consulta de contextos, neste caso, no grupo de características dinâmicas *Audio Message* presente no MC-D. Esta consulta de contextos permite verificar quais características estão presentes e qual é a maior prioridade para a seleção. A consulta de ativação de maior prioridade é considerada como um exemplo de entrada inferida da aplicação.

Por exemplo, o paciente encontra-se na sala de sua casa. A aplicação conclui que a característica (*Smart phone*) é a ideal para este fim e a mesma é ativada no MCC-D. Por sua vez, o paciente entende que há uma nova mensagem no *Smart phone* e decide ligar o *Stereo System* para ouvir o áudio. Automaticamente a aplicação percebe a entrada explícita e desativa o evento

na característica *Smart phone* e em seguida ativa outro evento para o *Stereo system*. Por fim, o usuário confirma o interesse de ouvir o recado e a aplicação executa a ação para transmitir o áudio (Figura 4.27). O código 4.4 detalha o *script* para simular esse cenário de notificação por mensagens de áudio.



(a) *Desenho onde o paciente é notificado por meio do alarme sonoro*



(b) *Característica Alarm recebe um evento de recado da característica Reminder*

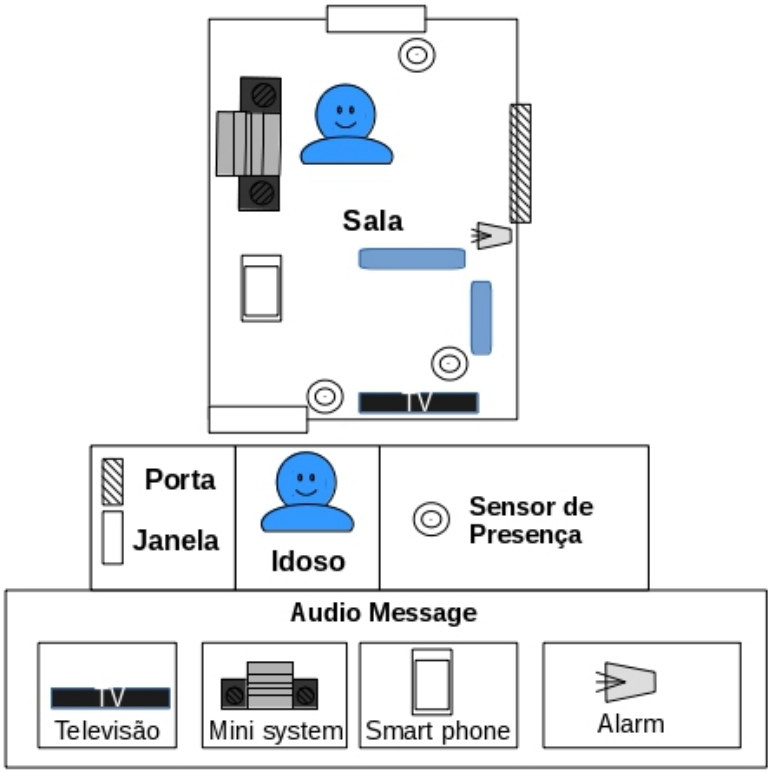
Figura 4.26: *Mudanças de estados de acordo com o local onde o paciente se encontra*

A característica de maior prioridade do grupo de *Audio message* é o

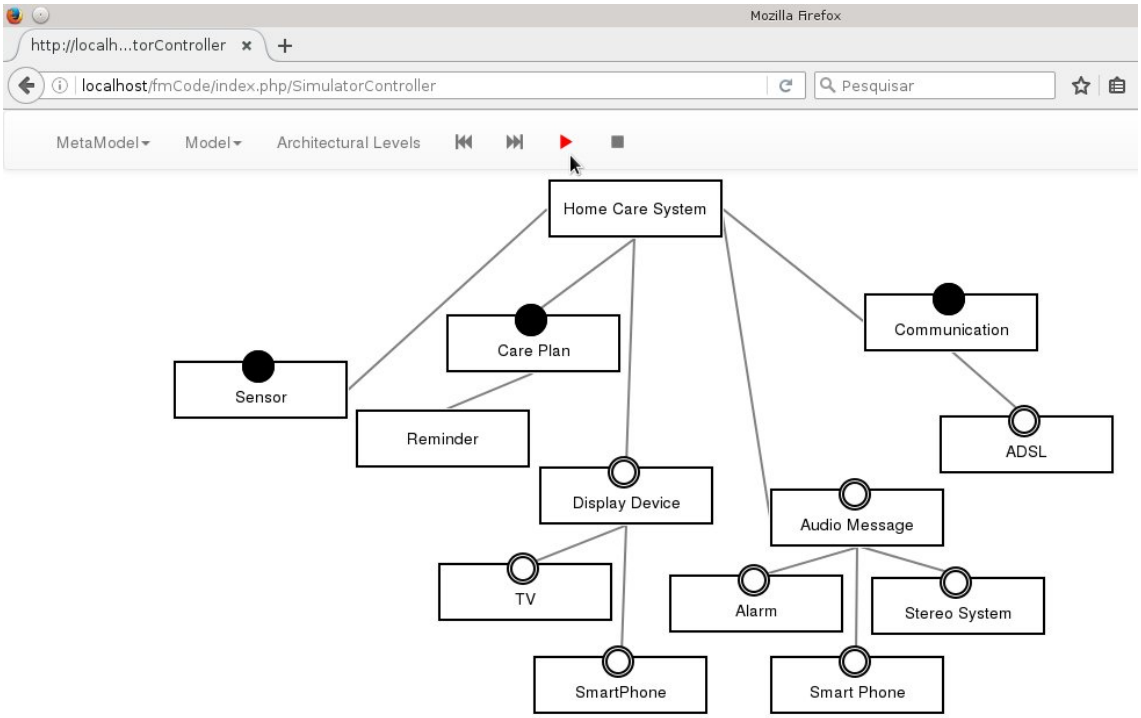
alarme sonoro, pois ele tem maior alcance. Em outro momento, o paciente encontra-se no jardim de sua casa e não tem nenhum dispositivo eletrônico próximo a ele. Neste momento, os sensores de presença informam na aplicação que ele está no jardim e existe um evento de recado importante para o paciente. O dispositivo mais próximo é o alarme sonoro e, e neste caso, a aplicação aciona o evento alarme, de som crescente, como, por exemplo, uma música, que começa com um volume baixo e segue aumentando de forma que não incomode outras pessoas (veja o desenho na Figura 4.26(a) e o MCC-D na Figura 4.26(b)).

O paciente percebe que o alarme de recados está acionado e vai até a sala onde encontram-se os dispositivos de *Display device* e de *Audio message*. A mensagem pode ser ouvida ou visualizada conforme a escolha do paciente (Figuras 4.27(a) e 4.27(b)). Tal mensagem é um lembrete de confirmação de que há uma consulta mensal de rotina mara cada duas horas depois do horário atual.

Este cenário é interessante para garantir que um determinado recado importante seja informado, seja por meio visual ou por áudio. A aplicação reconhece por meio de sensores qual local em que o paciente se encontra. Além disto, é capaz de realizar mudanças nas características em tempo de execução conforme a disponibilidade de contextos no ambiente. No próximo cenário, é abordado um exemplo de simulação de controle de temperatura para uma *homecare*.



(a) *Paciente escolhe o Stereo System para ouvir a mensagem*



(b) *O evento do alarme é desativado para acionar o Stereo System no MCC-D*

Figura 4.27: *Mudança de contexto do grupo de características Audio message*

Código 4.4: *Script para simular exemplos de notificação por mensagens de áudio*

```
1  //*****************************************************
2  // Simulação: Notificação por mensagens de áudio
3  //
4  // As características estáticas (mandatórias e opcionais)
5  // são ativadas em tempo de implantação. Quando a aplicação
6  // é executada, as características dinâmicas no script serão
7  // ativadas.
8  //*****************************************************
9
10 //
11 // Derivação de um produto em tempo de implantação.
12 //
13
14 add f28; // Home Care System - característica raiz
15 add f29; // Sensor - característica mandatória
16 add f30; // Care Plan - característica mandatória
17
18 add f38;     // Display Device - característica dinâmica
19 add fg39;    // Grupo de Display Device - grupo dinâmico
20 f38.enable; // define como ativo a característica Display Device
21 f40.enable; // TV - característica dinâmica
22 add f44;     // Adiciona o Reminder. A regra de composição requires
23              // exige que o Display Device esteja selecionado e ativo
                   primeiro.
24
25 add f45;     // Audio Message - característica dinâmica
26 add fg46;    // Grupo de Audio Message - grupo dinâmico
27 f45.enable; // define como ativo a característica Audio Message
28 f47.enable; // Define como ativo a característica Alarm
29
30 add f50;     // Communication - mandatória
31 add f51;     // Grupo de Communication - grupo dinâmico
32 f52.enable; // Define como ativo a característica ADSL
33
34 //
35 // Derivação em tempo de execução
36 //
37
38 // A aplicação ativa as novas características de audio message
39 // e de display no ambiente
40
41 f48.enable;  // A aplicação ativa a característica de Smart phone (áudio)
42 f49.enable;  // A aplicação ativa a característica Stereo System (áudio)
43 f42.enable;  // A aplicação ativa a característica Smart phone (visor)
```

```
44
45 // o alarm fica ativado em toda casa.
```

4.3.5 Controle de Temperatura e Umidade Sensível ao Contexto

O controle da temperatura e de umidade é importante para a saúde de idosos e de crianças, pois eles podem favorecer o surgimento de doenças ou alergias. Além disso, o controle de temperatura e umidade garantem um melhor conforto para o usuário na casa.

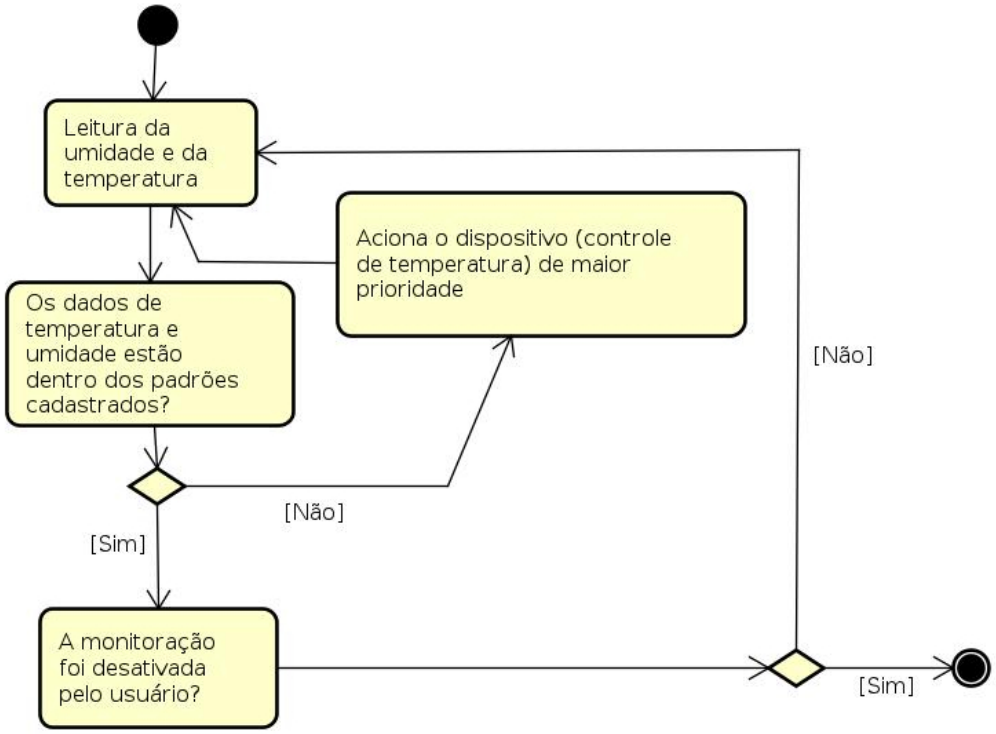
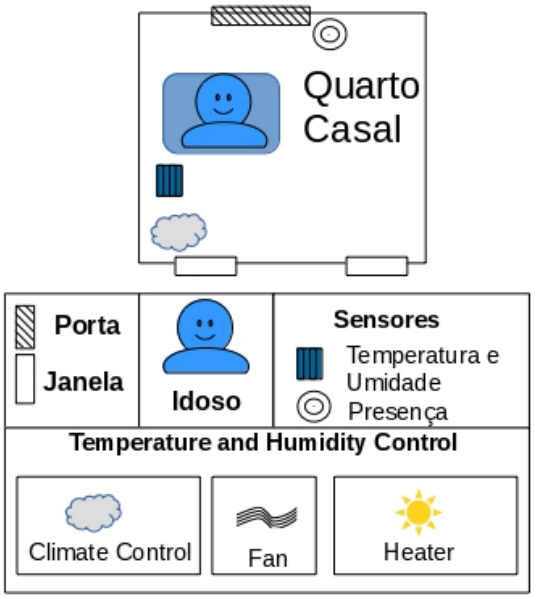


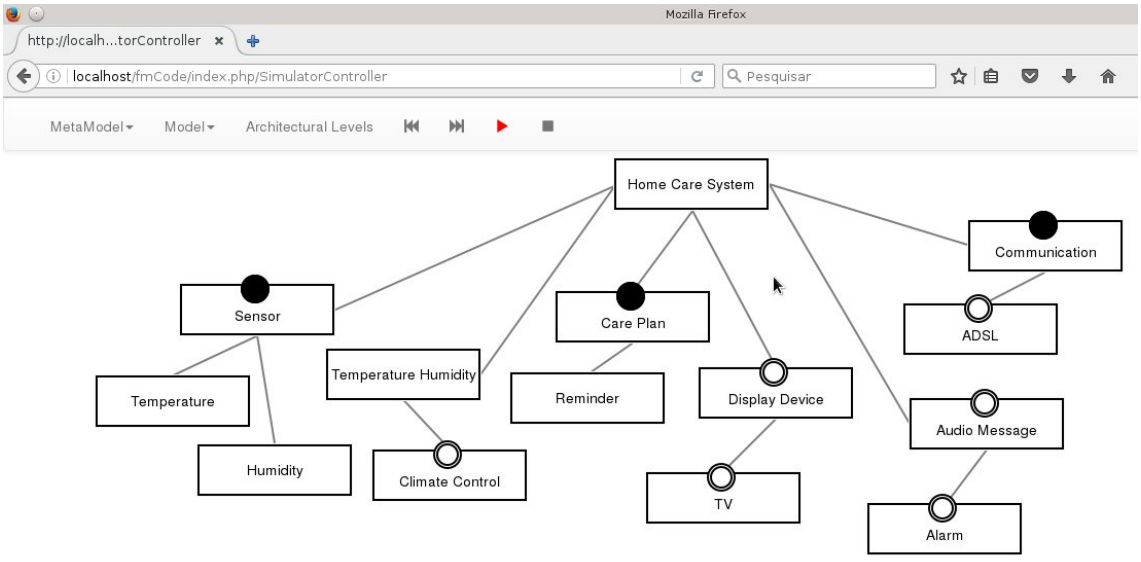
Figura 4.28: *Diagrama de atividades de controle de temperatura e umidade*

A característica de controle de temperatura tem um sensor de umidade e de temperatura e um grupo de características composto por: ventilador, climatizador, ar-condicionado e aquecedor, conforme o MC-D da Figura 4.14. O controle de temperatura percorre os passos de acordo com o diagrama de atividades da Figura 4.28, a aplicação realiza a leitura de dados de umidade e de temperatura, por meio de sensores. Caso a temperatura e/ou umidade estão diferentes dos valores cadastrados, a aplicação deve acionar os atuadores para atingir o valor programado. Porém, o paciente também pode desativar este controle, por meio de entradas explícitas da aplicação, caso

tenha interesse, como, por exemplo, utilizando o controle remoto. A temperatura e a umidade variam de acordo com horário e as estações do ano. Portanto, o paciente, um técnico ou um profissional de saúde podem configurar qual a temperatura ideal. Esta configuração inicial é necessária para as entradas inferidas e implícitas funcionarem sem a necessidade de se configurar manualmente a temperatura e/ou a umidade.



(a) *A aplicação ativa o climatizador no quarto*

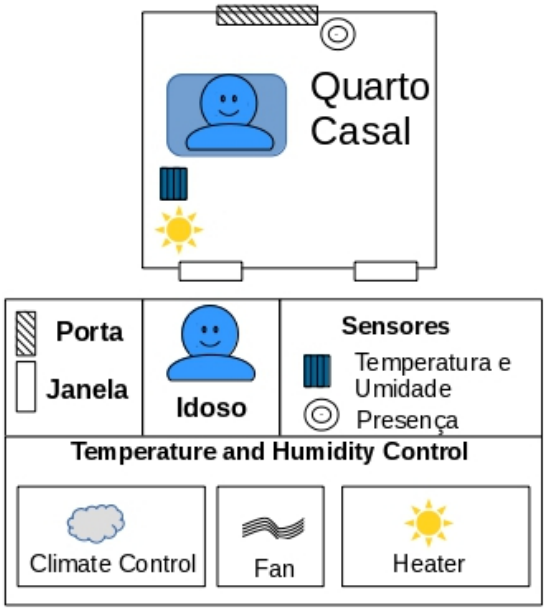


(b) *A característica Climate Control é ativada no MCC-D*

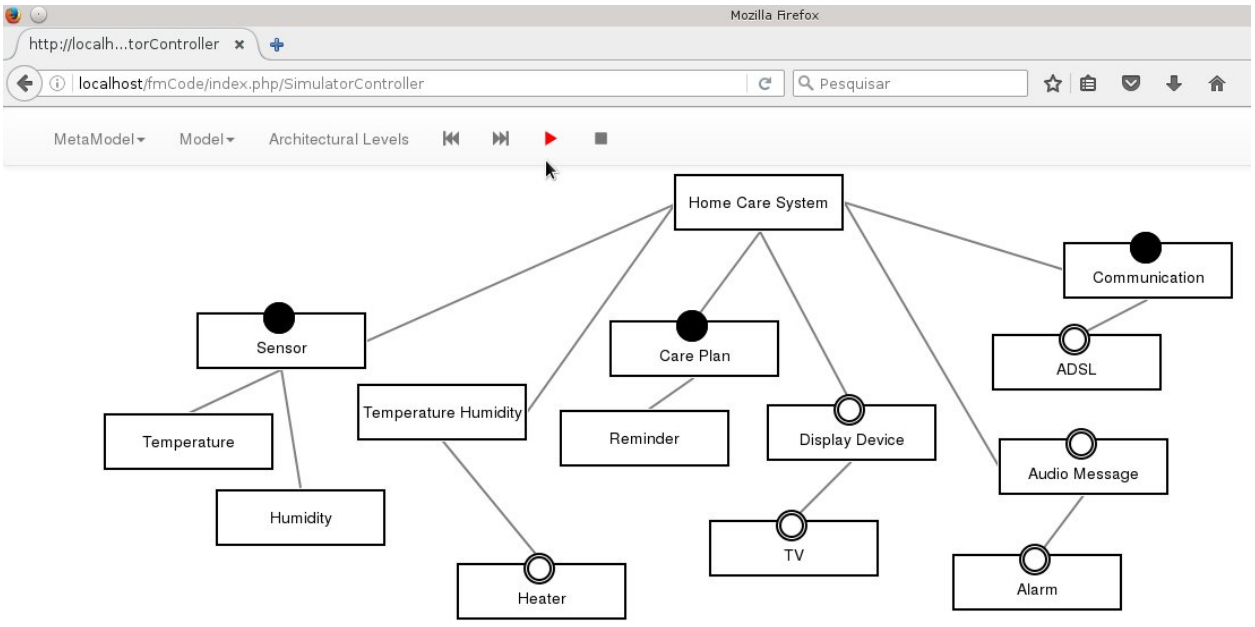
Figura 4.29: *Controle de temperatura e umidade pela característica Climate Control*

Por exemplo, quando o tempo estiver muito seco, os sensores de

temperatura e umidade e o sensor de presença informam o estado daquele local onde o paciente se encontra. (Figuras 4.29(a) e 4.29(b)). O código 4.5 detalha o *script* para simular esse cenário de controle de temperatura e umidade.



(a) *A aplicação ativa o aquecedor no quarto*



(b) *A característica Heater é ativada no MCC-D*

Figura 4.30: *Controle de temperatura pela característica Heater*

Código 4.5: *Script para simular exemplos de controle de temperatura e umidade*

```
//
// Derivação de um produto em tempo de implantação.
//

add f28; // Home Care System - característica raiz
add f29; // Sensor - característica mandatória
add f30; // Care Plan - característica mandatória

add f38;     // Display Device - característica dinâmica
add fg39;    // Grupo de Display Device - grupo dinâmico
f38.enable; // Define como ativo a característica Display Device
f40.enable; // TV - característica dinâmica
add f44;     // Adiciona o Reminder. A regra de composição requires
             // exige que o Display Device esteja selecionado
             // e ativo primeiro.

add f33;     // Temperature Humidity - característica estática opcional
add fg34;    // Grupo de Temperature Humidity - grupo dinâmico
f35.enable; // Define como ativo a característica Climate Control

add f31;    // Temperature - característica estática opcional
add f32;    // Humidity - característica estática opcional

add f45;     // Audio Message - característica dinâmica
add fg46;    // Grupo de Audio Message - grupo dinâmico
f45.enable; // Define como ativo a característica Audio Message
f47.enable; // Define como ativo a característica Alarm

add f50;     // Communication - mandatória
add f51;     // Grupo de Communication - grupo dinâmico
f52.enable; // Define como ativo a característica ADSL

//
// Derivação em tempo de execução
//

// A aplicação muda o estado das características dinâmicas
// para controlar a temperatura ambiente:

f35.disable;  // Desativa característica Climate Control
f37.enable;  // Ativa como ativo a característica Heater
```

4.4 Considerações Finais

A ferramenta proposta permite modelar MC, MCC, MC-D, MCC-D e realizar simulações de mudanças de estados de características estáticas ou dinâmicas. As simulações foram baseadas em cenários de uma aplicação *homecare*.

As propostas de ferramentas para modelagem de características são úteis para LPS, porém não lidam diretamente com LPSD. A ferramenta desta dissertação modela graficamente MC-D e MCC-D para uma LPSD diferenciando características estáticas e dinâmicas.

Além das propostas de representação, os metamodelos propostos formalizam os modelos de MC-D MCC-D. A proposta de ferramenta baseada na *web* não exige instalação e nem atualização de sistema operacional hospedeiro para utilizá-la em seus clientes, exigindo apenas acesso à Internet e um navegador.

Por fim, os trabalhos relacionados apresentados nesta Seção servem para conhecer o estado de desenvolvimento de ferramentas de modelagem voltadas para LPS.

CAPÍTULO 5

Conclusão

As propostas de representação de Modelos de Características Dinâmicas (MC-D) e de Modelos de Configuração de Características Dinâmicas (MCC-D) formalizadas em metamodelos podem colaborar com a natureza dinâmica das LPSDs. Os metamodelos para MC-D e MCC-D definem em alto nível como estes modelos devem se comportar em tempo de implantação e em tempo de execução.

A proposta de representação de MC-D permite representar características dinâmicas em nível de domínio e a proposta de representação de MCC-D, permite representar características dinâmicas em nível de aplicação. O simbolismo proposto para as características dinâmicas se diferencia daquele usado para as características de uma LPS, inclusive em relação à proposta de representação de grupos de características estáticas e dinâmicas para MC-D, garantindo a identificação das mesmas.

A ferramenta *web* proposta permite a interpretação de metamodelos Ecore e a modelagem de MC-D e MCC-D, por meio do módulo EcoreController. Esta ferramenta valida, mediante o módulo FValidator, regras de composição não presentes nos metamodelos ou nos modelos propostos. A criação de MCC-D foi simplificada por meio da criação de uma Linguagem de Domínio Específico (DSL), voltada para ativar e desativar características em tempo de execução ou em tempo de implantação (interpretada pelo módulo FCInterpreter). Esta linguagem é utilizada também para simular mudanças de estados das características estáticas ou dinâmicas. O módulo Simulator realiza estas simulações de cenários, incluindo aqueles relacionados a aplicações de *homecare*, e servem para exemplificar os metamodelos e modelos propostos. Estas simulações permitiram ainda exemplificar aplicações sensíveis ao contexto, envolvendo mudanças em tempo de implantação e adaptações em tempo de execução.

As pesquisas sobre requisitos em tempo de execução, sistemas autoadaptativos, Linha de Produto de Software Dinâmica e aplicações ubíquas pos-

sibilitaram a compreensão da proposta de uma abordagem para metamodelos, modelos e da própria ferramenta.

Em relação especificamente ao cenário escolhido de monitoramento da saúde em casa (*homecare*), vale a pena ressaltar que a Agência Nacional de Vigilância Sanitária, vinculada ao Ministério da Saúde, criou uma resolução de em janeiro de 2006 (Nº 11) [9] que determina os requisitos mínimos de segurança para os "Serviços de Atenção Domiciliar nas modalidades de Assistência e Internação Domiciliar". Tais requisitos não foram considerados nesse trabalho, no entanto, podem ser úteis ao se modelar uma aplicação dessa classe.

Ao longo da pesquisa surgiram oportunidades de complementação na pesquisa. Tais oportunidades são apresentadas a seguir.

A modelagem de MC-D e MCC-D necessita de um metamodelo para modelar aplicações sensíveis ao contexto com a integração a uma LPSD. A representação auxiliará nos estados de mudanças de contexto em relação à própria aplicação.

O EcoreController foi desenvolvido sobre uma plataforma baseada na web em padrão MVC. Esta abordagem abre uma oportunidade para o desenvolvimento de um *web-service* para interpretar metamodelos em Ecore, o que pode servir como colaboração para a ferramenta Eclipse e para o projeto EMF. Por meio de um *web-service* não será necessário o desenvolvimento de uma aplicação para interpretação de metamodelos na web.

Para uma melhor avaliação da ferramenta é importante realizar testes em nível M0, conforme o padrão OMG. Tais testes poderiam envolver protótipos com dispositivos embarcados, como, por exemplo, Arduino e Raspberry para simular adaptações estruturais da aplicação e mudanças de contexto.

A representação de MC-D e de MCC-D precisa aprimorar as suas representações gráficas, presentes na camada View. Por exemplo, como estas camadas não foram desenvolvidas empregando técnicas de *design* responsivo, exige-se novas propostas de representação em dispositivos diferentes, como, por exemplo, *smart phones* e *tablets*.

A probabilidade matemática pode auxiliar ao se estipular quantas derivações podem gerar um MC-D. E pode-se prever quais efeitos seus produtos podem oferecer nos níveis M1 e M0. Por exemplo, este estudo pode prever quais requisitos mínimos de hardware e de software que a aplicação precisaria para funcionar corretamente, isto é, para maximizar o uso de recursos da aplicação. Esta maximização pode gerar economia na contratação de serviços, energia e durabilidade da aplicação.

Referências Bibliográficas

[1] A BOWD, G. D.; D EY, A. K.; B ROWN, P. J.; D AVIES, N.; S MITH, M.; S TEGGLES, P. **Towards a better understanding of context and context-awareness.** *Springer*, 1999.

[2] AHO, A.; ULLMAN, J. **Compiladores - princípios, técnicas e ferramentas**. chapter 3, p. 38–45. LTC, 1995.

[3] Almeida, A.; Cavalcante, E.; Batista, T.; Cacho, N.; Lopes, F. **A component-based adaptation approach for multi-cloud applications**. In: *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, p. 49–54. IEEE, 2014.

[4] Andersson, J.; Baresi, L.; Bencomo, N.; de Lemos, R.; Gorla, A.; Inverardi, P.; Vogel, T. **Software engineering processes for self-adaptive systems**. In: *Software Engineering for Self-Adaptive Systems II*, p. 51–75. Springer, 2013.

[5] Atkinson, C.; Kuhne, T. **Model-driven development: a metamodeling foundation**. *Software, IEEE*, 20(5):36–41, 2003.

[6] BARESI, L.; QUINTON, C. **Dynamically evolving the structural variability of dynamic software product lines**. In: *10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, p. 7, 2015.

[7] BENCOMO, N.; BELAGGOUN, A. **Supporting decision-making for self-adaptive systems: from goal models to dynamic decision networks**. In: *Requirements Engineering: Foundation for Software Quality*, p. 221–236. Springer, 2013.

[8] Bencomo, N.; Hallsteinsen, S.; Santana, d. A. E. **A view of the dynamic software product line landscape**. *Computer*, 45(10):36–41, 2012.

[9] BRASIL. **Resolução RDC Nº11, de 26 de janeiro de 2006. Dispõe sobre o regulamento técnico de funcionamento de serviços que prestam atenção domiciliar.** *Diário Oficial da União*, 2006.

[10] Bresciani, P.; Perini, A.; Giorgini, P.; Giunchiglia, F.; Mylopoulos, J. **Tropos: An agent-oriented software development methodology**. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[11] Capilla, R.; Bosch, J.; Trinidad, P.; Ruiz-Cortés, A.; Hinchey, M. **An overview of dynamic software product line architectures and techniques: Observations from research and industry**. *Journal of Systems and Software*, 91:3–23, 2014.

[12] Carvalho, S. T. **Modelagem de linha de produto de software dinâmica para aplicações ubíquas**. *Tese de Doutorado. UFF Niterói*, 2013.

[13] Carvalho, S. T.; Copetti, A.; Loques Filho, O. G. **Sistema de computação ubíqua na assistência domiciliar à saúde**. *Journal Of Health Informatics*, 3(2), 2011.

[14] Carvalho, S. T.; Murta, L.; Loques, O. **Variabilities as first-class elements in product line architectures of homecare systems**. p. 33–39, 2012.

[15] Cetina, C.; Giner, P.; Fons, J.; Pelechano, V. **Prototyping dynamic software product lines to evaluate run-time reconfigurations**. *Science of Computer Programming*, 78(12):2399–2413, 2013.

[16] CHENG, B. H.; De LEMOS, R.; GIESE, H.; INVERARDI, P.; MAGEE, J.; ANDERSSON, J.; BECKER, B.; BENCOMO, N.; BRUN, Y.; CUKIC, B.; Serugendo, G. D. M.; DUSTDAR, S.; FINKELSTEIN, A.; GACEK, C.; GEIHS, K.; GRASSI, V.; KARSAI, G.; KIENLE, H. M.; KRAMER, J.; LITOIU, M.; MALEK, S.; MIRANDOLA, R.; MÜLLER, H. A.; PARK, S.; SHAW, M.; TICHY, M.; TIVOLI, M.; WEYNS, D.; WHITTLE, J. **Software engineering for self-adaptive systems: A research roadmap**. 2009.

[17] Da Silva, J. R. F.; da Silva, F. A. P.; Do Nascimento, L. M.; Martins, D. A.; Garcia, V. C. **The dynamic aspects of product derivation in dspl: A systematic literature review**. In: *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*, p. 466–473. IEEE, 2013.

[18] DE LEMOS, R.; GIESE, H.; MÜLLER, H. A.; SHAW, M.; ANDERSSON, J.; LITOIU, M.; SCHMERL, B.; TAMURA, G.; VILLEGAS, N. M.; VOGEL, T.; OTHERS. **Software engineering for self-adaptive systems: A second research roadmap**. In: *Software Engineering for Self-Adaptive Systems II*, p. 1–32. Springer, 2013.

[19] Dey, A. K.; Abowd, G. D.; Salber, D. **A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications**. *Human-computer interaction*, 16(2):97–166, 2001.

[20] Fernandes, P.; Werner, C.; Murta, L. G. P. **Feature modeling for context-aware software product lines.** p. 758–763, 2008.

[21] Fernandes, P.; Werner, C.; Teixeira, E. **An approach for feature modeling of context-aware software product line.** *J. UCS*, 17(5):807–829, 2011.

[22] Foote, B.; Rohnert, H.; Harrison, N. **Pattern languages of program design 4**. Addison-Wesley Longman Publishing Co., Inc., 1999.

[23] Foundation, T. E. **Package org.eclipse.emf.ecore**. http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html. Acessado em: 11 de maio de 2016.

[24] Hallsteinsen, S.; Hinchey, M.; Park, S.; Schmid, K. **Dynamic software product lines.** p. 93–95, 2008.

[25] Hallsteinsen, S.; Stav, E.; Solberg, A.; Floch, J. **Using product line techniques to build adaptive systems**. p. 10–pp, 2006.

[26] Hinchey, M.; Park, S.; Schmid, K. **Building dynamic software product lines**. *Computer*, (10):22–26, 2012.

[27] Kamsties, E.; Kneer, F.; Voelter, M.; Igel, B.; Kolb, B. **Feedback-aware requirements documents for smart devices**. In: *Requirements Engineering: Foundation for Software Quality*, p. 119–134. Springer, 2014.

[28] Kang, K. C.; Cohen, S. G.; Hess, J. A.; Novak, W. E.; Peterson, A. S. **Feature-oriented domain analysis (foda) feasibility study**. Technical report, DTIC Document, 1990.

[29] McGee-Lennon, M. R. **Requirements engineering for home care technology**. p. 1439–1442, 2008.

[30] Mendonca, M.; Branco, M.; Cowan, D. **Splot: software product lines online tools**. In: *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, p. 761–762. ACM, 2009.

[31] Murguzur, A.; Capilla, R.; Trujillo, S.; Ortiz, O.; Lopez-Herrejon, R. E. **Context variability modeling for runtime configuration of service-based dynamic software product lines**. In: *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools-Volume 2*, p. 2–9. ACM, 2014.

[32] Northrop, L. M. **Sei's software product line tenets**. *IEEE software*, (4):32–40, 2002.

[33] OMG. **Meta object facility (mof) specification version 1.4, april 2002 document number: formal/2004-04-03**. *Object Management Group (OMG)*, 2002.

[34] Oreizy, P.; Gorlick, M. M.; Taylor, R. N.; Heimbigner, D.; Johnson, G.; Medvidovic, N.; Wolf, A. L. **An architecture-based approach to self-adaptive software**. *IEEE Intelligent systems*, 3:54–62, 1999.

[35] Pressman, R. S. **Engenharia de software**. McGraw Hill Brasil, 2011.

[36] Sawyer, P.; Bencomo, N.; Whittle, J.; Letier, E.; Finkelstein, A. **Requirements-aware systems: A research agenda for re for self-adaptive systems**. In: *Requirements Engineering Conference (RE), 2010 18th IEEE International*, p. 95–103. IEEE, 2010.

[37] Schilit, B.; Adams, N.; Want, R. **Context-aware computing applications**. In: *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, p. 85–90. IEEE, 1994.

[38] Sommerville, I. **Engenharia de software**. São Paulo, Addison Wesley, 2010.

[39] Souza, V. E. S.; Lapouchnian, A.; Mylopoulos, J. **(Requirement) evolution requirements for adaptive systems**. In: *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, p. 155–164. IEEE Press, 2012.

[40] Steinberg, D.; Budinsky, F.; Merks, E.; Paternostro, M. **EMF: eclipse modeling framework**. Pearson Education, 2008.

[41] Trinidad, P.; Cortés, A. R.; Peña, J.; Benavides, D. **Mapping feature models onto component models to build dynamic software product lines.** In: *SPLC (2)*, p. 51–56, 2007.

[42] Van Deursen, A.; Klint, P.; Visser, J. **Domain-specific languages: An annotated bibliography.** *Sigplan Notices*, 35(6):26–36, 2000.

[43] Vieira, V.; Tedesco, P.; Salgado, A. C. **Modelos e processos para o desenvolvimento de sistemas sensíveis ao contexto**. *André Ponce de Leon F. de Carvalho, Tomasz Kowaltowski.(Org.). Jornadas de Atualização em Informática*, p. 381–431, 2009.

[44] Weiser, M. **The computer for the 21st century**. *Scientific american*, 265(3):94–104, 1991.

[45] Yang, Z.; Li, Z.; Jin, Z.; Chen, Y. **A systematic literature review of requirements modeling and analysis for self-adaptive systems**. In: *Requirements Engineering: Foundation for Software Quality*, p. 55–71. Springer, 2014.

APÊNDICE A

Metamodelo para Representação de MC-D e MCC-D

O código A.1 em XML contém a proposta de Metamodelo para Representação de MC-Ds e MCC-Ds, desenvolvido por meio da ferramenta EMF.

Código A.1: *Código XML Ecore para representar características dinâmicas*

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="dsl" nsURI="
          http://www.example.org/dsl" nsPrefix="dsl">
4   <eClassifiers xsi:type="ecore:EClass" name="Feature">
5     <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="
          ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
6     <eStructuralFeatures xsi:type="ecore:EAttribute" name="opt" eType="#//
          Optional"
7         defaultValueLiteral="mandatory"/>
8     <eStructuralFeatures xsi:type="ecore:EAttribute" name="bgp" eType="#//
          BindingTime"
9         defaultValueLiteral="static"/>
10    <eStructuralFeatures xsi:type="ecore:EReference" name="requires"
          upperBound="-1"
11        eType="#//Feature" eOpposite="#//Feature/isRequiredBy"/>
12    <eStructuralFeatures xsi:type="ecore:EReference" name="isRequiredBy"
          upperBound="-1"
13        eType="#//Feature" eOpposite="#//Feature/requires"/>
14    <eStructuralFeatures xsi:type="ecore:EReference" name="exlcudes"
          upperBound="-1"
15        eType="#//Feature" eOpposite="#//Feature/isExcludedBy"/>
16    <eStructuralFeatures xsi:type="ecore:EReference" name="isExcludedBy"
          upperBound="-1"
17        eType="#//Feature" eOpposite="#//Feature/exlcudes"/>
18    <eStructuralFeatures xsi:type="ecore:EReference" name="
          dynamicFatureGroup" upperBound="-1"
```

```
19          eType="#//FeatureGroup" containment="true"/>
20      <eStructuralFeatures xsi:type="ecore:EReference" name="isPartOf" eType=
            "#//FeatureGroup"
21          eOpposite="#//FeatureGroup/feature"/>
22      <eStructuralFeatures xsi:type="ecore:EReference" name="subFeature"
            upperBound="-1"
23          eType="#//Feature" containment="true"/>
24      <eStructuralFeatures xsi:type="ecore:EReference" name="
            staticFeatureGroup" upperBound="-1"
25          eType="#//FeatureGroup" containment="true"/>
26    </eClassifiers>
27    <eClassifiers xsi:type="ecore:EClass" name="DynamicFeatureModel"
          eSuperTypes="#//FeatureModel"/>
28    <eClassifiers xsi:type="ecore:EClass" name="FeatureModel">
29      <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="
            ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
30      <eStructuralFeatures xsi:type="ecore:EReference" name="rootFeature"
            lowerBound="1"
31          upperBound="-1" eType="#//Feature" containment="true"/>
32    </eClassifiers>
33    <eClassifiers xsi:type="ecore:EClass" name="FeatureGroup">
34      <eStructuralFeatures xsi:type="ecore:EAttribute" name="minCardinality"
            eType="ecore:EDataType http://www.eclipse.org/emf/2003/XMLType#//
            Integer"/>
35      <eStructuralFeatures xsi:type="ecore:EAttribute" name="maxCardinality"
            eType="ecore:EDataType http://www.eclipse.org/emf/2003/XMLType#//
            Integer"/>
36      <eStructuralFeatures xsi:type="ecore:EReference" name="feature"
            lowerBound="2"
37          upperBound="-1" eType="#//Feature" containment="true" eOpposite="
                #//Feature/isPartOf"/>
38    </eClassifiers>
39    <eClassifiers xsi:type="ecore:EEnum" name="BindingTime">
40      <eLiterals name="static" literal="static"/>
41      <eLiterals name="dynamic" value="1"/>
42    </eClassifiers>
43    <eClassifiers xsi:type="ecore:EEnum" name="Optional">
44      <eLiterals name="optional" literal="LITERAL0"/>
45      <eLiterals name="mandatory" value="1"/>
46    </eClassifiers>
47    <eClassifiers xsi:type="ecore:EClass" name="FeatureConfiguration">
48      <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="
            ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
49      <eStructuralFeatures xsi:type="ecore:EReference" name="
            featureSelectedMandatory"
50          lowerBound="1" upperBound="-1" eType="#//Feature"/>
```

```
51      <eStructuralFeatures xsi:type="ecore:EReference" name="rootFeature"
            lowerBound="1"
52          eType="#//Feature"/>
53      <eStructuralFeatures xsi:type="ecore:EReference" name="
            featureSelectedStatic"
54          upperBound="-1" eType="#//Feature"/>
55    </eClassifiers>
56    <eClassifiers xsi:type="ecore:EClass" name="DynamicFeatureConfiguration"
          eSuperTypes="#//FeatureConfiguration">
57      <eStructuralFeatures xsi:type="ecore:EReference" name="
            dynamicFeatureDisable"
58          upperBound="-1" eType="#//Feature"/>
59      <eStructuralFeatures xsi:type="ecore:EReference" name="
            dynamicFeatureEnable" upperBound="-1"
60          eType="#//Feature"/>
61    </eClassifiers>
62  </ecore:EPackage>
```

APÊNDICE B

Metamodelo para Representação de Características Dinâmicas

O código A.1 em XML contém a proposta de Metamodelo para Representação de MC-D, desenvolvido por meio da ferramenta EMF.

Código B.1: *Código XML Ecore do metamodelo de domínio*

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="domainmodel"
          nsURI="http://www.example.org/domainmodel" nsPrefix="domainmodel">
4   <eClassifiers xsi:type="ecore:EClass" name="Feature">
5     <eStructuralFeatures xsi:type="ecore:EReference" name="subFeature"
          upperBound="-1"
6         eType="#//Feature" containment="true"/>
7     <eStructuralFeatures xsi:type="ecore:EReference" name="isExcludedBy"
          upperBound="-1"
8         eType="#//Feature" eOpposite="#//Feature/excludes"/>
9     <eStructuralFeatures xsi:type="ecore:EReference" name="excludes"
          upperBound="-1"
10        eType="#//Feature" eOpposite="#//Feature/isExcludedBy"/>
11    <eStructuralFeatures xsi:type="ecore:EReference" name="requires"
          upperBound="-1"
12        eType="#//Feature" eOpposite="#//Feature/isRequiredBy"/>
13    <eStructuralFeatures xsi:type="ecore:EReference" name="isRequiredBy"
          upperBound="-1"
14        eType="#//Feature" eOpposite="#//Feature/requires"/>
15    <eStructuralFeatures xsi:type="ecore:EReference" name="
          staticFeatureGroup" upperBound="-1"
16        eType="#//FeatureGroup" containment="true"/>
17    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="
          ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
18    <eStructuralFeatures xsi:type="ecore:EReference" name="isPartOf" eType=
          "#//FeatureGroup"
```

```
19          eOpposite="#//FeatureGroup/feature"/>
20      <eStructuralFeatures xsi:type="ecore:EAttribute" name="opt" eType="#//
            Optional"
21          defaultValueLiteral="mandatory"/>
22      <eStructuralFeatures xsi:type="ecore:EAttribute" name="bgp" eType="#//
            BindingTime"
23          defaultValueLiteral="static"/>
24      <eStructuralFeatures xsi:type="ecore:EReference" name="
            dynamicFatureGroup" upperBound="-1"
25          eType="#//FeatureGroup" containment="true"/>
26    </eClassifiers>
27    <eClassifiers xsi:type="ecore:EClass" name="FeatureGroup">
28      <eStructuralFeatures xsi:type="ecore:EAttribute" name="minCardinality"
            eType="ecore:EDataType http://www.eclipse.org/emf/2003/XMLType#//
            Integer"/>
29      <eStructuralFeatures xsi:type="ecore:EAttribute" name="maxCardinality"
            eType="ecore:EDataType http://www.eclipse.org/emf/2003/XMLType#//
            Integer"/>
30      <eStructuralFeatures xsi:type="ecore:EReference" name="feature"
            lowerBound="2"
31          upperBound="-1" eType="#//Feature" containment="true" eOpposite="
                #//Feature/isPartOf"/>
32    </eClassifiers>
33    <eClassifiers xsi:type="ecore:EEnum" name="BindingTime">
34      <eLiterals name="static"/>
35      <eLiterals name="dynamic" value="1"/>
36    </eClassifiers>
37    <eClassifiers xsi:type="ecore:EEnum" name="Optional">
38      <eLiterals name="optional"/>
39      <eLiterals name="mandatory" value="1"/>
40    </eClassifiers>
41    <eClassifiers xsi:type="ecore:EClass" name="SubFeature" eSuperTypes="#//
          Feature"/>
42    <eClassifiers xsi:type="ecore:EClass" name="DynamicFeatureModel"
          eSuperTypes="#//FeatureModel"/>
43    <eClassifiers xsi:type="ecore:EClass" name="FeatureModel">
44      <eStructuralFeatures xsi:type="ecore:EReference" name="rootFeature"
            lowerBound="1"
45          upperBound="-1" eType="#//Feature" containment="true"/>
46    </eClassifiers>
47  </ecore:EPackage>
```

APÊNDICE C

Metamodelo para Representação de Configuração de Características Dinâmicas

O código A.1 em XML contém a proposta de Metamodelo para Representação de MCC-D, desenvolvido por meio da ferramenta EMF.

Código C.1: *Código XML Ecore do metamodelo de aplicação*

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="
           applicationmodel" nsURI="http://www.example.org/applicationmodel"
4      nsPrefix="applicationmodel">
5    <eClassifiers xsi:type="ecore:EClass" name="Feature">
6      <eStructuralFeatures xsi:type="ecore:EReference" name="isPartOf" eType=
           "#//FeatureGroup"
7          eOpposite="#//FeatureGroup/alternativeFeature"/>
8      <eStructuralFeatures xsi:type="ecore:EReference" name="featureGroup"
           upperBound="-1"
9          eType="#//FeatureGroup" containment="true"/>
10     <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="
           ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
11     <eStructuralFeatures xsi:type="ecore:EAttribute" name="opt" eType="#//
           Optional"
12         defaultValueLiteral="mandatory"/>
13     <eStructuralFeatures xsi:type="ecore:EAttribute" name="bgt" eType="#//
           BindingTime"
14         defaultValueLiteral="static"/>
15     <eStructuralFeatures xsi:type="ecore:EReference" name="subfeature"
           upperBound="-1"
16         eType="#//Feature" containment="true"/>
17   </eClassifiers>
18   <eClassifiers xsi:type="ecore:EClass" name="FeatureGroup">
```

```
19      <eStructuralFeatures xsi:type="ecore:EReference" name="
            alternativeFeature" lowerBound="2"
20          upperBound="-1" eType="#//Feature" containment="true" eOpposite="
                #//Feature/isPartOf"/>
21    </eClassifiers>
22    <eClassifiers xsi:type="ecore:EEnum" name="BindingTime">
23      <eLiterals name="static"/>
24      <eLiterals name="dynamic" value="1"/>
25    </eClassifiers>
26    <eClassifiers xsi:type="ecore:EEnum" name="Optional">
27      <eLiterals name="optional"/>
28      <eLiterals name="mandatory" value="1"/>
29    </eClassifiers>
30    <eClassifiers xsi:type="ecore:EClass" name="FeatureConfiguration">
31      <eStructuralFeatures xsi:type="ecore:EReference" name="rootFeature"
            lowerBound="1"
32          eType="#//Feature"/>
33      <eStructuralFeatures xsi:type="ecore:EReference" name="
            featureSelectedMandatory"
34          lowerBound="1" upperBound="-1" eType="#//Feature"/>
35      <eStructuralFeatures xsi:type="ecore:EReference" name="
            featureSelectedStatic"
36          upperBound="-1" eType="#//Feature"/>
37      <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="
            ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
38    </eClassifiers>
39    <eClassifiers xsi:type="ecore:EClass" name="DynamicFeatureConfiguration"
          eSuperTypes="#//FeatureConfiguration">
40      <eStructuralFeatures xsi:type="ecore:EReference" name="
            dynamicFeatureDisable"
41          upperBound="-1" eType="#//Feature"/>
42      <eStructuralFeatures xsi:type="ecore:EReference" name="
            dynamicFeatureEnable" upperBound="-1"
43          eType="#//Feature"/>
44    </eClassifiers>
45  </ecore:EPackage>
```

APÊNDICE D

Modelo Entidade Relacionamento

A Figura D.1 ilustra o MER da ferramenta proposta. Este modelo contém três tabelas, *model* para armazenar os modelos, *child* para armazenar as `eClassifiers` e os *attributes* para armazenar as `eStructuralFeatures`.

A tabela *model* contém colunas para armazenar algumas informações do metamodelo e do modelo. O metamodelo contém informações de versão do XMI, o endereço do XMI (`mode_xmlns_xmi`), o nome do metamodelo (`mode_xmlns`) e o local onde está salvo o metamodelo (`mode_locale_metamodel`). As configurações do modelo são armazenadas nas seguintes colunas: ponto de início do modelo (`mode_start_point`), o local onde o modelo será salvo, caso ele seja exportado (`mode_locale_model`) e, por fim, se o modelo depende de outro modelo, o mesmo deve armazenar a chave estrangeira em `mode_mode_id`. Caso o modelo há uma dependência de outro modelo, o valor padrão é *null*.

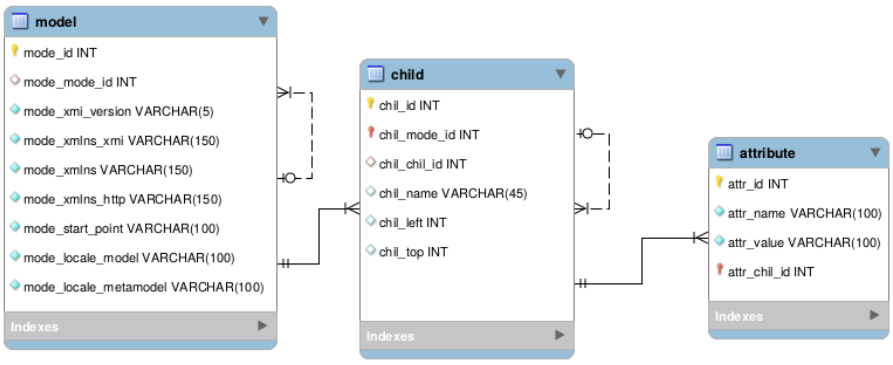


Figura D.1: *MER para persistência de modelos para o EcoreController*

A tabela *child* é uma referência aos *children* no XML presentes no metamodelo. Por exemplo, aplicação é persistida em três tipos de *child*, `Features`, `Subfeature` e `FeatureGroup` presentes nos metamodelos de domínio e de aplicação. O nome do modelo é armazenado na coluna `chil_name`. Uma *child* pode ter outras *child* (coluna `chil_chil_id`), a *child* mãe torna-se uma child

raiz. Toda *child* deve obrigatoriamente definir o modelo que ela pertence (`chil_mode`), isto é, apenas a uma única *child*. Opcionalmente a *child* pode ser uma representação gráfica e, nesta situação, deve armazenar suas posições: horizontal (`chil_left`) e vertical (`chil_top`).

Cada *child* pode ter N *attributes*, do português, atributo. O atributo contém um nome (`attr_nome`) e um valor (`attr_value`) Por exemplo, a característica *Reminder* contém os seguintes atributos opcional e mandatório. Outro exemplo, o grupo de característica *Display Device* contém atributos para definir o mínimo de cardinalidade e o máximo de cardinalidade.