Universidade Federal Fluminense

SÉRGIO TEIXEIRA DE CARVALHO

Modelagem de Linha de Produto de Software Dinâmica para Aplicações Ubíquas

NITERÓI

2013

SÉRGIO TEIXEIRA DE CARVALHO

Modelagem de Linha de Produto de Software Dinâmica para Aplicações Ubíquas

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Redes e Sistemas Distribuídos e Paralelos

Orientador:

Orlando Gomes Loques Filho

Coorientador:

Leonardo Gresta Paulino Murta

Universidade Federal Fluminense

NITERÓI

2013

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

C331 Carvalho, Sérgio Teixeira de
Modelagem de linha de produto de software : dinâmica para aplicações ubíquas / Sérgio Teixeira de Carvalho. – Niterói, RJ : [s.n.], 2013.
198 f.

Tese (Doutorado em Computação) - Universidade Federal Fluminense, 2012.
Orientador: Orlando Gomes Loques Filho, Leonardo Gresta Paulino Murta.

1. Arquitetura de software. 2. Computação ubíqua. 3. Computação pervasiva. 4. Rede de comunicação de computadores. I. Título.

CDD 005.1

Modelagem de Linha de Produto de Software Dinâmica para Aplicações Ubíquas

Sérgio Teixeira de Carvalho

Tese de Doutorado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Doutor em Computação.

Aprovada por:

Prof. Orlando Gomes Loques Filho / IC-UFF (Presidente)

Prof. Leonardo Gresta Paulino Murta / IC-UFF

Prof. Esteban Walter Gonzalez Clua / IC-UFF

Prof. Julius Cesar Barreto Leite / IC-UFF

Prof. Julio Cesar Sampaio do Prado Leite / DI-PUC-Rio

Prof. Joni da Silva Fraga / DAS-UFSC

Niterói, 07 de junho de 2013.

À Karen, Lya e Liza, minha vida.

Do Rigor na Ciência

"... Naquele Império, a Arte da Cartografia atingiu uma tal perfeição que o mapa duma só Província ocupava toda uma Cidade, e o mapa do Império, toda uma Província. Com o tempo, esses Mapas Desmedidos não satisfizeram e os Colégios de Cartógrafos levantaram um Mapa do Império que tinha o tamanho do Império e coincidia ponto por ponto com ele. Menos Apegadas ao Estudo da Cartografia, as Gerações Seguintes entenderam que esse extenso Mapa era Inútil e não sem Impiedade o entregaram às inclemências do Sol e dos Invernos. Nos Desertos do Oeste subsistem despedaçadas Ruínas do Mapa, habitadas por Animais e por Mendigos. Em todo o País não resta outra relíquia das disciplinas geográficas."

[Suárez Miranda: Viajes de Varones Prudentes, livro quarto, cap. XLV, Lérida, 1658. Fragmento selecionado por Jorge Luis Borges, "Do Rigor na Ciência", in "História Universal da Infâmia"].

"O esforço vale a pena. Você está dando um presente para as suas filhas."

Orlando Loques

"Tempus fugit"

Agradecimentos

Meus agradecimentos são muitos.

À Karen, minha esposa. Desenvolver uma tese não é uma tarefa fácil e eu jamais teria conseguido sem o seu apoio e amor incondicionais.

Às minhas filhinhas Lya e Liza, que mesmo sem saber ao certo o que estava acontecendo, me ajudavam com aquelas carinhas lindas e felizes.

A toda a minha família, especialmente à Dona Elci e ao "Seu" Teixeira, minha mãe e meu pai: vocês são meus pilares.

À Dona Vera Lígia e ao "José", minha sogra e meu sogro, que com muita paciência e compreensão nos ajudaram a cuidar das nossas filhinhas durante as minhas ausências.

Ao professor Orlando, meu mestre. As suas palavras, ensinamentos e orientações me conduziram para essa conquista.

Ao professor Leonardo Murta, pelo esmero e dedicação na orientação.

Aos professores Julius e Esteban, dos quais tive a sorte de ser aluno.

Ao professor Joni da Silva Fraga, por ter participado do meu exame de qualificação e por ter aceitado participar desta banca. Ao professor Julio Cesar Sampaio do Prado Leite, por ter aceitado participar desta banca.

À Ângela e à Maria, na secretaria da pós-graduação, por me acolher tão bem no meu primeiro ano de doutorado.

À Teresa e à Viviane pela disposição e energia junto à secretaria da pós-graduação. Tornaram-se grandes amigas, sempre prontas para me ajudar naquilo que fosse preciso.

Ao Carlos Eduardo do suporte, por me atender tão bem e de forma tão prestativa.

Aos meus amigos e *roommates* da "Casa do Fosso", Roger e Tiago. Vivemos momentos, conversas e "caminhadas de volta pra casa" impagáveis!

Ao Copetti, ao Douglas, à Renathinha e ao Vinícius Petrucci, por me receberem tão bem nas suas casas. Quando eu mais precisei, lá estavam eles de braços abertos.

Aos grandes amigos que ganhei de presente nesta trajetória: Anand, André Brandão, Carlão, Cíntia Caetano, Copetti, Daniel Madeira, David, Diego Brandão, Douglas, Edwin (contemporâneo do mestrado), Erick, Eyder, Facada, Flávia Bernardini, Gleiph, Gustavo Cabeça, Idalmis, Jacques, Juliana, Juliano Kazienko, Lincoln, Luciana Brugiolo, Matheus, Micheli, Puca, Rafael, Renathinha, Roger, Rômulo (contemporâneo do mestrado), Stênio, Teresa, Tiago, Thales, Vinícius Petrucci, Viviane, Zamith. Com eles pude compartilhar as minhas inquietações, anseios e alegrias. Valeu!

A todos do LabTempo, um lugar de muito trabalho e, ao mesmo tempo, muito companheirismo.

A todos do MediaLab, onde fiz grandes amigos e conheci o mundo dos *games*.

Ao Instituto de Informática da UFG por apoiar integralmente a minha decisão de cursar o doutorado, e à UFG por viabilizar a minha decisão.

À CAPES e ao CNPq, pelo apoio financeiro em parte do doutorado, e ao Programa de Pós-Graduação da UFF, pelo apoio financeiro para apresentação de artigos em conferências.

Resumo

Aplicações ubíquas são caracterizadas por requisitos que são variáveis ao longo da execução da aplicação, tais como as necessidades e preferências de seus usuários, e a qualidade e disponibilidade de recursos. Estas variações exigem respostas em termos de reconfigurações arquiteturais no nível do *software*, trazendo dificuldades na concepção da sua arquitetura. Uma solução em potencial é a abordagem de Linha de Produto de Software (LPS), na qual as características comuns e as variabilidades dos produtos são capturadas com o objetivo de facilitar o processo de desenvolvimento de novos produtos. Esta tese investiga e propõe o uso de técnicas de LPS na concepção de aplicações ubíquas, em especial, de técnicas de LPSs Dinâmicas (LPSD), nas quais a descrição de variabilidades dinâmicas é usada para diferenciar as configurações arquiteturais dos produtos em tempo de execução. A principal contribuição desta tese é o Metamodelo de Configuração de Variabilidades em LPS, concebido para a integração das características e dos elementos da Arquitetura da LPS. O Metamodelo proposto é formado eminentemente por Contratos Arquiteturais, os quais são modelados como elementos de primeira ordem para descrever variabilidades estáticas e variabilidades dinâmicas na forma de ações de adaptação arquitetural, e para guiar a derivação e a adaptação dinâmica das arquiteturas dos produtos. Com o Metamodelo, Modelos de Características, de Arquitetura e de Configuração são criados de uma forma integrada, sendo que este último reúne, além dos Contratos, regras de composição e de contexto, e mecanismos de seleção dinâmica de componentes. O Metamodelo proposto foi avaliado usando a técnica de questões de competência, tendo como base diferentes cenários de uso envolvendo variabilidades estáticas e dinâmicas do SCIADS (Sistema Computacional Inteligente de Assistência Domiciliar à Saúde), um sistema voltado ao telemonitoramento de pacientes domiciliares concebido pelo nosso grupo de pesquisa do Laboratório Tempo – Sistemas de Tempo Real e Embarcados, Universidade Federal Fluminense.

Palavras-chave: linha de produto de software dinâmico, arquitetura de software autoadaptável, computação ubíqua e pervasiva, sistemas distribuídos.

Abstract

Ubiquitous applications are characterized by requirements that vary throughout the execution of the application, such as the user needs and preferences, and the quality and availability of resources. These variations require responses in terms of architectural reconfigurations at the software level, leading to difficulties in the design of its architecture. A potential solution is the Software Product Line (SPL) approach, in which commonalities and variabilities of the products are captured in order to facilitate the development process of new products. This thesis investigates and proposes the use of SPL techniques in designing ubiquitous applications, and particularly, the use of Dynamic SPL (DSPL) techniques, in which the dynamic variabilities description is used to differentiate the architectural configurations of products at runtime. The main contribution of this thesis is the SPL Variability Configuration Metamodel, designed for the integration of features and SPL Architecture elements. The proposed metamodel is formed predominantly by Architectural Contracts, which are modeled as first-class elements to describe static and dynamic variabilities in the form of architectural adaptation actions, and to guide the derivation and dynamic adaptation of the product architectures. With the Metamodel, Feature Models, Architecture Models and Configuration Models are created in an integrated way, and the latter gathers, beyond Contracts, composition and context rules, and mechanisms for dynamic selection of components. The proposed Metamodel was assessed using the competency questions technique, based on different usage scenarios involving static and dynamic variabilities of SCIADS (Computational Intelligent System for Home Health Care) – from Portuguese, *Sistema Computacional Inteligente de Assistência Domiciliar à Saúde* – a patient telemonitoring system designed by our research group Tempo Laboratory – Real-Time and Embedded Systems, Universidade Federal Fluminense.

Keywords: dynamic software product line, self-adaptive software architecture, ubiquitous and pervasive computing, distributed systems.

Glossário

ADL	:	*Architectural Description Language*
CK	:	*Configuration Knowledge*
EMF	:	*Eclipse Modeling Framework*
HHS	:	*Home Health Station*
LPS	:	Linha de Produto de Software
LPSD	:	Linha de Produto de Software Dinâmica
SCIADS	:	Sistema Computacional Inteligente de Assistência Domiciliar à Saúde
SMF	:	*Supervisory Medical Facility*
UML	:	*Unified Modeling Language*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Este capítulo apresenta os objetivos e as principais contribuições desta tese, além das questões que motivaram o seu desenvolvimento. A organização do texto também é apresentada, incluindo uma breve descrição de cada um dos seus capítulos.

1.1 Contexto

Com o propósito de facilitar a contextualização das questões que motivaram o desenvolvimento deste trabalho, um cenário é apresentado a seguir. Ele descreve possíveis situações de uso de um sistema de telemonitoramento de pacientes domiciliares[1].

> *Marina está com 65 anos e mora sozinha. Ela tem pressão alta, diagnosticada aos 55 anos. Por causa disso, toma medicamentos regularmente para manter a pressão em valores considerados normais. Diante das solicitações de seu médico, ela usa um sensor sem fio capaz de aferir a pressão arterial e a frequência cardíaca. O sensor é de fácil manuseio e, para funcionar, basta que seja ajustado em torno do pulso. Há também sensores de temperatura, umidade, luminosidade e presença instalados pela casa, além de sensores que reconhecem atividades. Esses sensores são praticamente imperceptíveis, não exigindo da paciente qualquer interação.*
>
> *Dada a sua situação um tanto delicada em relação à pressão, Marina segue rigorosamente a prescrição médica definida em seu plano de cuidados. Para evitar esquecimentos, mensagens são apresentadas na TV alguns minutos antes do horário definido para tomar o medicamento. Se a TV não estiver disponível*

[1] Por uma questão de simplicidade, as informações médicas não são detalhadas.

(por exemplo, desligada), as mensagens são enviadas para o tablet ou para o celular, dependendo daquele que estiver mais próximo da paciente. Depois de tomar o medicamento, ela usa o controle remoto ou toca na tela para confirmar que cumpriu a prescrição. O sistema também envia mensagens lembrando a paciente de medir a sua pressão e frequência cardíaca.

O dia-a-dia de Marina é normal, tendo os seus dados coletados e enviados sistematicamente para o provedor de serviços de saúde, o qual mantém uma Central de Supervisão equipada com monitores para a visualização de todas as suas informações fisiológicas. Uma situação de anormalidade na saúde da paciente é imediatamente reportada na forma de alertas para a Central, a qual pode acionar o serviço de emergência, se for o caso.

Mais recentemente, valores elevados de frequência cardíaca têm sido apresentados por Marina, ocasionando frequentes alertas junto à Central. Com base nestes alertas e em outros fatores, o médico, diante da suspeita de insuficiência cardíaca, decide que a paciente deve iniciar imediatamente um controle preciso de seu peso.[2] *Uma balança é então selecionada para aquela paciente, fazendo com que o sistema implantado na casa configure a balança e passe a enviar mensagens lembrando a paciente de se pesar diariamente, e nos horários estabelecidos no plano de cuidados. Além disso, os dados das pesagens, como já ocorre com os de pressão arterial e frequência cardíaca, passam a ser enviados sistematicamente para a Central. Exames realizados em uma clínica especializada constatam a insuficiência cardíaca. O médico, então, prescreve novos medicamentos, além de manter a balança e incluir novos sensores, com o objetivo de detectar precocemente um possível agravamento da doença.*

O cenário descrito caracteriza uma aplicação ubíqua, cujos requisitos variáveis ao longo do tempo introduzem dificuldades na concepção da sua arquitetura. Esta tese lida com questões neste contexto, considerando mais especificamente o uso de técnicas de Linha de Produto de Software (LPS) – do inglês, *Software Product Line* (SPL) – na modelagem da arquitetura de aplicações dessa classe.

Os requisitos identificados no cenário estão relacionados com as necessidades individuais da paciente, suas preferências, seu ambiente residencial, e aspectos médicos, que

[2]De acordo com especialistas consultados, a insuficiência cardíaca é uma enfermidade progressiva que pode agravar-se com o tempo, sendo que um dos fatores que podem provocar o seu início é o longo histórico de pressão alta. Além disso, o organismo tenta compensar os efeitos da doença aumentando a frequência cardíaca, e retendo sal e água, o que provoca um rápido aumento de peso.

podem mudar de acordo com a evolução de seus problemas de saúde. Por exemplo, a hipertensão pode evoluir para insuficiência cardíaca, exigindo um novo sensor fisiológico apropriado, e assim novos recursos de *software*. Além disso, tais sistemas dependem da qualidade e disponibilidade de recursos (*e.g.*, medidores, sensores, dispositivos), as quais podem variar ao longo da execução da aplicação, uma vez que estes recursos podem ser adicionados, removidos, ou podem simplesmente falhar. Essas mudanças em relação às necessidades da paciente ou variações no ambiente exigem respostas em termos de reconfigurações arquiteturais no nível do *software*. Além de permitir personalizações antes da implantação (variabilidade estática), a arquitetura deve também permitir mudanças em tempo de execução (variabilidade dinâmica) [28].

Uma solução em potencial para estas questões é a abordagem de LPS [39, 99], uma vez que tem se mostrado uma forma eficiente de lidar com as diferentes necessidades dos usuários e restrições em termos dos recursos do ambiente [64, 63]. Em LPS, o foco está no desenvolvimento de uma família de produtos [39], onde uma arquitetura (denominada Arquitetura da LPS) deve ser descrita de uma forma que seja comum a todos os produtos [47] e, ao mesmo tempo, capture as diferenças entre eles. Para criar (derivar) a arquitetura de um produto específico, estas diferenças, denominadas variabilidades, devem ser resolvidas [129].

Tradicionalmente, LPSs não permitem que as arquiteturas de seus produtos sejam adaptadas em tempo de execução [76, 99, 129, 39]. Mais recentemente, porém, técnicas de LPS têm sido aplicadas no desenvolvimento de Sistemas Adaptativos [80, 96, 30, 31, 32, 105, 64, 12, 135, 128]. Um Sistema Adaptativo é caracterizado por gerar, durante a sua execução, diferentes configurações arquiteturais em decorrência de mudanças do contexto de execução ou de mudanças dos requisitos do usuário. Este aspecto é essencial em aplicações ubíquas, pois são sensíveis ao contexto [2] e demandam uma arquitetura adaptável dinamicamente. LPSs voltadas a Sistemas Adaptativos são conhecidas como LPSs Dinâmicas (LPSD) – do inglês, *Dynamic* SPL (DSPL) [63, 64]. Em LPSDs, cada produto corresponde a uma determinada configuração da arquitetura do sistema gerada em tempo de execução [3, 12], e os produtos são diferenciados uns dos outros por meio da descrição de variabilidades dinâmicas.

1.2 Motivação

O projeto SCIADS (Sistema Computacional Inteligente de Assistência Domiciliar à Saúde)[3], conduzido no contexto do nosso grupo de pesquisa[4], é um sistema voltado ao telemonitoramento de pacientes domiciliares [21, 20, 23], nos mesmos moldes do cenário apresentado na Seção 1.1. Os seus requisitos de personalização e as suas propriedades dinâmicas originaram a investigação e a proposta desta tese.

No SCIADS, variabilidades estáticas e variabilidades dinâmicas devem ser descritas de forma integrada na Arquitetura da LPS. Isso possibilita a criação de produtos em um processo cujas variabilidades estáticas são resolvidas primeiramente, deixando em aberto as variabilidades dinâmicas, para que estas sejam resolvidas posteriormente em tempo de execução. Essa integração, no nível da arquitetura, de variabilidades estáticas e dinâmicas, permite que um mesmo produto seja personalizado antes e após a sua implantação.

As variabilidades estáticas são usualmente descritas como pontos de variação [6, 47, 118, 60, 130], incluídos na Arquitetura da LPS com o objetivo de estabelecer as diferenças entre as Arquiteturas dos Produtos. Diferentemente, as variabilidades dinâmicas são descritas no contexto de LPSD, envolvendo, em grande parte das abordagens [30, 31, 32, 125, 80, 105], o uso de modelos de características para representar as variabilidades do sistema. Como tais modelos são eminentemente conceituais e estão em um nível de abstração mais alto [69, 45], se comparado ao nível arquitetural, uma associação deve ser feita entre características e elementos arquiteturais para que os produtos possam ser criados. Esta associação é muitas vezes conhecida como *Configuration Knowledge* (CK) [44].

Em geral, as abordagens que suportam variabilidades dinâmicas não se preocupam em representar um modelo para o CK, sendo que algumas apresentam técnicas que o definem de forma implícita. Há o uso de parametrizações [125, 80], de expressões anotadas diretamente nos componentes da arquitetura [128], de regras de composição definidas na própria Arquitetura da LPS [135], e de elementos de contexto representados no modelo de características [32, 33]. De forma geral, o entrelaçamento dos modelos dificulta a descrição, de modo independente, das variabilidades dinâmicas e das suas respectivas adaptações arquiteturais. Além disso, caso as variabilidades dinâmicas sejam críticas e complexas, como as existentes em sistemas de assistência à saúde [28, 83], a definição do CK nestes moldes pode tornar mais difícil modelar sistemas cujo comportamento deve ser seguro (*safety*) [28, 66, 77], como o próprio SCIADS. Uma forma de diminuir os riscos inerentes

[3] *http://www.tempo.uff.br/sciads*

[4] Laboratório Tempo – Sistemas de Tempo Real e Embarcados, *http://www.tempo.uff.br*.

a este processo de mapeamento é descrever as variabilidades dinâmicas especificando de uma forma clara e explícita as ações de adaptação arquitetural e as condições (*e.g.*, regras de composição, regras de contexto) sob as quais tais ações devem ser realizadas.

Diante do exposto, dois *Pontos* podem ser destacados:

1. *As abordagens estudadas não permitem descrever, de uma maneira integrada, variabilidades estáticas e dinâmicas no nível da Arquitetura da LPS*, o que pode causar um esforço extra, uma vez que uma abordagem voltada à descrição de variabilidades estáticas pode não ser apropriada para variabilidades dinâmicas, e vice-versa.

2. *As abordagens que suportam variabilidades dinâmicas não representam um modelo para o CK que seja explícito e independente dos modelos da LPS (características e arquitetura)*, e que tenha condições de integrar características, arquitetura, variabilidades estáticas e dinâmicas, e os processos de derivação e adaptação dinâmica de arquiteturas de produtos.

Estes pontos motivaram, no contexto desta tese, a investigação de Contratos Arquiteturais [29, 16, 84, 15] como uma forma de descrever as variabilidades no nível da Arquitetura da LPS. Os resultados obtidos a partir da descrição de variabilidades dinâmicas do SCIADS [26, 27], vislumbraram a construção de metamodelos [117, 120] visando representar e integrar características e Arquitetura da LPS por meio dos Contratos. Com essa abordagem, Contratos e seus elementos podem constituir o CK, estabelecendo um novo modelo independente dos demais modelos da LPS, denominado Modelo de Configuração. O uso de metamodelos permite expressar estes conceitos em um nível alto de abstração, e desenvolvê-los independentemente de detalhes de implementação e plataformas.

1.3 Objetivos

O objetivo desta tese é:

Definir um metamodelo, em um alto nível de abstração, para integrar as características e os elementos da Arquitetura da LPS, por meio do Modelo de Configuração, um CK descrito de forma independente dos demais modelos da LPS. O Modelo de Configuração é composto por Contratos, os quais são

modelados como elementos de primeira ordem e no nível da Arquitetura da LPS para: (i) descrever as variabilidades estáticas e dinâmicas na forma de ações de adaptação arquitetural, e as condições sob as quais tais adaptações devem ser realizadas; (ii) associar características a elementos da arquitetura; e (iii) guiar a derivação e a adaptação dinâmica das Arquiteturas dos Produtos.

A partir do metamodelo pode-se criar os Modelos de Características e de Arquitetura da LPS, e o Modelo de Configuração, de uma forma integrada. O Modelo de Configuração reúne, além dos Contratos, um conjunto de informações essenciais à configuração de variabilidades: regras de composição; informações e regras de contexto; e mecanismos de seleção de componentes e configurações arquiteturais.

O objetivo geral da tese pode ser dividido nos objetivos específicos abaixo:

1. Mostrar como variabilidades podem ser descritas na forma de Contratos no nível da Arquitetura da LPS.
2. Definir um metamodelo de características e um metamodelo de Arquiteturas de LPS.
3. Definir um metamodelo de configuração integrando características, Arquitetura da LPS e Contratos.
4. Definir diretrizes para a criação (instanciação) dos modelos da LPS, e para a derivação e adaptação dinâmica de Arquiteturas de Produtos.
5. Avaliar o metamodelo por meio de questões de competência [62, 127], onde instâncias do metamodelo, representadas por Diagramas de Objetos da UML (*Unified Modeling Language*), são criadas tendo como base cenários do SCIADS.

1.4 Contribuições

As principais contribuições desta tese são:

- O Metamodelo de Configuração de Variabilidades em LPS, concebido para a criação de modelos da LPS (Característica, Arquitetura e Configuração) voltados para sistemas cuja arquitetura seja adaptável diante de mudanças de contexto e mudanças dos requisitos dos usuários, tais como as aplicações ubíquas.

- A ampliação do escopo de utilização dos Contratos, permitindo que sejam aplicados na modelagem de Arquiteturas de LPS.
- Os resultados da descrição de variabilidades na forma de Contratos no nível da Arquitetura da LPS, envolvendo variabilidades dinâmicas do SCIADS.
- O Modelo de Características do SCIADS.
- Os resultados do uso da técnica de formulação de questões de competência na avaliação do Metamodelo.

Ao longo do desenvolvimento desta tese foi produzido um total de 12 (doze) trabalhos com alguns dos resultados obtidos, sendo 10 (dez) artigos [68, 28, 21, 20, 26, 25, 23, 122, 113, 19] e 02 (dois) relatórios técnicos [27, 22]. Dois artigos [20, 26] foram indicados para o prêmio de *Melhor Artigo* de suas respectivas conferências.

A seguir a lista de artigos:

1. *Runtime Monitoring and Auditing of Self-Adaptive Systems*, 25th International Conference on Software Engineering and Knowledge Engineering (SEKE 2013), junho de 2013 [68].
2. *Variabilities as First-Class Elements in Product Line Architectures of Homecare Systems*, 4th International Workshop on Software Engineering in Health Care (SEHC 2012), 34th International Conference on Software Engineering (ICSE 2012), junho de 2012 [28].
3. *Sistema de Computação Ubíqua na Assistência Domiciliar à Saúde*, Journal of Health Informatics (JHI), abril de 2011 [21].
4. *Um Sistema Computacional Inteligente de Assistência Domiciliar à Saúde*[5], XII Congresso Brasileiro de Informática na Saúde (CBIS 2010), outubro de 2010 [20].
5. *Dynamic Variability Management in Product Lines: An Approach Based on Architectural Contracts*[5], 4o. Simpósio Brasileiro de Componentes, Arquiteturas e Reuso de Software (SBCARS 2010), Congresso Brasileiro de Software: Teoria e Prática (CBSoft 2010), setembro de 2010 [26].

[5]Artigo indicado para o prêmio de *Melhor Artigo*.

6. *Arquitetura de Software para Sistemas Pervasivos de Assistência Domiciliar à Saúde*, X Workshop de Informática Médica (WIM 2010), XXX Congresso da Sociedade Brasileira de Computação (CSBC 2010), julho de 2010 [25].

7. *Monitoramento Remoto de Pacientes em Ambiente Domiciliar*, XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2010), maio de 2010 [23].

8. *Applying Context-Aware Techniques to Design Remote Assisted Living Applications*, International Journal of Functional Informatics and Personalised Medicine (IJFIPM), dezembro de 2009 [122].

9. *Using Discovery and Monitoring Services to Support Context-Aware Remote Assisted Living Applications*, International Conference on Computational Science and Engineering (CSE 2009), agosto de 2009 [113].

10. *E-Learning e Jogos Eletrônicos Interativos: Possibilidades para a Educação Médica*[6], XIX Simpósio Brasileiro de Informática na Educação (SBIE 2008), novembro de 2008 [19].

A seguir a lista de relatórios técnicos:

1. *A Contract-based Approach for Managing Dynamic Variability in Software Product Line Architectures*, Relatório Técnico, março de 2011 [27].

2. *Sistema de Assistência Domiciliar à Saúde Telemonitorada*, Relatório Técnico, abril de 2009 [22].

1.5 Organização

A tese está organizada em 07 (sete) capítulos, estruturados como se segue (Figura 1.1).

Introdução & Conceitos

– **Capítulo 1. Introdução (*este capítulo*).**
Apresenta os objetivos e as principais contribuições desta tese, além das questões que motivaram o seu desenvolvimento.

[6]Não está diretamente relacionado ao tema desta tese, porém é um dos marcos iniciais do SCIADS.

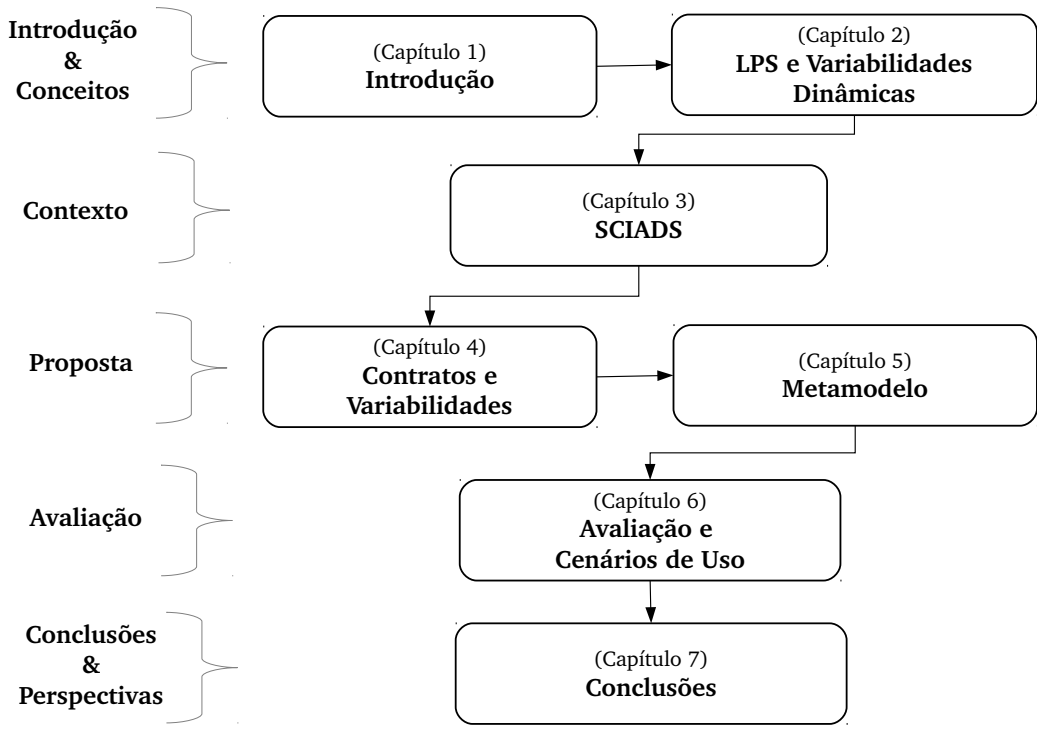


Figura 1.1: Organização da tese.

– **Capítulo 2. LPS e Variabilidades Dinâmicas.**

Apresenta os conceitos e os aspectos mais relevantes de LPS, Variabilidades Dinâmicas e Metamodelagem. Em relação a LPS, o capítulo apresenta conceitos compreendendo os Modelos de Características, Arquitetura e *Configuration Knowledge*, e as principais abordagens usadas para a modelagem de variabilidades. Conceitos de variabilidades dinâmicas são apresentados, incluindo a relação entre LPS e Sistemas Adaptativos, e as mais importantes abordagens que lidam com variabilidades dinâmicas em LPS. Isso inclui uma discussão destacando os trabalhos relacionados com esta tese. Por fim, é apresentada uma rápida visão sobre Metamodelagem, incluindo o EMF (*Eclipse Modeling Framework*)[7] [120, 95] e o meta-metamodelo Ecore [120], usados no desenvolvimento do Metamodelo proposto.

Contexto

– **Capítulo 3. SCIADS.**

Apresenta a estruturação geral do SCIADS e o seu Modelo de Características, o qual serve de base para boa parte dos exemplos e cenários apresentados nos demais capítulos da tese.

[7] *http://www.eclipse.org/emf*

Proposta

– **Capítulo 4. Contratos e Variabilidades.**

Detalha os conceitos e a estruturação dos Contratos, e como variabilidades podem ser descritas como Contratos no nível da Arquitetura da LPS.

– **Capítulo 5. Metamodelo de Configuração de Variabilidades em LPS.**

Detalha o Metamodelo proposto, destacando as três partes que o compõem: Metamodelo de Características, Metamodelo de Arquitetura e Metamodelo de Configuração, o qual inclui Contratos em sua estrutura. O capítulo apresenta também as diretrizes compreendendo os elementos e atividades que devem ser realizadas para a construção dos modelos da LPS, e para a derivação e adaptação dinâmica de Arquiteturas de Produtos.

Avaliação

– **Capítulo 6. Avaliação e Cenários de Uso.**

Apresenta a avaliação do Metamodelo de Configuração de Variabilidades em LPS, realizada com base na técnica de formulação de questões de competência. As questões são organizadas conforme os objetivos do Metamodelo, e as respostas compostas de explicações textuais e Diagramas de Objetos da UML, ambos envolvendo cenários com características do SCIADS.

Conclusões & Perspectivas

– **Capítulo 7. Conclusões.**

Apresenta as considerações finais e as principais contribuições desta tese, assim como as limitações e trabalhos futuros.

Capítulo 2

Linha de Produto de Software e Variabilidades Dinâmicas

Este capítulo aborda e inter-relaciona os principais conceitos utilizados no decorrer deste trabalho, detalha importantes abordagens envolvendo variabilidades dinâmicas em LPS, e apresenta uma discussão destacando as abordagens relacionadas com esta tese.

2.1 Introdução

Esta tese lida com Linhas de Produto de Software (LPS) modeladas para sistemas cuja arquitetura seja adaptável diante de mudanças de contexto e de mudanças de requisitos dos usuários. Além de LPS, a abordagem proposta combina outros conceitos, como representado na Figura 2.1.

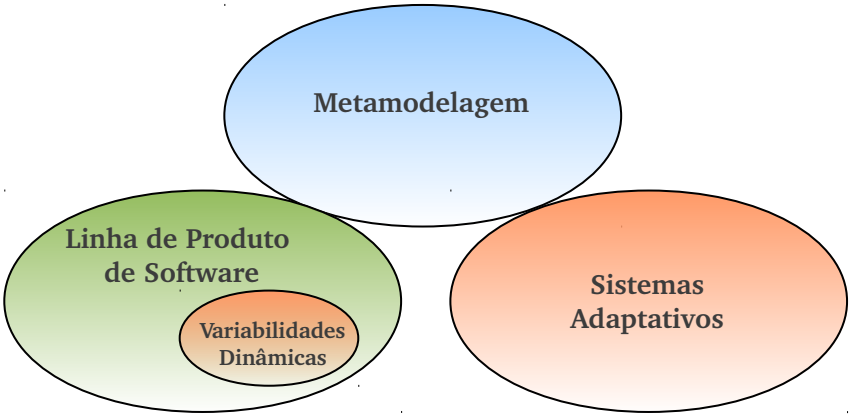


Figura 2.1: Principais conceitos envolvidos nesta tese.

Linhas de Produto de Software (LPS) são constituídas basicamente de dois mode-

los [99]: Modelo de Características [69, 46] e Modelo de Arquitetura [47, 60]. Adicionalmente, um terceiro modelo, o *Configuration Knowledge* (CK) [44] pode ser descrito para associar características e elementos da arquitetura.

O conceito de *Variabilidades Dinâmicas*, por sua vez, tem sido utilizado em LPSs voltadas a *Sistemas Adaptativos* [125, 80, 96, 97, 30, 31, 32, 105, 64, 12, 135, 128]. Há uma variedade de trabalhos abordando esse tema, com foco, principalmente, no suporte à reconfiguração, plataformas e monitoramento, mas com pouca atenção ao uso dos modelos da LPS de uma forma integrada.

Esta questão é tratada nesta tese com base no conceito de *Metamodelagem* [117, 120], sendo que parte do Metamodelo proposto (veja Capítulo 5) é formada pelos modelos apresentados na Seção 2.2 deste capítulo.

Este capítulo está organizado como se segue.

Seção 2.2

Conceitos de LPS, compreendendo os seus modelos – Características, Arquitetura e *Configuration Knowledge* (CK) – e as principais abordagens usadas para a modelagem de variabilidades.

Seção 2.3

Conceitos de variabilidades dinâmicas, apresentando a relação entre LPS e Sistemas Adaptativos, e as mais importantes abordagens que lidam com variabilidades dinâmicas em LPS.

Seção 2.4

Discussão tratando dos pontos em que as abordagens em destaque se relacionam com a proposta desta tese.

Seção 2.5

Uma rápida visão sobre metamodelagem, incluindo o EMF (*Eclipse Modeling Framework*) e o meta-metamodelo Ecore, usados no desenvolvimento do Metamodelo proposto.

Seção 2.6

Considerações finais.

2.2 Linha de Produto de Software

Muitas aplicações possuem conceitos e características similares, constituindo uma família de produtos de *software* [50, 103, 104][1]. Esse aspecto aliado às necessidades de redução de custo e esforço no desenvolvimento de sistemas constituem os princípios das LPSs. O uso das técnicas de LPS tem como foco a concepção, o desenvolvimento e a evolução de uma família de produtos de *software* relacionados [39, 99]. Todos os produtos de uma LPS compartilham entre si um conjunto de elementos mandatórios, e se diferenciam uns dos outros por meio de um conjunto de variabilidades.

Para se criar um produto específico, as variabilidades da LPS devem ser avaliadas e resolvidas. Por exemplo, ao criar um produto de uma LPS voltada ao monitoramento de pacientes, uma variabilidade a ser resolvida pode ser referente à existência ou não do sensor "botão de pânico" no produto. Este equipamento, geralmente portado pelo próprio paciente, emite um alarme de emergência para a central de supervisão médica ao ser pressionado. Em muitas situações, entretanto, o seu uso pode ser desnecessário, ou mesmo inapropriado, de acordo com o cuidado e/ou tratamento daquele paciente. O produto derivado possui em sua arquitetura todos os elementos mandatórios presentes na LPS, mais os elementos selecionados a partir da resolução das variabilidades. Uma vez que todas as variabilidades são resolvidas, o produto pode ser implantado e executado.

Para que tanto os elementos mandatórios quanto as variabilidades sejam representados em uma LPS, modelos de variabilidades devem ser criados para capturar as semelhanças e diferenças entre os produtos [129]. Para os propósitos desta tese, três modelos são empregados: Modelo de Características, também conhecido como Modelo de *Features*, Modelo de Arquitetura e *Configuration Knowledge* (CK). O primeiro permite modelar as variabilidades da LPS em termos de suas características [46, 56], e o segundo, em termos de seus elementos arquiteturais [47, 60]. O terceiro modelo, por sua vez, associa características a elementos arquiteturais [44]. Estes modelos são detalhados nas próximas subseções, além das principais abordagens para se modelar variabilidades em LPS.

2.2.1 Modelo de Características

As variabilidades, quando modeladas em um sistema de *software*, permitem que este seja estendido, personalizado ou configurado para uso em um contexto particular [121].

[1] Os trabalhos precursores de Dijkstra [50] e de Parnas [104, 103] apresentam as primeiras ideias sobre família de *software*.

Uma forma largamente usada de representar variabilidades é por meio do Modelo de Características, parte integrante da proposta de Kang *et al.* [69] denominada *Feature-Oriented Domain Analysis* (FODA). O Modelo de Características 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 [56].

Organizadas em um diagrama na forma de árvore, as características são divididas em três tipos: mandatórias, opcionais e alternativas. As características mandatórias são aquelas que devem estar presentes em todos os produtos deriváveis da LPS. Por outro lado, as opcionais e as alternativas podem ou não ser selecionadas na composição de um produto durante a sua derivação, representando, portanto, as variabilidades da LPS [69].

A Figura 2.2 apresenta um exemplo de Modelo de Características. Neste modelo, que é uma parte do modelo construído para o SCIADS (veja Capítulo 3), há quatro características mandatórias: *Alarm*, para enviar alertas para a central de supervisão; *Sensor*, representando os sensores que podem ser usados pelo paciente; *MedicalSensor* representando os sensores médicos; e *HealthProblem* identificando os possíveis problemas de saúde do paciente. Há ainda a característica opcional *PanicButton* (botão de pânico) e dois grupos de características, um composto das características alternativas *HeartRate* (sensor de frequência cardíaca) e *BloodPressure* (sensor de pressão arterial), e outro composto das características alternativas *Diabetes* (diabetes) e *Hypertension* (hipertensão).

Características alternativas são representadas em grupos e podem ser selecionadas conforme a cardinalidade definida para o grupo [46]. A expressão $< 1 - 2 >$ identifica a cardinalidade do grupo de características alternativas *HeartRate* e *BloodPressure*.

Uma cardinalidade é identificada por $< a - b >$, onde,

a é a quantidade mínima de características alternativas que deve ser selecionada, $a \geq 0$;

b é a quantidade máxima de características que deve ser selecionada, $b \geq 1$.

Dessa forma, a cardinalidade $< 1 - 2 >$ representa uma relação OR entre as características alternativas que compõem o grupo, uma vez que uma ou mais podem ser selecionadas. Outra relação que pode ser expressa em grupos é a XOR, onde uma e apenas uma característica alternativa pode ser selecionada. A cardinalidade, nesse caso, deve ser $< 1 - 1 >$. A representação de cardinalidades aqui adotada está baseada nas exten-

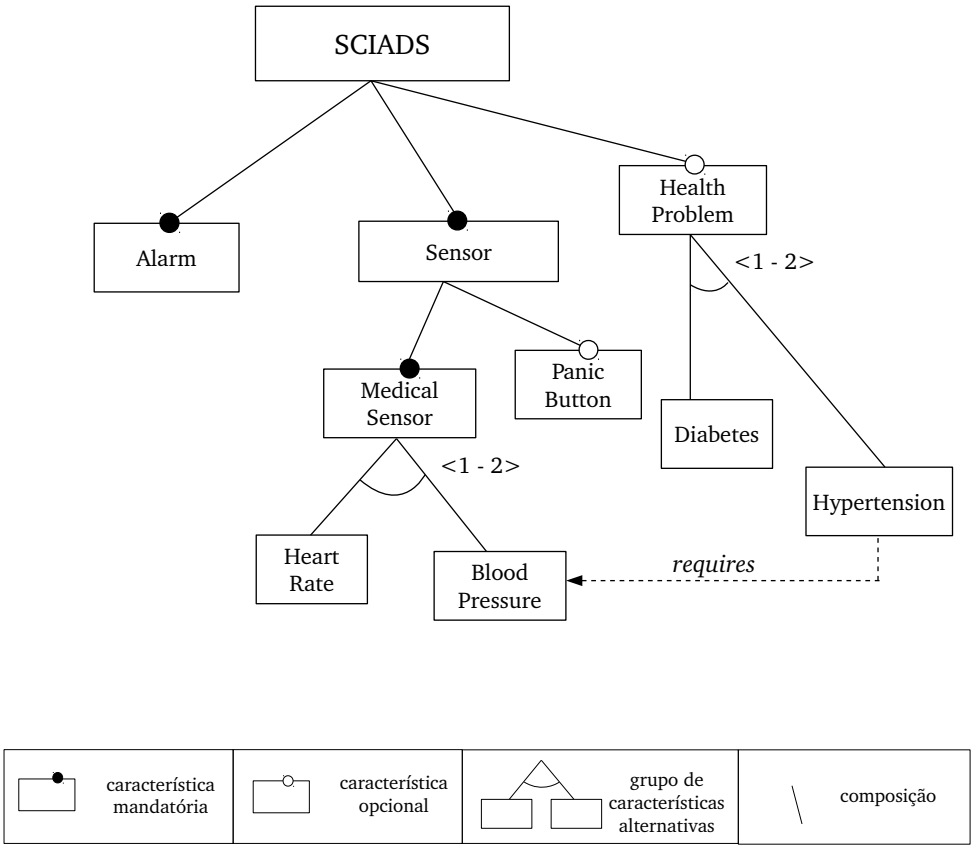


Figura 2.2: Exemplo de um Modelo de Características.

sões propostas por Czarnecki *et al.* [46, 45], apresentadas com o objetivo de aumentar a expressividade do modelo original, definido em [69].

As características podem também ser relacionadas por meio da definição de restrições de dependência. Duas declarações podem ser usadas para expressar dependências: *requires* e *excludes*. Uma determinada característica pode requerer uma ou mais características (*requires*), ou ainda excluir uma ou mais características (*excludes*). O modelo da Figura 2.2 apresenta um exemplo de uso da declaração *requires*, onde um paciente hipertenso requer a seleção do sensor de pressão arterial. A modelagem de restrições de dependência facilita a verificação de consistência dos produtos da LPS.

Ao derivar um produto obtém-se a Configuração de Características desse produto (do inglês, *Feature Configuration*), composta das características selecionadas durante o processo de derivação. A Figura 2.3 mostra a Configuração de Características de um produto derivado a partir do modelo da Figura 2.2. O produto foi criado para um paciente que precisa ser monitorado quanto à sua pressão arterial, possuindo, portanto, a característica alternativa *BloodPressure* selecionada, além das mandatórias *Alarm*, *Sensor*, e *MedicalSensor*. *FC* representa o conjunto formado pelas características que compõem o

produto.

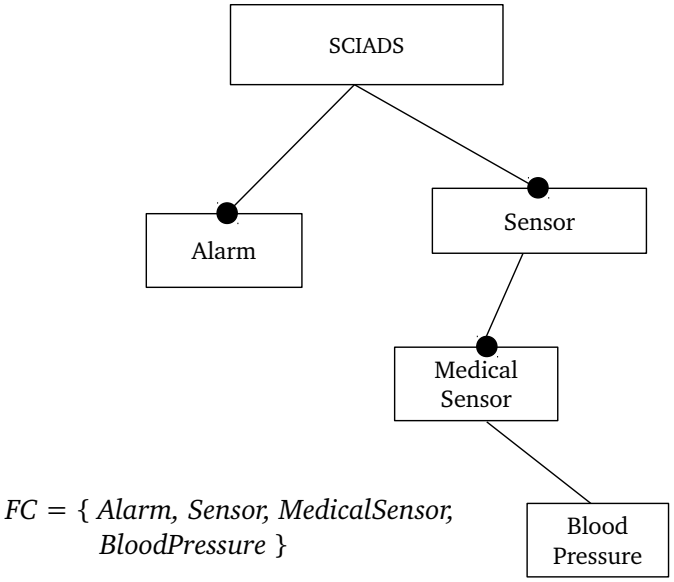


Figura 2.3: Exemplo de uma Configuração de Características.

Um metamodelo representando a estruturação do Modelo de Características é apresentado na Figura 2.4. Descrito na forma de um Diagrama de Classes da UML, o metamodelo mostra que um Modelo de Características (*FeatureModel*) é formado por uma ou mais características-raiz (*-root*), as quais podem ser compostas por subcaracterísticas (*SubFeature*) e por grupos de características (*FeatureGroup*). As subcaracterísticas podem também ser compostas por outras subcaracterísticas e ainda por grupos. Além disso, subcaracterísticas podem ser mandatórias ou opcionais (*optional*) e os grupos, por sua vez, têm em sua composição duas ou mais características alternativas (*-alternative*). Este metamodelo integra a abordagem proposta nesta tese, a qual é detalhada no Capítulo 5.

2.2.2 Modelo de Arquitetura

Além do Modelo de Características, uma LPS é constituída, basicamente, de um conjunto de componentes reutilizáveis e de uma Arquitetura de Software [99].

A arquitetura definida para um sistema de *software* especifica a estrutura do sistema, formada por componentes e conectores, e a configuração, por meio da qual instâncias de componentes e conectores são criadas e interligadas [123, 91, 24]. Para que componentes e conectores sejam interligados para formar a configuração da arquitetura, portas devem ser criadas identificando os seus pontos de interligação. As portas podem ser de saída, representando serviços requisitados pelo componente ou conector, ou podem ser de entrada, representando serviços fornecidos. A especificação dos elementos da Arquitetura

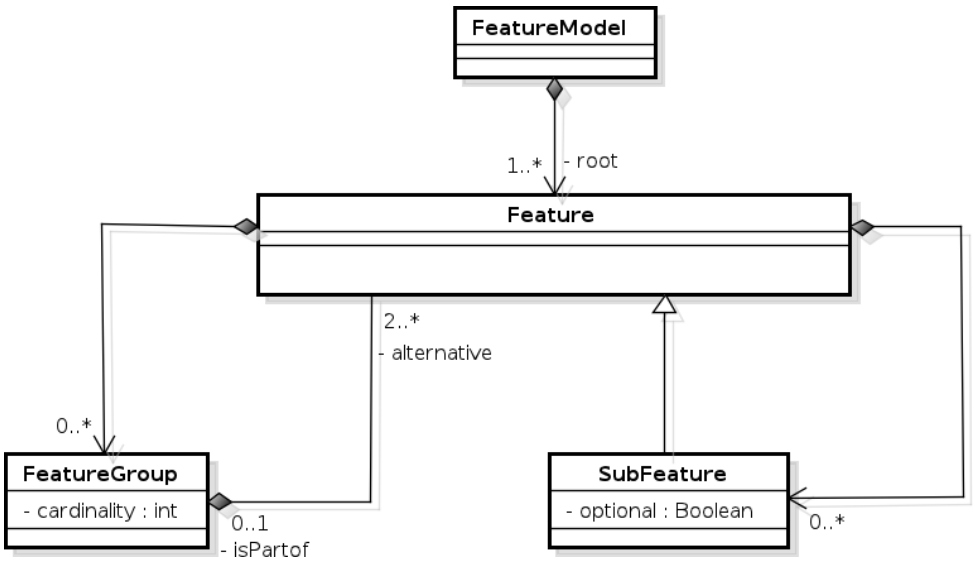


Figura 2.4: Metamodelo de Características.

de Software de um sistema pode ser feita por meio de uma Linguagem de Descrição Arquitetural (do inglês, *Architectural Description Language* - ADL). A Figura 2.5 mostra um metamodelo de Arquiteturas de Software, desenvolvido na forma de um Diagrama de Classes da UML.

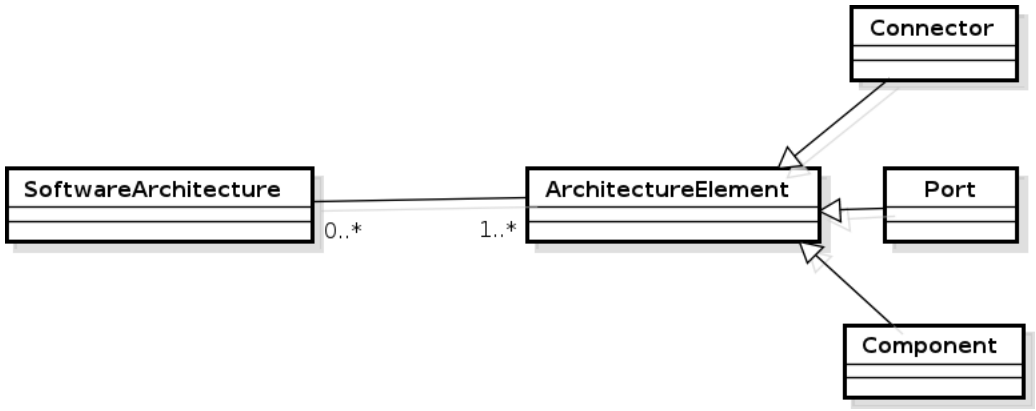


Figura 2.5: Metamodelo de Arquiteturas de Software.

Diferentemente da arquitetura concebida para sistemas isolados e não relacionados, a arquitetura de uma LPS precisa fornecer abstrações para representar a estrutura e o comportamento de todos os produtos que podem ser derivados da LPS. Para isso, ela precisa identificar as variabilidades que tornam as arquiteturas dos produtos diferentes entre si, e os elementos arquiteturais que correspondem a tais variabilidades.

A Figura 2.6 apresenta um metamodelo que acrescenta os conceitos de Arquitetura de LPS ao metamodelo de Arquiteturas de Software. As variabilidades da Arquitetura da LPS (*Variability*) são representadas na forma de pontos de variação (*VariationPoint*) e

na forma de opcionalidades (*Optionality*). Em um ponto de variação há dois ou mais elementos arquiteturais variantes (*-variant*) que podem ser selecionados durante a derivação. Em uma opcionalidade há um elemento arquitetural opcional (*-optional*) que pode ou não ser selecionado. Este metamodelo integra a abordagem proposta nesta tese, a qual é detalhada no Capítulo 5.

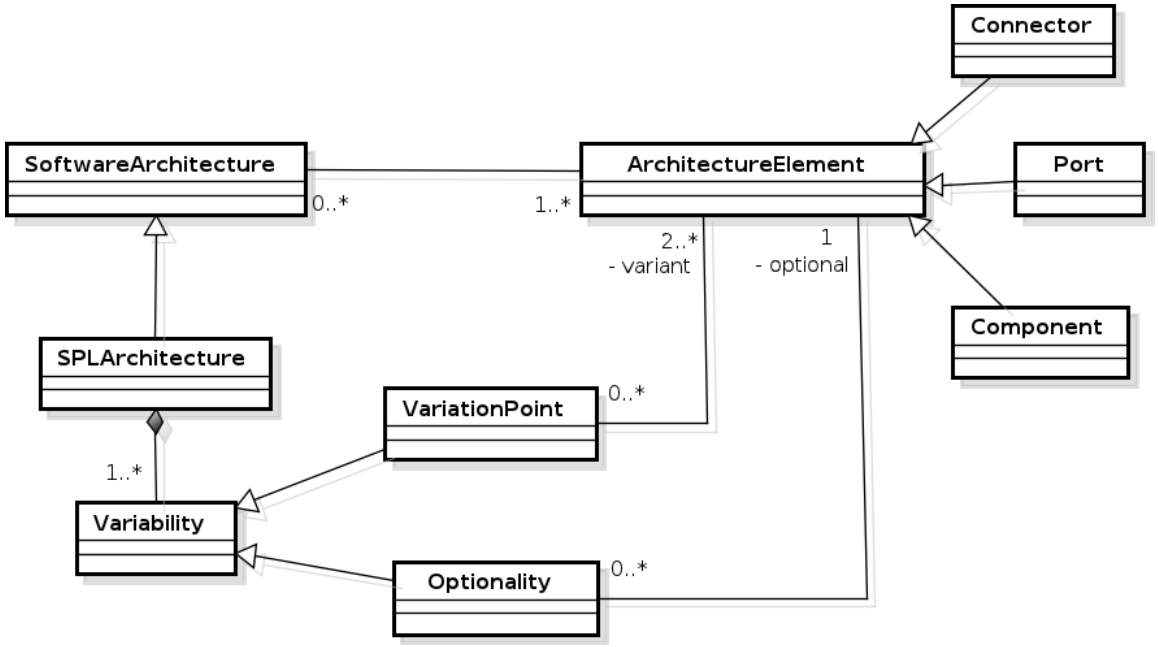


Figura 2.6: Metamodelo de Arquiteturas de LPS.

A Figura 2.7 mostra a arquitetura correspondente ao Modelo de Características mostrado na Figura 2.2. Os componentes *BloodPressureSensor* e *HeartRateSensor* e os seus respectivos conectores compõem um ponto de variação da Arquitetura da LPS. A arquitetura de um produto específico pode ser derivada a partir da Arquitetura da LPS por meio da incorporação dos elementos mandatórios e da seleção dos elementos correspondentes às variabilidades [47, 60]. Portanto, neste exemplo, pode-se derivar a arquitetura de um produto contendo o componente *BloodPressureSensor*, ou o componente *HeartRateSensor*, ou ainda ambos os componentes.

2.2.3 *Configuration Knowledge*

A Arquitetura da LPS representa os elementos que efetivamente colaboram para a execução de um produto. Diferentemente, o Modelo de Características representa um nível mais alto de abstração, sendo bastante empregado como modelo de apoio à análise de requisitos no desenvolvimento da LPS [116, 69].

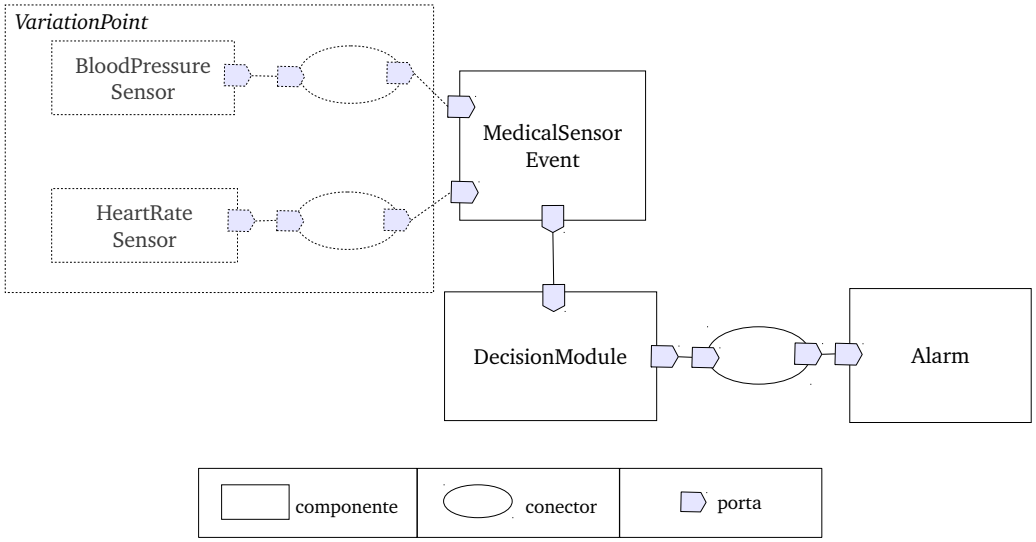


Figura 2.7: Exemplo de Arquitetura de LPS.

Uma vez construídos o Modelo de Características e a Arquitetura da LPS, características e elementos arquiteturais precisam ser associados uns aos outros para que produtos possam ser derivados a partir da seleção de características. O *Configuration Knowledge* (CK) [44] realiza essa associação, descrevendo, para as características, os elementos arquiteturais correspondentes e as regras e restrições de como combiná-los para formar a arquitetura de um produto. Dessa forma, o processo de derivação de um produto pode ser feito pela seleção de características (produzindo a Configuração de Características), seguida pela análise do CK, de onde se obtém os elementos arquiteturais necessários à derivação da Arquitetura do Produto.

A correspondência entre características e elementos arquiteturais na modelagem do CK pode ser *um-para-um*, *um-para-muitos* ou *muitos-para-muitos*. Em um relacionamento um-para-um, uma característica corresponde exatamente a um elemento arquitetural. Por exemplo, uma característica representando o recurso TV pode ser associada diretamente, na visão arquitetural, a um componente que implementa a TV.

Em um relacionamento um-para-muitos, uma característica corresponde a um ou mais elementos arquiteturais. Por exemplo, uma Balança pode ser uma característica associada a um componente correspondente à Balança e também a um conector que implementa o protocolo de comunicação da Balança (*e.g.*, *WiFi*, *Bluetooth*), ou seja, a dois elementos da arquitetura.

Em um relacionamento muitos-para-muitos, uma ou mais características correspondem a um ou mais elementos arquiteturais. Por exemplo, duas características, uma representando o sensor de pressão arterial e outra o protocolo de comunicação (*e.g.*, *WiFi*),

podem ser associadas a apenas um componente correspondente ao sensor de pressão arterial com *WiFi* integrado.

Na abordagem proposta nesta tese, o CK recebe a denominação de Modelo de Configuração. Ele associa características e elementos arquiteturais por meio de Contratos [84, 16, 15], elementos de primeira ordem no modelo e estruturados para descrever as variabilidades da Arquitetura da LPS (Figura 2.8). O Modelo de Configuração inclui, ainda, um conjunto de informações usadas para a derivação e para a adaptação dinâmica das Arquiteturas dos Produtos. O Capítulo 4 apresenta detalhes sobre a estruturação dos Contratos.

Os Modelos de Características, de Arquitetura e de Configuração, podem ser integrados em um metamodelo, mostrado na Figura 2.8, o qual forma a base do Metamodelo de Configuração de Variabilidades em LPS, apresentado em detalhes no Capítulo 5.

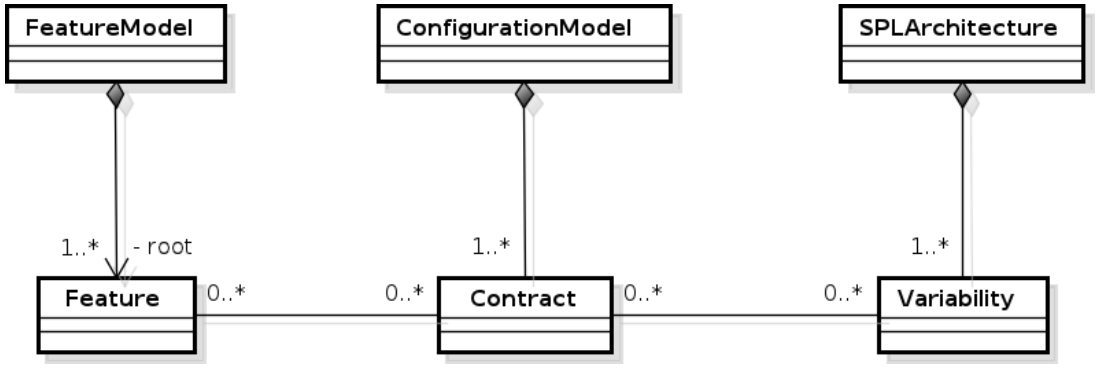


Figura 2.8: Metamodelo integrando Características, Arquitetura da LPS e Configuração.

2.2.4 Modelagem de Variabilidades

Entre as mais difundidas propostas para a modelagem de variabilidades, estão aquelas voltadas a características e aquelas voltadas à arquitetura [34]. Algumas das principais propostas são apresentadas a seguir.

O trabalho de Kang *et al.* [69], em parte apresentado na Subseção 2.2.1, é a primeira técnica a aplicar o conceito de características. No método proposto, as características podem representar, em um nível alto de abstração, as diferenças entre os produtos da LPS; o Modelo de Características inter-relaciona estas características com o objetivo de gerenciar tais diferenças.

Como uma extensão à técnica proposta por Kang *et al.*, os trabalhos de Czarnecki *et al.* [46, 45] trouxeram maior expressividade e formalismo aos Modelos de Características.

Em [46], os autores propõem, além da extensão em relação à definição de cardinalidades para os grupos de características, a noção de características solitárias, definidas como aquelas que não integram os grupos. O metamodelo apresentado na Figura 2.4 emprega o conceito de cardinalidade e também o de características solitárias, as quais podem ser modeladas sob a forma de subcaracterísticas.

Em [45] foi proposto o conceito de configuração em estágios (do inglês, *staged configuration*), um processo em que a derivação de um produto da LPS pode ser feita em estágios, onde o Modelo de Características inicial é especializado em outro, ainda contendo características opcionais e alternativas a serem selecionadas. Essa especialização pode ser sucessivamente realizada até a configuração final do produto. A cada estágio é gerado um Modelo de Características especializado, onde o conjunto de produtos descritos pelo modelo é um subconjunto de produtos descritos no Modelo de Características original. Essa abordagem é útil quando se precisa, em vez de derivar um produto final para um cliente, gerar um conjunto de características e conceitos mais especializados, de tal forma que este ainda consiga resolver variabilidades e efetivamente derivar os seus produtos finais. A noção de configuração em estágios inspirou a estruturação da Configuração de Características representada no Metamodelo proposto nesta tese (veja Capítulo 5). Como descrito na Subseção 5.3.2, a Configuração de Características de um produto pode ser gerada contendo, além das características selecionadas durante a derivação do produto (variabilidades resolvidas), também as características opcionais e alternativas que representam as variabilidades ainda a serem resolvidas em tempo de execução do produto.

Diferentemente da abordagem orientada a características, há propostas onde a modelagem de variabilidades é feita apenas em termos da Arquitetura da LPS, sem o emprego de características. Nesse sentido, Asikainen *et al.* [7, 6] propõem o Koalish, uma linguagem de modelagem para expressar LPS suportando a configuração automática de produtos. A linguagem estende a ADL Koala [130], adicionando mecanismos explícitos de modelagem de variabilidade. O Koalish não define pontos de variação, mas sim expressões de restrição que definem os tipos dos componentes e das interfaces arquiteturais que podem fazer parte da configuração de um produto. Uma configuração é um conjunto de instâncias destes tipos, sendo válida apenas se satisfizer as restrições definidas no modelo. O Metamodelo proposto nesta tese inclui a representação de tipos e instâncias para a Arquitetura da LPS, assim como a representação da configuração arquitetural dos produtos.

Outra abordagem voltada para modelar variabilidades em Arquiteturas de LPS está no trabalho de Dashofy *et al.* [47], onde a arquitetura e as suas variabilidades são modeladas

usando xADL 2.0, uma ADL construída com base em esquemas XML. Um esquema XML indica os pontos de variação da arquitetura, onde a estrutura pode variar incluindo-se ou excluindo-se um grupo de elementos. Variantes indicam pontos na arquitetura onde uma de várias alternativas podem ser substituídas por um elemento ou grupo de elementos. Cada elemento opcional ou variante é modelado associando uma condição de guarda para determinar a sua inclusão ou exclusão. Em relação à sua estrutura, a Arquitetura da LPS é descrita em termos dos componentes, dos pontos de conexão destes componentes, denominados interfaces, e dos conectores interligando as interfaces. Os elementos da arquitetura podem ter ainda a descrição do seu tipo. Muitos destes elementos têm representação no metamodelo da Figura 2.6: Pontos de Variação com seus elementos Variantes; Opcionalidades com seus elementos Opcionais; Componentes; Conectores; e Portas, as quais representam as interfaces.

Modelos de variabilidades também têm sido investigados no contexto da UML. A modelagem de características, por exemplo, pode ser realizada estendendo a especificação UML a fim de incorporar os conceitos de características. Isso é feito usando metamodelagem, onde características podem ser modeladas como metaclasses definidas com estereótipos, tais como <<*common feature*>>, <<*optional feature*>>, <<*alternative feature*>> [61]. Em [133], Weber e Gomaa propõem o *Variation Point Model* (VPM), onde variabilidades são definidas por meio de pontos de variação, os quais especificam as localizações onde podem ser introduzidas. VPM emprega modelos UML para descrever variabilidades e consiste de quatro visões (ou *views*): visão de requisitos, visão de componentes, visão estática e visão dinâmica. A visão de requisitos é definida em termos de casos de uso rotulados com o estereótipo <<*kernel*>> para representar os mandatórios, <<*optional*>> para representar os opcionais e <<*alternative*>> para representar os casos de uso alternativos. A visão de componentes mostra todos os pontos de variação associados a um componente em particular. A visão estática explora o Diagrama de Classes da UML identificando as classes por meio de estereótipos como <<*optional*>> e <<*variant*>>. Por fim, a visão dinâmica mostra o impacto que características opcionais e alternativas podem causar quando configuradas na Arquitetura da LPS. Outro trabalho bastante difundido é o de Halmans e Pohl [65] que também estende o metamodelo da UML adicionando estereótipos para representar variantes. De forma geral, o conceito de metamodelagem vem sendo bastante utilizado em LPS [8, 43, 124, 126, 115, 98].

Assim como estes trabalhos, o Metamodelo proposto nesta tese foi desenvolvido usando conceitos de metamodelagem. A sua construção segue o Ecore [120], um meta-metamodelo definido em um nível mais alto que a especificação UML. A Seção 2.5 apresenta uma visão

geral sobre metamodelagem e Ecore.

2.3 Variabilidades Dinâmicas

Tradicionalmente, as LPSs são modeladas com variabilidades estáticas [76, 99, 129, 39], ou seja, aquelas cujo tempo de vinculação (do inglês, *binding time*) é definido como *derivação* ou *implantação* [63]. Tais variabilidades somente podem ser resolvidas durante a derivação ou implantação de um produto, significando que, uma vez resolvidas e a Arquitetura do Produto implantada, esta não pode ser adaptada em tempo de execução.

Em outras palavras, não é possível reconfigurar a arquitetura deste produto por conta de mudanças do contexto, falhas, ou ainda por novas necessidades dos usuários. As mudanças não são possíveis nem mesmo se as características estiverem disponíveis no Modelo de Características e no Modelo da Arquitetura definidos em tempo de desenvolvimento da LPS. Isso ocorre, pois cada produto é originado a partir de escolhas feitas em tempo de derivação, não oferecendo possibilidade de mudança. A solução para casos onde as mudanças são necessárias passa pela parada total do produto (interrupção de sua execução), seguida pelo desenvolvimento e/ou implantação de um novo produto, a partir de uma nova resolução de variabilidades, e pela consequente configuração de uma nova arquitetura. Esse processo demanda tempo e, em certas situações, pode gerar riscos para os usuários de um sistema, como, por exemplo, para pacientes que precisam ser monitorados de forma contínua.

Existem, entretanto, LPSs onde a arquitetura precisa suportar variabilidades dinâmicas, ou seja, variabilidades que podem ser resolvidas em tempo de execução. Isso requer a adaptação dinâmica da arquitetura conforme as mudanças de requisitos [63]. Por exemplo, caso os sensores da Figura 2.7 sejam modelados como variabilidades dinâmicas, o profissional de saúde, ao perceber alguma mudança de saúde do paciente monitorado, pode indicar que ele use um novo dispositivo para medir sua frequência cardíaca periodicamente. O produto, originalmente configurado com o sensor de pressão arterial, pode ter sua arquitetura adaptada dinamicamente para atender a essa nova demanda. Outros exemplos de requisitos de adaptação, no contexto do SCIADS, são discutidos no Capítulo 3.

O suporte a variabilidades dinâmicas em Arquiteturas de LPS tem sido bastante investigado, originando as Linhas de Produto de Software Dinâmicas (LPSD) [63, 64], as quais integram os conceitos de LPS e Sistemas Adaptativos. Em uma LPSD, os pro-

dutos derivados podem ser reconfigurados em tempo de execução [3, 12], tendo as suas variabilidades resolvidas após a sua implantação. Assim, a vinculação dos elementos arquiteturais correspondentes a uma variabilidade não necessariamente deve ocorrer durante o desenvolvimento ou a implantação de um produto, podendo ocorrer também enquanto o produto está em execução.

As próximas subseções abordam a relação entre LPS e Sistemas Adaptativos, e apresentam as mais importantes abordagens que lidam com variabilidades dinâmicas em LPS.

2.3.1 Linha de Produto de Software e Sistemas Adaptativos

LPS e Sistemas Adaptativos, em princípio áreas bastante distintas, têm sido integradas por conta das similaridades em torno do gerenciamento de variabilidades [3, 12]. De um lado estão as LPSs, que empregam a noção de variabilidade visando, principalmente, o reuso de artefatos de *software* em domínios específicos e a redução de custos; de outro estão os Sistemas Adaptativos, com técnicas voltadas à concepção de sistemas mais dinâmicos e flexíveis que se adaptem em tempo de execução.

Um Sistema Adaptativo, quando em execução, pode gerar diferentes configurações arquiteturais, diante, principalmente, de duas situações:

- Mudanças do contexto de execução do sistema;
- Mudanças dos requisitos do usuário.

Na primeira situação, a decisão de gerar uma nova configuração é automática e depende do processamento das informações e das regras de contexto previamente modeladas. A definição de contexto, no escopo desta tese, está relacionada àquela proposta por Abowd *et al.* [1] e Dey [48]: qualquer informação que possa ser usada para caracterizar a situação de uma pessoa, lugar ou objeto relevante para a interação entre um usuário e uma aplicação, incluindo estes dois últimos.

Sistemas Adaptativos com a propriedade de identificar as informações de contexto e decidir automaticamente pela sua reconfiguração são caracterizados como autoadaptativos (do inglês, *Self-Adaptive Systems*). Ao detectar que as condições das regras são satisfeitas, o sistema realiza a adaptação, caracterizada pela reconfiguração de sua arquitetura [101], incluindo a verificação da qualidade e da disponibilidade dos recursos presentes no contexto de execução do sistema. Tais sistemas são também ditos Sensíveis ao Contexto,

pois utilizam as informações de contexto para fornecer serviços ou informação relevante conforme a atividade do usuário [1].

Mudanças dos requisitos do usuário durante a execução do sistema, por sua vez, caracterizam os Sistemas Adaptativos cuja adaptação é orientada ao usuário (*User-Driven Adaptation Systems*). Neste caso, a adaptação é iniciada por meio da intervenção do próprio usuário, fazendo com que o sistema reconfigure sua arquitetura para atender à mudança do requisito.

Em ambos os tipos de sistemas, a execução pode ser vista como uma máquina de estados [16, 137], onde os estados representam as diferentes configurações do sistema, e as transições representam os diferentes caminhos entre as possíveis configurações. Assim, o processo de reconfiguração é realizado identificando-se uma determinada configuração (estado) por meio das transições, às quais são associadas as condições (*e.g.*, informações e regras de contexto, preferências do usuário) que definem quando o sistema deve passar de uma configuração para outra.

Um desafio para este tipo de sistema é a identificação de todas as possíveis configurações que um sistema pode alcançar, e como modelar essas possibilidades [96]. Um problema que pode ocorrer é o alcance de uma configuração inconsistente, produzindo situações de risco para os seus usuários, como a parada parcial ou mesmo total da operação do sistema. Nesse sentido, o emprego de técnicas de LPS se faz interessante, visto que a modelagem de LPS torna possível a identificação prévia (antes da implantação) de todos os produtos que podem ser derivados da LPS [109]. Isso é significativo para um Sistema Adaptativo, pois cada produto da LPS corresponde a uma configuração possível que o sistema pode alcançar enquanto está em execução. Com essa perspectiva, um sistema pode ser modelado definindo os elementos que variam de uma configuração para outra, ou seja, entre as transições. Estes elementos correspondem às variabilidades dinâmicas da LPS.

Há uma variedade de trabalhos que empregam as notações e formalismos próprios da área de LPS na construção de Sistemas Adaptativos [125, 80, 96, 97, 30, 31, 32, 105, 64, 12, 135, 128]. Eles representam algumas das principais iniciativas em termos de gerenciamento de variabilidades dinâmicas. Os trabalhos estão divididos em dois grupos: aqueles cuja abordagem é orientada a característica, onde o desenvolvimento e a configuração do produto são feitos com base no Modelo de Características, e aqueles cuja abordagem é centrada na arquitetura [34], em que o desenvolvimento e a configuração do produto são feitos lidando com a Arquitetura da LPS, prescindindo do Modelo de Características.

A abordagem proposta nesta tese pode ser classificada como orientada a características, apesar de possibilitar que o desenvolvimento e a configuração de produtos também sejam centrados na arquitetura. A Tabela 2.1 identifica os trabalhos, os quais são detalhados nas duas próximas subseções.

Tabela 2.1: Abordagens que lidam com variabilidades dinâmicas.

	Abordagens
Orientadas a Característica	Trinidad *et al.* [125], Lee e Kang [80], Morin *et al.*[96, 97], Cetina *et al.* [30, 31, 32], Parra *et al.* [105, 106].
Centradas na Arquitetura	Hallsteinsen *et al.* [64], Bencomo *et al.* [12], White *et al.* [135], van der Hoek [128].

2.3.2 Abordagens Orientadas a Característica

Trinidad *et al.* [125] propõem o uso do Modelo de Características no sentido de modelar os estados de um produto. A modelagem de características é construída e mapeada para uma arquitetura, na qual um componente denominado *Configurator* é definido com o objetivo de fornecer um comportamento dinâmico ao produto. Este componente conhece o Modelo de Características e decide quais delas devem ser ativadas (selecionadas) ou desativadas, definindo implicitamente um CK, uma vez que as mapeia em uma relação um-para-um com os componentes da arquitetura.

Na proposta de Lee e Kang [80], o Modelo de Características é dividido em unidades denominadas FBU (*Feature Binding Unit*). Cada unidade é constituída por um conjunto de características de alguma forma inter-relacionadas. Depois de agrupadas, o tempo de vinculação é definido, entre três alternativas: desenvolvimento, instalação e operação do produto. A reconfiguração é feita com base nos tempos de vinculação definidos para as características agrupadas. Um reconfigurador, apresentado na forma de um modelo conceitual, também é proposto, com as funções de monitoramento e gerenciamento da configuração de produtos.

Morin *et al.* [96, 97] combinam técnicas de metamodelagem com orientação a aspectos para especificar e executar os sistemas dinamicamente. Eles definem um modelo para descrever variabilidades, com propriedades que se assemelham ao Modelo de Características, e um modelo orientado a aspectos para associá-las a um modelo de componentes do sistema. Aspectos são usados para descrever as adaptações que devem ser realizadas no

modelo de componentes, especificando quais variantes devem ser selecionadas de acordo com o contexto da aplicação em execução. Um dos principais focos da abordagem é tratar da questão da explosão do número possível de configurações de um sistema durante a sua execução. A modelagem orientada a aspectos é usada para derivar um conjunto de configurações, aplicando-se aspectos ao modelo, em um processo denominado *aspect weaving* [97]. As configurações geradas neste modelo são então usadas para adaptar o sistema automaticamente.

Em [30, 31, 32], Cetina *et al.* propõem uma abordagem onde diagramas de características são usados para descrever variabilidades funcionais de uma *smart home*. Cada característica é refinada em termos de componentes e canais de comunicação usando um operador denominado *superimposition*, o qual retorna os elementos arquiteturais relacionados à característica, em um mapeamento um-para-muitos entre uma característica e os elementos arquiteturais correspondentes. Uma característica quando ativada (selecionada) executa um plano de reconfiguração pré-determinado para reconfigurar a arquitetura. Para isso, no entanto, condições de contexto devem ser definidas no próprio Modelo de Características, onde cada característica deve ser diretamente rotulada com as condições que determinam quando ela deve ser ativada ou não.

Em [105], Parra *et al.* apresentam uma abordagem para derivar Arquiteturas de Produtos a partir da Configuração de Características, combinando técnicas de metamodelagem e orientação a aspectos. Cada característica selecionada adiciona elementos à Arquitetura do Produto, representados na forma de modelos de aspectos, os quais contêm, além dos elementos a serem adicionados à arquitetura, o conjunto de modificações a serem feitas para adicionar esses elementos. Há modelos para representar dados relacionados à adaptação dinâmica: informações de contexto, local onde a adaptação deve ocorrer e a modificação que deve ser realizada. O Modelo de Características é representado por um metamodelo e contém as restrições de dependência. Regras de contexto são usadas para especificar o comportamento do sistema e reconfigurar a arquitetura.

2.3.3 Abordagens Centradas na Arquitetura

O trabalho de Hallsteinsen *et al.* [64, 63] foi um dos primeiros no sentido de se construir Sistemas Adaptativos a partir das técnicas de LPS. Com uma abordagem arquitetural, eles propõem que as aplicações tenham uma representação de sua arquitetura em tempo de execução, para que uma plataforma distinta possa realizar as adaptações. Uma aplicação é modelada em termos de uma arquitetura de referência (componentes, conectores

e portas) e em termos de suas variabilidades. Variantes são escolhidas de acordo com o contexto. Empregam anotações associadas às portas dos componentes para representar as propriedades relativas à qualidade de serviço, tais como tempo de resposta, latência, enlace de comunicação. Uma função utilidade é usada para combinar essas propriedades com as necessidades do usuário. A plataforma de adaptação fornece um modelo conceitual e uma arquitetura de referência para aplicações adaptativas. As regras para permitir a representação em tempo de execução e para habilitar a reconfiguração dinâmica são construídas na própria arquitetura de referência da aplicação.

Também com uma abordagem arquitetural, Bencomo *et al.* [12] especificam o comportamento do sistema e a reconfiguração de sua arquitetura por meio de regras de contexto, definidas junto a um modelo de variabilidades denominado Modelo de Variabilidades Ortogonal, proposto por Pohl *et al.* [109]. Com a modelagem de variabilidades de LPS, eles definem como sistemas podem se adaptar em tempo de execução às mudanças do ambiente, representando as suas configurações como máquinas de estado. A variabilidade é estruturada em duas dimensões, estrutural e de ambiente. Esta última define as condições nas quais o sistema deve se adaptar (contexto). Usam um *middleware* para prover o suporte à variabilidade dinâmica.

A partir de uma Arquitetura de LPS desenvolvida para dispositivos móveis, White *et al.* [135] apresentam uma abordagem para a seleção de variantes tendo como critérios propriedades não funcionais, tais como o sistema operacional utilizado, o total de memória e o armazenamento. Os autores apresentam uma ferramenta denominada Scatter que seleciona variantes dinamicamente a partir da entrada dos requisitos de construção da Arquitetura da LPS e dos recursos disponíveis de um dispositivo móvel descoberto. A partir das propriedades não funcionais, a ferramenta seleciona o componente de *software* mais adequado ao dispositivo e faz o *download*. A seleção de variabilidades é feita no nível da arquitetura. Na Arquitetura da LPS são definidas as regras de composição, e um modelo é necessário para analisar os requisitos não funcionais de uma variante, a fim de evitar a seleção de variantes que atendem às regras de composição, mas cujos requisitos funcionais falham por causa da infraestrutura incompatível ou insuficiente. Os requisitos não funcionais são associados a cada componente da Arquitetura da LPS.

Voltado especificamente para o desenvolvimento de Arquiteturas de LPS, o trabalho de van der Hoek [128] apresenta uma abordagem para o gerenciamento de variabilidades modeladas com diferentes tempos de vinculação, às quais o autor chama de *any-time variability*. As variabilidades são modeladas na arquitetura associando uma condição de

guarda aos componentes arquiteturais, fazendo com que sejam selecionados ou não de acordo com a condição definida. A arquitetura e as suas variabilidades são modeladas usando xADL 2.0, uma ADL construída com base em esquemas XML, e um conjunto de ferramentas [47, 60]. Em termos de variabilidades dinâmicas, ele propõe que a reconfiguração em tempo de execução seja realizada por um componente responsável por detectar o contexto corrente e selecionar a configuração mais apropriada conforme o contexto. Uma vez selecionada, a configuração é instanciada usando um processo de versionamento, onde a configuração a ser instanciada é comparada com a configuração ativa. A questão da variabilidade dinâmica é vista sob a perspectiva de Gerenciamento de Configurações [40], com múltiplas versões sendo mantidas do mesmo produto em execução.

2.4 Discussão

Diante das abordagens apresentadas, três pontos são relevantes aos objetivos desta tese, sendo que os dois primeiros se referem diretamente aos *Pontos* destacados no Capítulo 1, Seção 1.2.

- A descrição de variabilidades estáticas e dinâmicas no nível da Arquitetura da LPS;
- Modelo para o CK;
- A representação usada para a Arquitetura da LPS.

2.4.1 Variabilidades no Nível da Arquitetura

A Arquitetura da LPS representa as arquiteturas de todos os seus produtos por meio da descrição de variabilidades, como apresentado na Seção 2.2.2. Na proposta desta tese, as variabilidades são descritas na forma de Contratos [84, 16, 15], os quais definem ações de adaptação arquitetural, como detalhado no Capítulo 4. Mais especificamente, eles permitem a descrição de variabilidades estáticas e variabilidades dinâmicas de forma integrada e no nível da Arquitetura da LPS.

As abordagens apresentadas não permitem descrever, de uma maneira integrada, as variabilidades estáticas e dinâmicas no nível da arquitetura. Há abordagens que descrevem variabilidades estáticas usando ADLs [6, 47, 118, 60, 130], as quais, em geral, fazem a distinção entre os elementos mandatórios e as variabilidades usando pontos de variação descritos na forma de expressões de restrição (*e.g.*, condição de guarda).

Entre as abordagens que suportam variabilidades dinâmicas, o trabalho de van der Hoek [128] permite a descrição, também, de variabilidades estáticas, por meio da ADL xADL 2.0 [47, 60]. No entanto, este trabalho não explora como as informações de contexto são identificadas, e na representação das variabilidades mostrada no trabalho, não há elementos que identificam como estas informações influenciam na configuração dinâmica do sistema. No Metamodelo proposto nesta tese, as informações de contexto são modeladas para representar elementos e entidades de contexto, e de uma forma que possam ser usadas na definição de regras que influem na adaptação dinâmica da Arquitetura do Produto. Além do mais, a abordagem proposta em [128] é centrada em arquitetura (veja Tabela 2.1), enquanto a proposta desta tese está voltada, principalmente, para LPSs orientadas a característica (veja Capítulo 5, Seção 5.2).

De qualquer modo, misturar diferentes abordagens durante o desenvolvimento da Arquitetura da LPS pode causar um esforço extra, pois uma abordagem para descrever variabilidades estáticas pode não ser apropriada para descrever variabilidades dinâmicas, e vice-versa. Outra dificuldade está na descrição de variabilidades cujo tempo de vinculação é decidido no momento da derivação da Arquitetura do Produto. Este é o caso de algumas variabilidades do SCIADS (veja Capítulo 3), que precisam ser descritas de uma forma que possam ser tratadas como estáticas ou como dinâmicas. Por exemplo, o ponto de variação da arquitetura da Figura 2.7 pode ser tratado como uma variabilidade estática que, uma vez resolvida, vincula estaticamente o elemento da arquitetura selecionado (*e.g.*, *BloodPressureSensor*), ou como uma variabilidade dinâmica, onde os dois elementos envolvidos no ponto de variação podem ser vinculados dinamicamente durante a execução do produto.

2.4.2 *Configuration Knowledge*

Na proposta desta tese, o *Configuration Knowledge* (CK) recebe a denominação de Modelo de Configuração, o qual é composto de Contratos e de um conjunto de informações, tais como restrições de dependência e regras de contexto. Os Contratos associam características a elementos arquiteturais, e guiam a derivação e a adaptação dinâmica das Arquiteturas dos Produtos. O Modelo de Configuração é definido de forma explícita e independente dos Modelos de Características e Arquitetura da LPS (veja Figura 2.8).

Em muitas abordagens, o CK, por outro lado, é descrito de forma implícita, como em [9], onde os autores assumem um mapeamento um-para-um entre características e elementos arquiteturais, sem descrever explicitamente um modelo para o CK, mas apenas

relacionando as características a módulos de *software* com o mesmo nome da característica.

Ainda de forma implícita, os componentes da arquitetura podem ser implementados com algum tipo de parametrização de acordo com as características. Kang *et al.* [69], Trinidad *et al.* [125] e Lee e Kang [80] apresentam técnicas para se realizar essa parametrização. Pode-se ainda descrever a Arquitetura da LPS anotando as características em cada componente, ou ainda, expressões que permitam selecionar os componentes de acordo com regras, como realizado em [6, 47, 60, 128]. Caso as condições definidas nas expressões sejam satisfeitas, os respectivos elementos arquiteturais são selecionados e incluídos à Arquitetura do Produto.

Essas soluções implícitas para o CK, no entanto, podem poluir os modelos, especialmente o da Arquitetura da LPS, com informações não diretamente relacionadas às suas funcionalidades. Por exemplo, na própria arquitetura da Figura 2.7, uma condição de guarda pode ser descrita para cada elemento variante que compõe o ponto de variação. O elemento *BloodPressureSensor* pode ter a seguinte condição de guarda,

$$HealthProblem = "hypertension"$$

denotando que ele deve ser selecionado para a Arquitetura do Produto que esteja sendo criada para um paciente cujo problema de saúde seja hipertensão.

O uso de expressões traz problemas relacionados à falta de encapsulamento, uma vez que a modelagem de um determinado produto requer a verificação de diversos pontos de variação, os quais podem estar espalhados por toda a Arquitetura da LPS na forma de expressões. Uma condição de guarda, como a mostrada anteriormente, precisa ser descrita para cada problema de saúde de um paciente específico e, além disso, associada a cada um dos elementos arquiteturais (*e.g.*, sensores médicos) que devem fazer parte da Arquitetura do Produto voltado para pacientes com aquele problema de saúde (*e.g.*, hipertensão). Com isso, a mesma condição de guarda se espalha por vários elementos da Arquitetura da LPS e, dependendo da quantidade de guardas modelados e de como eles estejam entrelaçados em fórmulas mais complexas, pontos de variação individuais podem se tornar extremamente difícieis de ser rastreados. Consequentemente, essa abordagem pode dificultar que a evolução das variabilidades, representadas pelos pontos de variação, ocorra independentemente da arquitetura, e vice-versa.

Poucas abordagens que suportam variabilidades dinâmicas (veja Tabela 2.1) se preocupam em representar o CK. Trinidad *et al.* [125] definem um CK implícito, mapeando

características e componentes da arquitetura definindo interfaces e parâmetros. Em [128], van der Hoek representa as informações de configuração como guardas identificados junto aos próprios componentes da arquitetura. White *et al.* [135] mantêm as regras de composição na própria Arquitetura da LPS, enquanto Parra *et al.* [105, 106] definem modelos de aspectos associados a características e a elementos da arquitetura, não mantendo nestes modelos as restrições de dependência entre características.

Ainda com relação às abordagens que suportam variabilidades dinâmicas, muitas não deixam explícito como e onde são modeladas as informações e regras de contexto, essenciais em aplicações dinâmicas. Em [128, 135] não são detalhados como o contexto é representado, e em [125] isso não é citado. Cetina *et al.* [32, 33] representam contexto no próprio Modelo de Características. Outros usam uma plataforma ou *middleware* para representar o contexto e prover o suporte à variabilidade dinâmica, como é o caso de [12, 64, 105]. Os trabalhos [81, 80] indicam o uso de um reconfigurador que monitora e gerencia a configuração.

2.4.3 Representação da Arquitetura

A representação da arquitetura empregada nesta tese contém componentes, conectores e portas, como apresentado no Metamodelo de Arquiteturas de LPS da Figura 2.6, Seção 2.2.2. Esta representação possui semelhanças com aquelas usadas por Dashofy *et al.* [47], van der Hoek [128] e Hallsteinsen *et al.* [64]. O Capítulo 5 apresenta o Metamodelo da Arquitetura completo, incluindo a representação de tipos, instâncias e a forma com que a interligação entre componentes e conectores é realizada.

Entre as abordagens centradas na arquitetura, a representação arquitetural possui variações. Em [128], van der Hoek usa uma ADL na descrição de arquiteturas, as quais são modeladas usando componentes e conectores. Hallsteinsen *et al.* [64] empregam também este estilo, mas não usam uma ADL para descrever a arquitetura, mas sim um modelo de variação onde entidades interagem umas com as outras fornecendo e usando serviços por meio de portas. A abordagem de White *et al.* [135] é baseada na composição de componentes para a formação da arquitetura, utilizando uma ferramenta visual de modelagem para capturar as variabilidades e definir requisitos não funcionais. Por último, Bencomo *et al.* [12] usam um modelo de variabilidades estrutural descrito com uma linguagem que, segundo os autores, é essencialmente uma ADL personalizada para o *middleware* adaptativo usado no suporte à reconfiguração.

As abordagens orientadas a características empregam uma representação com poucas

variações em relação ao Modelo de Características apresentado na Seção 2.2.1. Em termos de representação da arquitetura, no entanto, há diferenças entre as abordagens.

Cetina *et al.* [32, 33] empregam uma arquitetura onde os componentes são interligados pelo que chamam de canais de comunicação. Em Trinidad *et al.* [125], a arquitetura é baseada em componentes e modelada com o Diagrama de Componentes da UML. Lee e Kang [80] usam o estilo arquitetural C2 [90] para a definição da arquitetura. Morin *et al.* [96] descrevem arquiteturas baseadas em componente e definem um metamodelo para representá-las. Parra *et al.* [105, 106] representam a arquitetura com artefatos que correspondem a modelos parciais do produto, os quais são usados para gerar o produto a partir da Configuração de Características. A arquitetura segue um modelo orientado a serviços.

2.5 Uma Visão Geral de Metamodelagem

O Metamodelo de Configuração de Variabilidades em LPS, proposto nesta tese e apresentado no Capítulo 5, foi construído usando o EMF[2] [120, 95]. A partir do Metamodelo é possível instanciar Modelos de Características, Modelos de Arquitetura e Modelos de Configuração. Este último permite a derivação e a adaptação dinâmica de Arquiteturas de Produtos. Com o Metamodelo definido usando o Ecore, um meta-metamodelo empregado pelo EMF para elaborar metamodelos, os modelos podem ser instanciados a partir da geração de uma API e portados para XMI (XML *Metadata Interchange*)[3], assim como terem código-fonte em Java gerados. Isso pode ser feito usando editores especializados do EMF, ou ainda ferramentas visuais como o GMF (*Graphical Modeling Framework*)[4], para modelar graficamente a instância de um modelo específico.

A Tabela 2.2 apresenta o conceito de metamodelagem [100]. O nível *M3* define meta-metamodelos, os quais são usados para definir metamodelos, representados no nível *M2*, como, por exemplo, a especificação *UML* e o próprio *Metamodelo de Configuração de Variabilidades em LPS*, proposto nesta tese. Um exemplo de meta-metamodelo é o *MOF* (*Meta-Object Facility*)[5], padrão de metamodelagem estabelecido pelo OMG (*Object Management Group*)[6]. Outro exemplo de meta-metamodelo é o próprio *Ecore*. O Ecore surgiu como uma implementação do MOF, e evoluiu por conta da experiência resultante

[2] *http://www.eclipse.org/emf*
[3] *http://www.omg.org/specs/XMI*
[4] *http://www.eclipse.org/modeling/gmp*
[5] *http://www.omg.org/mof*
[6] *http://www.omg.org*

da construção de ferramentas [120].

Tabela 2.2: Níveis de metamodelagem. Adaptada a partir da Arquitetura de Metadados MOF [100].

Nível	Descrição	Elementos
M3	**Meta-metamodelos** (*e.g.*, MOF; *Ecore*).	Classes, atributos e associações MOF e *Ecore*.
M2	**Metamodelos** (*e.g.*, UML; *Metamodelo de Configuração de Variabilidades em LPS*).	Classes, atributos e associações da UML e do *Metamodelo de Configuração de Variabilidades em LPS*.
M1	**Modelos** (*e.g.*, Diagrama de Classe da UML; *Modelo de Características*, *Modelo da Arquitetura*, *Modelo de Configuração*).	Classe "Paciente"; *característica BloodPressure*, *componente BloodPressureSensor*, *Contrato MedicalSensorContract*.

Os metamodelos do nível *M2* são usados para definir os modelos do nível *M1*. A partir da UML pode-se definir, por exemplo, *Diagramas de Classes* e *Diagramas de Objetos*. No caso do EMF, uma vez definidos os metamodelos a partir do Ecore, os modelos do nível *M1* podem ser instanciados e validados, ou de forma programática ou por meio de um editor, tendo como referência o metamodelo da aplicação construído. Com as ferramentas de integração do EMF, o trabalho de modelagem e o trabalho de implementação se aproximam, tornando-se partes integradas em uma mesma atividade.

Com isso, os modelos são considerados entidades de primeira ordem no desenvolvimento de aplicações. Em outros termos, os modelos não são apenas artefatos de documentação que auxiliam na atividade de desenvolvimento, mas sim parte integrante do *software* [117]. A ideia é fazer com que o arquiteto possa se concentrar em modelos de alto nível, se protegendo da complexidade inerente à implementação nas diferentes plataformas.

2.6 Considerações Finais

Os três principais conceitos envolvidos nesta tese, Linha de Produto de Software, Sistemas Adaptativos e Metamodelagem, foram inter-relacionados neste capítulo. Da mesma forma, as mais relevantes abordagens envolvendo variabilidades dinâmicas em LPS foram discutidas.

Muitas das abordagens que lidam com variabilidades dinâmicas são voltadas, principalmente, para o suporte à reconfiguração em tempo de execução, com propostas de plataformas ou mecanismos para realizar o monitoramento do ambiente e fornecer a reconfiguração automática do sistema. No entanto, elas não se dedicam diretamente a questões relacionadas à concepção de modelos, em especial modelos que integrem características, arquitetura, variabilidades estáticas e dinâmicas, e de processos de derivação e de adaptação dinâmica da arquitetura de produtos.

O próximo capítulo apresenta o SCIADS, projeto que motivou a investigação de Contratos como forma de descrever variabilidades no nível da Arquitetura da LPS e o desenvolvimento do Metamodelo de Configuração de Variabilidades em LPS.

Capítulo 3

Sistema Computacional Inteligente de Assistência Domiciliar à Saúde

Este capítulo apresenta a estruturação geral do SCIADS e o seu Modelo de Características, o qual serve de base para boa parte dos exemplos e cenários apresentados nos demais capítulos desta tese.

3.1 Introdução

O SCIADS (Sistema Computacional Inteligente de Assistência Domiciliar à Saúde) [21, 20, 23] é um sistema voltado ao telemonitoramento de pacientes domiciliares. Com uma infraestrutura formada por sensores usados pelo paciente (*e.g.*, sensor de pressão arterial, acelerômetro), sensores instalados no ambiente domiciliar (*e.g.*, sensor de temperatura, sensor de umidade), além de dispositivos de visualização (*e.g.*, TV, *tablet*), o sistema, entre outras funcionalidades, coleta dados do paciente, identifica a sua situação de saúde, e envia alertas e alarmes de emergência para uma central de supervisão.

O SCIADS tem um papel essencial no contexto desta tese: os seus requisitos de personalização e, sobretudo, as propriedades dinâmicas de sua arquitetura, ambos observados durante a pesquisa e o desenvolvimento, motivaram a investigação e a proposta desta tese [28, 27, 26, 25, 18].

Os requisitos de personalização têm relação com os conceitos de LPS, no sentido de prover e implantar produtos personalizados para cada paciente e sua residência. Cada paciente, nesse contexto, possui requisitos específicos, por exemplo, em relação aos seus problemas de saúde, ou ainda, em relação à distribuição e uso dos sensores na residência.

As propriedades dinâmicas da arquitetura, por sua vez, possuem relação com as variabilidades dinâmicas da LPS. Como o SCIADS é um sistema cuja concepção está baseada nos conceitos de computação ubíqua [134], sua arquitetura deve ser adaptável dinamicamente diante de mudanças do contexto de execução do sistema. Este cenário originou, em termos da modelagem de uma LPS para o SCIADS:

- A investigação de Contratos [83, 16] na descrição de variabilidades da Arquitetura da LPS (Capítulo 4);
- O desenvolvimento do Metamodelo de Configuração de Variabilidades em LPS, para integrar características, arquitetura e Contratos em um nível mais alto de abstração, e apoiar a derivação e a adaptação dinâmica das Arquiteturas dos Produtos (Capítulo 5).

Este capítulo está organizado como se segue.

Seção 3.2
A motivação de se desenvolver um sistema como o SCIADS e alguns requisitos básicos tratados pelos sistemas ubíquos de assistência domiciliar à saúde.

Seção 3.3
Visão geral do SCIADS, a estruturação do protótipo [23] implementado pelo nosso grupo de pesquisa, e uma discussão envolvendo alguns trabalhos e soluções relacionados.

Seção 3.4
O Modelo de Características do SCIADS e uma descrição das variabilidades do sistema, com destaque para as questões de personalização e de adaptação dinâmica.

Seção 3.5
Considerações finais.

3.2 Motivação e Requisitos Básicos

A quantidade de pessoas idosas tem crescido substancialmente nos últimos anos, além do número de pacientes com doenças crônicas, como diabetes e doenças cardíacas. Este cenário, somado ao alto custo das internações hospitalares e do gerenciamento de doenças

crônicas [119], tem aumentado a demanda por serviços de saúde, exigindo cada vez mais da atual infraestrutura existente.

A assistência domiciliar à saúde, definida como a provisão de serviços de saúde às pessoas de qualquer idade em casa ou outro local não institucional [49], é reconhecida como uma das formas de se reduzir o número de internações hospitalares [72]. Nesse sentido, o uso da tecnologia de computação ubíqua pode contribuir na implementação e melhoria desse tipo de assistência, uma vez que, por meio de sensores utilizados no ambiente domiciliar, o paciente pode ser monitorado continuamente e em qualquer parte da casa. Dados fisiológicos como pressão arterial e frequência cardíaca, atividades realizadas pelo paciente, como caminhar, dormir e comer, e condições do ambiente, como temperatura e umidade, podem ser obtidos continuamente. Cuidadores, enfermeiros e médicos têm a oportunidade de acompanhar, por meio do serviço de telemonitoramento, o dia-a-dia do paciente e a evolução do seu tratamento. Para o paciente pode significar um número menor de visitas ao consultório médico e períodos mais curtos de hospitalização [71].

Com esta motivação teve início o desenvolvimento do SCIADS, um sistema que interliga o paciente, em sua residência, aos provedores de serviços de saúde, como profissionais, prestadores de assistência domiciliar, e instituições médicas, integrando, por meio de uma infraestrutura baseada em computação ubíqua, diversos aspectos relevantes ao monitoramento remoto da saúde do paciente em seu ambiente domiciliar.

Analisando trabalhos relevantes relacionados à área de computação ubíqua na assistência domiciliar à saúde, *e.g.*, [72, 131, 102, 54], em conjunto com especialistas da área, foram identificados alguns requisitos básicos que devem ser tratados em um sistema como o SCIADS:

- Reconhecimento das atividades realizadas pelo paciente;
- Identificação da situação de saúde do paciente considerando os seus dados fisiológicos e a sua atividade;
- Aproximação do profissional de saúde e paciente, aumentando a adesão ao tratamento.

A identificação da situação de saúde se fundamenta em pesquisas voltadas à coleta de dados fisiológicos e ao reconhecimento de atividades realizadas pelo paciente [4, 35]. A importância de se considerar a atividade está relacionada à necessidade de precisão na identificação da real situação de saúde do paciente. Por exemplo, um paciente com uma

frequência cardíaca de $120bpm$ pode estar varrendo a casa, caracterizando uma atividade doméstica de esforço. Mesmo com a alta frequência cardíaca é possível, dependendo do paciente, que isso seja normal.

A adesão ao tratamento, por sua vez, refere-se ao cumprimento das prescrições médicas por parte do paciente [70, 94, 14]. Um plano de cuidados é definido para o paciente contendo as prescrições elaboradas pelo profissional de saúde, tais como: identificação das medições a serem realizadas, como pressão arterial e frequência cardíaca, assim como a sua periodicidade; medicamentos prescritos acompanhados da dosagem, horário e forma de administração; orientações de dieta e de exercícios físicos; e outras recomendações personalizadas conforme o tratamento. O aumento da adesão pode ser conseguido por meio de serviços e tecnologias inteligentes que permitam o profissional de saúde acompanhar de perto a evolução do paciente, por exemplo, recebendo continuamente informações a respeito da sua situação de saúde e do seu cumprimento do plano de cuidados. Também relevante para o aumento da adesão é o emprego de notificações ao paciente, lembrando-o de realizar as prescrições contidas no plano de cuidados [67]. Outra forma de aumentar a adesão é tornar disponíveis informações aos seus familiares, permitindo que estes se envolvam e participem do tratamento.

3.3 O SCIADS

O SCIADS é um sistema com uma estruturação flexível que pode integrar sensores, dispositivos e serviços inteligentes, visando:

- A contínua identificação da situação de saúde do paciente;
- O envio de alertas e alarmes de emergência a profissionais de saúde e/ou a uma central de supervisão;
- O envio de notificações e lembretes direcionados ao paciente e associados ao plano de cuidados, elaborado pelo profissional de saúde.

As próximas subseções apresentam uma visão geral do sistema, o protótipo e alguns trabalhos e soluções relacionadas.

3.3.1 Visão Geral do Sistema

A Figura 3.1 apresenta a visão geral de funcionamento do SCIADS.

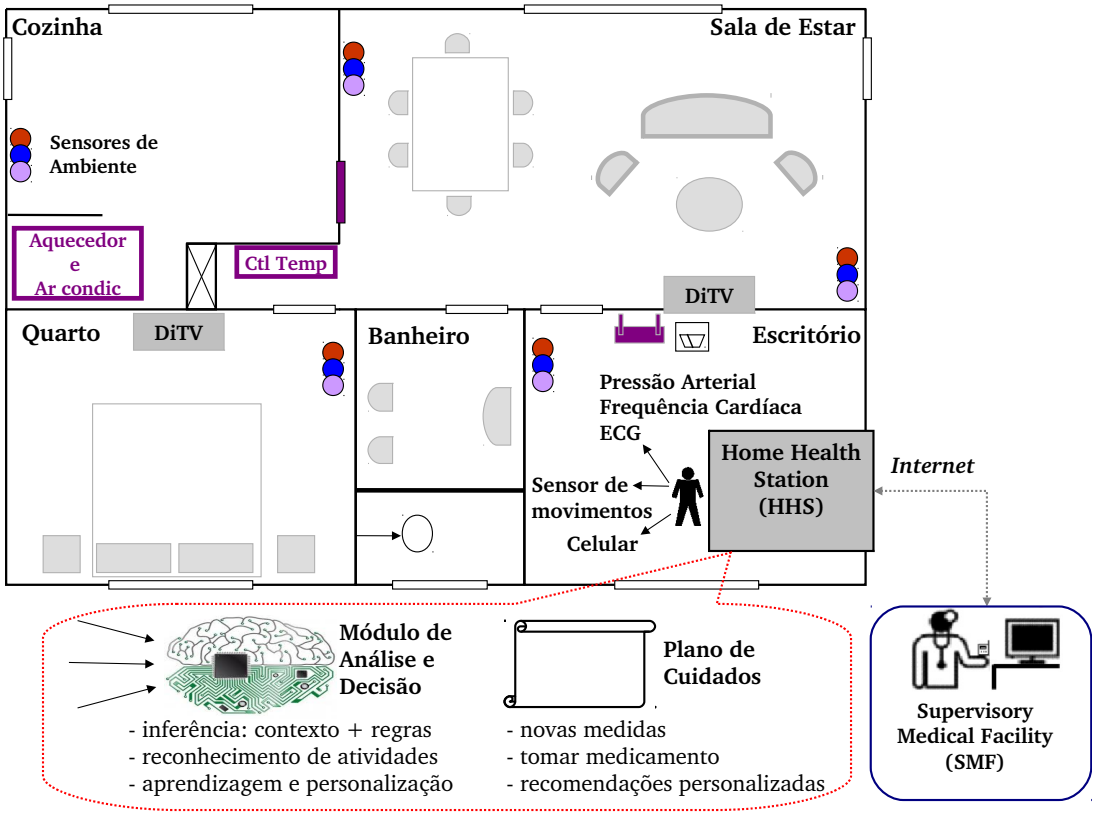


Figura 3.1: Visão geral do SCIADS

Durante o monitoramento, a *HHS* (*Home Health Station* - Central de Saúde Residencial), localizada na residência, recebe e processa os dados fisiológicos e de atividades do paciente, e os dados do ambiente, tais como temperatura e umidade do local. Dispositivos e sensores integrados à rede de comunicação residencial coletam estes dados com o objetivo de determinar a situação de saúde do paciente. A eventual identificação de anormalidade pode ativar um dispositivo (*e.g.*, *DiTV* - TV Digital) ou, dependendo da gravidade, enviar um alerta ou um alarme de emergência para a *SMF* (*Supervisory Medical Facility* - Central de Supervisão Médica).

Os dados coletados na *HHS* são representados por variáveis nebulosas e analisados pelo *Módulo de Análise e Decisão*, o qual emprega técnicas de inteligência artificial baseadas em regras médicas produzidas em cooperação com especialistas. Implementado no contexto da Tese de Doutorado de Alessandro Copetti [41], integrante do nosso grupo de pesquisa, o módulo associa as variáveis fisiológicas modeladas às variáveis de atividades e às variáveis do ambiente. Como exemplo, pode-se citar as recomendações médicas relacionadas à pressão arterial para uma pessoa que realiza atividades domésticas. Uma vez reconhecida que uma atividade doméstica esteja sendo realizada pelo paciente nos minutos precedentes à medição de sua pressão arterial, o resultado da medida com desvio um pouco superior

à média pode ser considerado normal, pois espera-se um aumento natural da pressão arterial nestas circunstâncias. O protótipo ora implementado (veja Seção 3.3.2) está voltado a pacientes com hipertensão, mas a estrutura modular do SCIADS permite sua personalização para diferentes problemas de saúde. Em princípio, para lidar com um paciente diabético, por exemplo, basta produzir um novo conjunto de regras médicas a serem processadas pelo *Módulo de Análise e Decisão*, e integrar os sensores médicos apropriados.

Para que o monitoramento tenha início, um *Plano de Cuidados* deve ser definido contendo uma série de prescrições elaboradas pelo profissional de saúde, o qual pode também, a qualquer momento, modificar o plano original, prescrevendo, por exemplo, novos horários para a medicação ou ainda nova periodicidade de uso dos dispositivos de medição.

De acordo com o conteúdo do *Plano de Cuidados*, o SCIADS pode gerar notificações ao paciente, lembrando-o de cumprir os procedimentos definidos. Lembretes para a realização de um procedimento podem ser enviados e exibidos em dispositivos configurados no sistema, tais como TV digital, aparelhos de telefonia celular ou dispositivos do tipo *tablet*. O envio destas notificações depende da configuração da *HHS* e de informações do contexto, como, por exemplo, a localização do paciente dentro de sua casa.

O SCIADS pode ainda incluir componentes relacionados à visualização e à interação com o *Plano de Cuidados* e com os dados coletados junto aos sensores. Em alguns casos, o paciente de doença crônica se sente mais motivado e com mais chance de aumentar a sua adesão ao tratamento quando tem a possibilidade de conhecer seus dados e perceber seu progresso. Nesse sentido, o sistema precisa ser configurável para permitir que estas informações sejam apresentadas ao paciente de uma forma simples e fácil de ser compreendida e assimilada, não exigindo esforço de interação.

Todos os dados coletados são armazenados na própria *HHS*, a qual mantém uma base de dados com o histórico individualizado do paciente. A base de dados pode também ser armazenada na *SMF*, ou ainda na nuvem (do inglês, *cloud*), termo relacionado ao conceito de Computação em Nuvem (do inglês, *Cloud Computing*), o qual se refere ao hardware e aos sistemas de *software* mantidos em centros de dados que fornecem serviços por meio da *Internet*[1] [5].

[1]Questões relacionadas à segurança têm sido bastante discutidas no contexto das aplicações baseadas em nuvem, especialmente das aplicações de saúde, como pode ser observado em [138]. Esta tese não aborda essas questões.

Além de armazenar e processar dados localmente, a *HHS* age como um *gateway* de comunicação entre a residência do paciente, os provedores de saúde e os seus familiares. Essa funcionalidade de comunicação permite:

- Enviar alertas e alarmes de emergência para a *SMF* ou para o profissional de saúde;
- Enviar periodicamente os dados do paciente (fisiológicos, atividades e situação de saúde) para a *SMF*;
- Receber o *Plano de Cuidados* elaborado pelo profissional de saúde;
- Disponibilizar os dados aos familiares do paciente;
- Estabelecer uma conexão direta do paciente com o profissional de saúde e com os seus familiares.

A *SMF* recebe a informação processada por várias *HHS*s e pode armazenar dados de longo prazo. Isso torna possível, com o apoio de operadores especializados, o monitoramento de diversos pacientes e outros tipos de avaliação, tais como verificação de tendências, e análises estatísticas e comparativas envolvendo dados de vários pacientes.

Determinadas funcionalidades da aplicação, como a obtenção de dados dos sensores, as notificações ao paciente, o envio de mensagens e alarmes ao profissional de saúde, e a transmissão de dados e informação de emergência da *HHS* para a *SMF*, necessitam de um suporte diversificado de comunicação. Tais funcionalidades apresentam requisitos específicos de qualidade e disponibilidade, bem como a demanda de programação e integração também específicos.

3.3.2 Protótipo

Nas fases iniciais de desenvolvimento do SCIADS, foi implementado um protótipo voltado à realização dos requisitos destacados na Seção 3.2. Com o objetivo de executar experimentos em laboratório, o protótipo[2] desenvolvido pelo nosso grupo de pesquisa [23, 52, 88] tem como foco a contínua identificação da situação de saúde do paciente associada à definição de um plano de cuidados. O protótipo coleta os dados fisiológicos e de atividades do paciente e os analisa usando o Módulo de Análise e Decisão. Para a coleta de dados foram empregados equipamentos com comunicação sem fio, como um dispositivo de medição de pressão arterial e frequência cardíaca, e sensores de movimento.

[2] Disponível no endereço *http://www.tempo.uff.br/sciads*.

A Figura 3.2 identifica os módulos de *software*, implementados com as seguintes funcionalidades:

- *Coleta* dos dados fisiológicos e de atividade por meio de dispositivos de medição e sensores;
- *Análise* dos dados a fim de identificar (decidir) a situação de saúde do paciente;
- *Armazenamento* dos dados e da situação de saúde do paciente;
- *Distribuição* dos dados e da situação de saúde do paciente para a *SMF*.

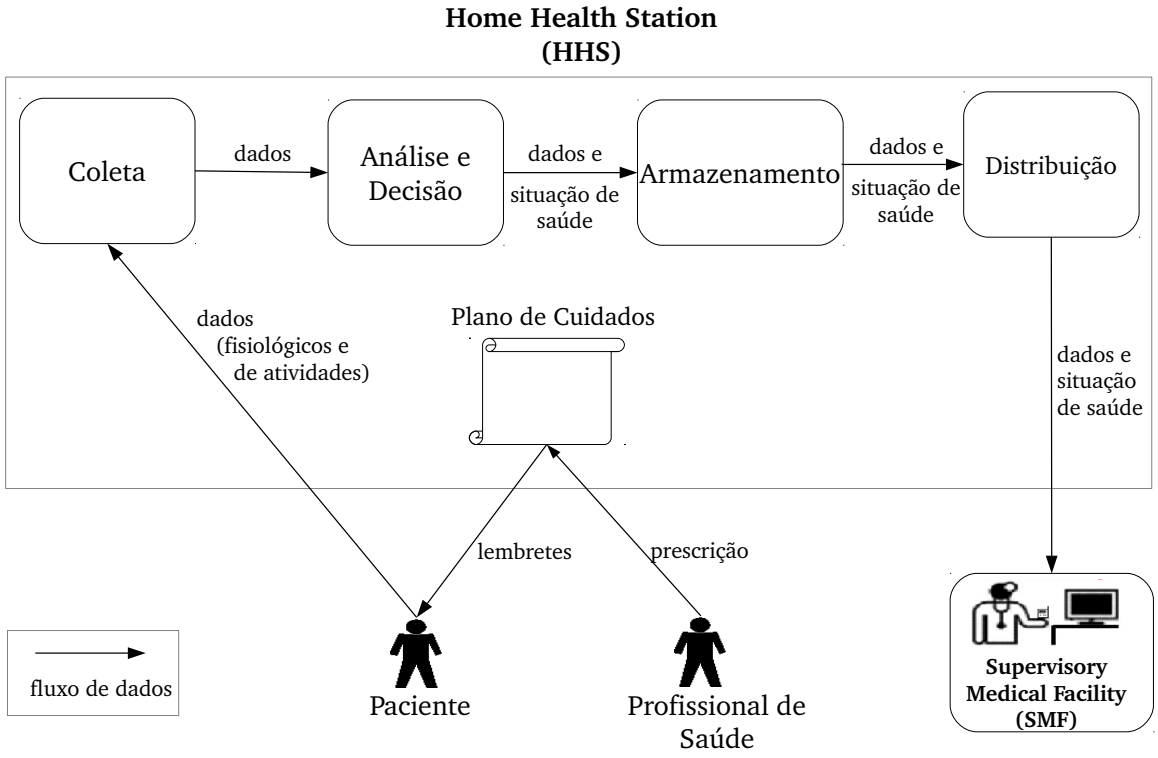
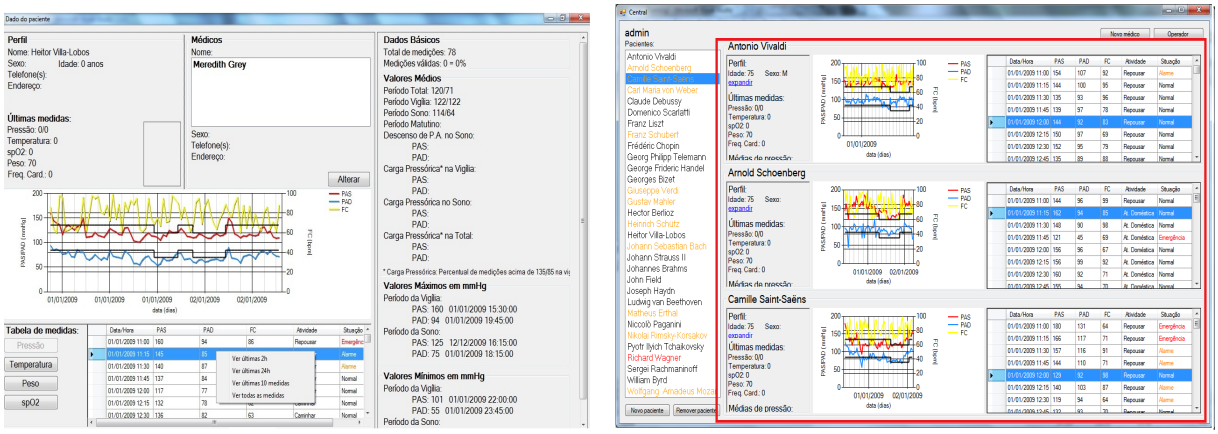


Figura 3.2: Estrutura geral do Protótipo.

Os módulos operam na *HHS* e iniciam a execução de acordo com a programação preestabelecida no *Plano de Cuidados*, o qual pode ser inserido pelo profissional de saúde. Supondo que a programação estabeleça que o paciente deve medir sua pressão arterial e frequência cardíaca duas vezes ao dia, às 8h e às 18h, o protótipo irá iniciar um ciclo de execução, em cada um destes horários.

Interfaces especializadas também estão presentes no protótipo, provendo visões dos dados apropriadas a cada um dos atores do sistema. A Figura 3.3 mostra a visualização, na *SMF*, de dados consolidados de vários pacientes obtidos a partir de experimentos

realizados com diferentes valores de pressão arterial, de frequência cardíaca e de atividades realizadas pelos pacientes. Módulos para identificação de usuários por nome e senha, e para cadastramento de pacientes, profissionais de saúde, operadores e administradores, também foram implementados. Mais detalhes sobre a construção do protótipo estão disponíveis em [23, 52, 88].



(a) (b)

Figura 3.3: Visualização na SMF: a) dados de um paciente; b) dados de vários pacientes.

3.3.3 Trabalhos e Soluções Relacionados

Diversos trabalhos têm sido desenvolvidos aplicando a tecnologia de computação ubíqua na assistência à saúde. Uma revisão da literatura na área pode ser encontrada em [102]. Em [82] é apresentado um sistema de monitoramento projetado para reagir quando limites preestabelecidos dos dados fisiológicos do paciente são atingidos. O sistema não associa os dados fisiológicos com os dados de atividade para analisar a situação do paciente, como no SCIADS, portanto ele pode não reagir diante de valores que representam riscos ao paciente.

O uso conjunto de dados fisiológicos e atividades físicas está presente em [79], que propõe uma arquitetura para aplicações ubíquas de assistência à saúde. Os autores discutem a importância da obtenção de dados e da sua análise, no entanto, não contemplam mecanismos para identificar situações críticas do paciente.

A plataforma apresentada por Chung *et al.* [37] considera medidas realizadas sobre o eletrocardiograma e as associa com as atividades físicas "caminhando" e "correndo". No entanto, ela é específica para coleta e análise de sinais de ECG, enquanto que o SCIADS permite a personalização para diferentes tipos de dados de acordo com o problema de saúde do paciente (veja o Capítulo 4).

Um sistema que integra dados fisiológicos e atividades, empregando diferentes técnicas e algoritmos para captura da atividade física do paciente, é apresentado em [51]. Diferentemente do SCIADS, a ênfase do projeto está no emprego de variadas plataformas de sensores, não fornecendo um sistema de notificações ou lembretes ao paciente.

O projeto AlarmNet [136] tem pontos em comum com o SCIADS, no entanto, possui prioridades relacionadas à coleta e processamento de dados em tempo real e componentes que possibilitam o gerenciamento eficiente de energia. Além disso, não há qualquer menção a serviços de notificações ou lembretes relacionados a um plano de cuidados.

Um aspecto importante que diferencia o SCIADS desses sistemas é a sua capacidade de integrar:

- Os requisitos próprios dessa classe de aplicações;
- A contínua identificação da situação de saúde do paciente;
- O plano de cuidados com o sistema de notificações e lembretes, serviços capazes de estabelecer uma rotina diária para o paciente, auxiliando-o na sua adesão ao tratamento;
- A flexibilidade em relação a diferentes problemas de saúde, obtida por meio do Módulo de Análise e Decisão (Seção 3.3.1).

Em termos de perspectivas, o objetivo é permitir a interoperabilidade do SCIADS com outros sistemas de informação, por meio da adoção de Padrões de Informática em Saúde. Uma forma de prover interoperabilidade semântica ao SCIADS é o uso do modelo dual, proposto pela Fundação OpenEHR[3], cuja arquitetura é análoga à da norma CEN/ISO 13606. A sua principal característica é a separação entre o modelo de persistência, conhecido como modelo de referência, e o modelo do domínio, definido por arquétipos, que são definições formais de conceitos clínicos. Com base no modelo dual, pode-se desenvolver uma representação de dados e um conjunto de arquétipos para a estruturação destes dados, assim como uma solução de persistência de dados clínicos dos pacientes, conforme está descrito em [58], um dos trabalhos do nosso grupo de pesquisa. Para a troca de informações com outros sistemas, os arquétipos podem ser integrados em mensagens HL7[4], padrão para a troca de informações entre sistemas de informação em saúde heterogêneos.

[3] *http://www.openehr.org*
[4] *http://www.hl7.org*

Alguns trabalhos têm explorado essa integração [92], a qual permite que os arquétipos possam ser processados e interpretados por qualquer sistema que use HL7.

Outro aspecto a ser investigado quanto à sua viabilidade refere-se à adoção da arquitetura de referência proposta pela Continua Health Alliance[5], uma organização sem fins lucrativos formada por um grupo de companhias, tanto da área de saúde quanto da área tecnológica. Essa arquitetura fornece uma estruturação e terminologias que permitem estabelecer diretrizes de interoperabilidade relacionadas aos dispositivos médicos [108]. A arquitetura está baseada nos padrões ISO/IEEE 11073 *Personal Health Data*[6], voltados à especificação de formatos e procedimentos de troca de dados entre os dispositivos médicos usados em uma residência, como, por exemplo, medidores de pressão arterial, glucômetros e balanças.

3.4 Linha de Produto para o SCIADS

De forma geral, sistemas da mesma classe do SCIADS têm sido desenvolvidos sem levar em conta as diferenças existentes entre os pacientes [83, 54], como, por exemplo, problemas de saúde e seus requisitos específicos. Uma solução passa pelo uso de técnicas de LPS. A concepção de uma LPS para o SCIADS torna possível a criação de diferentes produtos para diferentes pacientes, de acordo com as especificidades de cada um, e, ao mesmo tempo, preservando um núcleo estrutural de elementos de *software* comum a todos os produtos. Cada produto deve ser criado de forma personalizada, inclusive em relação à residência (leiaute da residência, distribuição e tipo dos sensores e dispositivos, etc.). Em outras palavras, cada residência com o seu paciente pode ser considerada como um produto de uma LPS.

Adicionalmente, os produtos podem ter suas arquiteturas reconfiguradas em tempo de execução, diante de mudanças do contexto (*e.g.*, mudança da localização de um dispositivo) ou dos requisitos dos usuários (*e.g.*, mudança do problema de saúde do paciente). Tais propriedades caracterizam o SCIADS como um Sistema Adaptativo, como descrito no Capítulo 2, Seção 2.3.1. Portanto, variabilidades dinâmicas devem ser descritas no nível da Arquitetura da LPS para que os produtos tenham arquiteturas adaptáveis em tempo de execução.

As próximas subseções apresentam o Modelo de Características do SCIADS e uma

[5] *http://www.continuaalliance.org*
[6] *http://www.11073.org*

descrição das variabilidades do sistema.

3.4.1 Modelo de Características

A Figura 3.4 apresenta o Modelo de Características do SCIADS, desenvolvido, principalmente, a partir da experiência adquirida com o desenvolvimento do protótipo [23], e do conhecimento obtido junto a especialistas da área de saúde [21, 41].

A partir do modelo pode-se verificar que, para um determinado paciente em sua residência, os requisitos de monitoramento e as características de *software* necessárias ao seu suporte podem variar bastante. A derivação de produtos deve ser feita de forma que cada um seja personalizado/configurado para um paciente em específico, tanto em relação às necessidades e preferências do próprio paciente, quanto em relação à residência.

3.4.2 Variabilidades do Sistema

O SCIADS é um sistema que requer um alto grau de adaptabilidade. As variabilidades do modelo podem ser assim descritas:

- Regras médicas podem ser configuradas conforme o problema de saúde do paciente (*HealthProblem*). Caso um dos problemas de saúde seja selecionado, o sensor médico (*MedicalSensor*) correspondente também deve ser, por conta da relação de dependência do tipo *requires* definida no modelo. Por exemplo, o sensor de pressão arterial (*BloodPressure*) deve ser selecionado, caso hipertensão (*Hypertension*) seja o problema de saúde do paciente.

- Sensores de ambiente adequados também podem ser selecionados (*EnvironmentSensor*: *Temperature*, *Humidity*, *Luminosity*).

- Sensores de movimento, como, por exemplo, do tipo acelerômetro (*Accelerometer*), podem ser usados para detectar a atividade do paciente na casa.

- O módulo de decisão determina a situação de saúde do paciente a partir dos dados coletados dos sensores selecionados, e também das regras médicas, caso algum problema de saúde (*HealthProblem*) seja selecionado.

- Em casos de situação de emergência, detectada pelo módulo de decisão ou pelo botão de pânico (*PanicButton*), quando selecionado, a HHS deve enviar um alarme

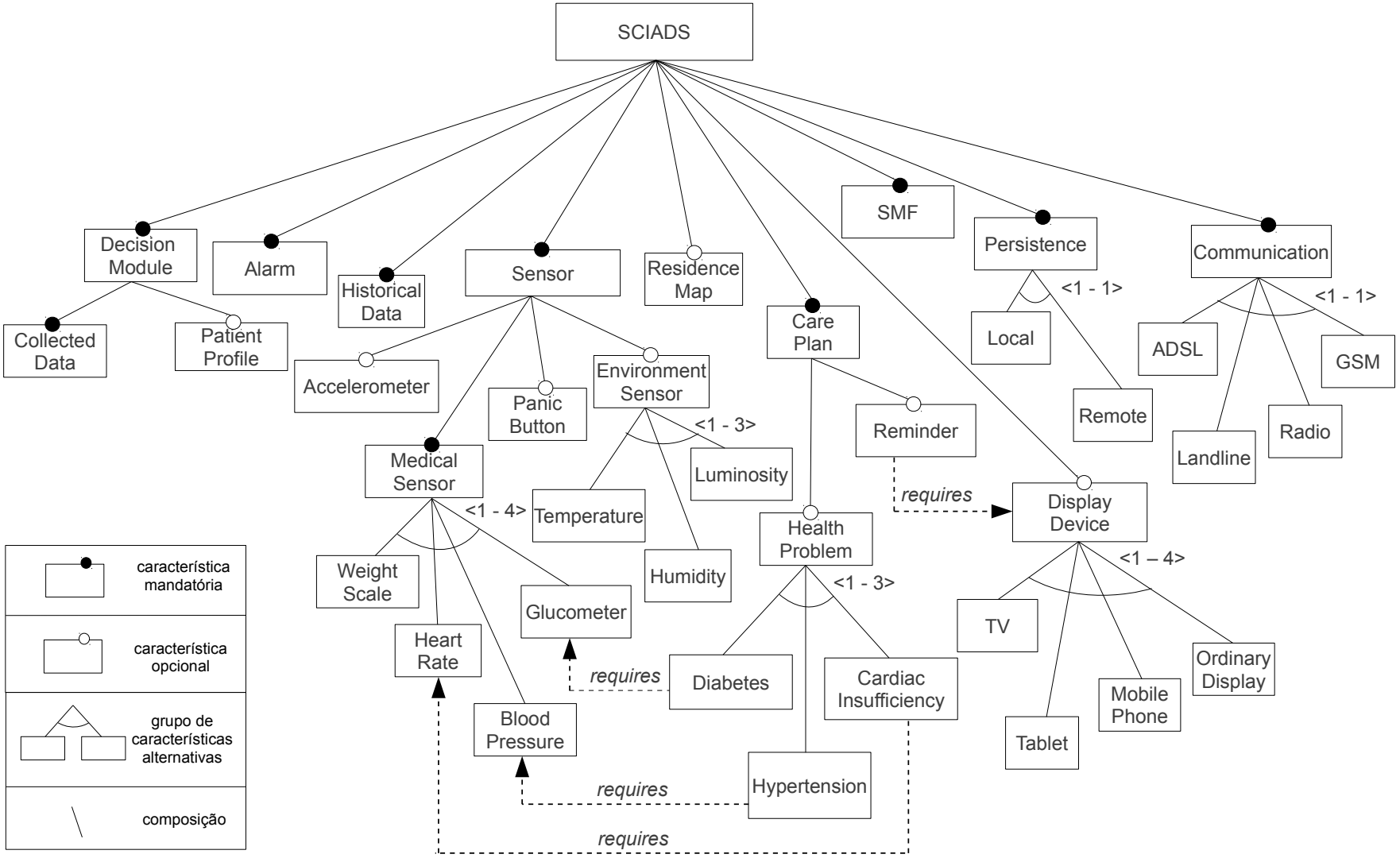


Figura 3.4: Modelo de Características de uma LPS para o SCIADS.

para a SMF, usando uma tecnologia de comunicação disponível (*Communication*: *ADSL*, *Landline*, *Radio*, *GSM*).

- A persistência de todos os dados capturados pelos sensores (*Persistence*) deve ser feita localmente na própria HHS (*Local*) ou remotamente, isto é, na SMF (*Remote*). Realizada de forma contínua, a persistência estabelece um histórico completo e individualizado do paciente, incluindo a sua situação de saúde.

- O plano de cuidados deve ser definido com as prescrições elaboradas pelo profissional de saúde. Para lembrar o paciente de realizar as prescrições no seu dia-a-dia, a característica relacionada ao serviço de lembretes (*Reminder*) pode ser selecionada, o que exige também, por causa da relação de dependência do tipo *requires*, a seleção de pelo menos um dispositivo de visualização para mostrar ao paciente as mensagens de notificação (*DisplayDevice*: *TV*, *Tablet*, *MobilePhone*, *OrdinaryDisplay*).

Para derivar um produto personalizado para um paciente em sua residência, deve-se, em princípio, realizar a seleção de um conjunto de sensores médicos especializados, considerando um problema de saúde em particular, quando for o caso, e de uma tecnologia de comunicação entre a HHS e a SMF. Além disso, um plano de cuidados personalizado deve ser definido de acordo com as recomendações médicas aplicadas ao paciente. Durante a derivação, podem ser selecionados ainda sensores de ambiente e dispositivos de visualização. Resolvidas as variabilidades, neste caso variabilidades estáticas, uma vez que estão sendo resolvidas em tempo de derivação, a Arquitetura do Produto pode ser implantada.

Dessa forma, a arquitetura resultante da derivação está acoplada às necessidades daquele paciente e às características da sua residência naquele momento do tempo, não sendo capaz de realizar adaptações em tempo de execução. Por conta disso, se, no futuro, o problema de saúde do paciente mudar (*e.g.*, de hipertensão para insuficiência cardíaca) ou mesmo se novos dispositivos forem adicionados à residência, o produto em execução deve ser interrompido e uma nova versão ser derivada e implantada.

Isso pode exigir por parte dos usuários do sistema (*e.g.*, o próprio paciente, cuidadores, familiares), um esforço por conta da pouca automação que pode existir no gerenciamento dessa operação. Os usuários, por exemplo, podem ser obrigados a acompanhar de perto os procedimentos, realizando solicitações e verificando junto ao provedor de serviços a implantação do novo produto, causando, em muitos casos, desconforto e aborrecimento. Além disso, dependendo do paciente e das suas necessidades de monitoramento, a demora envolvida na implantação de um novo produto pode resultar em riscos à saúde do paciente.

Diante disso, em vez de implantar um novo produto, a Arquitetura do Produto em execução pode ser adaptada em pontos específicos determinados por variabilidades dinâmicas. O problema de saúde do paciente pode, por exemplo, ser definido como uma variabilidade dinâmica a ser resolvida durante a execução do produto, uma vez que mudanças dos requisitos em relação aos problemas de saúde do paciente exigem modificações nos recursos previamente configurados (*e.g.*, sensores médicos, dispositivos), determinando a reconfiguração (adaptação dinâmica) do produto.

Dispositivos de visualização, quando usados para apresentar mensagens do sistema, podem também ser considerados como variabilidades dinâmicas. A seleção do dispositivo a ser usado pode ser feita durante a derivação do produto, onde uma TV disponível na residência pode, por exemplo, ser selecionada. Por outro lado, caso o paciente possua outros dispositivos (*e.g.*, celular, *tablet*), a seleção pode ser postergada para a execução do produto. Neste caso, ela é realizada de acordo com informações do contexto (*e.g.* localização do dispositivo) e regras pré-definidas (*e.g.*, *selecionar o dispositivo que esteja mais próximo do paciente e que seja o seu preferido*). A seleção do dispositivo em tempo de execução implica na adaptação dinâmica da Arquitetura do Produto a fim de configurar o dispositivo selecionado.

A adaptação dinâmica da arquitetura lida ainda com requisitos relacionados à disponibilidade de recursos. Por exemplo, para prover mecanismos de tolerância a falhas para a comunicação entre a residência e a central, são necessárias adaptações dinâmicas da arquitetura a fim de selecionar a melhor entre as tecnologias de comunicação alternativas (*e.g.*, ADSL, rádio, GSM), incluindo o caso de falhas entre elas. Portanto, a comunicação pode também ser considerada como uma variabilidade dinâmica no modelo.

Um aspecto relevante para o SCIADS, ao se prover variabilidades estáticas e dinâmicas, é o arranjo de produtos que pode ser coberto pela Arquitetura da LPS, envolvendo desde produtos cuja arquitetura não admite adaptações (Produto Estático), até aqueles cuja arquitetura suporta adaptações (Produto Dinâmico). Por exemplo, com base em requisitos do usuário, um Produto Estático é derivado para atender a um paciente com hipertensão, contendo um sensor de pressão arterial e a comunicação entre HHS e SMF configurada com ADSL. Neste caso, o produto foi derivado resolvendo-se as variabilidades estaticamente, implicando em uma arquitetura que não suporta adaptações em tempo de execução. Um Produto Dinâmico pode ser derivado com as mesmas características, porém, a sua arquitetura permite adaptações em pontos determinados pelas variabilidades dinâmicas, suportando, por exemplo, mudanças do problema de saúde do paciente.

3.5 Considerações Finais

Este capítulo apresentou a estruturação geral do SCIADS, incluindo o protótipo implementado e o Modelo de Características, além de uma discussão em relação às suas variabilidades.

A necessidade de personalização e a questão da adaptação dinâmica do SCIADS, orientaram parte da pesquisa e desenvolvimento deste trabalho de tese. Durante o desenvolvimento do Modelo de Características, os exemplos e cenários discutidos envolvendo variabilidades dinâmicas, despertaram o interesse pelo estudo em termos da Arquitetura da LPS, em especial, pelas técnicas de descrição de variabilidades no nível arquitetural, discutidas no Capítulo 2. Neste ínterim, Contratos [83, 16], um conceito que tem sido desenvolvido no nosso grupo de pesquisa, foram investigados e aplicados na descrição de variabilidades dinâmicas no nível da Arquitetura da LPS [26, 27].

O próximo capítulo detalha os conceitos de Contratos e os resultados quanto ao seu uso na descrição de variabilidades dinâmicas, incluindo exemplos e cenários do SCIADS. O Capítulo 5 detalha o Metamodelo que integra características, arquitetura e Contratos.

Capítulo 4

Contratos e Variabilidades

Este capítulo apresenta como variabilidades podem ser descritas como Contratos no nível da Arquitetura da LPS.

4.1 Introdução

De forma geral, o termo contrato se refere à especificação do comportamento de artefatos de *software*. Um contrato permite expressar, basicamente, as condições que devem ser satisfeitas para que seja executada uma ação (rotina de *software*), e os efeitos causados pela execução desta ação. A especificação das condições e efeitos faz com que o comportamento esperado de um artefato possa ser mais precisamente especificado, permitindo determinar previamente se ele pode ou não ser usado em um certo contexto [93, 13]. O conceito de contrato foi primeiramente introduzido por Meyer [93], o qual usa o termo *Design by Contract* para apresentar técnicas voltadas à melhoria da confiabilidade em sistemas de *software*.

Em diversos trabalhos realizados pelo nosso grupo de pesquisa [29, 84, 16, 15, 107, 17, 42, 78], contratos têm sido desenvolvidos como elementos definidos no nível da descrição arquitetural de uma aplicação, com o objetivo de expressar ações de adaptação da arquitetura.

Durante o desenvolvimento desta tese, conceitos relacionados a tais Contratos[1] foram revisitados no contexto de Arquiteturas de LPS, mais especificamente em termos da descrição de variabilidades. Resultados obtidos a partir deste trabalho e do desenvolvimento de Contratos descrevendo variabilidades dinâmicas do SCIADS [26, 27], mostraram que

[1] Estes contratos têm a denominação de Contratos Arquiteturais. Nesta tese, para efeito de simplificação, eles são chamados apenas de Contratos.

Contratos podem ser modelados para descrever variabilidades no nível da Arquitetura da LPS. Isso amplia o escopo de utilização dos Contratos, pois além de utilizados na modelagem arquitetural de aplicações isoladas e não relacionadas [84, 16], podem também ser utilizados na modelagem de Arquiteturas de LPS.

Diante disso, a pesquisa evoluiu em direção ao desenvolvimento do Metamodelo de Configuração de Variabilidades em LPS (apresentado no próximo capítulo), onde, por meio dos Contratos, características da LPS são associadas a elementos da arquitetura, e vice-versa, fazendo com que modificações no nível das características (*e.g.*, a seleção de uma característica) provoquem mudanças no nível arquitetural, e modificações no nível da arquitetura (*e.g.*, mudança do contexto) produzam reflexos no nível das características.

Este capítulo está organizado como se segue.

Seção 4.2

Conceito e estruturação dos Contratos, assim como os principais componentes da infraestrutura de suporte e execução.

Seção 4.3

Como variabilidades podem ser descritas na forma de Contratos.

Seção 4.4

Considerações finais.

4.2 Contratos

Contratos são elementos que expressam as operações (*e.g.*, primitivas, *scripts*) capazes de realizar adaptações da arquitetura de uma aplicação em tempo de execução. Eles permitem expressar as condições que devem ser satisfeitas para que seja executada a adaptação, e os efeitos causados pela sua execução.

A linguagem utilizada para a construção de Contratos é a ADL CBabel[2] [16, 84], originada a partir da linguagem Babel [85] incorporando-se conceitos relacionados à especificação de qualidade de serviço em sistemas distribuídos, propostos por Frølund e Koistinen em [59], onde expressões de restrição e negociação são usadas para estabelecer o relacionamento formal entre as partes que usam ou fornecem recursos. Estes conceitos foram reformulados para o contexto de descrição de arquiteturas de *software* [84, 29],

[2]Esta tese não aborda os detalhes da ADL CBabel. O Apêndice A mostra parte da sua BNF referente à descrição de Contratos.

onde o comportamento individual dos elementos deve ser claramente separado da interação entre eles na arquitetura [74]. A CBabel segue essa direção ao permitir a descrição da arquitetura inicial da aplicação e dos Contratos, de forma que serviços possam ser estabelecidos para adaptar essa arquitetura frente a diferentes situações de contexto e de qualidade que venham a ocorrer durante o tempo de vida da aplicação.

As próximas subseções detalham a estruturação de um Contrato e a infraestrutura de suporte e execução.

4.2.1 Estruturação

Contratos são definidos, essencialmente, por serviços, regras e negociação. Um serviço descreve um conjunto de operações de adaptação da arquitetura, enquanto as regras determinam, com base em informações do contexto, se um serviço em particular pode ser executado ou não. A negociação, por sua vez, descreve uma ordem de execução entre os serviços.

A Figura 4.1 apresenta como um Contrato é estruturado. Cada serviço (*service*, *linhas 03 a 15*) é descrito com operações de adaptação arquitetural. A execução destes serviços e, por conseguinte, das operações de adaptação, somente ocorrem se as regras definidas em seus respectivos perfis são satisfeitas (*profile1*, *linha 06* e *profile2*, *linha 13*). Estas regras são descritas usando informações de contexto estruturadas em uma ou mais categorias (*category*). A ordem de execução dos serviços é definida na cláusula de negociação (*negotiation*) descrita nas *linhas 17 a 21*. Os próximos parágrafos apresentam o mecanismo básico de funcionamento do Contrato, visando facilitar a compreensão dos Contratos desenvolvidos para os cenários do SCIADS, descritos na Seção 4.3. A semântica dos Contratos é discutida e apresentada de forma detalhada em [15, 110, 84], incluindo o desenvolvimento de uma semântica operacional [15].

Quando o Contrato da Figura 4.1 é instanciado, durante a execução de uma aplicação, a negociação tem início visando escolher o primeiro serviço a ser executado, neste caso *service1*. A sua escolha se deve por ser o serviço preferencial do Contrato, ou seja, aquele definido na primeira linha da cláusula de negociação (*linha 18*) e à esquerda do símbolo de transição "->".

Antes que o serviço *service1* seja executado, no entanto, as regras descritas no perfil *profile1*, a ele associado na *linha 06*, devem ser avaliadas. Se as regras são satisfeitas, as operações de adaptação descritas em *service1* são executadas. Se, no entanto, uma das

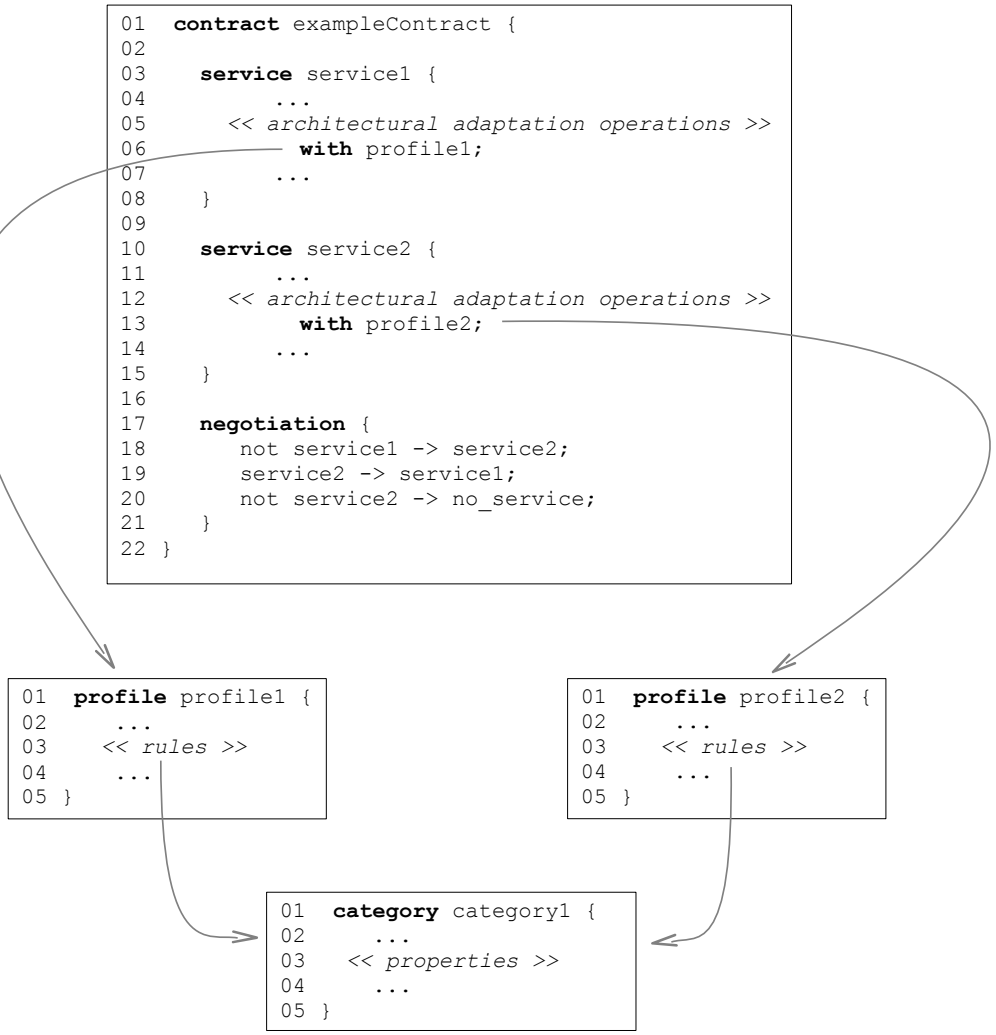


Figura 4.1: Estrutura geral de um Contrato.

regras não é satisfeita, *service1* não pode ser executado, fazendo com que a negociação escolha outro serviço, no caso, *service2*. Isso é feito por meio da transição de *service1* para *service2*, representada pelo símbolo "->" da *linha 18* da negociação. O operador *not* indica que a transição deve ocorrer quando o serviço *service1* não puder ser executado.

A escolha de *service2* para execução faz com que as regras descritas no perfil *profile2*, a ele associado na *linha 13*, sejam avaliadas. Se as regras são satisfeitas, as operações de adaptação descritas em *service2* são executadas. No entanto, *service2* pode ter sua execução interrompida se as regras do perfil *profile1* associado ao serviço *service1* voltarem a ser satisfeitas, por conta de uma mudança do contexto. Isso está descrito na *linha 19* da negociação, a qual define a transição de *service2* para *service1*.

O operador *not* da *linha 20* indica que, caso *service2* não possa ser executado, uma transição deve ocorrer para o estado *no_service*. Quando o estado *no_service* é alcançado, significa que nenhum dos serviços descritos no Contrato pode ser executado. Nesse caso,

o *Contract Manager* (veja Subseção 4.2.2) pode iniciar uma nova negociação assim que o serviço preferencial (*service1*) se tornar disponível novamente [84, 29], ou reportar o problema para o nível da aplicação, onde poderá ser tratado, ou ainda para Contratos especializados.

Um Contrato é construído, portanto, a partir dos seguintes elementos:

- Categorias: definem, em um nível alto de abstração, as propriedades dos recursos associados ao contexto de execução da aplicação (*e.g.*, localização do paciente, localização de dispositivos). Por exemplo, *DisplayDevice* pode ser uma categoria, e *location* uma propriedade desta categoria. Estas propriedades podem ser usadas para especificar relacionamentos semânticos entre diferentes tipos de recursos, a fim de estabelecer ontologias que podem ser usadas por Serviços de Registro e Descoberta de Recursos, como descrito em [122, 111].

- Perfis: definem regras de contexto, as quais quantificam e estabelecem condições que devem ser satisfeitas para que seja realizada a adaptação da arquitetura. A quantificação restringe cada propriedade de acordo com a sua descrição, funcionando como uma instância de valores aceitáveis para determinada categoria. Predicados lógicos podem ser descritos envolvendo as propriedades. Por exemplo, a regra *DisplayDevice.location == Bedroom* significa que uma adaptação ocorre somente quando o dispositivo específico estiver no quarto.

- Serviços: definem ações de adaptação que devem ser realizadas no nível da arquitetura da aplicação, por meio de uma ou mais operações (*e.g.*, primitivas *instantiate* e *link* da CBabel). As ações têm como consequência a configuração ou reconfiguração dinâmica da arquitetura da aplicação, a qual é realizada dependendo dos perfis associados. Por exemplo, tendo como base a regra definida anteriormente (*DisplayDevice.location = Bedroom*), um componente apenas pode ser instanciado e vinculado à arquitetura da aplicação, se o determinado dispositivo estiver no quarto. Cada serviço define um possível estado de operação para a arquitetura da aplicação.

- Negociação: descreve uma política para a transição entre os serviços. Por exemplo, se a arquitetura está configurada com o componente *BloodPressureSensor* e um novo sensor é configurado, uma transição deve ocorrer para adaptar a arquitetura da aplicação em tempo de execução. Máquinas de estados podem ser usadas para definir uma ordem particular de estabelecimento dos serviços, ou funções utilidade podem ser usadas para guiar o mecanismo de adaptação com o objetivo de selecionar

a melhor alternativa [107, 78]. De acordo com o descrito na cláusula de negociação, quando um serviço de maior preferência não puder mais ser mantido, um serviço com menor preferência será estabelecido. O retorno para um serviço de maior preferência também pode ser descrito, permitindo que um serviço de melhor qualidade seja estabelecido se os perfis associados se tornarem válidos, ou seja, se as suas regras de contexto forem satisfeitas.

A Figura 4.2 apresenta um metamodelo com os elementos que formam um Contrato. Contratos (*Contract*) são compostos de serviços (*Service*) e negociação (*Negotiation*). Serviços são associados às regras (*Rule*) por meio de perfis (*Profile*). Um perfil é composto de uma ou mais regras, e estas são estruturadas por propriedades (*Property*), as quais são definidas por categorias (*Category*). Categorias e suas propriedades representam os recursos que fazem parte do contexto de execução de uma aplicação.

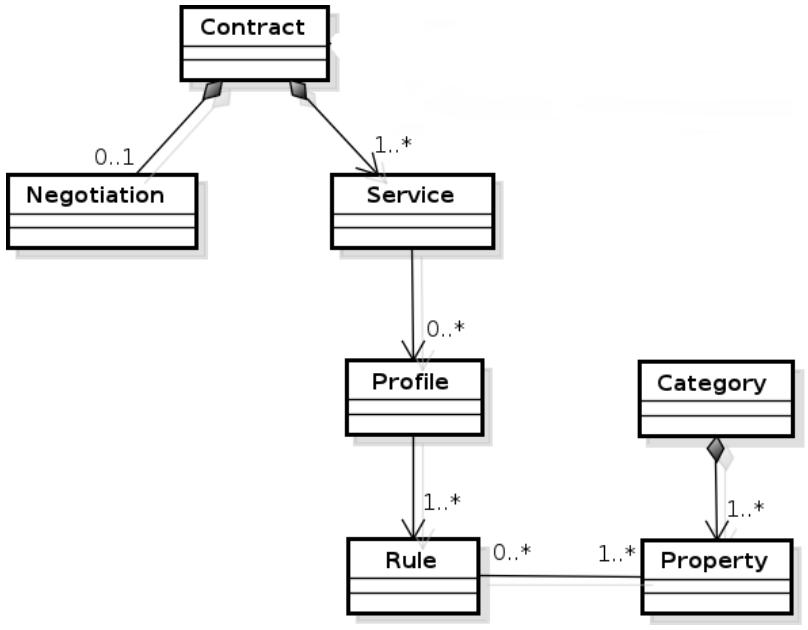


Figura 4.2: Metamodelo de Contratos.

A modelagem de Contratos da Figura 4.2 integra o Metamodelo proposto no Capítulo 5, juntamente com a modelagem de características e de Arquitetura da LPS, apresentadas no Capítulo 2, Figuras 2.5 e 2.6, respectivamente.

4.2.2 Suporte e Execução

Para que os Contratos efetivamente realizem as adaptações arquiteturais neles descritas, uma infraestrutura de suporte é necessária, com o monitoramento do contexto e o

controle das adaptações. Embora a infraestrutura desenvolvida para o suporte e execução dos Contratos não seja o foco desta tese, vale ressaltar as suas principais funções e componentes.

As principais funções da infraestrutura são:

- Interpretar as especificações descritas nos Contratos e armazená-las como informações de metanível associadas à arquitetura da aplicação em execução;
- Prover mecanismos de reflexão e adaptação dinâmica, os quais permitem gerenciar e adaptar as configurações arquiteturais a fim de atender as demandas definidas nos Contratos;
- Monitorar e gerenciar os Contratos associados à arquitetura da aplicação em execução.

A Figura 4.3 apresenta uma visão geral da interação entre os principais componentes da infraestrutura de suporte: Serviço de Contexto (do inglês, *Context Service*), Gerenciador de Contratos (do inglês, *Contract Manager*), Serviço de Descoberta de Recursos (do inglês, *Resource Discovery Service*) e Configurador (do inglês, *Configurator*).

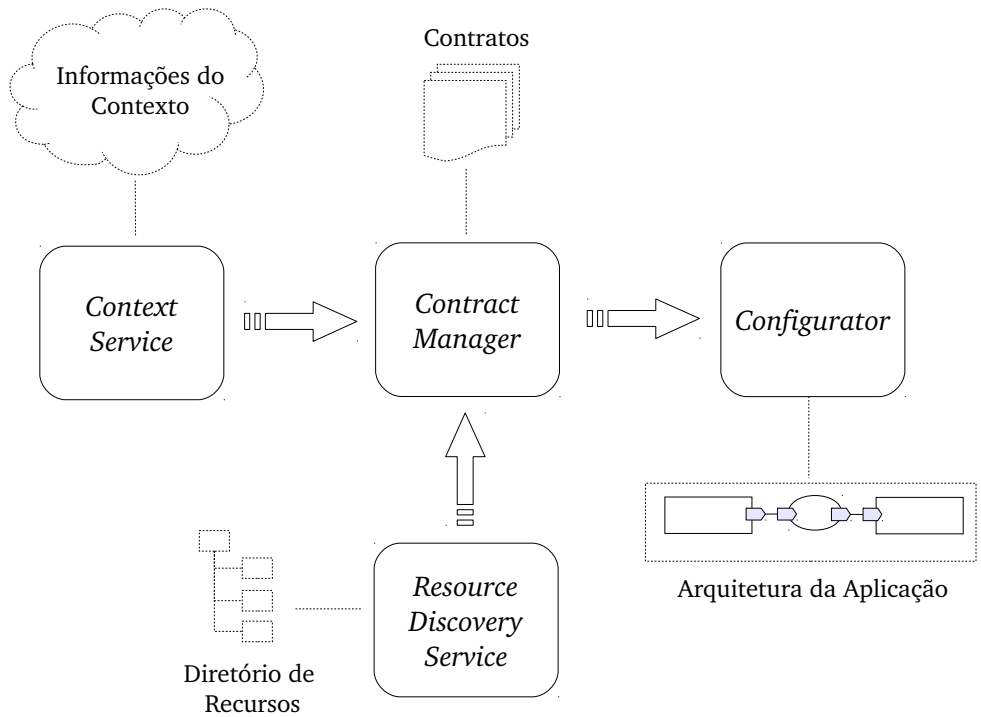


Figura 4.3: Visão geral da interação entre os principais elementos da infraestrutura de suporte.

Context Service monitora as *Informações do Contexto* de execução da aplicação, as quais são obtidas e enviadas para o *Contract Manager*. Por exemplo, o *Context Service* pode enviar, durante a operação da aplicação, a informação da localização de um determinado dispositivo na residência (*e.g.*, a TV está no quarto).

Contract Manager é responsável pela seleção dos serviços e pelo gerenciamento das políticas descritas nos *Contratos*. As condições descritas nos perfis na forma de regras devem ser avaliadas. Se as condições são satisfeitas, o perfil é validado e o serviço correspondente é selecionado e colocado em execução; caso contrário, outro serviço deve ser selecionado de acordo com a cláusula de negociação. O mesmo ocorre se, durante a operação de um determinado serviço, o contexto muda e o perfil se torna inválido. Neste caso, o serviço não pode ser mantido em execução e, portanto, é encerrado. Por exemplo, um serviço que instancia (*instantiate*) e vincula (*link*) um componente TV é selecionado para execução somente se o paciente estiver no quarto.

Resource Discovery Service, por sua vez, é usado quando o serviço selecionado pelo *Contract Manager* para execução não nomeia, em suas operações de adaptação, qual o elemento arquitetural que deve ser configurado junto à *Arquitetura da Aplicação*. Por exemplo, um serviço pode ser descrito para configurar na arquitetura, dentre todos os dispositivos do tipo *DisplayDeviceType* (*e.g.*, TV, celular, *tablet*), apenas aquele que esteja no mesmo ambiente em que estiver o paciente.

Em casos como este, uma busca deve ser feita no *Diretório de Recursos* por uma instância que atenda a estas condições definidas em perfis. Este componente é responsável pelo armazenamento das informações e pelo processamento de consultas e registros de recursos [17]. Quando um recurso entra no ambiente, como, por exemplo, um novo sensor ou dispositivo, este se registra no *Diretório de Recursos* com o seu tipo e as suas propriedades de contexto, aqui representadas na forma de categorias e propriedades. Consultas podem então ser feitas junto ao diretório, pelo tipo e também pelos valores das propriedades de contexto fornecidos pelo *Context Service*.

Por fim, o *Configurator* traduz as descrições do *Contrato* (construídas em CBabel) correspondentes ao serviço selecionado em ações que efetivamente realizam as adaptações / configurações na arquitetura. Os trabalhos de Cardoso *et al.* [16], Loques *et al.* [84], Rodrigues *et al.* [114], Corradi [42] e Cardoso [17] apresentam os detalhes do funcionamento desta infraestrutura e dos seus componentes.

4.3 Descrevendo Variabilidades na Forma de Contratos

A associação de Contratos à arquitetura de uma aplicação envolve determinados elementos da arquitetura e a torna configurável em tempo de execução. A arquitetura tem a sua configuração modificada diante da execução de serviços que adicionam e/ou removem alguns dos seus elementos.

A adaptação arquitetural realizada por um serviço pode ser vista como uma reconfiguração que ocorre em apenas uma porção restrita da arquitetura da aplicação, e de forma pré-planejada, ou seja, definida durante o desenvolvimento da arquitetura e dos próprios Contratos. Para se desenvolver a arquitetura de uma aplicação e associá-la a Contratos, deve-se criar uma arquitetura inicial para a aplicação e, durante o seu desenvolvimento, levantar os pontos da arquitetura passíveis de modificações em tempo de execução. Estes pontos capturados em tempo de desenvolvimento da arquitetura podem então ser construídos na forma de Contratos e serviços de adaptação.

Por exemplo, uma aplicação cliente-servidor em um ambiente ubíquo pode ter a sua arquitetura inicialmente constituída por um grupo de servidores distribuídos e por clientes acessando informações fornecidas por estes servidores. Ao descrever essa arquitetura, pode-se definir um Contrato com serviços que permitam acrescentar novos componentes ao grupo de servidores, caso o fluxo de requisições aumente junto ao grupo, impedindo assim uma sobrecarga ou mesmo a parada da aplicação. Neste exemplo, o Contrato está atuando diretamente em um determinado ponto da arquitetura (grupo de servidores), identificando as possíveis configurações que podem ser feitas neste ponto frente ao aumento das requisições. Os trabalhos de Petrucci e Loques [107], e de Cardoso *et al.* [16] mostram exemplos nesse sentido.

Os Contratos foram originalmente concebidos para lidar, principalmente, com requisitos não funcionais, como o mostrado no exemplo (veja em [36] um estudo detalhado a respeito de requisitos não funcionais). A estruturação em termos de serviços e regras permite, no entanto, a construção de Contratos independentemente do tipo de requisito. Por exemplo, um Contrato pode ser desenvolvido para configurar apenas alguns pontos da arquitetura para lidar com variações relacionadas ao tipo de monitoramento que um paciente precisa na sua residência, como, por exemplo, monitorar a pressão arterial, o nível de glicose, ou a frequência cardíaca, conforme a evolução dos seus problemas de saúde [28, 27].

Construídos para configurar somente determinados pontos da arquitetura da apli-

cação, os Contratos fazem com que ela tenha um comportamento dinâmico nas partes delimitadas por estes pontos, e seja estática nas demais. Tais pontos guardam semelhança com a concepção de pontos de variação e opcionalidades, que, como descrito no Capítulo 2, são as formas de se representar variabilidades em Arquiteturas da LPS. Dessa forma, variabilidades podem ser descritas como Contratos definindo-se configurações no nível da Arquitetura da LPS. Em especial, os Contratos favorecem a descrição de variabilidades dinâmicas, pois eles têm a propriedade de adaptar arquiteturas em tempo de execução e com base em regras de contexto.

Com o objetivo de mostrar como variabilidades dinâmicas podem ser descritas na forma de Contratos, as subseções a seguir apresentam três cenários desenvolvidos no contexto do SCIADS. O primeiro lida com as alternativas em termos de tecnologia de comunicação disponíveis entre a casa e a SMF (*Supervisory Medical Facility*), enquanto o segundo envolve os dispositivos usados para apresentar mensagens ao paciente, tais como TV, *tablet* e celular. O terceiro cenário, por sua vez, está relacionado a mudanças nos problemas de saúde do paciente. Para cada cenário são apresentadas:

- A descrição do Núcleo da Arquitetura, contendo os elementos arquiteturais mandatórios;
- A descrição de um ou mais Contratos representando as variabilidades dinâmicas da arquitetura.

O *Núcleo da Arquitetura* e os *Contratos* formam a *Arquitetura da LPS*, como indicado na Figura 4.4.

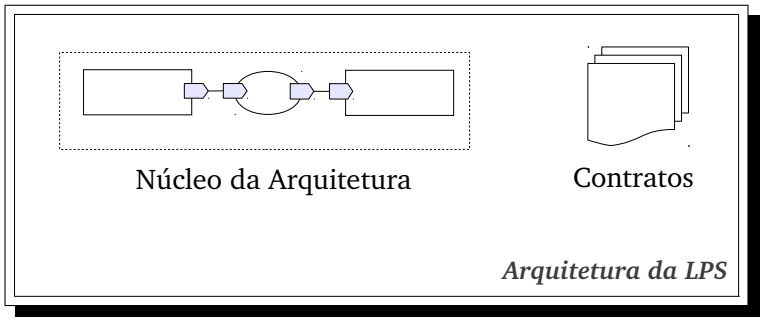
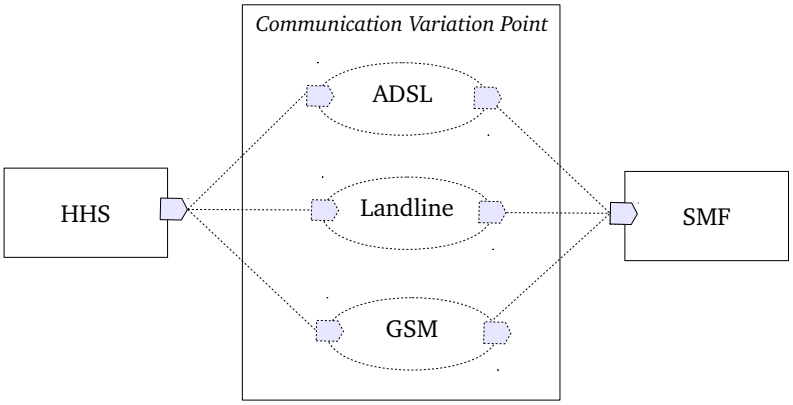


Figura 4.4: O Núcleo da Arquitetura e os Contratos formam a Arquitetura da LPS.

4.3.1 Comunicação

No SCIADS, a transmissão de dados de emergência da HHS (*Home Health Station*), localizada na casa do paciente, para a SMF (*Supervisory Medical Facility*), a central de

supervisão médica, pode sofrer descontinuidades, geralmente associadas à qualidade do serviço prestado pelos provedores de acesso, uma vez que primariamente a comunicação é feita por meio da Internet *"commodity"*. Neste sentido, pode-se prover características relacionadas à tolerância a falhas utilizando canais alternativos, seja via rede celular, ou mesmo via linha fixa. A Figura 4.5 mostra a Arquitetura da LPS.



(a)

```
01  module CoreArchitecture {
02     instantiate HHSType as HHS;
03     instantiate SMFType as SMF;
04  }
```

(b)

Figura 4.5: Comunicação entre HHS e SMF: a) Arquitetura da LPS; b) Descrição do núcleo da arquitetura.

Os conectores pontilhados da Figura 4.5-a representam as variantes do ponto de variação *Communication*, as quais estão disponíveis na forma de tecnologias de comunicação que podem ser selecionadas para a interligação entre os componentes *HHS* e *SMF*. A Figura 4.5-b apresenta a descrição do núcleo da arquitetura, no qual são definidos os componentes *HHS* e *SMF* (*linhas 02 e 03*).

O conector de interligação entre os componentes *HHS* e *SMF* deve ser dinamicamente estabelecido pelo Contrato descrito na Figura 4.6. O Contrato define três diferentes serviços, sendo que cada serviço descreve uma variante de comunicação (*linhas 03 a 05*, *linhas 07 a 09*, *linhas 11 a 13*). O suporte necessário para cada serviço está encapsulado em um conector (construção *link*), seguindo a regra de contexto descrita no seu respectivo perfil. Quando um determinado conector é estabelecido, faz com que a HHS seja interligada a este conector e este interligado à SMF. A associação entre o Contrato e o núcleo da arquitetura está definida por meio dos elementos arquiteturais HHS e SMF.

A Figura 4.7 apresenta a categoria *Transport* constituída da propriedade *technology*

```
01  contract CommunicationContract {
02
03    service sADSL {
04       link HHS to SMF with ADSLProfile;
05    }
06
07    service sLandline {
08       link HHS to SMF with LandlineProfile;
09    }
10
11    service sGSM {
12       link HHS to SMF with GSMProfile;
13    }
14
15    negotiation {
16     not sADSL -> sLandline;
17     sLandline -> sADSL;
18     not sLandline -> sGSM;
19     sGSM -> sLandline;
20     not sGSM -> no_service;
21    }
22 }
```

Figura 4.6: Contrato do ponto de variação *Communication*.

(*linhas 01 a 03*), e os perfis *ADLSProfile* (*linhas 04 a 06*), *LandlineProfile* (*linhas 07 a 09*) e *GSMProfile* (*linhas 10 a 12*). Para cada opção de enlace há uma interface de comunicação e um sensor específico que detecta se o respectivo enlace está disponível ou não. Além disso, o *Contract Manager*, componente da infraestrutura de suporte que avalia as regras descritas nos perfis, pode estar em execução em quaisquer dos nós do sistema distribuído, incluindo aquele onde a própria HHS está em execução.

```
01 category Transport {
02   technology: enum (ADSL, Landline, GSM);
03 }

04 profile ADSLProfile {
05   Transport.technology = ADSL;
06 }

07 profile LandlineProfile {
08   Transport.technology = Landline;
09 }

10 profile GSMProfile {
11   Transport.technology = GSM;
12 }
```

Figura 4.7: Categoria *Transport* e perfis *ADSLProfile*, *LandlineProfile* e *GSMProfile*.

A cláusula de negociação do Contrato *CommunicationContract*, mostrado na Figura 4.6, estabelece a ordem na qual os serviços são avaliados, sendo que a *linha 16* determina *sADSL* como o serviço preferencial. Caso este serviço não esteja disponível, o serviço *sLandline* é estabelecido. Na sequência, *caso sADSL* se torne disponível novamente, ele é reestabelecido pelo *Contract Manager*, mesmo se o serviço *sLandline* estiver em exe-

cução, conforme a transição definida na *linha 17*. As *linhas 18 e 19* impõem o mesmo mecanismo, mas dessa vez envolvendo os serviços *sLandline* e *sGSM*. Por fim, se o serviço *sGSM* não estiver disponível, nenhum serviço pode ser estabelecido (*no_service*), exigindo que os operadores localizados na *SMF* sejam alertados a fim de que procedimentos sejam realizados no sentido de garantir a segurança do paciente. Para este propósito, a *SMF* deve incluir um mecanismo adicional que monitore a conectividade com a *HHS*.

A Figura 4.8 mostra duas possíveis configurações para a arquitetura de um produto em tempo de execução. Para estabelecer o enlace de comunicação, o conector é estabelecido e mantido no canal preferencial definido pelo Contrato (Figura 4.8-a). Entretanto, se *ADSL* falha durante a operação do produto, torna-se necessário substituir o conector a fim de manter a comunicação entre a *HHS* e *SMF*. De acordo com o Contrato, o serviço *sLandline* deve adicionar outro conector correspondente à linha fixa, resultando na arquitetura mostrada na Figura 4.8-b. Além disso, o conector correspondente a *ADSL* tenta se restabelecer novamente, conforme a cláusula de negociação do Contrato.

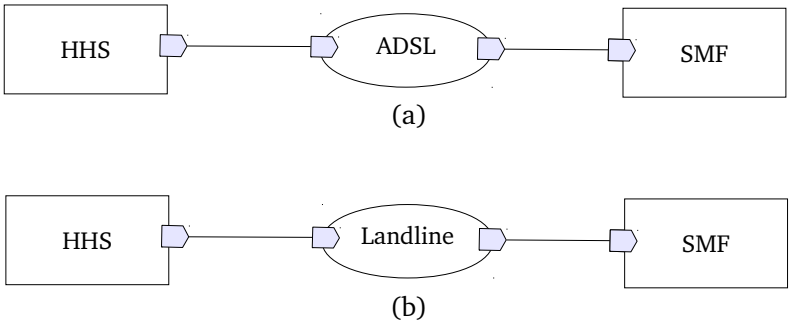
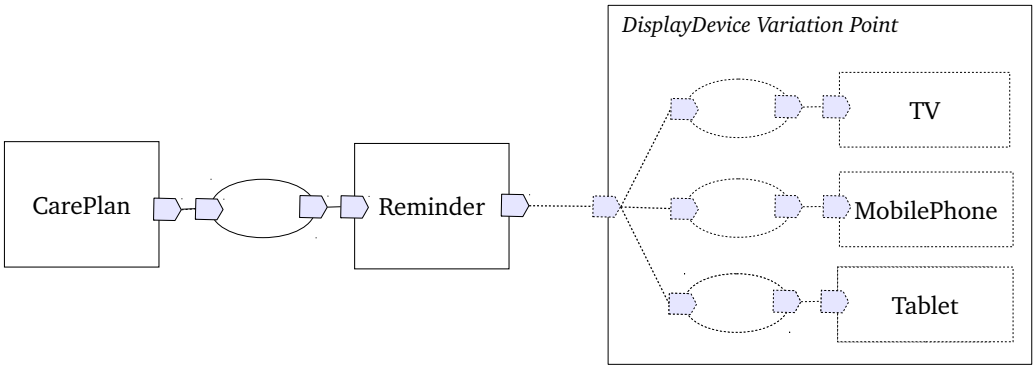


Figura 4.8: Configurações para a arquitetura de um produto com: a) conector *ADSL*; b) conector *Landline*.

4.3.2 Notificação Sensível ao Contexto

O segundo cenário manipula uma variabilidade relacionada aos dispositivos TV, celular e *tablet*, usados para notificar o paciente lembrando-o de realizar as suas prescrições médicas, tais como tomar um medicamento, fazer uma medida com um sensor fisiológico (*e.g.*, pressão arterial, frequência cardíaca) e outras recomendações personalizadas conforme o tratamento. O plano de cuidados mantém essas prescrições e deve interagir com um notificador, o qual envia as mensagens aos dispositivos disponíveis na casa do paciente. Os dispositivos são componentes da Arquitetura da LPS e, neste cenário, são representados como variabilidades dinâmicas. Assim, a seleção de qual dispositivo usar para apresentar as mensagens ao paciente deve ser feita em tempo de execução e de acordo com regras de contexto.

A Figura 4.9-a apresenta a Arquitetura da LPS e a Figura 4.9-b mostra a descrição do núcleo da arquitetura, estabelecido com os componentes *CarePlan* e *Reminder*. As *linhas 02 e 03* descrevem a instanciação destes componentes e a *linha 04* os interliga.



(a)

```
01  module CoreArchitecture {
02    instantiate CarePlanType as CarePlan;
03    instantiate ReminderType as Reminder;
04    link CarePlan to Reminder;
05  }
```

(b)

Figura 4.9: Notificação sensível ao contexto: a) Arquitetura da LPS; b) Descrição do núcleo da arquitetura.

Para que os dispositivos sejam selecionados em tempo de execução, o ponto de variação *DisplayDevice* da Arquitetura da LPS (Figura 4.9-a) deve ser associado a Contratos que definem regras para a seleção. Neste cenário, a seleção do dispositivo e a sua interligação ao componente *Reminder* estão condicionadas a uma situação de contexto: o dispositivo deve estar próximo ao paciente. Além disso, a seleção depende das preferências do paciente descritas no perfil.

Duas categorias estão definidas na Figura 4.10, *Patient* e *DisplayDevice*, com as propriedades de contexto que devem ser avaliadas pelo Contrato associado à arquitetura. Os valores a serem consultados junto ao Serviço de Contexto estão definidos no perfil *NearestDeviceProfile*, também apresentado na Figura 4.10.

As categorias *Patient* e *DisplayDevice* definem a propriedade *location* por meio de um tipo enumerado (*linhas 02 e 05*) indicando os ambientes da residência onde o paciente e os dispositivos podem estar localizados. A categoria *DisplayDevice* define ainda o tipo do dispositivo (*type* na *linha 06*), com *TVType*, *MobilePhoneType* e *TabletType*. Essa definição é importante nesse cenário para se construir regras de contexto que filtrem pelo tipo os dispositivos recuperados pelo Serviço de Descoberta de Recursos (veja *Discovery Resource Service* descrito na Seção 4.2.2).

```
01 category Patient {
02   location: enum (Livingroom, Bedroom);
03 }

04 category DisplayDevice {
05   location: enum (Livingroom, Bedroom);
06   type: enum (TVType, MobilePhoneType, TabletType);
07 }

08 profile NearestDeviceProfile {
09   DisplayDevice.location = Patient.location;
10   DisplayDevice.type = (TVType > MobilePhoneType > TabletType);
11 }
```

Figura 4.10: Categorias *Patient* e *DisplayDevice*; perfil *NearestDeviceProfile*.

O perfil *NearestDeviceProfile*, por sua vez, contém dois critérios para a seleção dos dispositivos. O primeiro critério (*linha 09*) seleciona um determinado dispositivo apenas se o paciente estiver no mesmo ambiente em que o dispositivo estiver. O segundo critério (*linha 10*) seleciona o dispositivo preferido pelo paciente, respeitando a regra de precedência que define a TV como o dispositivo preferido, seguido pelo telefone celular e pelo *tablet*. Ambos os critérios são usados em conjunto para selecionar a melhor configuração. Por exemplo, se o paciente estiver no quarto e uma TV e um *tablet* também estiverem no quarto, a TV é escolhida, tendo em vista a regra descrita no perfil.

A Figura 4.11 mostra um Contrato construído para selecionar os dispositivos. A operação *select()* (*linha 04*) recupera, por meio do Serviço de Descoberta de Recursos os componentes do tipo *DisplayDeviceType* (primeiro parâmetro) correspondentes aos dispositivos implantados na residência. *DisplayDeviceType* representa o Domínio de Recursos de interesse a ser descoberto junto ao *Diretório de Recursos* (veja Figura 4.3). Neste cenário, ele está definido como um supertipo de *TVType*, *MobilePhoneType* e *TabletType*, os tipos dos componentes *TV*, *MobilePhone* e *Tablet*, respectivamente (Figura 4.12). Assim, a definição de *DisplayDeviceType* como o Domínio de Recursos faz com que o Serviço de Descoberta de Recursos retorne uma lista de dispositivos, incluindo TVs, telefones celulares e *tablets* disponíveis na residência. Caso, por exemplo, se quisesse retornar uma lista contendo apenas as TVs da residência, o Domínio de Recursos poderia ser definido como *TVType*.

Depois de recuperar os dispositivos, a operação *select()* escolhe, dentre eles, aquele mais próximo ao paciente e que tenha a sua preferência, de acordo com as regras definidas no perfil *NearestDeviceProfile*, apresentado na Figura 4.10. O dispositivo selecionado (*@nearest_device*) é então dinamicamente interligado ao componente *Reminder* (*linha 05*), resultando na adaptação da Arquitetura do Produto.

```
01 contract DisplayDeviceContract {
02
03  service DisplayDeviceService {
04    @nearest_device = select (DisplayDeviceType, NearestDeviceProfile);
05    link Reminder to @nearest_device;
06  }
07
08  negotiation {
09    not DisplayDeviceService -> no_service;
10  }
11 }
```

Figura 4.11: Contrato para selecionar o dispositivo mais próximo ao paciente.

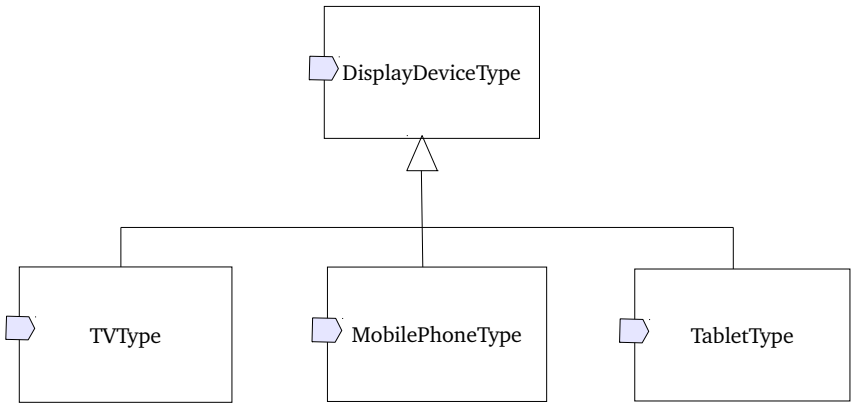


Figura 4.12: Hierarquia de tipos para os dispositivos.

O serviço *DisplayDeviceService* (*linhas 03 a 06*) verifica continuamente as mudanças de contexto por meio do Serviço de Contexto. Por exemplo, se a TV do quarto estiver desligada ou não estiver operacional, o serviço interliga dinamicamente o componente *Reminder* ao *Tablet*, caso este também esteja no quarto. Se a qualquer momento, no entanto, nenhum dispositivo puder atender às condições das regras de contexto descritas no perfil, o estado *no_service* é alcançado, conforme a cláusula de negociação (*linha 09*). Neste caso, operadores da SMF devem ser alertados enquanto os dispositivos são restabelecidos.

4.3.3 Evolução dos Problemas de Saúde do Paciente

O terceiro cenário lida com a adaptação dinâmica causada pela evolução dos problemas de saúde do paciente. A Figura 4.13 mostra a Arquitetura da LPS, formada pelos pontos de variação *MedicalSensor* e *HealthProblem* e pelos elementos mandatórios *Persistence* e *Alarm*.

O objetivo dessa arquitetura é identificar a situação de saúde do paciente em monitoramento com base em regras médicas definidas nos componentes *Hypertension* e *CardiacInsufficiency*. Estes componentes são interligados, por meio do componente *MedicalSen-*

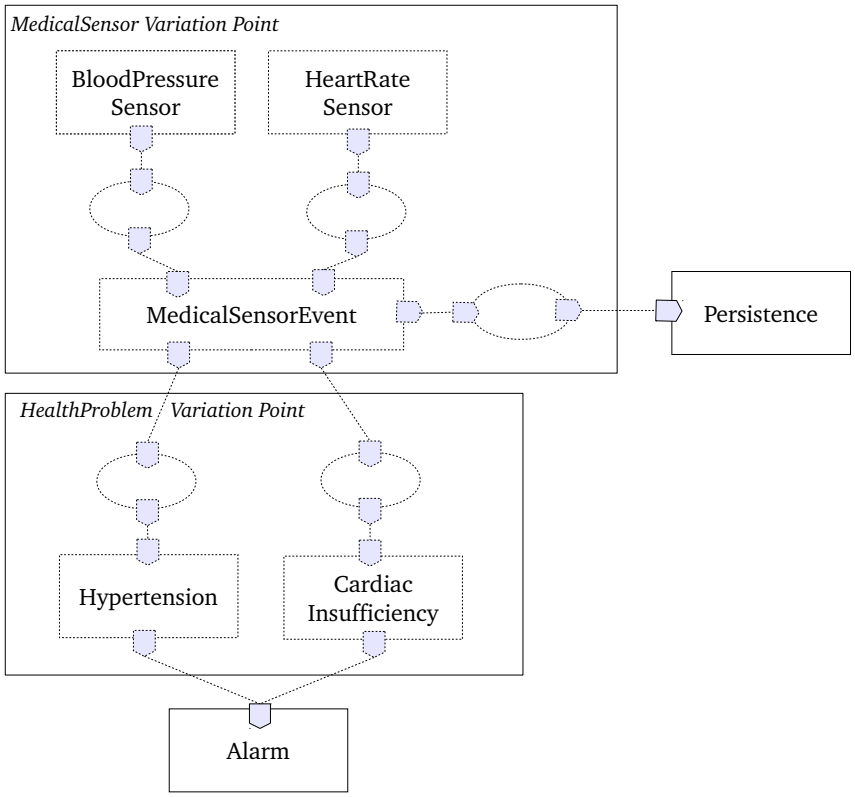


Figura 4.13: Problemas de saúde do paciente: Arquitetura da LPS.

sorEvent, aos sensores médicos apropriados conforme o problema de saúde do paciente: paciente com hipertensão usa o sensor de pressão arterial (*BloodPressureSensor*) e paciente com insuficiência cardíaca usa o sensor de frequência cardíaca (*HeartRateSensor*). Os dados do paciente coletados pelos sensores são persistidos por meio do componente *Persistence*, e também enviados para *Hypertension* e *CardiacInsufficiency*, para que a situação de saúde do paciente seja avaliada e enviada para a SMF por meio do componente *Alarm*.

Os elementos correspondentes às variabilidades são selecionados em tempo de execução e de acordo com a evolução dos problemas de saúde do paciente, conforme descrito nos Contratos associados à Arquitetura da LPS. A Figura 4.14 apresenta um Contrato construído para lidar com as necessidades de pacientes com hipertensão. O serviço *HypertensionService* (*linhas 03 a 11*) adapta a Arquitetura do Produto, selecionando o sensor de pressão arterial (componente *BloodPressureSensor*) e as regras médicas relacionadas à hipertensão (componente *Hypertension*). Esta adaptação somente ocorre quando o paciente tem o seu problema de saúde definido como hipertensão, situação avaliada pelo perfil *HypertensionProfile*, nas *linhas 05 e 07*.

A categoria envolvendo os problemas de saúde do paciente e os perfis *Hypertension-*

```
01 contract HypertensionContract {
02
03  service HypertensionService {
04    instantiate BloodPressureSensorType as BloodPressureSensor
05          with HypertensionProfile;
06    instantiate HypertensionType as Hypertension
07          with HypertensionProfile;
08    instantiate MedicalSensorEventType as MedicalSensorEvent;
09    link BloodPressureSensor to MedicalSensorEvent;
10    link MedicalSensorEvent to Hypertension;
11  }
12
13  negotiation {
14    not HypertensionService -> no_service;
15  }
16 }
```

Figura 4.14: Contrato para um paciente com hipertensão.

Profile e *CardiacInsufficiencyProfile* são apresentados na Figura 4.15. Os perfis *HypertensionProfile* e *CardiacInsufficiencyProfile* estabelecem regras associadas à propriedade *health_problem* da categoria *Patient* (*linhas 08 e 13*). A regra da *linha 08* faz com que o serviço *HypertensionService* seja estabelecido e, ao mesmo tempo, o torna sensível a qualquer mudança do problema de saúde que venha a ocorrer.

```
01 category Patient {
02
03   health_problem: enum (hypertension, cardiac_insufficiency;
04
05 }
                          (a)


06 profile HypertensionProfile {
07
08   Patient.health_problem = hypertension;
09
10 }
                          (b)


11 profile CardiacInsufficiencyProfile {
12
13   Patient.health_problem = cardiac_insufficiency;
14
15 }
                          (c)
```

Figura 4.15: a) Categoria *Patient*; b) Perfil para um paciente com hipertensão; c) Perfil para um paciente com insuficiência cardíaca.

A análise dos dados e os avisos emitidos pelo sistema durante a sua execução podem revelar, com o passar do tempo, mudanças na saúde do paciente a ponto de requerer modificações, por parte do profissionl de saúde, do problema de saúde do paciente. Estas modificações podem, inclusive, exigir a adição de novos componentes. Caso o profissional de saúde decida atualizar o problema de saúde do paciente, por exemplo para insuficiência

cardíaca, torna-se necessário monitorar, além da pressão arterial do paciente, também a sua frequência cardíaca. A Figura 4.16 traz o Contrato que adapta a arquitetura para este caso.

```
01 contract CardiacInsufficiencyContract {
02
03  service CardiacInsufficiencyService {
04    instantiate HeartRateSensorType as HeartRateSensor
05           with CardiacInsufficiencyProfile;
06    instantiate CardiacInsufficiencyType as CardiacInsufficiency
07           with CardiacInsufficiencyProfile;
08    instantiate MedicalSensorEventType as MedicalSensorEvent;
09    link HeartRateSensor to MedicalSensorEvent;
10    link MedicalSensorEvent to CardiacInsufficiency;
11  }
12
13  negotiation {
14    not CardiacInsufficiencyService -> no_service;
15  }
16 }
```

Figura 4.16: Contrato para um paciente com insuficiência cardíaca.

Neste Contrato, dois novos componentes são adicionados e interligados a outros componentes da arquitetura (*linhas 04 a 07*), considerando que um novo problema de saúde foi detectado (*linha 13* do perfil *CardiacInsufficiencyProfile*). Quando o serviço *CardiacInsufficiencyService* é implantado, a Arquitetura do Produto resultante da adaptação permite o monitoramento de um paciente com insuficiência cardíaca, e também mantém disponíveis recursos para o monitoramento relacionado à hipertensão.

Vale ressaltar que ambos os Contratos verificam continuamente a informação de contexto correspondente aos problemas de saúde do paciente, usando as regras descritas nos perfis. Se, a qualquer momento, uma regra não for satisfeita, ou, se um componente não estiver disponível (*e.g.*, um sensor falha ou não está instalado na residência), o serviço de adaptação é interrompido e o estado *no_service* é alcançado (*linha 14* da Figura 4.14, e *linha 14* da Figura 4.16). Em consequência disso, parte da Arquitetura do Produto, estabelecida pelo serviço de adaptação, se torna inativa. Por exemplo, se o profissional de saúde decide não mais monitorar o paciente com hipertensão, a Arquitetura do Produto será adaptada mantendo somente os componentes relacionados à insuficiência cardíaca.

Este cenário foi construído envolvendo dois Contratos. Novos Contratos, no entanto, podem ser criados e associados com a Arquitetura da LPS. Por exemplo, pode ser definido um novo Contrato para lidar com pacientes diabéticos.

4.3.4 Discussão

Nos cenários apresentados, Contratos foram associados às variabilidades dinâmicas das respectivas arquiteturas. Com isso, dada uma variabilidade (ponto de variação ou opcionalidade), pode-se determinar as suas possíveis configurações arquiteturais, descritas nos serviços do Contrato. Por exemplo, o ponto de variação *Communication* da arquitetura do Cenário 1 (Figura 4.5) foi representado pelo Contrato *CommunicationContract*. Da mesma forma, o ponto de variação *DisplayDevice* da arquitetura do Cenário 2 (Figura 4.9) foi representado pelo Contrato *DisplayDeviceContract*.

A experiência com a construção dos Contratos envolvendo variabilidades dinâmicas do SCIADS trouxe perspectivas. Além de ser associado a variabilidades da Arquitetura da LPS, um Contrato pode também ser associado a uma ou mais características do Modelo de Características, produzindo assim uma correspondência entre características, Contratos e variabilidades da Arquitetura da LPS. Além disso, Contratos podem ser construídos para lidar também com variabilidades estáticas e serem usados para auxiliar no processo de derivação da Arquitetura de Produtos a partir do Modelo de Características. Esta propriedade faz com que a abordagem proposta seja classificada como orientada a características, ou seja, com o desenvolvimento e a configuração de produtos sendo feitos com base em características, como definido no Capítulo 2, Seção 2.3.2. São com estes princípios que Contratos e seus elementos podem constituir o *Configuration Knowledge* (CK), formando o Modelo de Configuração, como descrito no Capítulo 2, Seção 2.2.3.

Outra perspectiva que vale a pena ser ressaltada, apesar de não estar entre os objetivos desta tese, diz respeito ao potencial dos Contratos em facilitar a verificação da arquitetura de uma aplicação [15]. Um Contrato é especificado contendo essencialmente: *(i)* as condições que devem ser satisfeitas para que a variabilidade seja resolvida; *(ii)* as ações de adaptação que devem ser realizadas junto à Arquitetura do Produto para resolver a variabilidade; e *(iii)* os efeitos causados pela resolução da variabilidade. Essa estruturação facilita a construção de Contratos que expressem e quantifiquem o comportamento esperado [15, 13] das variabilidades. Isso pode propiciar a verificação do comportamento da Arquitetura da LPS em relação a propriedades de *safety*, um aspecto essencial em sistemas de assistência à saúde [28, 66, 77], tais como o SCIADS, uma vez que lidam com vidas humanas.

4.4 Considerações Finais

Este capítulo apresentou os conceitos de Contratos, de que forma são estruturados e como podem descrever variabilidades. O próximo capítulo detalha o Metamodelo de Configuração de Variabilidades em LPS, proposto para integrar Características, Arquitetura da LPS e Configuração.

Capítulo 5

Metamodelo de Configuração de Variabilidades em LPS

Este capítulo detalha o Metamodelo proposto nesta tese e as diretrizes compreendendo os elementos e atividades que devem ser realizadas para a construção dos modelos da LPS, e para a derivação e adaptação dinâmica de Arquiteturas de Produtos.

5.1 Introdução

O Metamodelo de Configuração de Variabilidades em LPS integra os diferentes conceitos de três metamodelos desenvolvidos no decorrer desta tese:

- Características, apresentado na Figura 2.4;
- Arquiteturas de LPS, apresentado Figura 2.6;
- Contratos, apresentado na Figura 4.2.

O Metamodelo proposto permite a criação dos modelos da LPS de uma forma integrada, reunindo os elementos essenciais dos metamodelos anteriores: características; elementos arquiteturais; variabilidades estáticas e dinâmicas; e Contratos. Os Contratos, em especial, possuem um papel fundamental, pois descrevem as variabilidades no nível da Arquitetura da LPS e as associa a características, além de lidar com a derivação e a adaptação dinâmica das Arquiteturas dos Produtos.

Os principais objetivos do Metamodelo proposto são:

- Integrar características, arquitetura e Contratos em um nível alto de abstração;
- Permitir a criação dos modelos da LPS: Modelo de Características, Modelo de Arquitetura e Modelo de Configuração;
- Representar as variabilidades estáticas e dinâmicas no nível da Arquitetura da LPS e associá-las a características;
- Apoiar a derivação e a adaptação dinâmica das Arquiteturas dos Produtos.

Além do próprio Metamodelo, a proposta inclui diretrizes que servem como guias na identificação dos caminhos que devem ser seguidos, no contexto do Metamodelo, para a construção dos modelos da LPS, e para a derivação e adaptação dinâmica de Arquiteturas de Produtos.

Este capítulo está organizado como se segue.

Seção 5.2
Metamodelo completo implementado no EMF, com detalhes das três partes que o compõem: Metamodelo de Características, Metamodelo de Arquitetura e Metamodelo de Configuração.

Seção 5.3
Diretrizes compreendendo os elementos e atividades que devem ser realizadas para a construção dos modelos da LPS e para a derivação de Arquiteturas de Produtos.

Seção 5.4
Representação da Configuração de Características para Produtos Estáticos e para Produtos Dinâmicos, e as diretrizes voltadas à adaptação dinâmica de Arquiteturas de Produtos.

Seção 5.5
Considerações finais.

5.2 O Metamodelo

As classes do Metamodelo estão distribuídas em três partes independentes e, ao mesmo tempo, integradas. A primeira parte refere-se ao Metamodelo de Características, representado pela classe *FeatureModel*. A segunda, por sua vez, refere-se ao Metamodelo de

Configuração, representado pela classe *ConfigurationModel* e a terceira, por fim, refere-se ao Metamodelo da Arquitetura, representado pela classe *ArchitectureModel*. A partir dessas classes são instanciados os respectivos modelos – Características, Configuração e Arquitetura. A Figura 5.1 mostra um fragmento do Metamodelo, destacando as três classes.

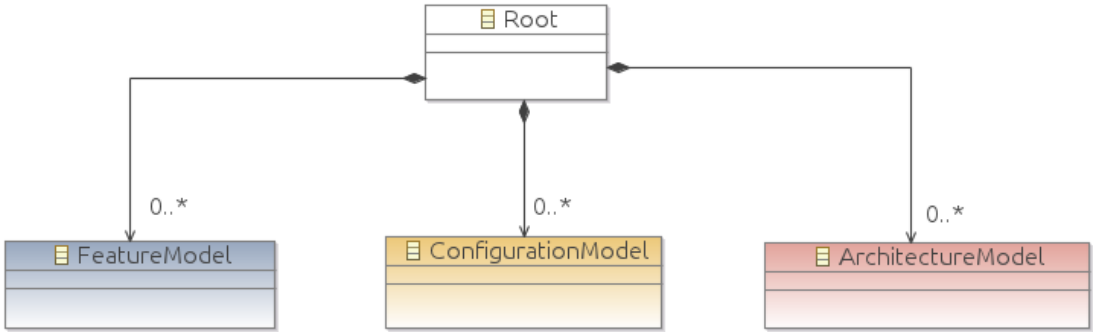


Figura 5.1: Fragmento do Metamodelo identificando os modelos que são instanciados.

O Modelo de Características possibilita que a derivação da Arquitetura do Produto seja feita por meio da seleção de características. Com base na Configuração de Características resultante, a Arquitetura do Produto é então derivada empregando o Modelo de Configuração, mais especificamente, usando os Contratos, associados tanto às características quanto às variabilidades descritas no nível arquitetural. A Configuração de Características serve de base, ainda, para as reconfigurações dinâmicas da Arquitetura do Produto, pois, caso seja modificada (*e.g.*, usuário seleciona uma nova característica dinâmica), a Arquitetura do Produto é adaptada. Estas propriedades classificam o Metamodelo proposto no grupo daqueles cuja abordagem é orientada a características (veja Capítulo 2, Seção 2.3.2).

O Modelo da Arquitetura, por sua vez, organiza os elementos arquiteturais em opcionalidades e pontos de variação. Os Contratos representam as variabilidades por meio de ações de adaptação, descritas de forma que possam ser realizadas durante a derivação e durante a execução da Arquitetura do Produto. Essa abordagem é essencial, principalmente por permitir que as variabilidades dinâmicas sejam configuradas junto à Arquitetura do Produto, tornando-a reconfigurável diante das mudanças do contexto de execução e das modificações feitas na Configuração de Características.

Com essa estruturação, uma LPS pode ser modelada desde as suas características até os seus elementos arquiteturais, de forma integrada e com recursos que possibilitam a derivação de arquiteturas estáticas e dinâmicas para os produtos. A Figura 5.2 apresenta o Metamodelo completo desenvolvido usando o EMF. As próximas subseções detalham o

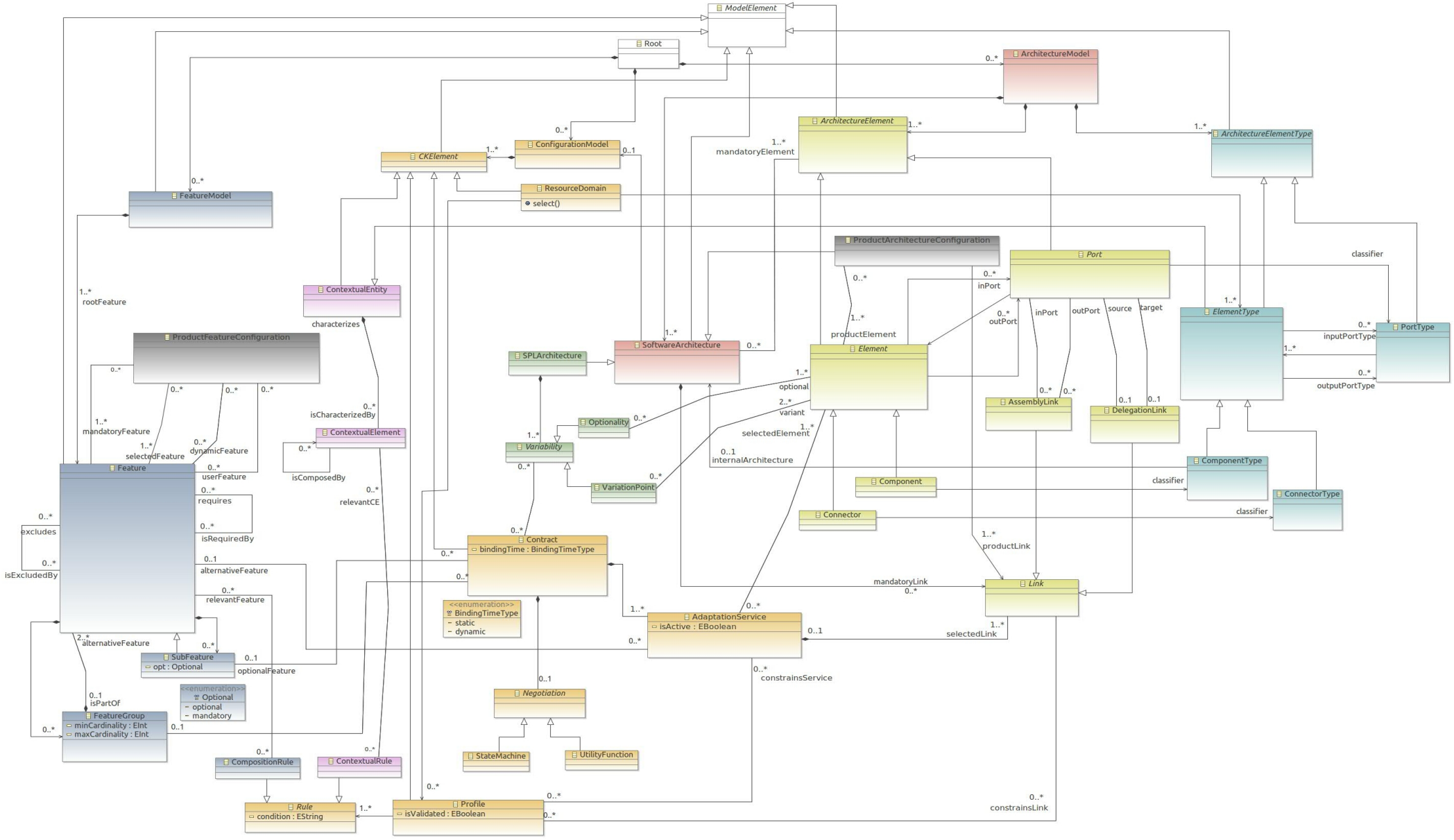


Figura 5.2: Metamodelo de Configuração de Variabilidades em LPS - Diagrama Completo.

Metamodelo, destacando cada uma das partes que o compõem.

5.2.1 Características

O Metamodelo de Características, apresentado na Figura 5.3, adota parte da proposta de Kang *et al.* [69] e parte da proposta de Czarnecki *et al.* [45] para representar as características e as suas inter-relações. Os seus elementos são detalhados nas próximas subseções.

5.2.1.1 Características Mandatórias, Opcionais e Alternativas

Modelos de Características são instanciados a partir da classe *FeatureModel* do Metamodelo, definindo-se características-raiz (papel *rootFeature*, classe *Feature*), subcaracterísticas (classe *SubFeature*) e características alternativas (papel *alternativeFeature*, classe *Feature*); veja Figura 5.3.

As características do tipo raiz permitem organizar um modelo em diferentes estruturas hierárquicas. As subcaracterísticas, por sua vez, compõem outras características (classe *Feature*), podendo ser opcionais ou mandatórias, conforme o atributo *opt* da classe *SubFeature*. Por fim, as características alternativas são aquelas estabelecidas em grupos (papel *isPartOf*, classe *FeatureGroup*).

A classe *FeatureGroup* possui os atributos *minCardinality* e *maxCardinality*, para que sejam definidas a quantidade mínima e a quantidade máxima de características alternativas que podem ser selecionadas para derivar a Configuração de Características para um produto. Relações OR e XOR são representadas por meio das cardinalidades. Por exemplo, no Modelo de Características do SCIADS, há um grupo definido com a cardinalidade <1-1>, representando a relação XOR, e há grupos com a cardinalidade <1-2> e <1-3>, representando a relação OR entre as características alternativas. Ao empregar o Metamodelo, o primeiro grupo é definido usando os atributos *minCardinality=1* e *maxCardinality=1*, enquanto que o segundo é definido com *minCardinality=1* e *maxCardinality=2*. O mecanismo de cardinalidade e de agrupamento adotado pelo Metamodelo está baseado na proposta de Czarnecki *et al.* [45], e ainda no conceito de grupo, descrito em [112] como um conjunto de características anotado com uma cardinalidade que especifica um intervalo envolvendo a quantidade de características que podem ser selecionadas daquele conjunto.

Comuns aos modelos de características [109, 69], as restrições de dependência são

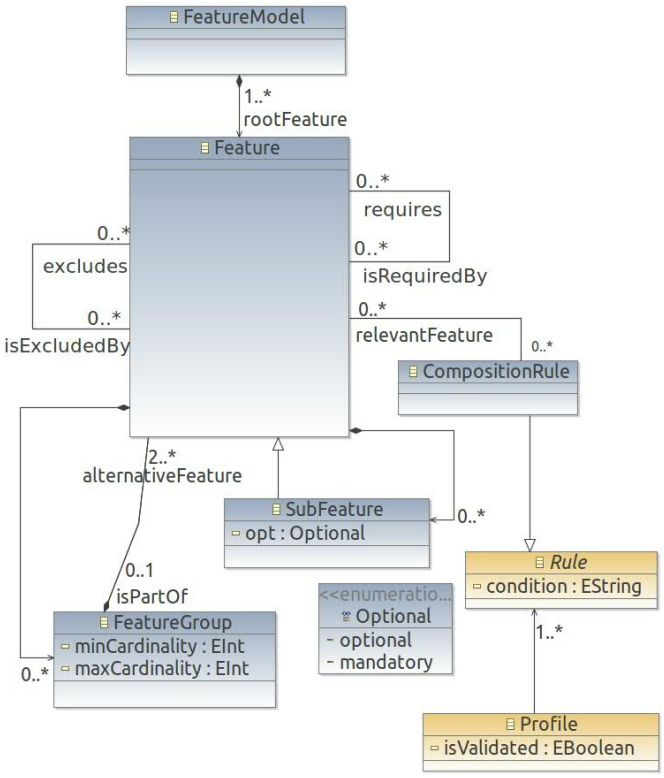


Figura 5.3: Metamodelo de Características.

representadas pelas autoassociações da classe *Feature*, as quais definem tanto as características que dependem quanto aquelas que são dependentes de outras. Dada uma característica existente no SCIADS, por exemplo *Hypertension*, ao navegar no modelo, descobre-se qual(is) a(s) característica(s) que requerem e/ou excluem *Hypertension* (*requires* e *excludes*), e qual(is) são requeridas e/ou excluídas por *Hypertension* (*isRequiredBy* e *isExcludedBy*).

5.2.1.2 Regras de Composição

As restrições de dependência entre características, assim como as relações OR e XOR em grupos, são descritas como Regras de Composição (classe *CompositionRule*, associação *relevantFeature*), as quais compõem os perfis (classe *Profile*), elementos do Metamodelo de Configuração. Proposições lógicas são usadas para descrever as regras, consistindo de um conjunto de símbolos ou variáveis representando as características, e um conjunto de conectores lógicos (*e.g.*, NOT, AND, OR) inter-relacionando os valores das variáveis, como descrito em [10, 87, 86]. Além de conter as restrições de dependência do tipo *requires*/*excludes* e as relações OR e XOR, as regras de composição devem incluir também as restrições envolvendo características opcionais, tanto aquelas compostas por grupos de características quanto aquelas que fazem parte de outras características opcionais (au-

toassociação da classe *Feature*). Estas últimas devem ter regras para verificar, durante a configuração de um produto, o estado da característica opcional hierarquicamente superior.

A tradução das relações de um Modelo de Características para fórmulas proposicionais vem sendo bastante explorada como uma forma de automatizar a detecção de inconsistências em modelos [11]. Elas possibilitam ainda a correta configuração de um produto da LPS, no sentido de prover produtos cuja arquitetura seja compatível com as restrições definidas no Modelo de Características.

5.2.1.3 Representação das Variabilidades

As variabilidades são representadas no Metamodelo pelas características definidas como opcionais (atributo *opt=optional*, classe *SubFeature*) e pelos grupos de características alternativas (atributos *minCardinality* e *maxCardinality*, classe *FeatureGroup*).

Além disso, as variabilidades podem ser estáticas ou dinâmicas, conforme o tempo de vinculação definido nos Contratos correspondentes às características (Figura 5.4). Variabilidades estáticas são representadas por características associadas a Contratos Estáticos (atributo *bindingTime=static*, classe *Contract*), enquanto que variabilidades dinâmicas são representadas por características associadas a Contratos Dinâmicos (atributo *bindingTime=dynamic*, classe *Contract*). Tais características também são classificadas quanto ao tempo de vinculação, em características estáticas e características dinâmicas, respectivamente.

O tempo de vinculação é, em geral, tratado apenas como uma informação adicional em Modelos de Características [45, 69]. No entanto, no Metamodelo aqui proposto ele é fundamental, pois permite classificar a variabilidade que será manipulada no nível da Arquitetura do Produto. Para as variabilidades dinâmicas, em especial, um ou mais Contratos Dinâmicos devem ser construídos com cláusulas de negociação e serviços capazes de identificar as porções da arquitetura que devem se adaptar em tempo de execução. A Seção 5.2.3 aborda Contratos Dinâmicos, serviços e negociação.

5.2.2 Arquitetura

A estruturação do Metamodelo de Arquitetura tem como fundamento a concepção básica de Arquitetura de Software como um conjunto de componentes interconectados por conectores [123, 91], onde, tanto os componentes quanto os conectores possuem por-

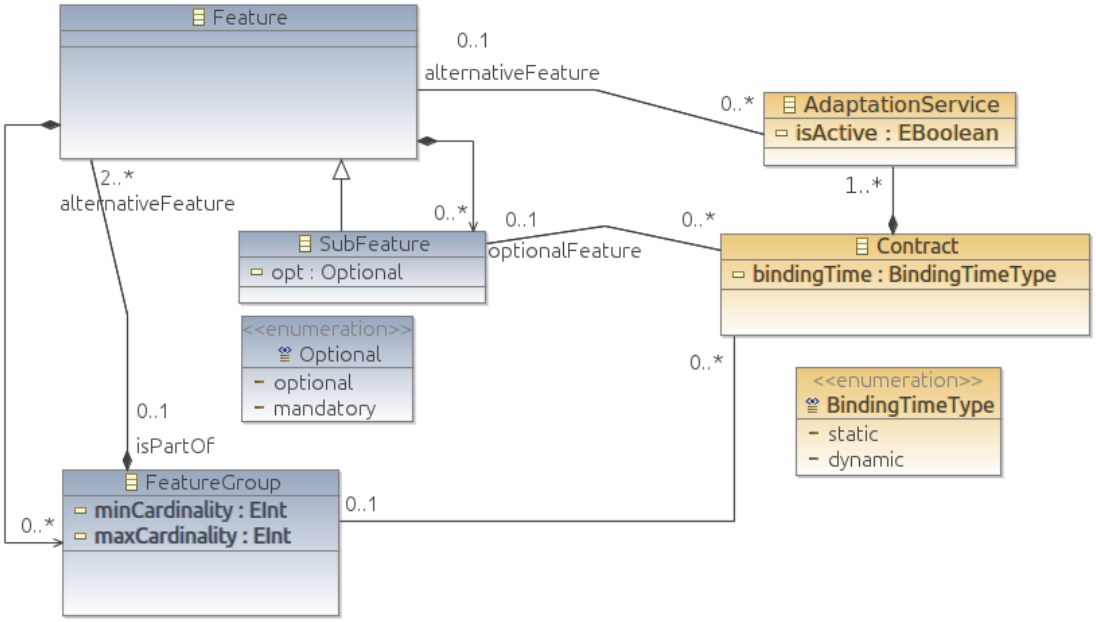


Figura 5.4: Associação entre Características e Contratos.

tas de entrada e portas de saída que identificam os pontos de interligação e possibilitam as interações. Ao criar um Modelo de Arquitetura a partir do Metamodelo, os elementos definidos pelas classes *Component*, *Connector* e *Port* são interligados para formar a Arquitetura da LPS.

A Figura 5.5 apresenta o Metamodelo de Arquitetura, cujos elementos são detalhados nas próximas subseções.

5.2.2.1 Instâncias e Tipos dos Elementos Arquiteturais

No Metamodelo de Arquitetura, a classe *SoftwareArchitecture* é ponto de partida para a criação de Modelos Arquiteturais. Uma Arquitetura de Software é fundamentalmente estruturada por instâncias de componentes, de conectores e de portas [38, 55]. Diante disso, os elementos da arquitetura são representados no Metamodelo de duas formas: na forma de tipo (ou classe) e na forma de instância. Essa distinção é essencial pois possibilita a construção de arquiteturas interligando instâncias criadas a partir de um tipo previamente definido. A Figura 5.6 destaca a representação de tipos e de instâncias.

Os tipos estão representados pela classe abstrata *ArchitectureElementType*, a qual é parte do Modelo de Arquitetura representado pela classe *ArchitectureModel*. A partir desta classe abstrata são criados tipos para componentes (classe *ComponentType*), para conectores (classe *ConnectorType*) e para portas (classe *PortType*).

As instâncias de componentes, conectores e portas são representadas pelas classes

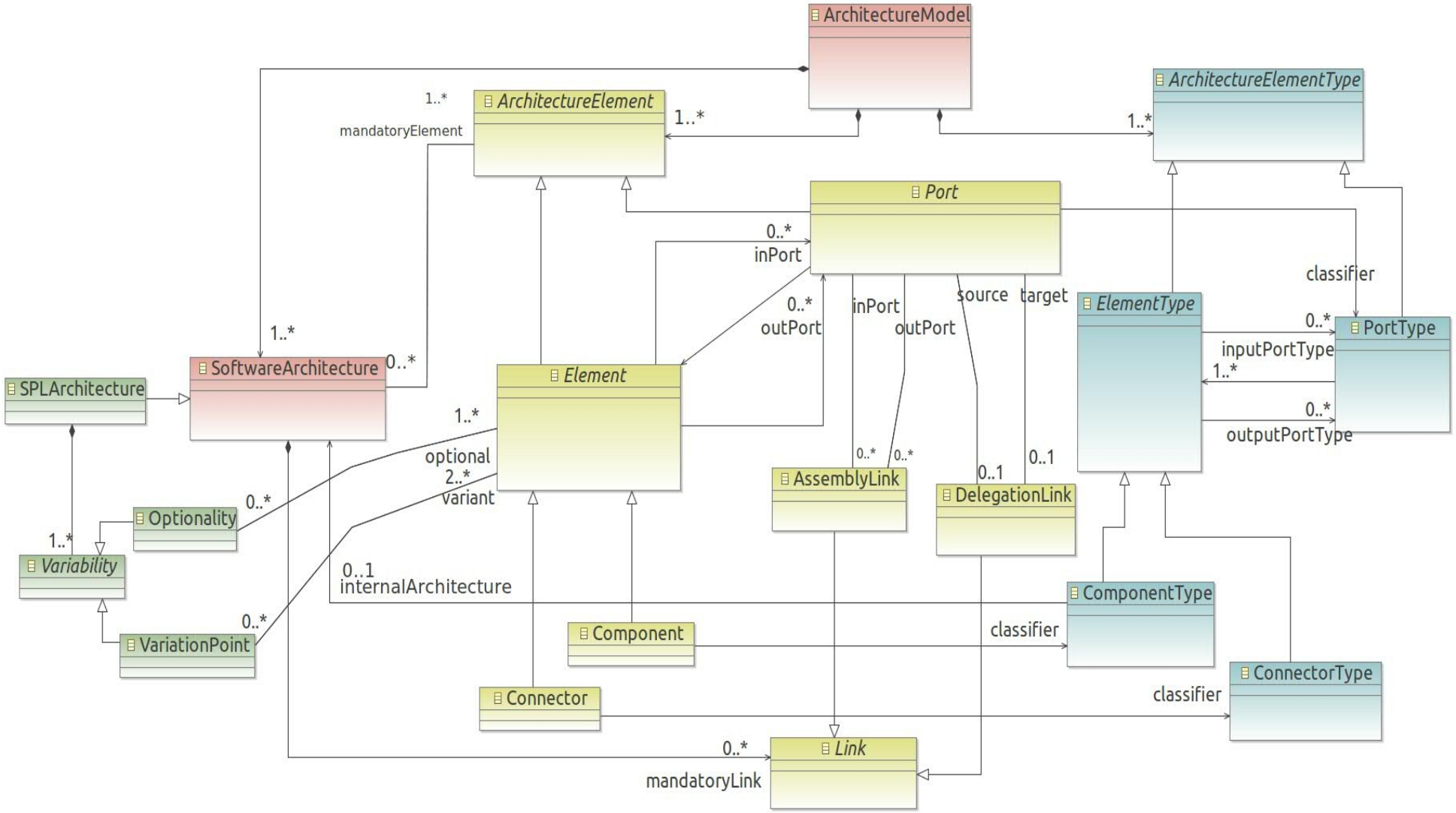


Figura 5.5: Metamodelo de Arquitetura.

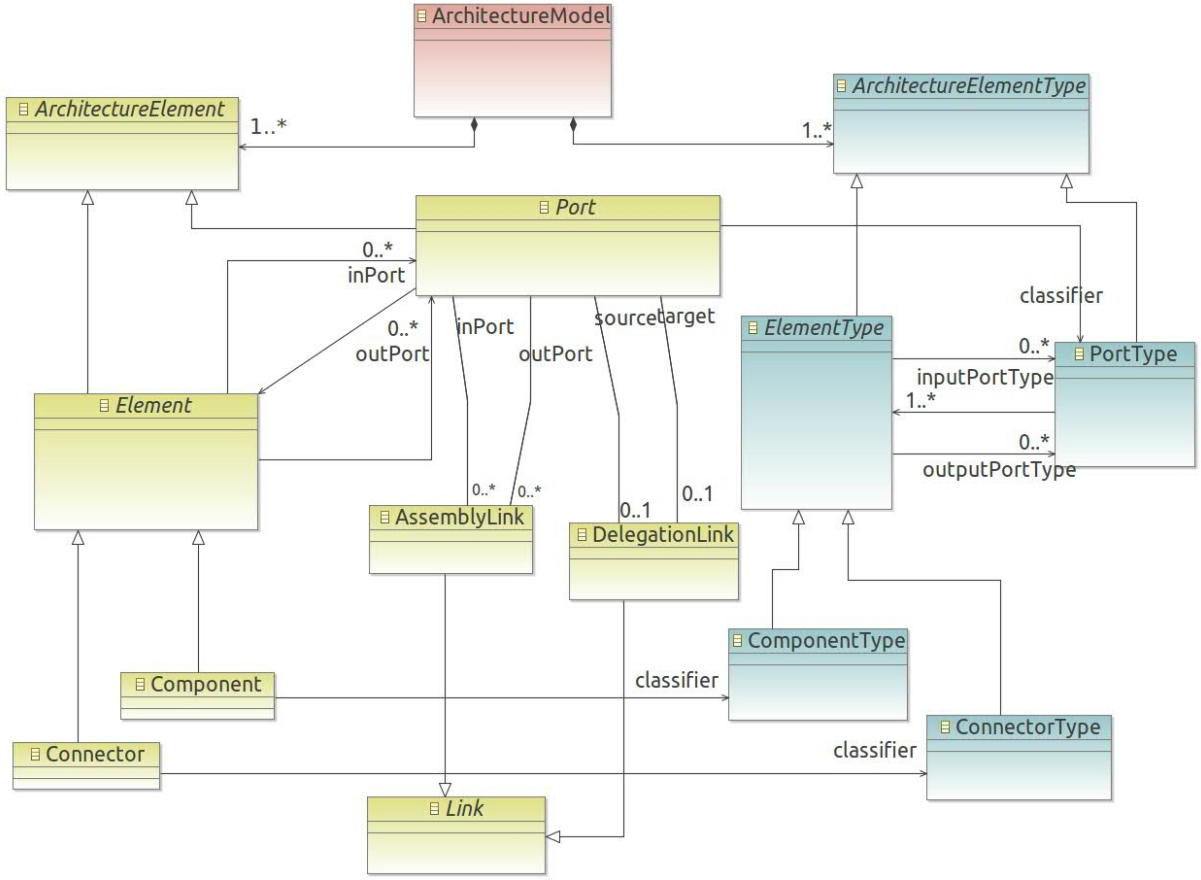


Figura 5.6: Representação de tipos e instâncias para os elementos da arquitetura.

Component, *Connector* e *Port*, respectivamente, e possuem associações que determinam os seus respectivos classificadores (*classifier*). O relacionamento tipo-instância dos elementos da arquitetura se assemelha ao relacionamento classe-objeto da modelagem orientada a objetos.

Durante a criação da Arquitetura de Software, representada pela classe *SoftwareArchitecture*, as instâncias dos elementos são criadas, assim como a interconexão entre elas, representada pela classe abstrata *Link*. Essa solução traz uma separação clara entre tipos e instâncias de elementos arquiteturais, permitindo principalmente que os tipos sejam reutilizados em outras arquiteturas.

5.2.2.2 Interligação de Componentes e Conectores

Em um Modelo de Arquitetura instanciado a partir do Metamodelo, tanto os componentes quanto os conectores podem ter portas de entrada e portas de saída. Além disso, os componentes são interligados a outros de forma direta ou por meio de um ou mais conectores explícitos. Um conector é interligado a um componente ou a outro conector, produzindo uma arquitetura flexível em termos de suas ligações.

No Metamodelo, há duas formas de interligar elementos da arquitetura, definidas pelas classes *AssemblyLink* e *DelegationLink* (Figura 5.7). A primeira é a mais comum e representa uma interligação cuja origem (papel *outPort*) deve ser uma porta de saída e cujo destino (papel *inPort*) deve ser uma porta de entrada, indicando a interação entre um elemento que requer e outro elemento que fornece serviços. A outra forma de interligação é empregada para a modelagem da arquitetura interna de componentes (veja Subseção 5.2.2.3). Ambas as classes são subclasses concretas da classe abstrata *Link*, a qual é parte da arquitetura representada pela classe *SoftwareArchitecture*.

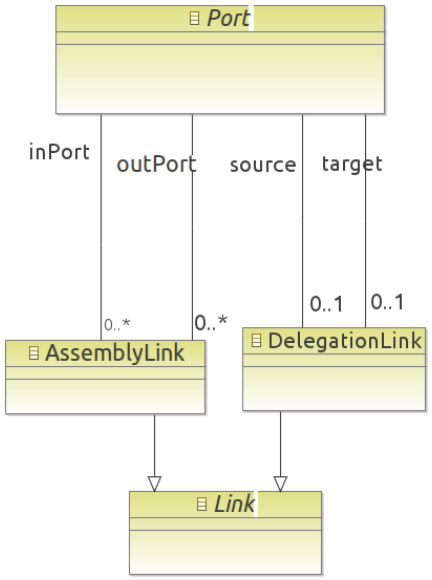


Figura 5.7: Interligação dos elementos da arquitetura com as classes *AssemblyLink* e *DelegationLink*.

5.2.2.3 Arquitetura Interna para Tipo de Componente

Uma arquitetura pode ser definida para a estrutura interna de um Tipo Componente (*ComponentType*). Para isso, deve-se usar a associação *internalArchitecture* para criar a arquitetura, a qual é então estabelecida com os seus elementos e interligações. Além de se criar as interligações usando a classe *AssemblyLink* (veja Subseção 5.2.2.2), deve-se também criar interligações usando a classe *DelegationLink*.

A classe *DelegationLink* permite definir a interligação dos elementos internos do *ComponentType*, cuja arquitetura está sendo modelada, com as portas do próprio *ComponentType*. A interligação usando *DelegationLink* é feita entre as portas de entrada do *ComponentType* e as portas de entrada de seus elementos internos – *source* e *target* representando portas de entrada – e entre as portas de saída do *ComponentType* e as portas de saída de seus elementos internos – *source* e *target* representando portas de saída (Figura 5.7).

Essas ligações indicam que o comportamento descrito nas portas de entrada e de saída do componente devem ser realizados pelos elementos internos ao componente.

5.2.2.4 Núcleo da Arquitetura de uma LPS

As associações *mandatoryElement* e *mandatoryLink* da classe *SoftwareArchitecture* formam o núcleo da arquitetura de uma LPS (Figura 5.8) . A primeira associação, estabelecida com a classe abstrata *ArchitectureElement*, representa os elementos mandatórios da arquitetura da LPS, os quais podem ser componentes (*Component*), conectores (*Connector*) e portas (*Port*). A segunda associação, por sua vez, estabelecida com a classe abstrata *Link*, representa as interligações entre esses elementos mandatórios.

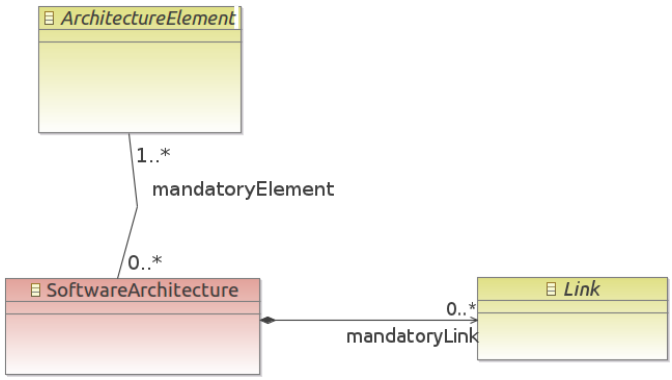


Figura 5.8: Elementos e interligações mandatórias formam o núcleo da Arquitetura da LPS.

5.2.2.5 Representação das Variabilidades

As classes do Metamodelo até aqui apresentadas dão condições de se criar diferentes modelos para Arquiteturas de Software com tipos, instâncias e interligações dos seus elementos. Para representar a arquitetura de uma LPS, no entanto, é necessário que se represente também as suas variabilidades, com o objetivo de se reutilizar a mesma arquitetura entre os diferentes produtos derivados da LPS. Nesse sentido, o Metamodelo segue os mesmos princípios de Arquitetura de Software acrescidos da representação de variabilidades. Isso é feito definindo-se a classe *SPLArchitecture* como uma subclasse de *SoftwareArchitecture*, e a classe abstrata *Variability* com um destaque para dois tipos de variabilidade: Opcionalidade (classe *Optionality*) e Ponto de Variação (classe *Variation Point*) (Figura 5.9).

Tanto a Opcionalidade quanto o Ponto de Variação têm associação com os elementos da arquitetura. Para que uma Opcionalidade seja representada na Arquitetura da

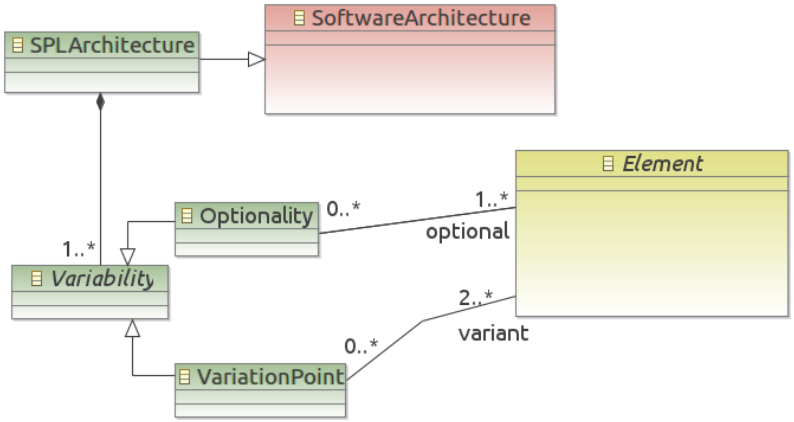


Figura 5.9: Opcionalidades e Pontos de Variação na Arquitetura da LPS.

LPS, deve estar associada a pelo menos um elemento da arquitetura, neste caso denominado de opcional (papel *optional*), enquanto que, para um Ponto de Variação, devem estar associados pelo menos dois elementos da arquitetura, denominados variantes (papel *variant*).

Os conceitos de opcionalidade e ponto de variação em Arquiteturas de LPS guardam semelhança com os conceitos de característica opcional e alternativa, presentes no Metamodelo de Características. Apesar dos modelos atuarem em diferentes níveis, uma vez que Modelos de Características são mais conceituais e tratam do espaço-problema, e Arquiteturas de LPS estão mais centradas nos elementos que efetivamente contribuem para o espaço-solução, a metamodelagem da adaptação arquitetural, proposta nesta tese, explora essa similaridade. O objetivo é associar características com elementos da arquitetura por meio da implementação dos Contratos, serviços de adaptação e regras. Vale a pena ressaltar que no Metamodelo os elementos mandatórios não são representados no contexto da classe *SPLArchitecture*, mas sim junto à classe *SoftwareArchitecture*, como descrito na Subseção 5.2.2.4.

5.2.3 Configuração

O Metamodelo de Configuração, apresentado na Figura 5.10, reúne o conhecimento necessário para a configuração de variabilidades de uma LPS, de uma forma independente dos demais metamodelos (Características e Arquitetura).

Esse conhecimento, chamado *Configuration Knowledge* (CK) (veja Capítulo 2), contém os seguintes elementos:

- Contratos e seus Serviços de Adaptação, para representar as variabilidades na forma

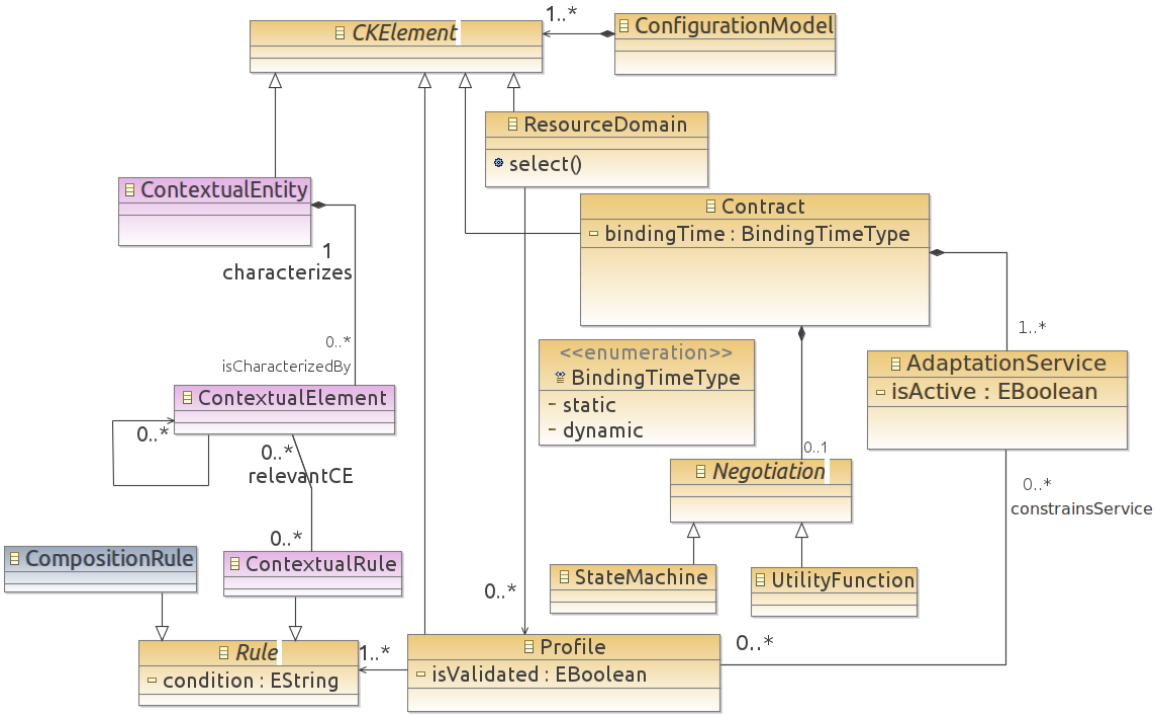


Figura 5.10: Metamodelo de Configuração.

de adaptações arquiteturais;

- Perfis, para representar as Regras de Composição e as Regras de Contexto;
- Entidades e Elementos de Contexto, para representar a Modelagem de Contexto dos produtos;
- Negociação, para representar os critérios para se escolher dinamicamente os serviços de adaptação.

Os elementos do CK são definidos de forma explícita sem qualquer entrelaçamento com os demais modelos da LPS (Características e Arquitetura). As próximas subseções detalham os elementos do CK.

5.2.3.1 Contratos e Serviços de Adaptação

Um dos focos do Metamodelo está na modelagem de variabilidades dinâmicas e estáticas na forma de Contratos, elementos de primeira ordem modelados no nível da Arquitetura da LPS. O Contrato permite associar as variabilidades da arquitetura às características da LPS, e representar as ações de adaptação que devem ser realizadas junto à Arquitetura do Produto quando de sua configuração. Estas ações são explicitamente definidas no Contrato, e de forma independente dos modelos da LPS.

Contratos são construídos tanto para os pontos de variação quanto para as opcionalidades da Arquitetura da LPS. A Figura 5.11 destaca a associação de Contratos com as variabilidades da arquitetura.

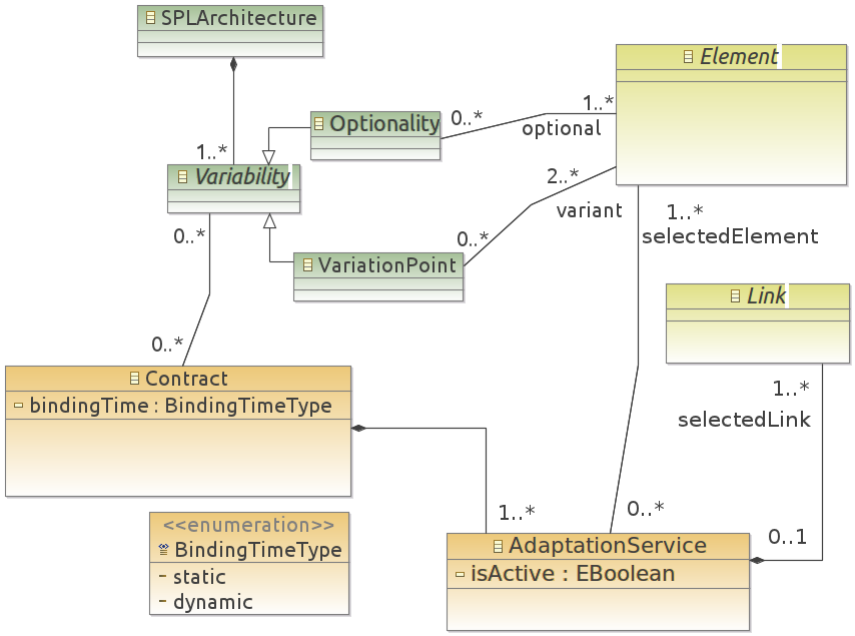


Figura 5.11: Contratos associados às variabilidades da Arquitetura da LPS.

Ao associar um Contrato a um determinado ponto de variação, permite-se a realização de diferentes configurações envolvendo suas variantes. Por exemplo, um Contrato pode ser modelado para remover ou adicionar componentes da arquitetura, conforme requerido no respectivo ponto de variação. Essas configurações são definidas nos serviços de adaptação do respectivo Contrato (classe *AdaptationService*), onde são descritos os mecanismos de adaptação envolvendo um ou mais componentes da arquitetura correspondentes às variantes. Estes serviços têm acesso aos elementos da arquitetura por meio da associação (*selectedElement*), e às interligações entre eles por meio da associação (*selectedLink*). Ao associar um Contrato a uma opcionalidade, os serviços também são descritos com mecanismos que envolvem componentes da Arquitetura da LPS, mas dessa vez, somente aqueles correspondentes ao elemento opcional.

Um Contrato é classificado em estático ou dinâmico, de acordo com o tempo de vinculação definido no seu atributo *bindingTime*. Contratos Estáticos são descritos para lidar com as variabilidades estáticas, enquanto Contratos Dinâmicos lidam com variabilidades dinâmicas. Ao reunir ambos os tipos de Contratos, estáticos e dinâmicos, Modelos de Arquitetura de LPS mais completos podem ser criados e, por conseguinte, mais personalizáveis e flexíveis. Isso torna possível a derivação, desde arquiteturas estáticas sem qualquer variabilidade, passando por arquiteturas contando apenas com variabilidades

dinâmicas, até arquiteturas de produtos híbridas, onde variabilidades são resolvidas tanto em tempo de derivação quanto em tempo de execução.

5.2.3.2 Correspondência Característica-Arquitetura

Os Contratos e seus serviços de adaptação exercem o papel de ligação entre características e elementos arquiteturais, facilitando a correspondência característica-arquitetura.

Dada uma característica, os seus respectivos elementos arquiteturais são alcançados por meio dos Contratos e dos serviços, os quais representam a adaptação arquitetural que deve ser feita para a característica. Cada serviço conhece os elementos da arquitetura correspondentes àquela característica por meio da associação *selectedElement*, e alcança a característica correspondente por meio das associações *optionalFeature* e *alternativeFeature*. A Figura 5.12 destaca a associação entre características, Contratos e elementos da arquitetura.

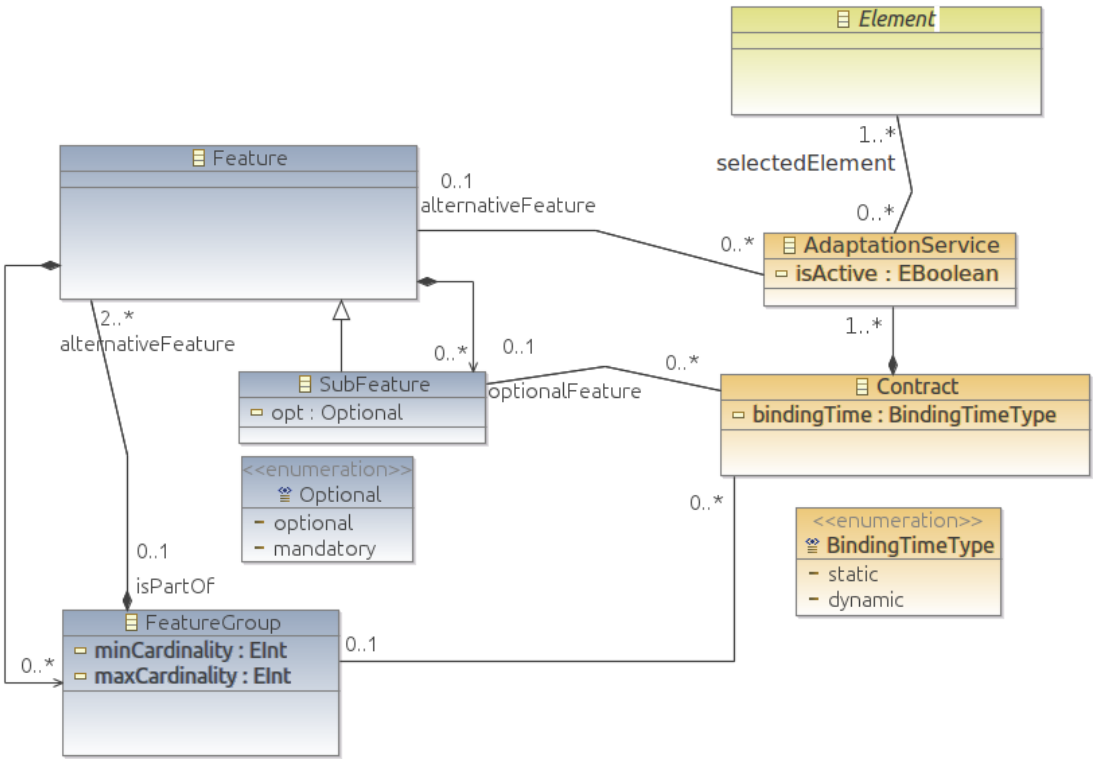


Figura 5.12: Associação entre características, Contratos e elementos da arquitetura.

Uma característica opcional alcança os elementos arquiteturais por meio da associação entre a característica e os seus Contratos (papel *optionalFeature*), e da associação entre estes Contratos e o serviço de adaptação. Isso é possível pois, cada Contrato associado a uma característica opcional deve ter apenas um serviço modelado para a respectiva

característica. Uma característica alternativa, por outro lado, alcança os elementos arquiteturais correspondentes por meio da associação direta entre a característica alternativa e o serviço de adaptação (papel *alternativeFeature*). É importante ressaltar que esse mecanismo foi construído para representar variabilidades, não prevendo a correspondência de características mandatórias a elementos da arquitetura.

Diferentemente das características, os elementos de uma arquitetura são representados pela classe abstrata *Element*, a qual possui a associação *selectedElement* com a classe *AdaptationService*. Essa associação indica que, dado um elemento, *Component* ou *Connector*, os serviços de adaptação que os selecionaram (configuraram) são identificados por meio da associação *selectedElement* e, por conseguinte, alcançam as características opcionais correspondentes por meio dos Contratos associados (via *optionalFeature*) e as características alternativas por meio da associação *alternativeFeature*. Vale ressaltar que esse mecanismo não está previsto para os elementos mandatórios da arquitetura.

5.2.3.3 Modelagem de Contexto

O Metamodelo de Configuração inclui a modelagem do contexto de execução para os produtos da LPS. É importante ressaltar, no entanto, que não é objetivo desta tese detalhar todos os aspectos relacionados à modelagem de contexto, mas sim apresentar um modelo que represente essencialmente entidades e elementos de contexto. Detalhes a respeito de como modelar contexto, incluindo metamodelagem, são encontrados em [132].

A modelagem proposta estende a concepção de Categorias, elementos usados na construção dos Contratos apresentados no Capítulo 4. Um Modelo de Contexto deve ser definido por duas classes, como apresentado na Figura 5.13: entidade de contexto (*ContextualEntity*) e elemento de contexto (*ContextualElement*). Uma entidade de contexto representa um objeto que pode ser identificado e que seja relevante no ambiente de execução do produto, enquanto um elemento de contexto representa uma propriedade usada para caracterizar uma entidade de contexto. Por exemplo, no domínio do SCIADS, um ambiente da casa (*e.g.*, sala de estar) é uma entidade, enquanto que temperatura, umidade e dimensões, são elementos do contexto.

Elementos arquiteturais também podem ser definidos como entidades de contexto, como representado na Figura 5.13, onde *ElementType* é uma subclasse de *ContextualEntity*. Isso faz com que componentes e conectores se comportem também como entidades de contexto. Por exemplo, a localização de uma TV pode ser representada como um elemento de contexto definido como parte do componente TV por meio da associação

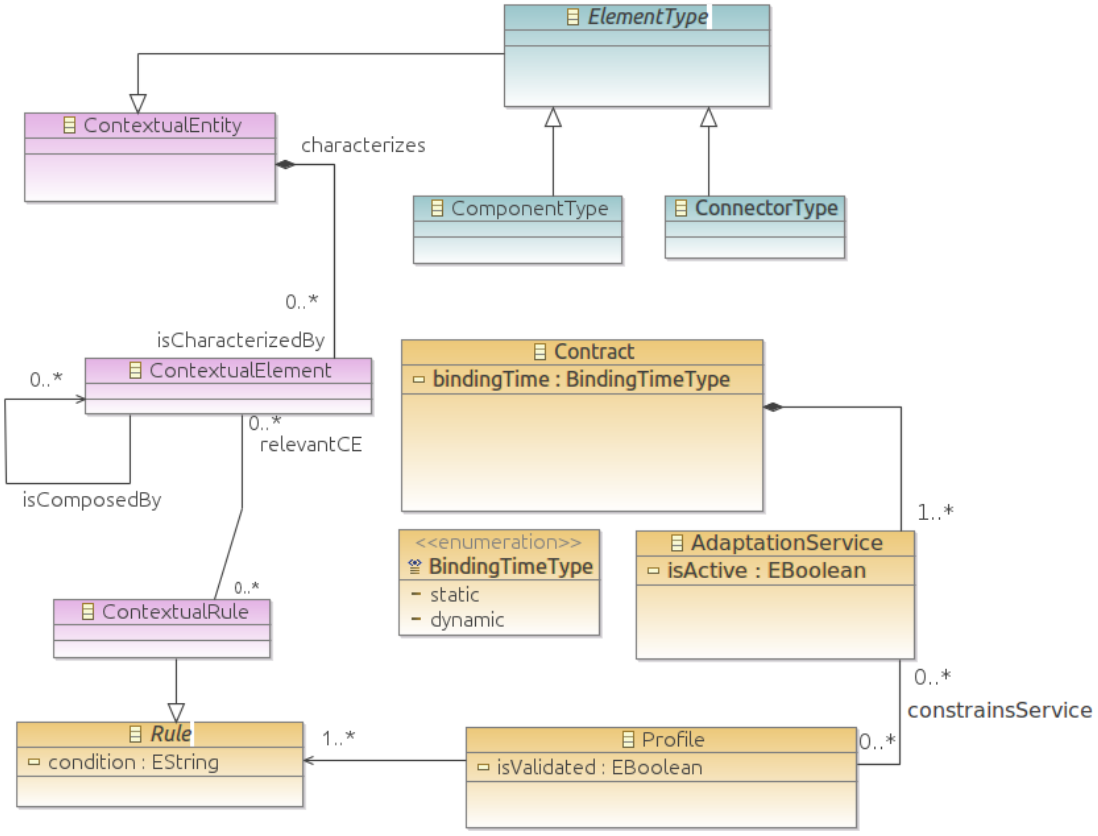


Figura 5.13: Modelagem de Contexto.

isCharacterizedBy. Essa representação permite que recursos sejam descobertos a partir do Tipo dos componentes ou conectores, como descrito na Seção 5.2.3.5.

As Regras de Contexto (classe *ContextualRule*) são construídas a partir destas informações modeladas, tendo como variáveis os elementos de contexto (associação *relevantCE*) cujos valores são avaliados pelas regras. Assim como as regras de composição, as regras de contexto também fazem parte dos perfis.

5.2.3.4 Perfil

Perfis (classe *Profile*) são compostos por regras de contexto ou por regras de composição (Figura 5.14). Independentemente do tipo da regra, um perfil torna-se válido (classe *Profile*, atributo *isValidated=true*) quando as condições definidas em todas as regras que o compõem são satisfeitas. Em outras palavras, um perfil é válido quando a operação de conjunção realizada entre os resultados lógicos de todas estas condições produzir um valor *true*.

Quando composto por regras de contexto, o perfil controla a ativação dos serviços de adaptação por meio da associação *constrainsService*. Um serviço de adaptação é ati-

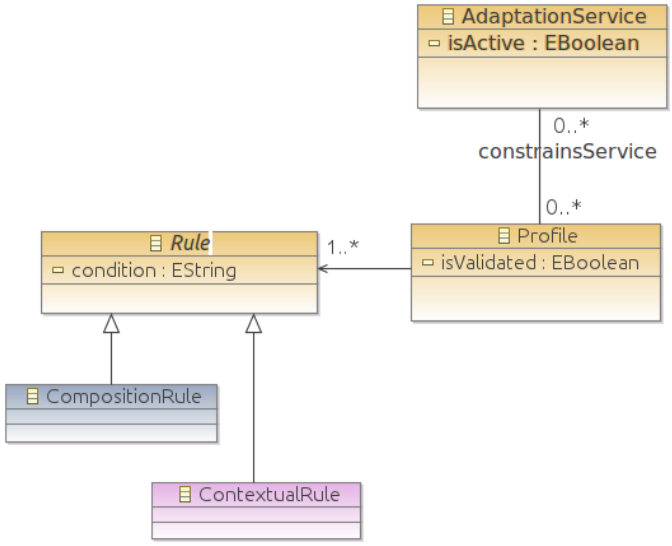


Figura 5.14: Perfis e regras.

vado (atributo *isActive* de *AdaptationService*) e as suas ações de adaptação arquitetural são executadas somente se todos os perfis a ele associados forem válidos, ou seja, se as condições definidas nas regras de contexto forem satisfeitas. Mesmo depois de ativado, um serviço continua dependente dos seus perfis para que permaneça ativo. Por exemplo, um serviço é desativado caso um dos seus perfis se torne inválido por conta de suas regras não serem mais satisfeitas. Nesta situação, a configuração arquitetural constituída pelo serviço é desfeita, uma vez que a sua execução é interrompida. Os componentes usados por este serviço podem, no entanto, continuar disponíveis para que sejam utilizados por outros serviços. É importante destacar que, para a infraestrutura de suporte realizar operações como essas, os componentes envolvidos devem estar em um estado que forneçam meios de preservar a consistência do produto em execução [73, 75].

Quando composto por regras de composição, o perfil válido indica que a Configuração de Características gerada durante a derivação de um produto é válida. Em outras palavras, indica que todas as regras de composição associadas às características da Configuração de Características estão satisfeitas. Essa validação evita que um serviço de adaptação gere, ao ser executado, uma arquitetura incompatível com as restrições de dependência e as relações OR e XOR definidas. Por exemplo, um serviço de adaptação modelado para adicionar um componente à arquitetura de um produto, somente é ativado se a característica correspondente a este componente estiver com as suas restrições de dependência resolvidas, ou seja, com as suas regras de composição satisfeitas.

A Seção 5.3.1 descreve as diretrizes do Metamodelo em relação à criação de perfis e a Seção 5.4.3 descreve as diretrizes em relação ao uso de perfis nas mudanças de contexto e

na criação da Configuração de Características.

5.2.3.5 Domínio de Recursos

Em Sistemas Ubíquos, recursos, como, por exemplo, sensores ou dispositivos, podem continuamente entrar e sair do ambiente. Um determinado recurso, ao entrar no ambiente, se registra automaticamente em um Diretório de Recursos (veja Seção 4.2.2), com o seu tipo e as suas propriedades de contexto, definidas nas classes *ContextualEntity* e *ContextualElement*. Com isso, o recurso fica disponível para que seja posteriormente configurado junto à Arquitetura do Produto em execução.

Nesse sentido, uma característica dinâmica pode ser modelada para representar um ou mais recursos com essas propriedades, e associada a Contratos Dinâmicos cujos serviços de adaptação usem um mecanismo que descubra os recursos registrados junto ao Diretório de Recursos e os configure na arquitetura. O Diretório de Recursos armazena os Tipos e as instâncias de recursos disponíveis, permitindo realizar buscas delimitadas pelo Tipo e também pelos valores das propriedades de contexto dos recursos, com o objetivo de retornar aquelas instâncias que satisfazem as condições descritas nas regras de contexto.

Para que a Descoberta de Recursos funcione, entretanto, deve-se definir um Domínio de Recursos, com um escopo, grupo ou arranjo de recursos a serem recuperados do diretório. No Metamodelo, o Domínio de Recursos é modelado usando-se a classe *ResourceDomain* (Figura 5.15). Essa classe possui uma associação com a classe abstrata *ElementType*, identificando portanto que a definição do Domínio de Recursos está relacionada aos Tipos dos elementos da arquitetura (*ComponentType* e *ConnectorType*). A classe *ResourceDomain* também está associada a perfis (*Profile*), o que permite que regras de contexto sejam definidas como critérios para a descoberta dos recursos. Além disso, a classe oferece uma operação para executar a descoberta dos recursos do domínio (operação *select()*).

Por exemplo, no contexto do SCIADS, produtos desenvolvidos para serem implantados em uma plataforma que ofereça Diretório, Registro e Descoberta de Recursos, podem ser configurados com a característica dinâmica TV associada a um Contrato que permita a descoberta e a configuração na Arquitetura do Produto de novas TVs que venham a se registrar no seu ambiente de execução.

Para que todas as TVs da casa possam ser descobertas pela operação *select()*, *ResourceDomain* deve ser associada ao Tipo *TVType*, neste exemplo, uma instância de

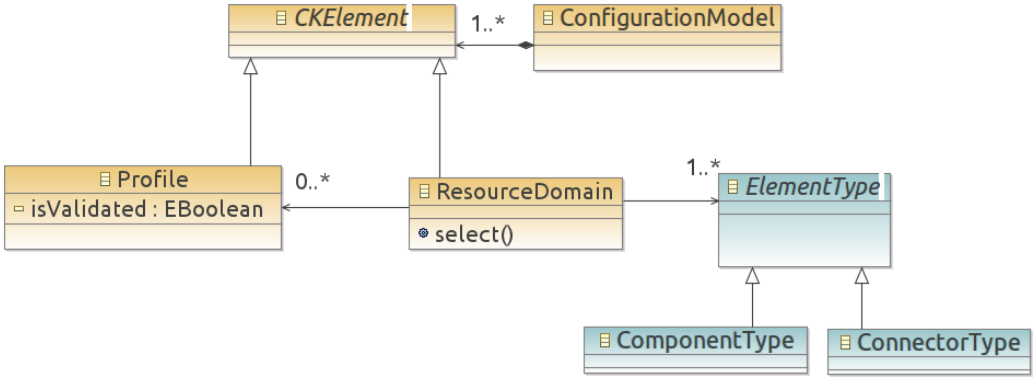


Figura 5.15: Domínios de Recursos são representados pela classe *ResourceDomain*.

ComponentType que representa o Tipo de componente para TV. Ou ainda, caso se queira realizar a descoberta apenas das TVs que estejam mais próximas ao paciente, regras de contexto devem ser criadas para satisfazer a essa condição envolvendo a localização das TVs e do paciente. Esta informação é definida em *ContextualEntity* e *ContextualElement*, de forma que seja usada no processo de descoberta das TVs. As regras criadas são associadas a um perfil que, por sua vez, é associado a *ResourceDomain*. O Capítulo 6 apresenta Diagramas de Objeto da UML desenvolvidos a partir do Metamodelo que demonstram esse cenário.

5.2.3.6 Negociação

Um serviço é ativado quando os perfis a ele associados são válidos, ou seja, quando as regras que o compõem são satisfeitas. Se, em um mesmo Contrato, mais de um serviço se tornar ativo, é necessário modelar algum mecanismo utilizando critérios como máquinas de estado ou funções de utilidade, para definir quais e/ou em que ordem devem ser executados. Na classe abstrata *Negotiation*, associada a *Contract*, podem ser modelados esses mecanismos.

Em *Negotiation*, mostrada na Figura 5.16, pode ser modelada uma máquina de estados (classe *StateMachine*), onde cada estado representa um serviço de adaptação. Assim, um serviço ao ser desativado por conta, por exemplo, do perfil tornar-se inválido, é interrompido e uma transição feita para outro serviço definido na máquina de estados. A modelagem por máquina de estados é útil quando se deseja que apenas um serviço de adaptação esteja ativo no escopo de um Contrato. Outra maneira de modelar a negociação é usando funções utilidade (classe *UtilityFunction*) para estabelecer políticas para a seleção de serviços. Um serviço é iniciado de acordo com propriedades especificando pesos que expressem a sua importância relativa a outros serviços do Contrato. O Meta-

modelo não impõe uma técnica de seleção de serviços de um Contrato, mas sugere que as subclasses de *Negotiation* sejam instanciadas para Contratos Dinâmicos.

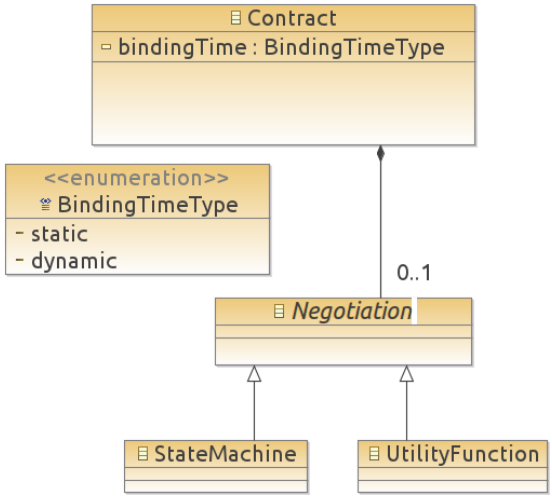


Figura 5.16: Negociação de serviços.

5.3 Modelagem da LPS

Esta seção descreve diretrizes compreendendo os elementos do Metamodelo e atividades que podem ser realizadas para modelar a LPS. As diretrizes estão organizadas considerando um processo constituído de duas etapas: a construção dos modelos da LPS (Características, Arquitetura e Configuração) e a derivação de produtos da LPS (Configuração de Características e Arquitetura).

5.3.1 Construção dos Modelos da LPS

A concepção de uma LPS passa eminentemente, em termos de modelagem, pela construção de um Modelo de Características e de um Modelo de Arquitetura. No contexto do Metamodelo proposto nessa tese, a modelagem inclui ainda a construção do Modelo de Configuração.

Ao modelar LPS empregando o Metamodelo, deve-se desenvolver, inicialmente, o Modelo de Características e, na sequência, descrever os elementos arquiteturais mandatórios com o objetivo de formar o núcleo da Arquitetura da LPS. A partir do Modelo de Características, uma análise deve ser feita para identificar as variabilidades do modelo e, por conseguinte, quais as características que devem ser modeladas como estáticas e quais que devem ser modeladas como dinâmicas. Uma vez identificadas as características, Con-

tratos Estáticos e Contratos Dinâmicos devem ser descritos para cada uma delas, levando em conta o tempo de vinculação definido. Além dos Contratos, os demais elementos do Modelo de Configuração devem ser definidos.

Durante esse processo, a descrição dos Contratos (estáticos e dinâmicos) é feita separadamente da descrição dos elementos mandatórios da arquitetura. Esta propriedade de separação de interesses torna o desenvolvimento e a evolução dos Contratos independentes do núcleo da Arquitetura da LPS.

A diretriz para a atividade de construção dos Modelos da LPS é constituída das ações a seguir enumeradas. A Figura 5.17 apresenta um Diagrama de Atividades da UML organizando estas ações e identificando os objetos produzidos.

1. Desenvolver o Modelo de Características da LPS (classe *FeatureModel*), composto de um conjunto M de características mandatórias, de um conjunto V de características opcionais e alternativas representando as variabilidades estáticas e dinâmicas, tal que $M \neq \varnothing$, $V \neq \varnothing$, $M, V \subset F$, onde F é o conjunto de todas as características do modelo.

2. Descrever regras de composição entre características opcionais e alternativas (classe *CompositionRule*) a partir das restriçoes de dependência (classe *Feature*, associações *requires* e *excludes*) e das relações OR e XOR (classe *FeatureGroup*) do Modelo de Características.

3. Descrever perfis (classe *Profile*) reunindo as regras de composição.

4. Descrever os Tipos dos elementos da Arquitetura da LPS (classe *ArchitectureElementType*).

5. Descrever o núcleo da Arquitetura da LPS, constituído de elementos e interligações mandatórios (classe *SPLArchitecture*, associações *mandatoryElement* e *mandatoryLink*).

6. Descrever as variabilidades da Arquitetura da LPS, constituídos de elementos opcionais e variantes (classe *SPLArchitecture*, associações *optional* e *variant*).

7. Descrever Contratos Estáticos (classe *Contract*, atributo *bindingTime=static*) e seus serviços de adaptação, constituídos de elementos e interligações representando as variabilidades estáticas de V (classe *AdaptationService*, associações *selectedElement* e *selectedLink*).

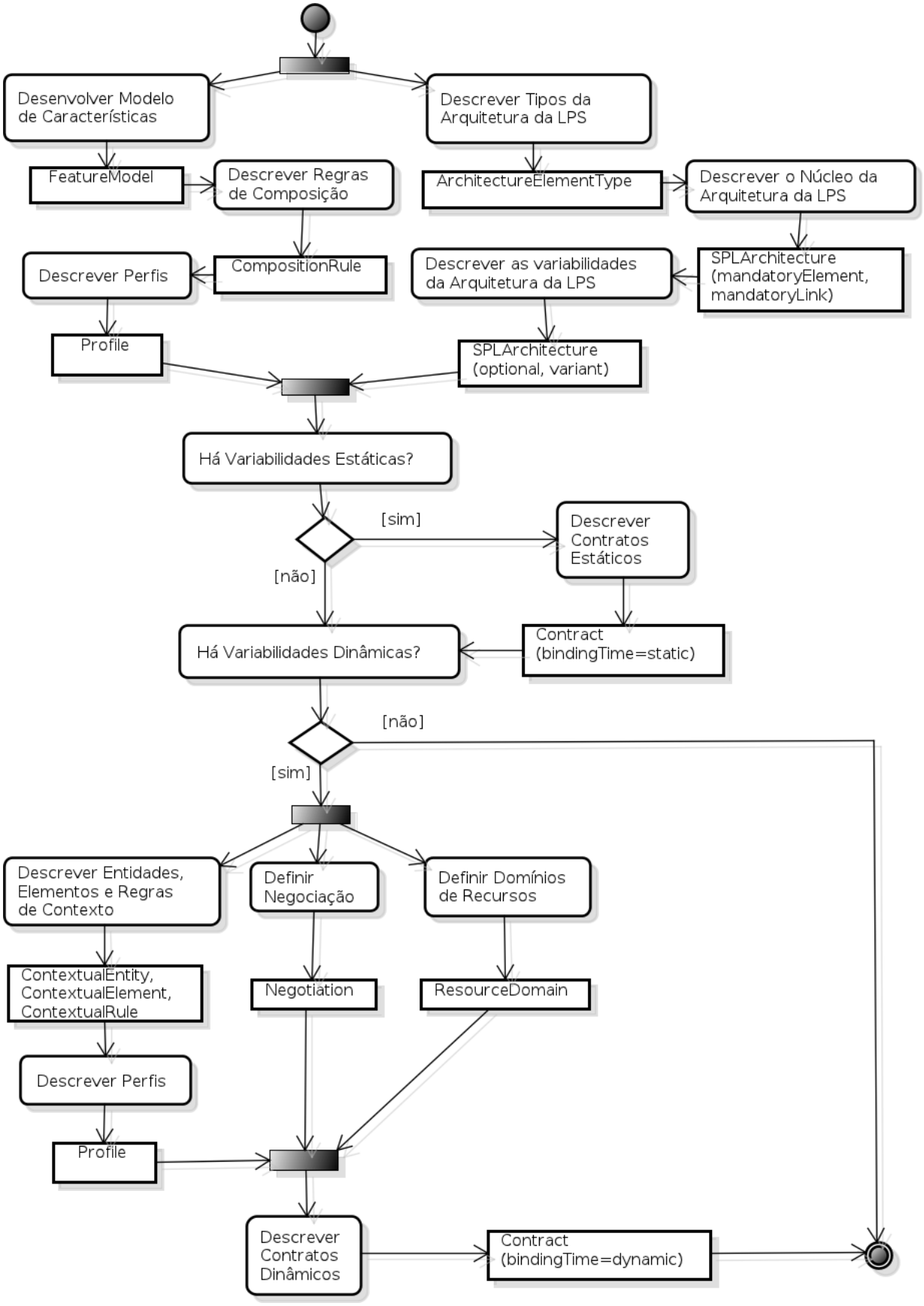


Figura 5.17: Diagrama de Atividades com ações para a construção da LPS.

8. Descrever entidades, elementos e regras de contexto (classes *ContextualEntity*, *ContextualElement* e *ContextualRule*).

9. Descrever perfis (classe *Profile*) incluindo as regras de contexto.

10. Definir um mecanismo de negociação de serviços para Contratos Dinâmicos (classe *Negotiation*).

11. Definir domínios de recursos (classe *ResourceDomain*) a serem empregados nos Contratos Dinâmicos.

12. Descrever Contratos Dinâmicos (classe *Contract*, atributo *bindingTime=dynamic*) e seus serviços de adaptação, constituídos de elementos e interligações representando as variabilidades dinâmicas de V (classe *AdaptationService*, associações *selectedElement* e *selectedLink*).

5.3.2 Derivação do Produto

Com os modelos da LPS construídos, a segunda etapa do processo ocorre quando um novo produto deve ser criado. A derivação do produto tem início com a resolução das variabilidades definidas no Modelo de Características. Isso é feito selecionando-se as características e gerando uma Configuração de Características para o produto. Na sequência, a Arquitetura do Produto é gerada a partir da sua Configuração de Características, e com base no núcleo da Arquitetura da LPS e no Modelo de Configuração. As próximas subseções detalham a geração da Configuração de Características e da Arquitetura do Produto.

5.3.2.1 Configuração de Características

Ao criar a Configuração de Características de um produto, deve-se levar em conta o tempo de vinculação de cada característica definido durante o desenvolvimento da LPS. Características dinâmicas, sejam opcionais ou alternativas, são selecionadas tanto em tempo de derivação quanto em tempo de execução do produto, enquanto características estáticas são selecionadas apenas em tempo de derivação.

A Configuração de Características do produto está representada no Metamodelo pela classe *ProductFeatureConfiguration*, mostrada na Figura 5.18. Ela possui quatro associações com a classe *Feature*: *mandatoryFeature*, *selectedFeature*, *dynamicFeature* e *userFeature*.

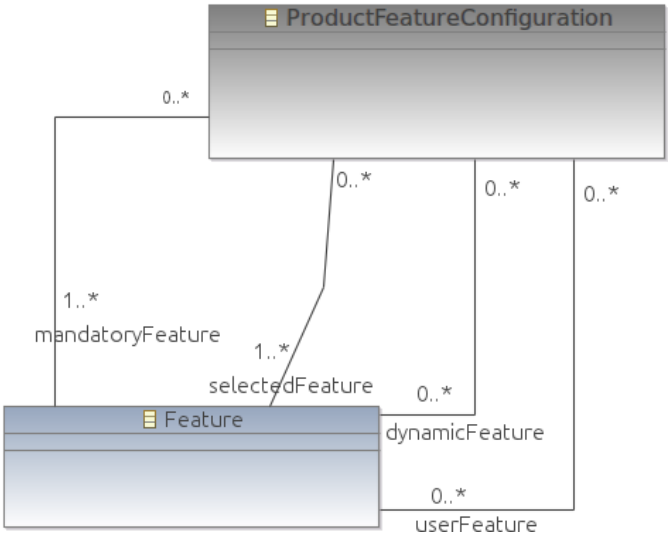


Figura 5.18: Classe *ProductFeatureConfiguration* representa a Configuração de Características de um produto.

- *mandatoryFeature* contém todas as características mandatórias definidas no Modelo de Características da LPS. As características desta associação são vinculadas já na implantação do produto.

- *selectedFeature* contém as características estáticas e dinâmicas selecionadas durante a derivação do produto, as quais são vinculadas já na sua implantação. Além destas características, esta associação recebe as características dinâmicas que forem selecionadas durante a execução do produto.

- *dynamicFeature*: contém as características dinâmicas que podem ou não ser selecionadas durante a execução do produto. As características desta associação são vinculadas apenas em tempo de execução.

- *userFeature*: contém as características dinâmicas modificadas pelo usuário em tempo de execução e que não podem ser modificadas por mudanças do contexto. Esta associação faz com que uma configuração feita pelo usuário não seja desfeita por mudanças de contexto.

A representação para a Configuração de Características tem inspiração na noção de configuração em estágios [45], onde as características de um produto são selecionadas de forma sucessiva até a sua configuração final, como descrito na Subseção 2.2.4. No caso da Configuração de Características, além de se selecionar as características que efetivamente vão compor o produto (*selectedFeature*), características dinâmicas podem ser selecionadas a *posteriori*, ou seja, em tempo de execução (*dynamicFeature*). As características sele-

cionadas em ambas as associações, *selectedFeature* e *dynamicFeature*, devem ser validadas frente às regras de composição definidas durante o processo de desenvolvimento da LPS, apresentado na Figura 5.17. Para isso, são usados os perfis contendo estas regras. A associação *userFeature*, diferentemente das demais, não é empregada na construção da Configuração de Características, mas apenas quando o usuário seleciona uma característica dinâmica, modificando a Configuração de Características em tempo de execução (veja Subseção 5.4.2).

A diretriz para a atividade de construção da Configuração de Características de um produto é constituída das ações enumeradas a seguir. A Figura 5.19 apresenta um Diagrama de Atividades da UML organizando estas ações.

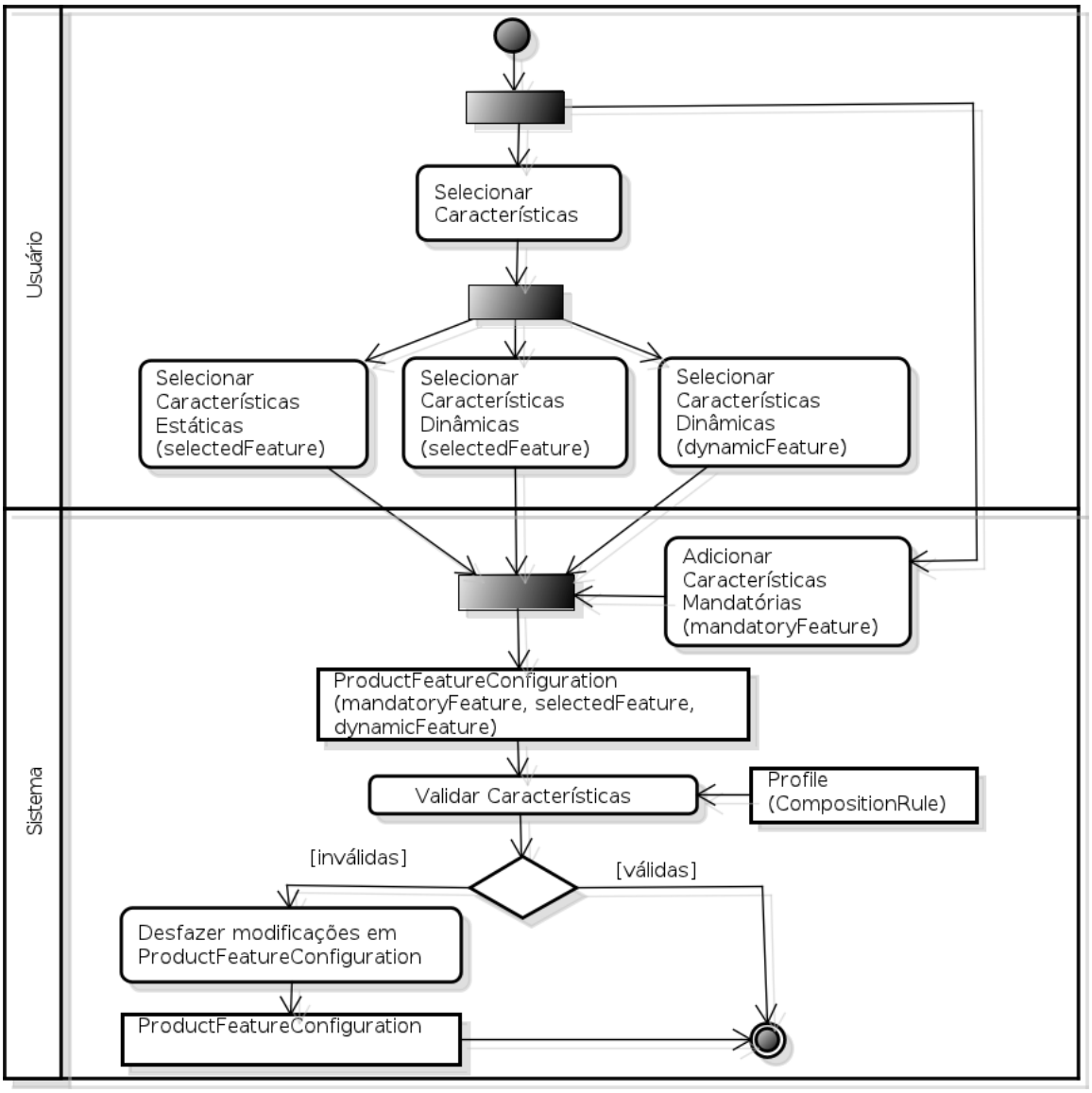


Figura 5.19: Diagrama de Atividades com ações para a geração da Configuração de Características.

1. Adicionar as características mandatórias à Configuração de Características. Para isso, todas as características mandatórias definidas no Modelo de Características (classe *Feature*, atributo *opt=mandatory*) devem ser adicionadas à associação *mandatoryFeature*.

2. Selecionar as características estáticas e adicioná-las à associação *selectedFeature*.

3. Selecionar as características dinâmicas que devem ser vinculadas já na implantação do produto e adicioná-las às associações *selectedFeature* e *dynamicFeature*.

4. Selecionar as características dinâmicas a serem vinculadas apenas durante a execução do produto e adicioná-las à associação *dynamicFeature*.

5. Validar as características de *selectedFeature* e *dynamicFeature* usando os perfis (*Profile*) contendo as regras de composição.

5.3.2.2 Arquitetura do Produto

A Arquitetura do Produto está representada no Metamodelo pela classe *ProductArchitectureConfiguration*, mostrada na Figura 5.20. Esta classe está definida como subclasse de *SoftwareArchitecture*, para que possa herdar as associações que representam as interligações e os elementos mandatórios da LPS (*mandatoryElement* e *mandatoryLink*).

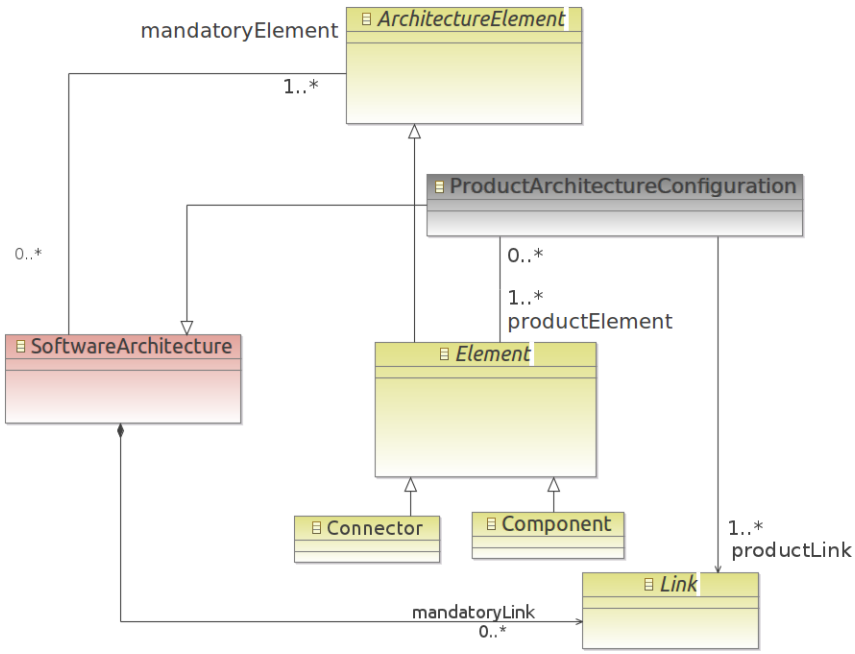


Figura 5.20: Classe *ProductArchitectureConfiguration* representa a Arquitetura do Produto.

A classe *ProductArchitectureConfiguration* contém ainda associações que correspondem às variabilidades da arquitetura: uma associação com a classe abstrata *Element* (*productElement*) e outra com a classe abstrata *Link* (*productLink*). Os elementos e interligações que compõem essas associações, somados àqueles que representam o núcleo mandatório da Arquitetura da LPS, formam o conjunto de todos os elementos e interligações configuradas na Arquitetura do Produto.

Para derivar a arquitetura de um produto, deve-se inicialmente buscar os elementos e interligações mandatórios do núcleo da Arquitetura da LPS, e associá-los a *mandatoryElement* e *mandatoryLink* da Arquitetura do Produto. Como resultado, obtém-se a sua Arquitetura Inicial, ou seja, contendo apenas elementos mandatórios.

Na sequência, a derivação continua com base na Configuração de Características, onde, para cada característica selecionada que compõe a associação *selectedFeature*, deve-se buscar o respectivo Contrato com os seus respectivos serviços de adaptação, definidos durante a modelagem dos Contratos. Os serviços de cada um dos Contratos são então executados, adaptando de forma incremental a Arquitetura Inicial do produto por meio de adições e remoções de elementos arquiteturais. As adaptações ocorrem conforme os elementos e as interligações pré-definidas pelas associações da classe *AdaptationService*: cada elemento de *selectedElement* é associado a *productElement* e cada interligação de *selectedLink* é associada a *productLink*. Ao final deste passo, obtém-se a Arquitetura Base do produto constituída dos elementos do núcleo da Arquitetura da LPS acrescidos dos elementos e interligações configurados pelos serviços de adaptação dos Contratos.

O processo de derivação continua obtendo-se, para cada característica dinâmica da associação *dynamicFeature*, os seus Contratos Dinâmicos correspondentes, definidos quando da construção do Modelo de Configuração. Estes Contratos são então implantados junto com a Arquitetura Base do produto. Assim, a Arquitetura Base do produto será adaptada em tempo de execução pelos Contratos Dinâmicos.

A diretriz para a atividade de derivação da arquitetura de um produto é constituída das ações enumeradas a seguir. A Figura 5.21 apresenta um Diagrama de Atividades da UML organizando as ações.

1. Derivar a Arquitetura Inicial do produto usando o Núcleo da Arquitetura da LPS (classe *SPLArchitecture*, associações *mandatoryElement* e *mandatoryLink*).

2. Adaptar a Arquitetura Inicial do produto, aplicando cada característica configurada em *selectedFeature* por meio dos respectivos Contratos e serviços de adaptação.

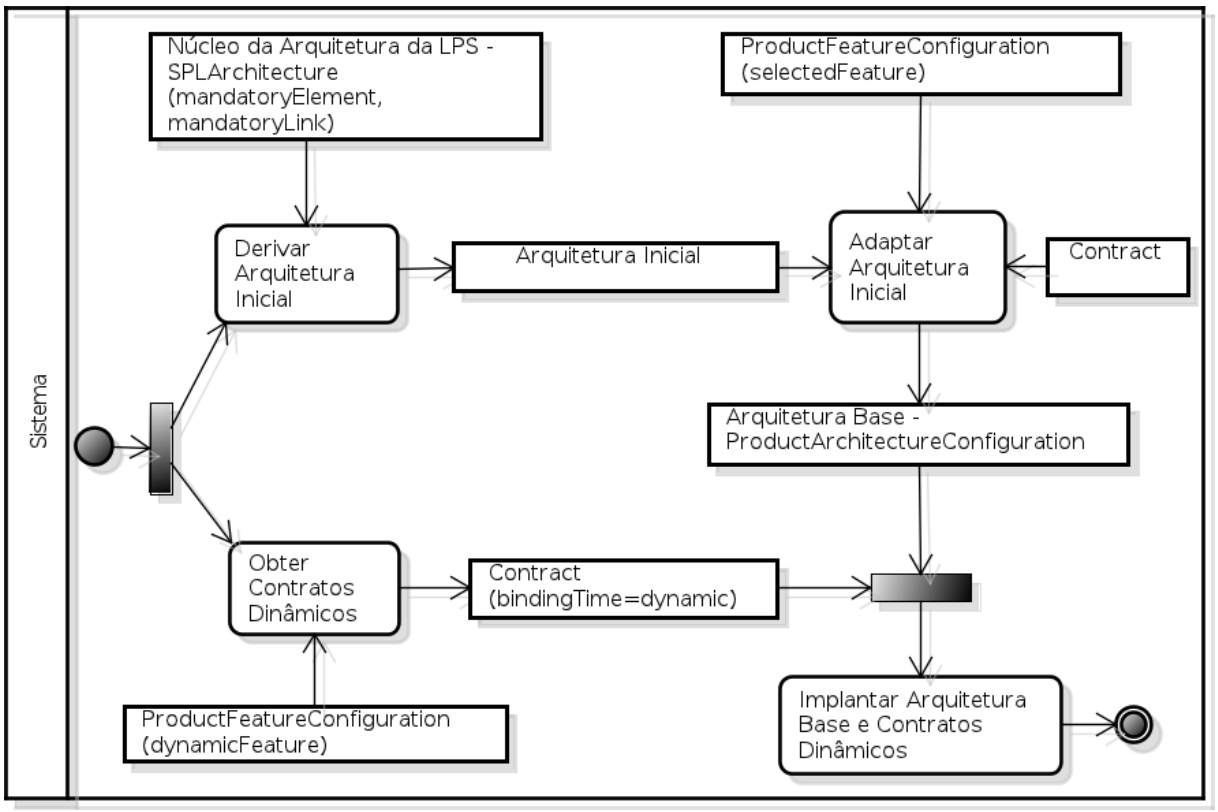


Figura 5.21: Diagrama de Atividades com ações para a geração da Arquitetura do Produto.

Após sucessivas adaptações aplicadas sobre a *Arquitetura Inicial*, o resultado será a *Arquitetura Base* do produto.

3. Obter os Contratos Dinâmicos com os Serviços de Adaptação correspondentes às características dinâmicas configuradas na associação *dynamicFeature*.

4. Implantar a Arquitetura Base do produto e os Contratos Dinâmicos obtidos.

A derivação, portanto, produz uma arquitetura dinâmica para o produto, formada pela Arquitetura Base e os Contratos Dinâmicos, os quais permitem adaptações futuras à luz de novas demandas que venham a ocorrer em tempo de execução. Vale ressaltar que se nenhuma variabilidade dinâmica for configurada para o produto (associação *selectedFeature* composta apenas de características estáticas e características dinâmicas selecionadas durante a derivação do produto, associação $dynamicFeature = \varnothing$), a derivação resulta em uma arquitetura estática para o produto (formada apenas pela Arquitetura Base) com todas as variabilidades resolvidas durante a sua derivação.

5.4 Adaptação Dinâmica

O Metamodelo em conjunto com o processo de derivação descrito na Seção 5.3.2, deve gerar Produtos Estáticos (com arquitetura estática) ou Dinâmicos (com arquitetura dinâmica), dependendo da Configuração de Características definida. Caso ela contenha somente características associadas a *selectedFeature*, uma arquitetura estática é gerada pelo Modelo de Configuração. Se, no entanto, uma ou mais características também forem associadas a *dynamicFeature*, uma arquitetura dinâmica é gerada para o produto, com suas adaptações sendo guiadas pelos Contratos Dinâmicos. Produtos Estáticos e Produtos Dinâmicos podem, portanto, ser representados em termos de suas características configuradas.

5.4.1 Produto Estático e Produto Dinâmico

Considerando um Modelo de Características composto de um conjunto M de características mandatórias, de um conjunto S de características estáticas e de um conjunto D de características dinâmicas, tal que $M, S, D \subset F$, onde F é o conjunto total de características do modelo, a Configuração de Características para um Produto Estático é assim definida:

. $$FC = M \cup SF \tag{1}$$

tal que $FC, SF \subseteq F$ e $M \neq \varnothing$, onde:

FC é o conjunto de todas as características configuradas na Arquitetura do Produto;

M é o conjunto de características mandatórias definidas no Modelo de Características;

SF é o conjunto de características selecionadas durante a derivação do produto.

O conjunto SF é assim definido:

. $$SF = sFC \cup dFC \tag{2}$$

tal que $sFC \neq \varnothing$, $sFC \subseteq S$ e $dFC \subseteq D$, onde:

sFC é o conjunto de características estáticas selecionadas durante a derivação do produto;

dFC é o conjunto de características dinâmicas selecionadas durante a derivação do produto.

A Configuração de Características para um Produto Dinâmico, por sua vez, é definida por dois conjuntos:

FC, conjunto definido em (1) e

DF, conjunto de características dinâmicas que podem ou não ser selecionadas durante a execução do produto,

tal que $DF \neq \varnothing$ e $DF \subseteq D$.

Uma vez que o conjunto DF representa variabilidades dinâmicas não resolvidas durante a derivação do produto, n configurações $(FC_1, FC_2, ..., FC_n)$ devem ser derivadas durante a sua execução:

$$FC_1 = M \cup SF_1 = M \cup (sFC \cup dFC_1);$$
$$FC_2 = M \cup SF_2 = M \cup (sFC \cup dFC_2;$$
$$...$$
$$FC_n = M \cup SF_n = M \cup (sFC \cup dFC_n);$$

tal que $dDF_n \subseteq DF, n \geq 1$, onde:

dFC_1, $dFC_2, ..., dFC_n$, são conjuntos de características dinâmicas selecionadas durante a execução do produto, seja pelo usuário ao modificar a Configuração de Características do produto, seja por mudanças do contexto. As características que compõem estes conjuntos são obtidas do conjunto DF.

As n configurações $FC_1, FC_2, ..., FC_n$ podem ser vistas como diferentes versões da arquitetura do Produto Dinâmico, resultantes do processo de adaptação arquitetural. Cada versão n tem o seu próprio conjunto dFC_n, enquanto que os conjuntos M, sFC e dFC não variam de configuração para configuração.

As próximas subseções apresentam as diretrizes do Metamodelo com atividades voltadas para a adaptação dinâmica de Arquiteturas de Produtos (Produtos Dinâmicos), considerando duas situações:

- Em resposta a mudanças feitas pelo usuário na Configuração de Características do produto;
- Em resposta a mudanças do contexto de execução do produto.

Na segunda situação, a Configuração de Características do produto também deve ser modificada com o objetivo de refletir a adaptação arquitetural realizada.

5.4.2 Mudanças na Configuração de Características

As características selecionadas e configuradas na arquitetura de um produto são identificadas por meio da associação *selectedFeature* da classe *ProductFeatureConfiguration*, definida durante a derivação do produto como descrito na Seção 5.3.2. Para modificar a Configuração de Características do produto em tempo de execução, o usuário deve selecionar uma determinada característica dinâmica da associação *dynamicFeature*, também definida durante a derivação do produto. Essa seleção faz com que a característica seja inserida na associação *selectedFeature* modificando assim a Configuração de Características do produto. Além disso, essa mesma característica é inserida na associação *userFeature* para fazer com que futuras mudanças de contexto não modifiquem esta característica (veja Seção 5.4.3).

Quando a Configuração de Características é modificada pelo usuário, selecionando uma determinada característica dinâmica de *dynamicFeature*, o Contrato Dinâmico associado a esta característica é iniciado, fazendo com que o serviço correspondente entre em execução e realize as ações de adaptação envolvendo os elementos da arquitetura (*selectedElement*) e as interligações (*selectedLink*). Uma vez em execução, o serviço mantém a configuração estabelecida até que o usuário remova a seleção da característica correspondente.

As ações envolvidas, desde a mudança da Configuração de Características até a adaptação da Arquitetura do Produto, segue uma sequência, seja para selecionar, seja para remover uma característica dinâmica. Para selecionar uma característica dinâmica, a sequência a seguir é realizada. A Figura 5.22 apresenta um Diagrama de Atividades da UML organizando estas ações.

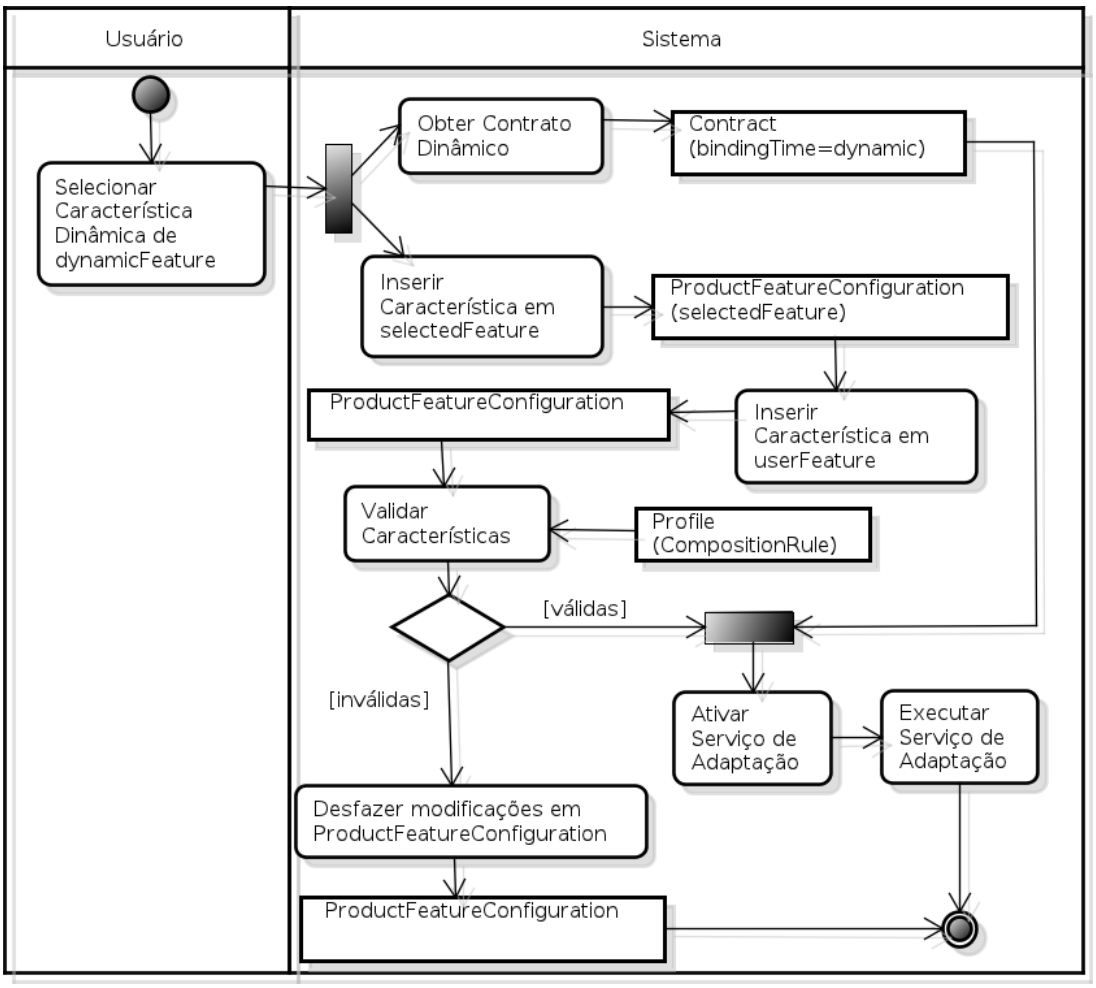


Figura 5.22: Diagrama de Atividades com ações para a adaptação arquitetural realizada quando uma característica dinâmica é selecionada.

1. Selecionar uma das características dinâmicas da associação *dynamicFeature* (*ação do usuário*).

2. Inserir a característica dinâmica selecionada na associação *selectedFeature*.

3. Inserir a característica dinâmica selecionada na associação *userFeature*.

4. Validar as características de *selectedFeature* e *dynamicFeature* usando os perfis (*Profile*) contendo as regras de composição.

5. Obter o Contrato Dinâmico e o Serviço de Adaptação correspondente à característica dinâmica selecionada.

6. Ativar o Serviço de Adaptação independentemente do perfil associado (se houver) estar válido.

7. Executar o Serviço de Adaptação conforme a cláusula de negociação definida para o Contrato.

Em outra situação, caso se queira remover, por exemplo, um sensor da configuração do produto, a característica dinâmica correspondente deve ser removida da associação *selectedFeature*. Também neste caso, a característica é inserida em *userFeature*. Para remover uma característica dinâmica, a seguinte sequência de ações é realizada. A Figura 5.23 apresenta um Diagrama de Atividades da UML organizando as ações.

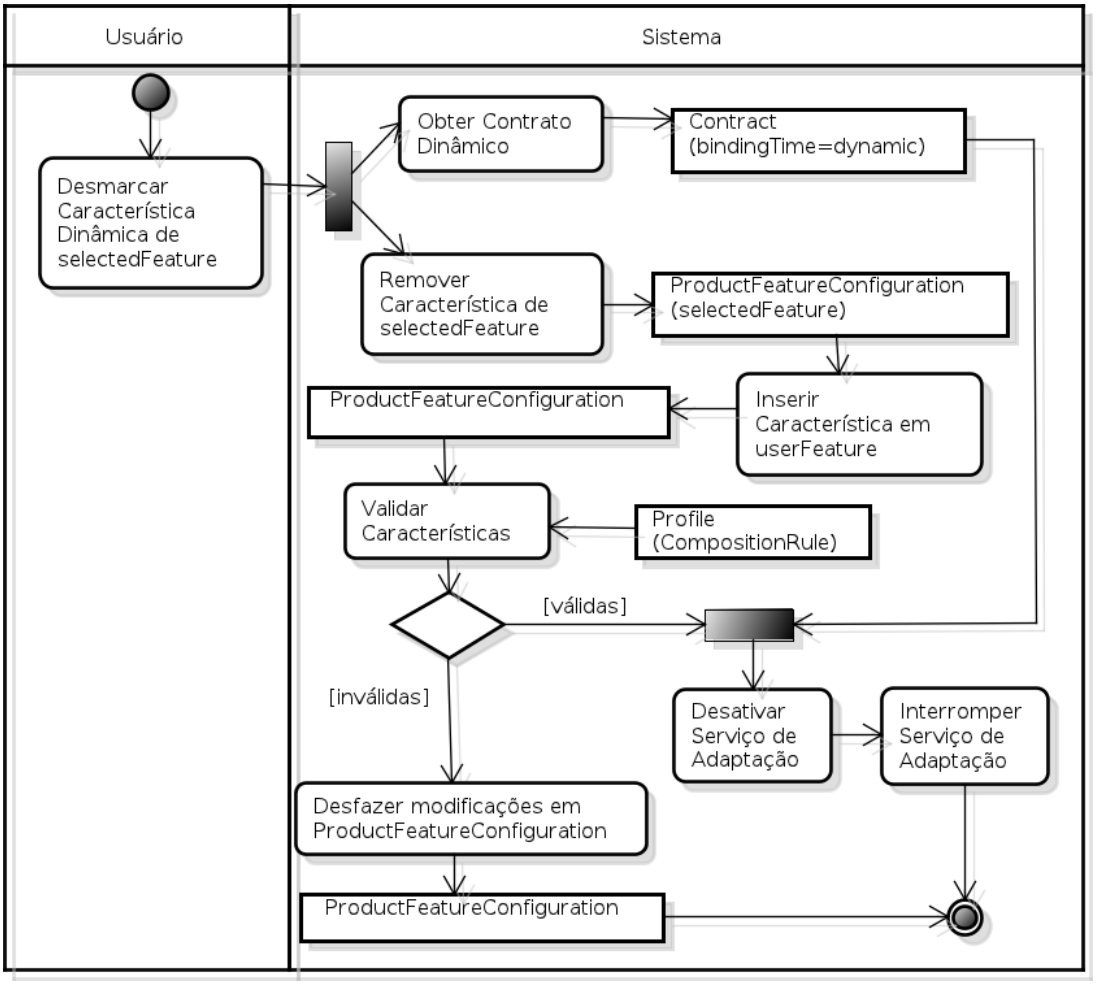


Figura 5.23: Diagrama de Atividades com ações para a adaptação arquitetural realizada quando uma característica dinâmica é removida.

1. Desmarcar uma das características dinâmicas da associação *selectedFeature* (*ação do usuário*).

2. Remover a característica dinâmica selecionada da associação *selectedFeature*.

3. Inserir a característica dinâmica na associação *userFeature*.

4. Validar as características *selectedFeature* e *dynamicFeature* usando os perfis (*Profile*) contendo as regras de composição.

5. Obter o Contrato Dinâmico e o Serviço de Adaptação correspondente à característica dinâmica removida.

6. Desativar o Serviço de Adaptação independentemente do perfil associado (se houver) estar válido.

7. Interromper a execução do serviço de adaptação.

5.4.3 Mudanças do Contexto

As entidades e os elementos do contexto são continuamente coletados durante a execução do produto. Essas informações são usadas junto às regras de contexto requeridas para validar os perfis.

Quando um determinado perfil se torna válido, os serviços de adaptação que possuem esse perfil associado são ativados e então executados seguindo o mecanismo de negociação definido em *Negotiation*. Com isso, a Arquitetura do Produto é adaptada. Por outro lado, um perfil que se torna inválido faz com que os serviços ativos que dependem desse perfil sejam interrompidos. A interrupção desfaz as configurações arquiteturais estabelecidas pelos serviços, gerando também adaptações arquiteturais.

Adaptações na Arquitetura do Produto estimuladas por mudanças de contexto provocam ainda a modificação da sua Configuração de Características. Como a adaptação da arquitetura é feita por um serviço, para se modificar a Configuração de Características, deve-se primeiramente alcançar a característica correspondente àquele serviço e, por conseguinte, alcançar a Configuração de Características. As associações *optionalFeature* e *alternativeFeature* auxiliam na identificação da respectiva característica, como descrito na Subseção 5.2.3.2.

Caso o serviço em questão tenha sido ativado por algum perfil, a característica dinâmica correspondente deve ser adicionada à associação *selectedFeature* de *ProductFeatureConfiguration*. Por outro lado, caso o serviço tenha sido desativado por algum perfil e consequentemente interrompido, a respectiva característica dinâmica deve ser removida da associação *selectedFeature*. Esta operação somente ocorre, no entanto, para características que não tenham sido modificadas diretamente pelo usuário, ou seja, que não façam parte da associação *userFeature*. Como apresentado na Seção 5.4.2, características dinâmicas

selecionadas ou removidas pelo usuário são inseridas nesta associação.

É importante destacar que a implementação das ações relacionadas à adaptação arquitetural (*e.g.*, ativar e desativar serviços de adaptação), incluindo as apresentadas na subseção anterior, pode depender de questões específicas da Arquitetura do Produto e, neste caso, tais especificidades precisam ser explicitamente programadas quando da criação do produto. Estas ações representam um guia no sentido de identificar o caminho que deve ser seguido, no contexto do Metamodelo, para realizar a adaptação da arquitetura.

Em resumo, a adaptação da arquitetura em decorrência da mudança do contexto segue as ações a seguir. A Figura 5.24 apresenta um Diagrama de Ativadades da UML organizando as ações.

1. Avaliar os perfis diante da mudança de contexto.
2. Obter os serviços de adaptação cujos perfis associados tenham mudado quanto ao atributo *isValidated*.
3. Selecionar as características dinâmicas da associação *dynamicFeature* correspondentes aos serviços de adaptação obtidos.
4. Verificar se as características dinâmicas estão na associação *userFeature*.
5. Se (*Profile*, *isValidated=true*), inserir as características dinâmicas selecionadas na associação *selectedFeature*.
6. Se (*Profile*, *isValidated=false*), remover as características dinâmicas selecionadas da associação *unselectedFeature*.
7. Validar as características de *selectedFeature* usando os perfis (*Profile*) contendo as regras de composição.
8. Se (*Profile*, *isValidated=true*), ativar e executar os serviços de adaptação obtidos.
9. Se (*Profile*, *isValidated=false*) desativar e interromper os serviços de adaptação obtidos.

5.5 Considerações Finais

O Metamodelo de Configuração de Variabilidades em LPS foi detalhado neste capítulo com destaque para os três metamodelos que o integram: Características, Arquitetura e

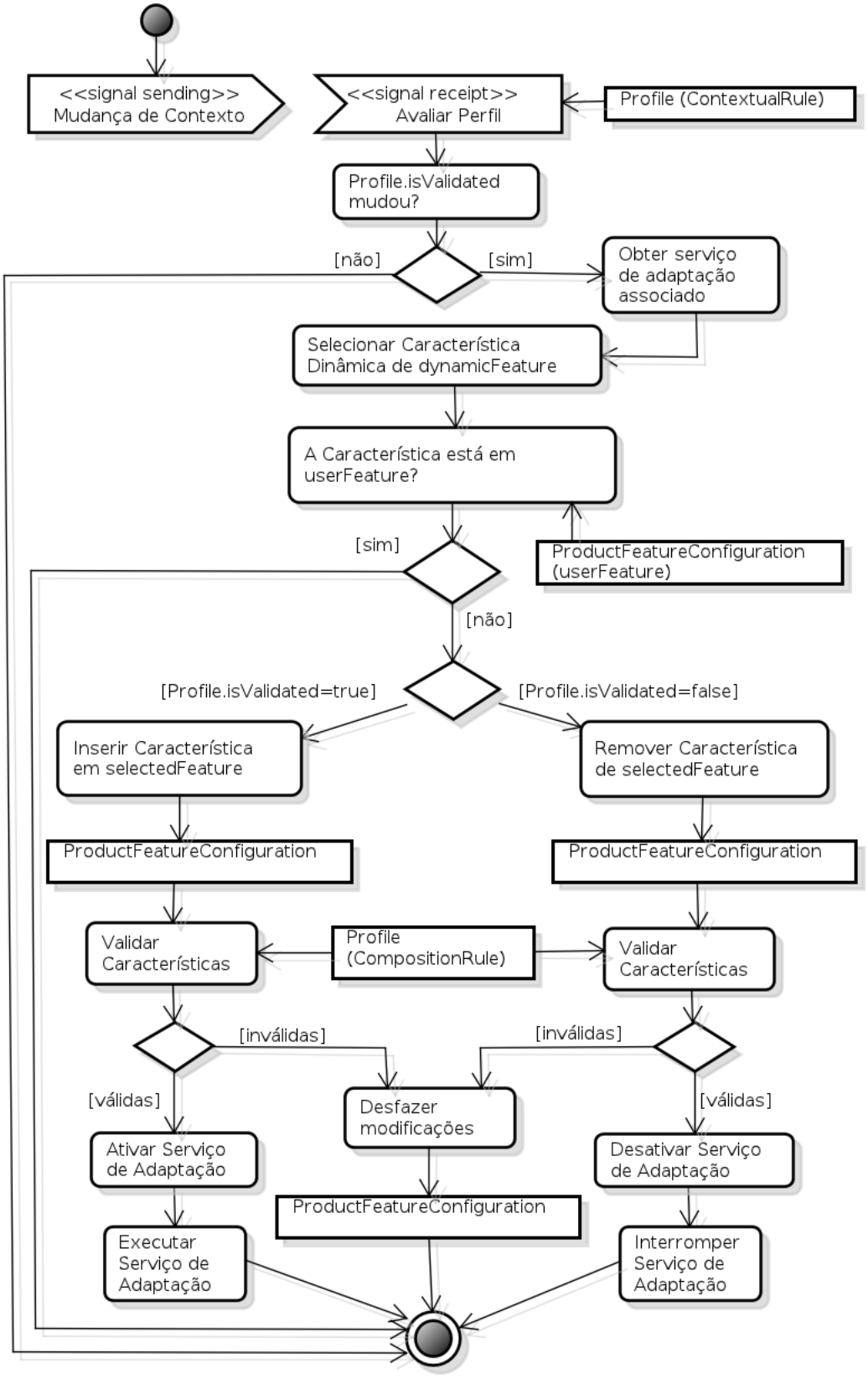


Figura 5.24: Diagrama de Atividades com ações para a adaptação arquitetural realizada em decorrência da mudança do contexto.

Configuração. Estes metamodelos permitem que os respectivos modelos da LPS sejam criados, e que as Arquiteturas dos Produtos sejam derivadas e adaptadas dinamicamente.

Na abordagem proposta, a derivação das Arquiteturas dos Produtos é realizada, principalmente, por meio da seleção de características, tendo, portanto, o Modelo de Características agindo como um guia no processo de derivação. Essa propriedade caracteriza a proposta como orientada a características (veja Capítulo 2). No entanto, como as variabilidades estáticas e dinâmicas são modeladas no nível da Arquitetura da LPS na forma de Contratos, o desenvolvimento e a configuração de Arquiteturas de Produtos são realizados de forma centrada na arquitetura, prescindindo-se do Modelo de Características. Neste caso, a seleção de elementos da arquitetura é feita por meio de regras definidas em perfis (*Profile*). É essencial ressaltar que estas regras não provocam os problemas que as expressões de restrição causam nas outras abordagens, como descrito no Capítulo 2, pois um perfil não representa um ponto de variação e não é associado a componentes individualmente, mas sim a serviços que representam configurações arquiteturais. Além disso, Contratos são associados a pontos de variação ou opcionalidades, o que traz independência entre a descrição da variabilidade, realizada no contexto dos serviços de adaptação dos Contratos, e as regras de composição e/ou regras de contexto.

Em termos de desenvolvimento, o Metamodelo proposto foi construído usando o EMF e o meta-metamodelo Ecore. O EMF oferece editores para se descrever metamodelos e validá-los quanto à sua conformidade em relação ao Ecore. As ferramentas do EMF oferecem ainda formas de criar modelos a partir do metamodelo em desenvolvimento e fazer validações. Um Modelo de Características, por exemplo, pode ser criado e validado diante das especificações do Metamodelo proposto neste capítulo. No desenvolvimento desta tese, além do uso de editores e outras ferramentas, também foram codificados programas em Java para manipular diretamente as classes e os demais elementos do Metamodelo. O Apêndice B mostra o arquivo XMI Ecore do Metamodelo, a partir do qual código-fonte em Java das classes do Metamodelo e instâncias dos modelos são gerados.

O próximo capítulo apresenta a avaliação e cenários de uso do Metamodelo proposto.

Capítulo 6

Avaliação e Cenários de Uso

Este capítulo apresenta a avaliação do Metamodelo de Configuração de Variabilidades em LPS, envolvendo cenários com características do SCIADS e tendo como base os objetivos do Metamodelo: construção dos modelos da LPS, derivação e implantação de produtos, correspondência característica-arquitetura e adaptação dinâmica da Arquitetura do Produto.

6.1 Introdução

A avaliação do Metamodelo apresentado no Capítulo 5 consiste na formulação de questões de competência [62, 127] diretamente relacionadas aos objetivos do Metamodelo.

A técnica de formulação de questões de competência tem sido bastante utilizada no contexto do desenvolvimento de ontologias, sendo primeiramente sugerida por Grüninger e Fox [62], e posteriormente detalhada por Uschold *et al.* [127]. Ela consiste na criação de perguntas que auxiliem na especificação dos requisitos da ontologia a ser desenvolvida. Uma vez implementada a ontologia, as questões de competência são então formalizadas de tal maneira que possam ser executadas frente a um conjunto de dados de teste instanciados a partir da ontologia. O objetivo é verificar a sua conformidade em relação às especificações. Além das questões em si, alguns exemplos de aplicações (cenários) podem também ser fornecidos, visando auxiliar na caracterização das funcionalidades da ontologia.

No contexto desta tese, foram formuladas questões de competência onde Diagramas de Objetos da UML são utilizados para formalizar a execução dos modelos criados a partir do Metamodelo. O uso de Diagramas de Objetos traz a possibilidade de representar

cada modelo (*e.g.*, Modelo de Características) por meio da identificação das classes e das associações necessárias à sua realização. A notação de objetos é interessante nesse contexto, uma vez que o Metamodelo é um modelo EMF, essencialmente um subconjunto do Diagrama de Classes da UML [120].

As questões compreendem ainda cenários de uso do Metamodelo, no caso envolvendo características do SCIADS. As respostas às questões, tanto na forma de explicações textuais quanto na forma de Diagramas de Objetos, são apresentadas no contexto dos cenários, provendo assim formas concretas de verificar como o Metamodelo pode ser usado para tratar os problemas estabelecidos pelas questões.

As questões formuladas estão diretamente focadas nos problemas que se deseja solucionar, visando avaliar a competência do Metamodelo frente aos seus objetivos. A competência aqui destacada diz respeito à cobertura de questões que o Metamodelo pode responder ou de tarefas que ele pode apoiar.

Este capítulo está organizado como se segue.

Seção 6.2 Questões relacionadas à Construção dos Modelos da LPS.

Seção 6.3 Questões relacionadas à Derivação e Implantação de Produtos.

Seção 6.4 Questões relacionadas à Correspondência Característica-Arquitetura.

Seção 6.5 Questões relacionadas à Adaptação Dinâmica da Arquitetura do Produto.

Seção 6.6 Resumo dos resultados.

Seção 6.7 Considerações finais.

6.2 Construção da LPS

O Metamodelo deve permitir a criação dos Modelos de Características e de Arquitetura da LPS, assim como de um Modelo de Configuração com condições de representar variabilidades estáticas e dinâmicas no nível da Arquitetura da LPS. As questões relacionadas à construção da LPS são apresentadas a seguir.

6.2.1 *Como construir um Modelo de Características?*

A Figura 6.1 apresenta o Diagrama de Objetos que representa parte do Modelo de Características do SCIADS, apresentado no Capítulo 3, Figura 3.4.

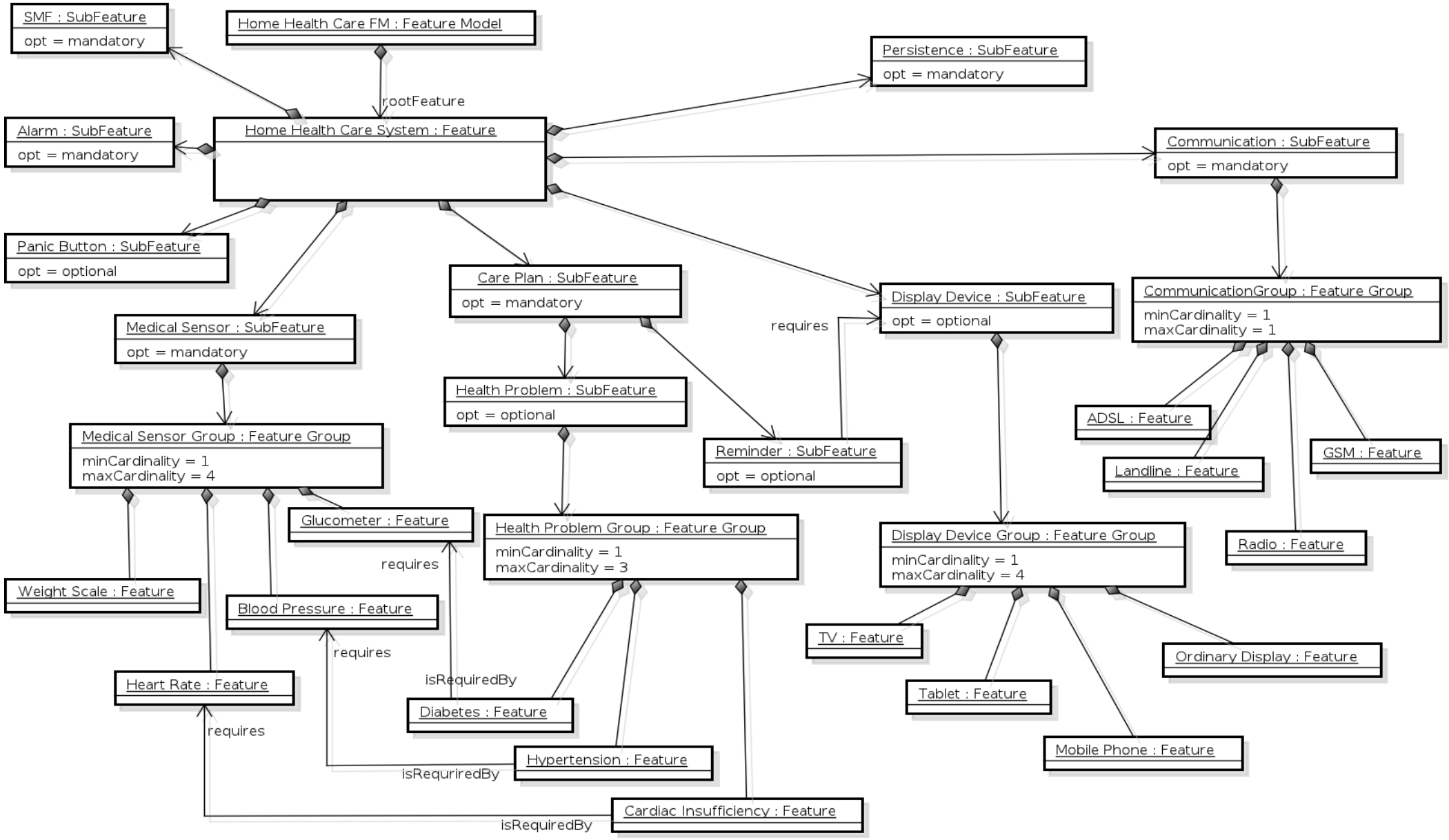


Figura 6.1: Parte do Modelo de Características do SCIADS criada a partir do Metamodelo.

Os detalhes sobre como representar as características, as relações OR e XOR, as restrições de dependência e as variabilidades, são apresentados nas próximas subseções.

6.2.1.1 Características Mandatórias, Opcionais e Alternativas

O Diagrama de Objetos da Figura 6.2 destaca três características definidas no modelo da Figura 6.1. A primeira e a segunda características, *Alarm* e *PanicButton* respectivamente, são folhas da árvore, sendo que *Alarm* é mandatória (*opt=mandatory*) e *PanicButton* é opcional (*opt=optional*).

A terceira característica, *MedicalSensor*, também é mandatória (*opt=mandatory*), no entanto, é composta por um grupo, *MedicalSensorGroup*, formado pelas características alternativas *HeartRate*, *BloodPressure* e *Glucometer*. *Alarm* e *PanicButton* são denominadas de características solitárias, por não serem compostas de outras características.

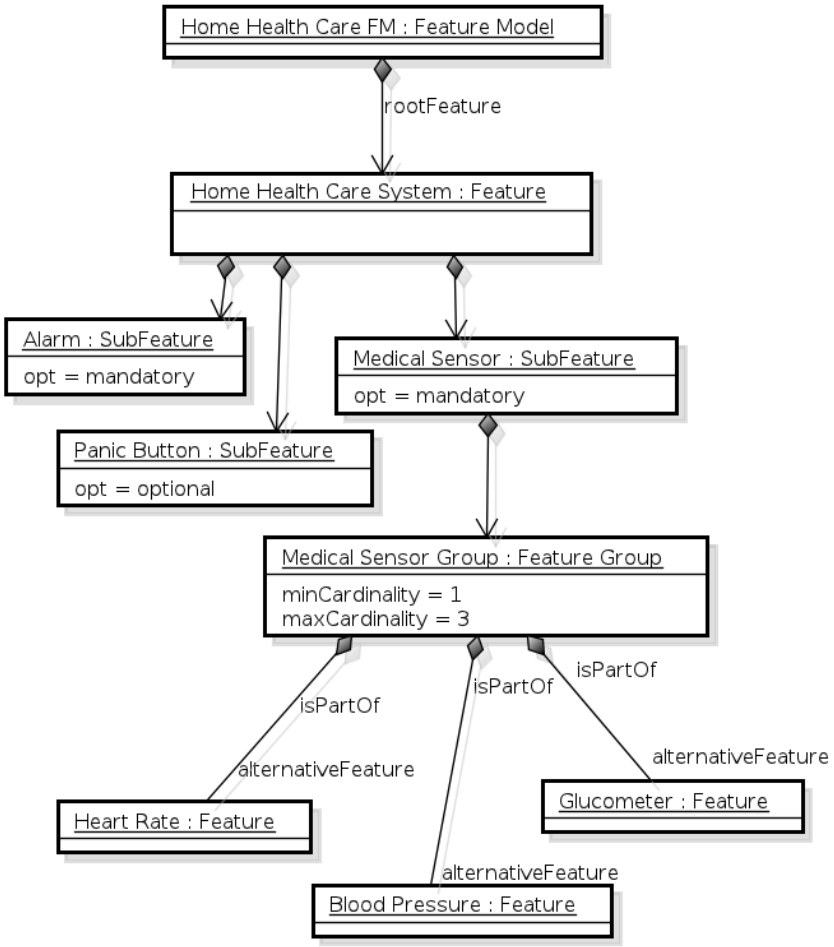


Figura 6.2: Representação de características mandatórias, opcionais e alternativas.

6.2.1.2 Relações OR e XOR em Grupos de Características

O Diagrama de Objetos da Figura 6.3 destaca dois grupos, *MedicalSensorGroup* e *CommunicationGroup*. *MedicalSensorGroup* possui a cardinalidade <1-3> (*minCardinality=1*; *maxCardinality=3*), identificando uma relação OR, onde no mínimo um e no máximo três características alternativas devem ser selecionadas durante a derivação. *CommunicationGroup*, por sua vez, representa uma relação XOR, por causa da cardinalidade <1-1> (*minCardinaliy=1*; *maxCardinality=1*). Neste caso, uma e somente uma tecnologia de comunicação pode ser selecionada. Com o objetivo de simplificar o diagrama, foram suprimidos os papéis nas associações entre as características alternativas e os grupos.

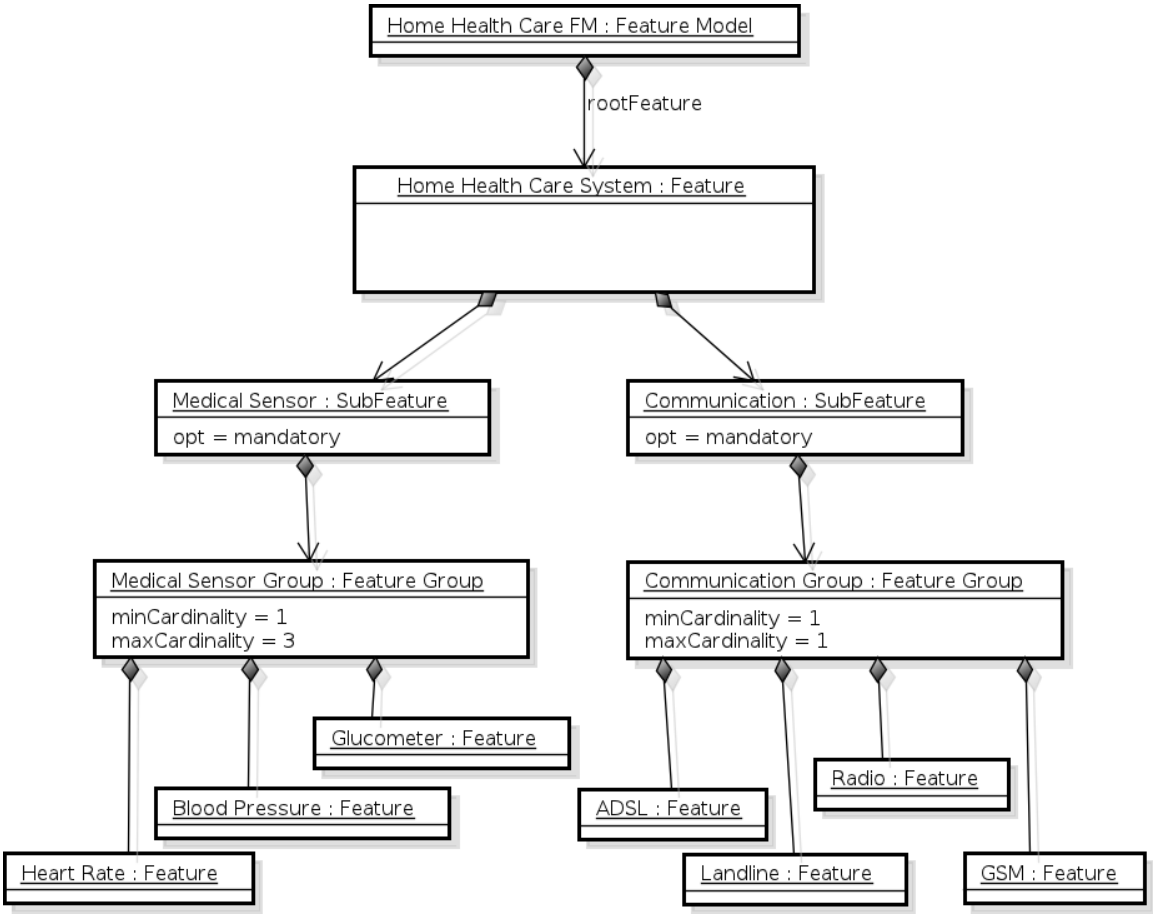


Figura 6.3: Representação de relações OR e XOR em grupos de características.

6.2.1.3 Restrições de Dependência

O Diagrama de Objetos da Figura 6.4 destaca as restrições de dependência entre características. No diagrama, a associação *requires* faz com que a seleção de *Diabetes*, uma característica alternativa representando um problema de saúde do paciente, requeira que

Glucometer também seja selecionada. Outras duas, *Hypertension* e *CardiacInsufficiency*, também possuem restrições de dependência em relação às características alternativas que compõem o grupo *MedicalSensorGroup*.

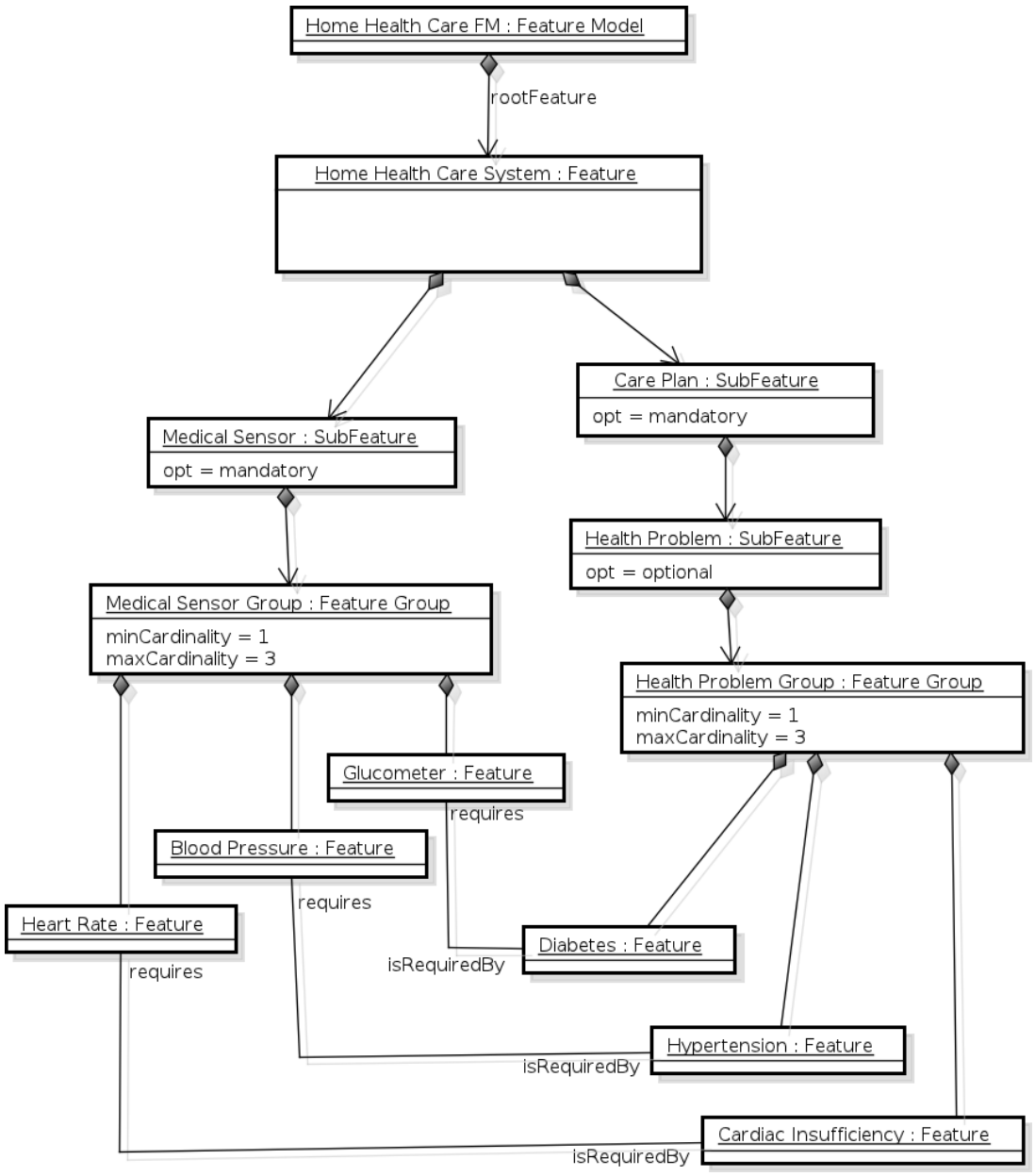


Figura 6.4: Modelo de Características com restrições de dependência do tipo *requires*.

6.2.1.4 Representação das Variabilidades

As variabilidades são representadas pelas características opcionais e alternativas. No Diagrama de Objetos da Figura 6.5 a característica opcional *PanicButton* e as alternativas *ADSL*, *Landline*, *Radio* e *GSM*, representam variabilidades, uma vez que a primeira é opcional, e as outras compõem um grupo cuja cardinalidade permite a seleção de uma

das características. Contratos estão associados (*PanicButtonContract* e *CommunicationGroupContract*) para que as variabilidades sejam representadas no nível da Arquitetura da LPS (veja a Seção 6.2.3). As demais características do diagrama são mandatórias (*Alarm* e *Communication*), não representando, portanto, variabilidades. Vale observar que, apesar de *Communication* ser mandatória, a variabilidade aparece entre as características alternativas organizadas no grupo *CommunicationGroup*.

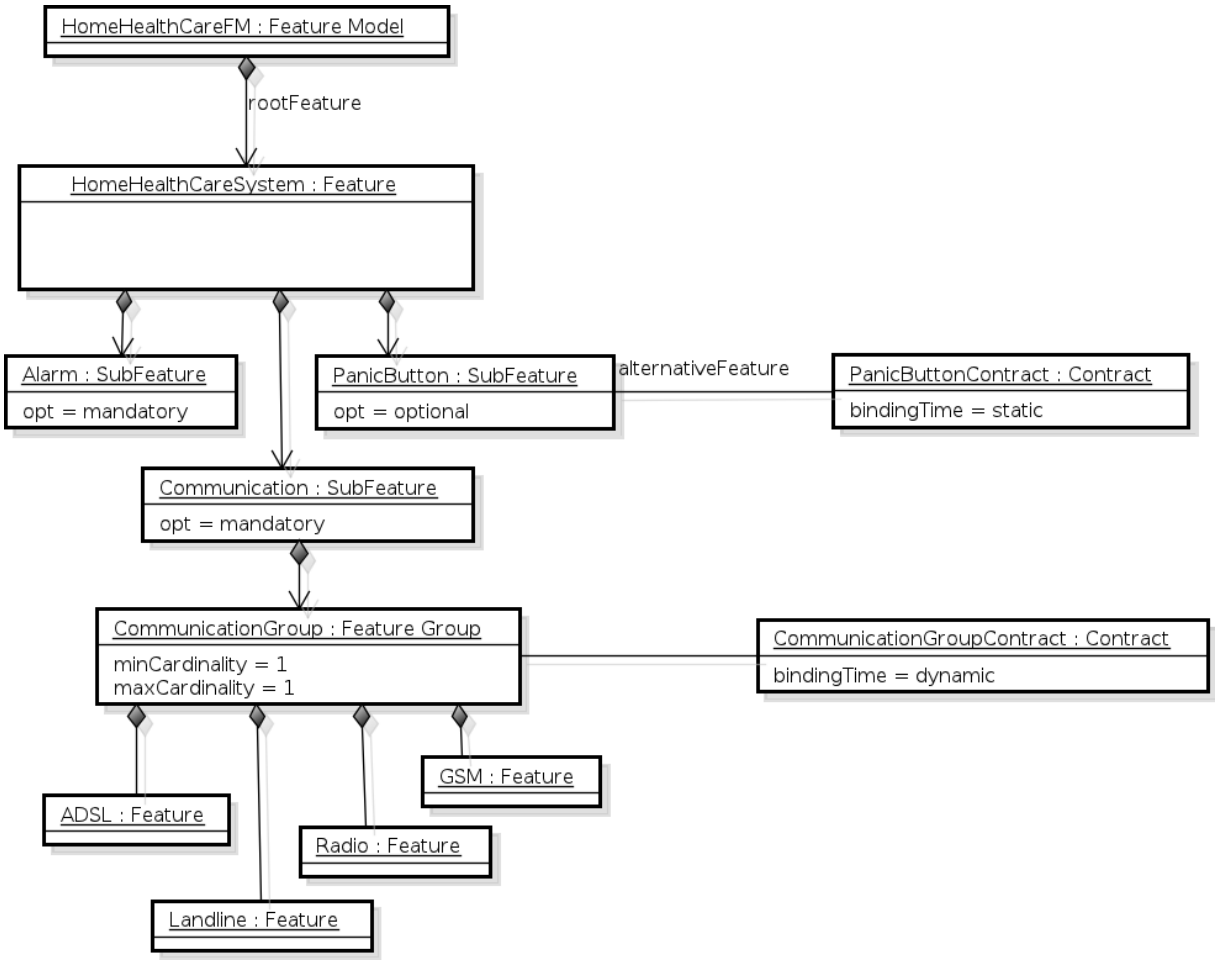


Figura 6.5: Contratos associados a características.

As variabilidades também são representadas conforme o tempo de vinculação das características. Ainda no mesmo diagrama da Figura 6.5, *PanicButton* representa uma variabilidade estática, pois está associada ao Contrato Estático *PanicButtonContract* (*bindingTime=static*). Por outro lado, *ADSL*, *Landline*, *Radio* e *GSM*, representam variabilidades dinâmicas, pois o grupo *CommunicationGroup* está associado ao Contrato Dinâmico *CommunicationGroupContract* (*bindingTime=dynamic*). Por conta disso, *PanicButton* pode ser classificada como uma característica estática e as demais como características dinâmicas. Mais detalhes sobre como associar Contratos a características estão na Seção 6.2.3.

6.2.2 *Como construir um Modelo de Arquitetura?*

Um Modelo de Arquitetura da LPS pode ser criado instanciando-se o Metamodelo de Arquitetura. Os detalhes sobre a representação dos elementos arquiteturais, incluindo a forma de interligá-los e a criação do núcleo da Arquitetura da LPS, e a representacação das variabilidades são apresentados nas próximas subseções.

6.2.2.1 Instâncias e Tipos dos Elementos Arquiteturais

Para se criar um Modelo de Arquitetura deve-se primeiramente criar os Tipos para os componentes, conectores e portas, e depois instanciá-los, gerando os elementos que efetivamente vão ser interligados para estruturar a Arquitetura da LPS. A Figura 6.6 mostra um Diagrama de Objetos representando dois componentes: *CarePlan* (instância) e o seu Tipo *CarePlanType*, *Alarm* (instância) e o seu Tipo *AlarmType*. Estão também representadas a porta de saída *send* (instância) e o seu Tipo *SendType*, e a porta de entrada *receive* (instância) e o seu Tipo *ReceiveType*. Os Tipos dos elementos da arquitetura são suprimidos em muitos dos Diagramas de Objetos apresentados neste capítulo, com o objetivo de torná-los mais simples.

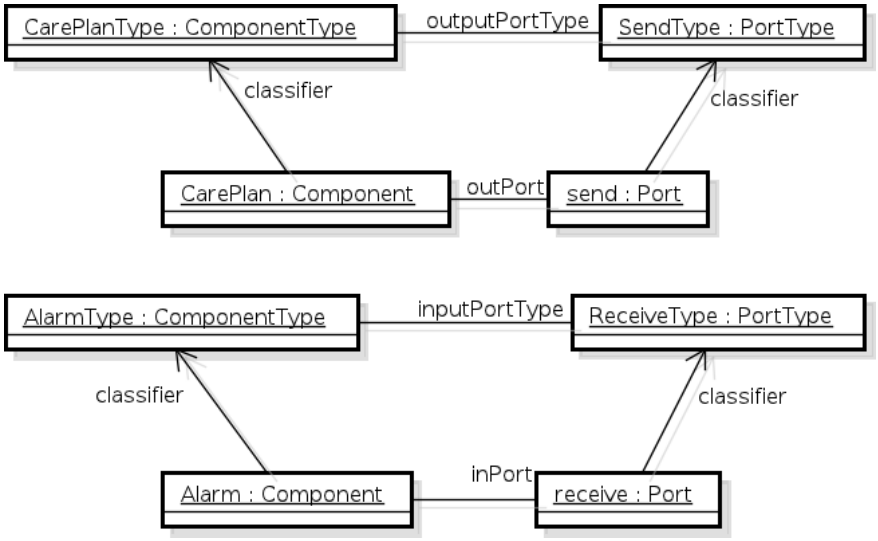


Figura 6.6: Representação de instâncias e Tipos.

6.2.2.2 Interligação de Componentes e Conectores

O Diagrama de Objetos da Figura 6.7 mostra como interligar a porta de saída do componente *CarePlan* à porta de entrada do componente *Alarm*. Ambos os componentes são instâncias dos classificadores *CarePlanType* e *AlarmType*, respectivamente. As portas

dos componentes também possuem classificadores, *SendType* e *ReceiveType*. *CarePlan* e *Alarm* são interligados pelo objeto *Link*.

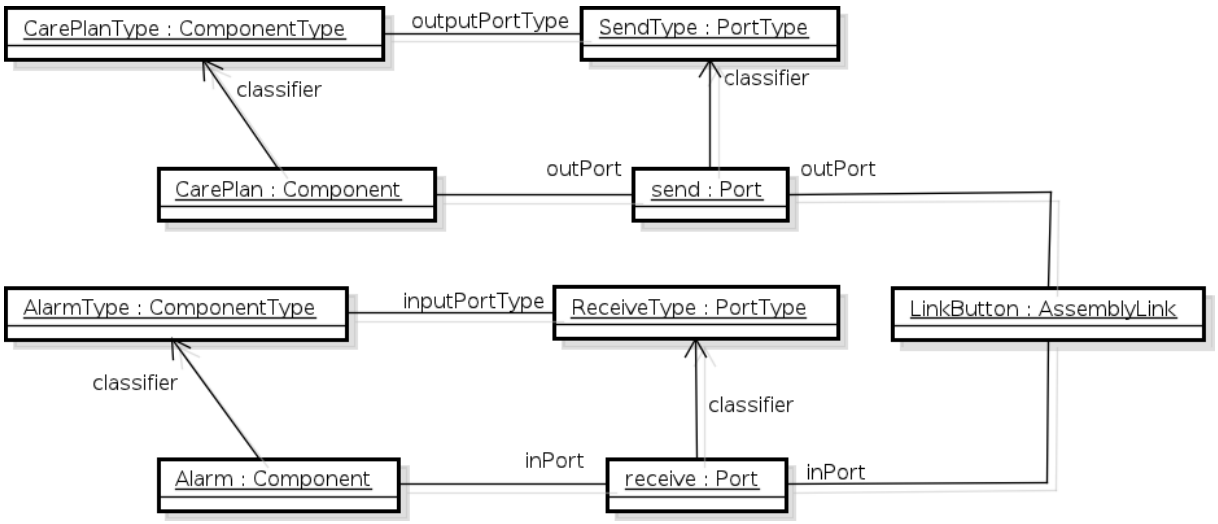


Figura 6.7: Interligação entre a porta de saída de *CarePlan* e a porta de entrada de *Alarm*.

6.2.2.3 Arquitetura Interna para Tipo de Componente

A Figura 6.8 apresenta o Diagrama de Objetos com a arquitetura interna modelada para o *ComponentType SMFType*. No modelo, *SMFInputPortType* é o Tipo da porta de entrada e *SMFOutputPortType* é o Tipo da porta de saída de *SMFType*; *inData* é a porta de entrada e *outData* é a porta de saída do componente *SMF*, instância de *SMFType*; *DataProcessing* com a sua porta de entrada *ReceiveDataInput* e *DataDistribution* com a sua porta de saída *DistribOutput* são componentes da arquitetura interna de *SMFType*. A linha pontilhada entre os elementos internos indica as interligações entre eles, suprimidas para tornar mais simples a representação. A arquitetura interna contém ainda as interligações *InputLink* e *OutputLink*, ambas do tipo *DelegationLink*, as quais ligam a porta de entrada e a porta de saída do componente SMF às respectivas portas dos elementos internos.

6.2.2.4 Núcleo da Arquitetura de uma LPS

O núcleo da arquitetura de uma LPS pode ser modelado a partir da classe *SoftwareArchitecture* e suas associações *mandatoryElement* e *mandatoryLink*. O Diagrama de Objetos da Figura 6.9 mostra o núcleo da Arquitetura da LPS (classe *SPLArchitecture*) referente ao Modelo de Características apresentado na Figura 6.1.

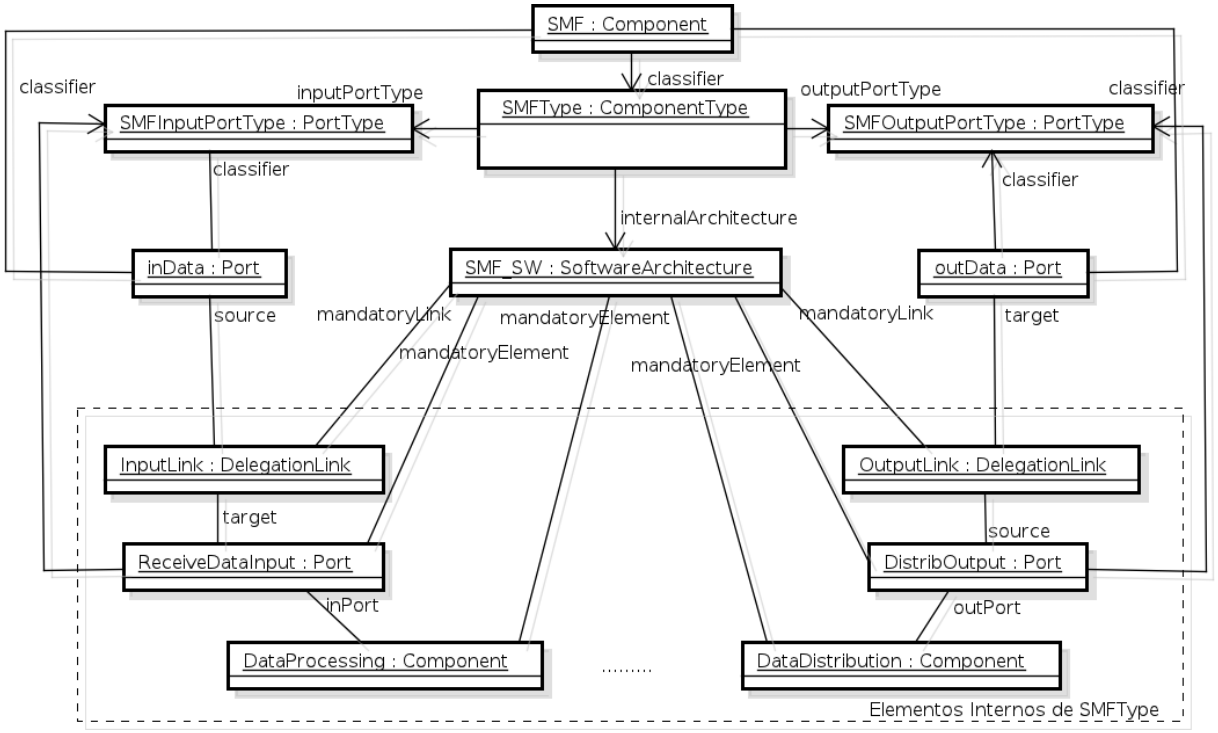


Figura 6.8: Modelagem da arquitetura interna do *ComponentType SMFType*.

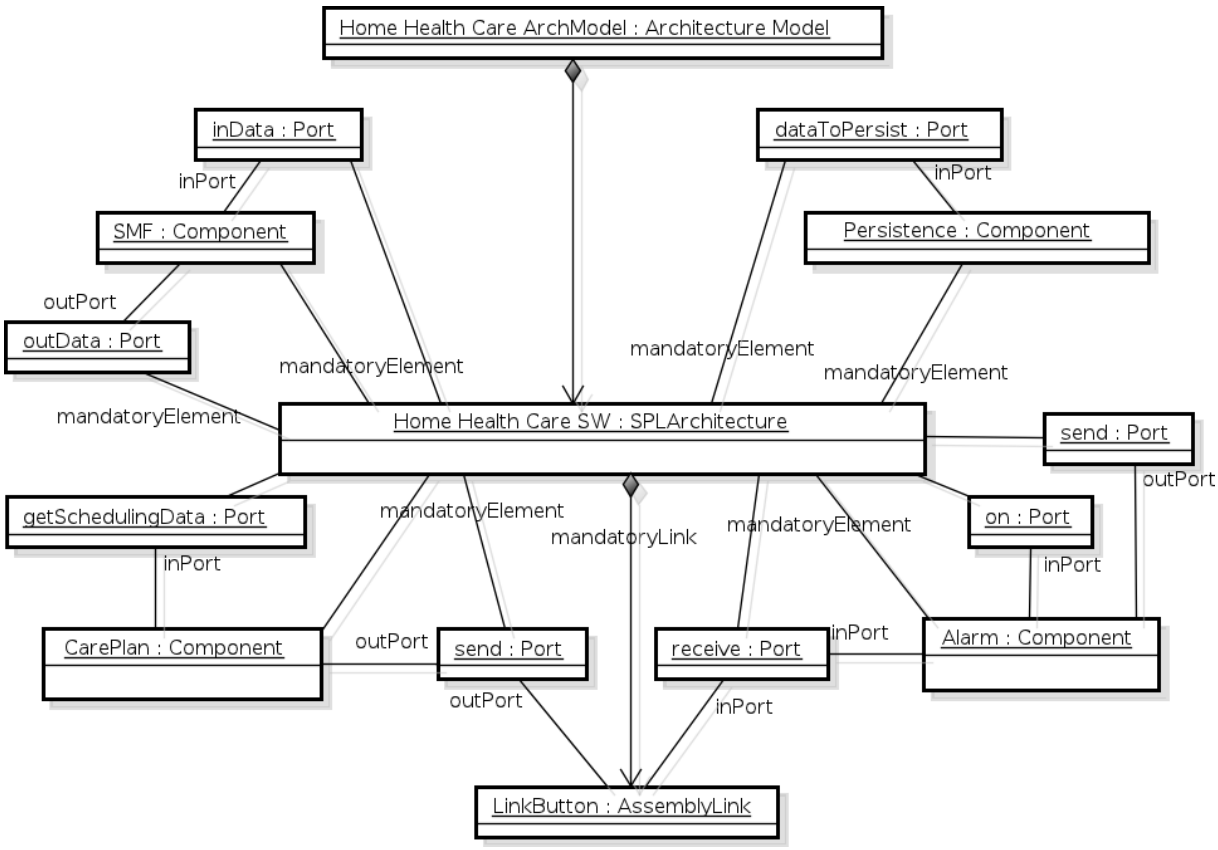


Figura 6.9: Núcleo da Arquitetura da LPS: elementos mandatórios.

6.2.2.5 Representação das Variabilidades

O Diagrama de Objetos da Figura 6.10 apresenta como elementos arquiteturais são organizados para representar opcionalidades e pontos de variação. *PanicButtonOptional* é uma opcionalidade formada pelo componente opcional *PanicButton*, e *CommunicationVP* é um ponto de variação formado pelos elementos variantes *ADSL*, *Landline*, *Radio* e *GSM*. Ambos os tipos de variabilidades estão associados a Contratos, *PanicButtonContract* e *CommunicationGroupContract*, os mesmos Contratos também associados, no diagrama da Figura 6.5, à característica *PanicButton* e ao grupo de características alternativas *CommunicationGroup*.

A Subseção 6.2.3 mostra como as variabilidades são representadas por adaptações arquiteturais, empregando estes Contratos. Para tornar o diagrama mais simples, as ligações entre os componentes foram suprimidas.

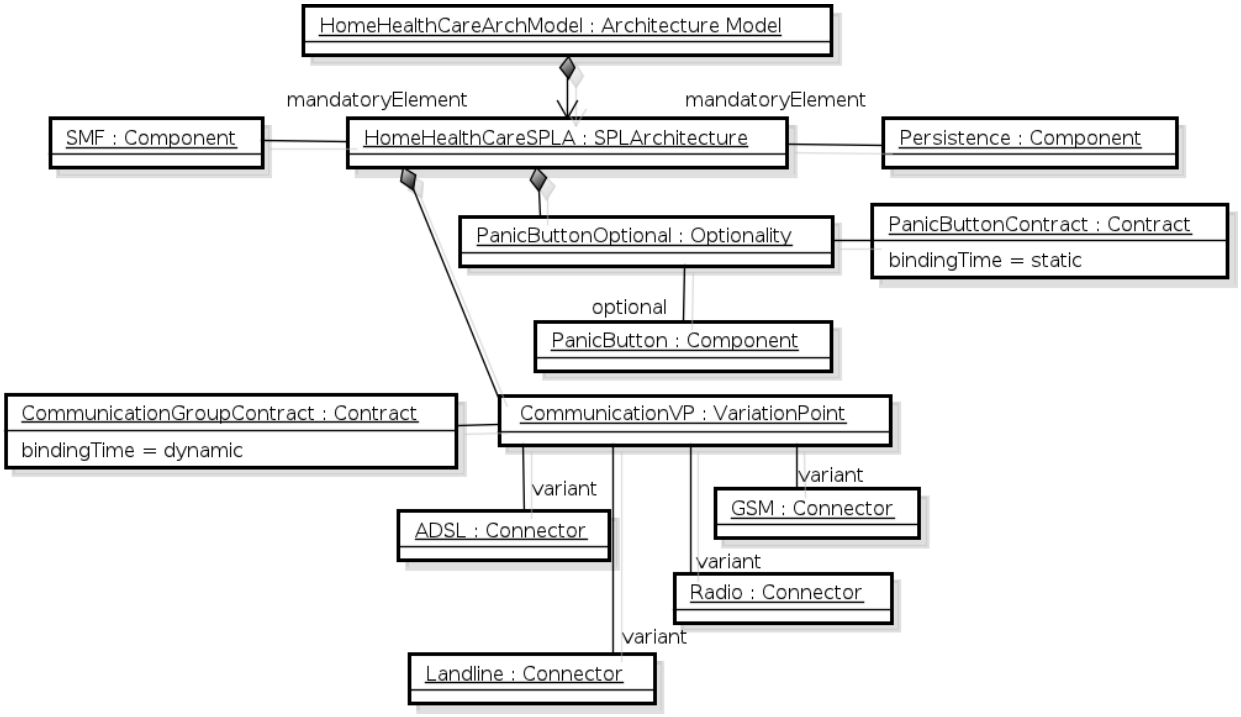


Figura 6.10: Arquitetura da LPS com um ponto de variação e uma opcionalidade.

6.2.3 *Como construir um Modelo de Configuração?*

Um Modelo de Configuração pode ser construído instanciando-se o Metamodelo de Configuração. Com este modelo, a arquitetura de um produto da LPS pode ser derivada a partir da configuração (seleção) de características do Modelo de Características. Caso haja características dinâmicas na Configuração de Características, o modelo permite que

a Arquitetura do Produto seja adaptada em tempo de execução, seja pela intervenção do usuário, modificando a Configuração de Características, seja pela mudança do contexto de execução do produto.

6.2.3.1 Representação de Variabilidades Estáticas e Dinâmicas

Para mostrar como as variabilidades são representadas, seis Contratos associados a características do Modelo de Características da Figura 6.1 são desenvolvidos a seguir.

A) Característica Opcional PanicButton

A Figura 6.11 apresenta um Diagrama de Objetos onde o Contrato Estático *PanicButtonContract* (classe *Contract*, atributo *bindingTime=static*) está associado, no nível da Arquitetura da LPS, à opcionalidade *PanicButtonOptional*. O Contrato também está associado à característica opcional *PanicButton*, conforme o diagrama da Figura 6.5.

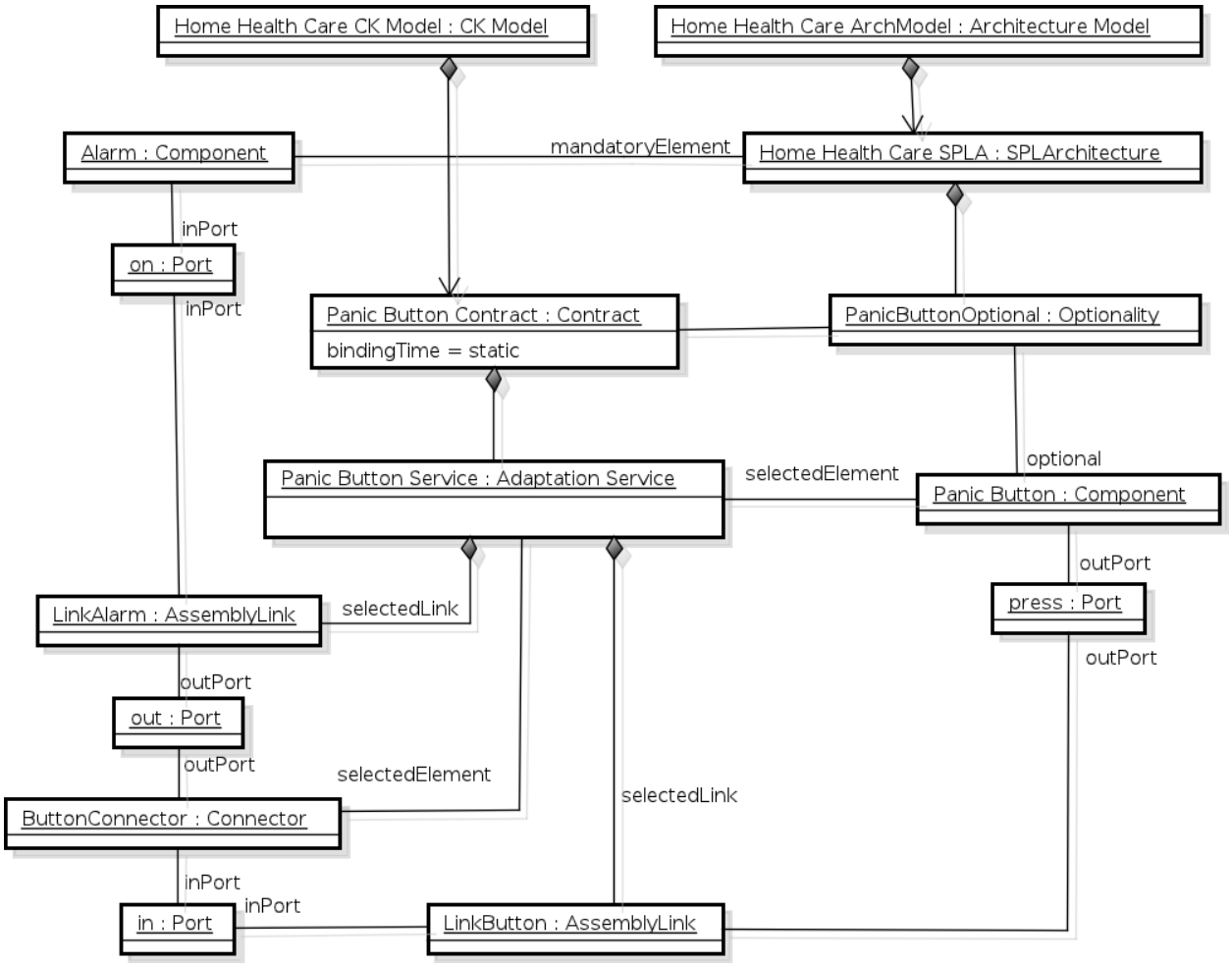


Figura 6.11: Contrato *PanicButtonContract* representando a característica opcional *PanicButton*.

O serviço *PanicButtonService* define que, caso a característica *PanicButton* seja selecionada durante a derivação, o componente arquitetural *PanicButton* deve ser selecionado (associação *selectedElement*) e também interligado (associação *selectedLink*) ao conector *ButtonConnector* (via objeto *LinkButton*), e este, por sua vez, deve ser interligado (associação *selectedLink*) ao componente mandatório *Alarm* (via objeto *LinkAlarm*). Assim, a seleção da característica opcional *PanicButton* durante a derivação de um produto faz com que a arquitetura deste produto tenha uma configuração contendo o componente *PanicButton* interligado ao componente mandatório *Alarm* por meio do conector *ButtonConnector*.

B) Característica Opcional Reminder

A Figura 6.12 traz um Contrato também modelado para uma característica opcional, neste caso a *Reminder*, uma subcaracterística para configurar um sistema de lembrete para o paciente.

O Contrato *ReminderContract* possui o serviço *ReminderService*, o qual configura o componente *Reminder* interligado ao componente mandatório *CarePlan*. *Reminder* obtém os dados agendados em *CarePlan* e, com eles, cria e envia mensagens de aviso para *ReminderEvent*, um componente baseado em eventos que notifica outros componentes a ele interligados.

C) Grupo de Características CommunicationGroup

Diferentemente dos Contratos anteriores, *CommunicationGroupContract*, apresentado na Figura 6.13, está associado ao ponto de variação *CommunicationVP* da Arquitetura da LPS e ao grupo de características alternativas *CommunicationGroup*, conforme mostrado no diagrama da Figura 6.5. Com o propósito de simplificar a apresentação do diagrama, foram suprimidas as características alternativas *Landline* e *Radio*.

O Contrato Dinâmico possui dois serviços, *ADSLService* e *GSMService*, os quais definem as configurações arquiteturais que devem ser realizadas caso as respectivas características (*ADSL* e *GSM*) sejam selecionadas. Os elementos arquiteturais *ADSL* e *GSM* são variantes que devem ser configuradas como conectores para se estabelecer a comunicação entre os componentes mandatórios *Alarm* e *SMF*.

De acordo com o Modelo de Características, apenas uma característica de comunicação pode ser selecionada, devido à relação XOR definida entre elas (Subseção 6.2.1.2). Caso *ADSL* seja selecionada, o serviço *ADSLService* é ativado (atributo *isActive*). A partir disso, o serviço adapta a arquitetura selecionando o conector *ADSL* (*selectedElement*)

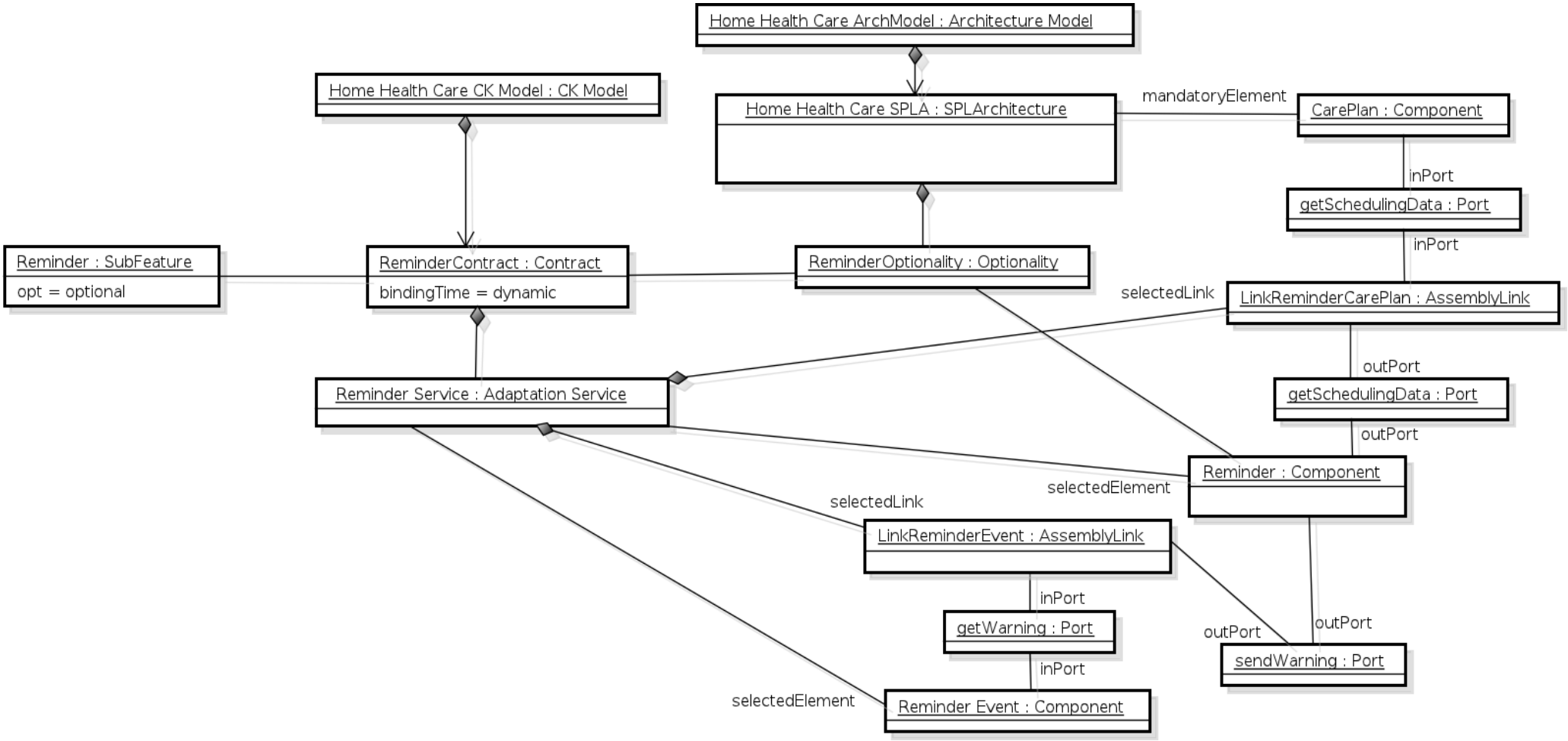


Figura 6.12: Contrato *ReminderContract* representando a característica opcional *Reminder*.

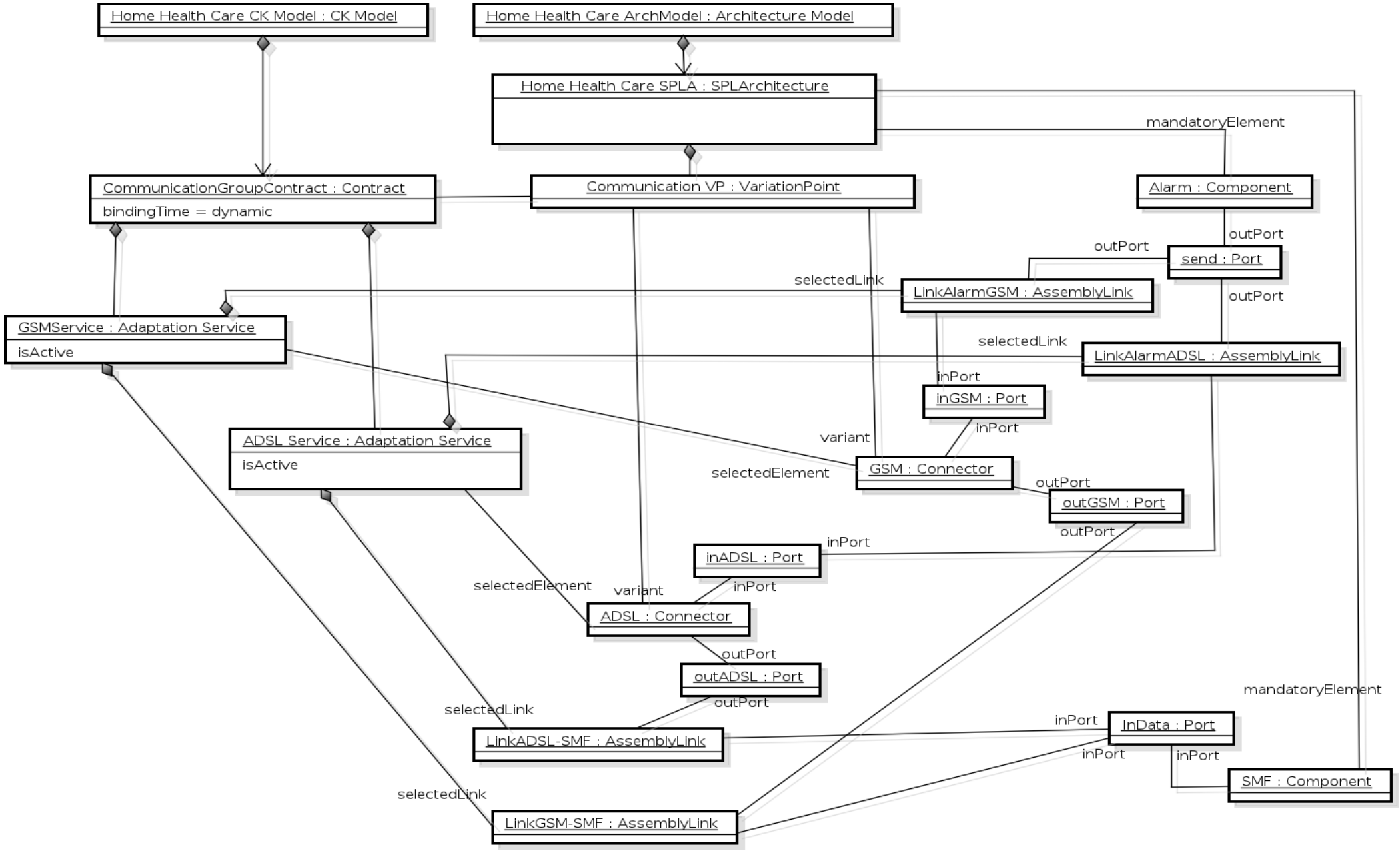


Figura 6.13: Contrato *CommunicationGroupContract* representando o grupo de características *CommunicationGroup*.

e o interligando (*selectedLink*) a *Alarm* (*LinkAlarmADSL*) e a *SMF* (*LinkADSL-SMF*). Por outro lado, a seleção da característica *GSM* faz com que o serviço *GSM* seja ativado (atributo *isActive*) e adapte a arquitetura, selecionando o conector *GSM* (*selectedElement*) e realizando a sua interligação (*selectedLink*) com *Alarm* (*LinkAlarmGSM*) e com *SMF* (*LinkGSM-SMF*).

D) Grupo de Características MedicalSensorGroup

A Figura 6.14 traz outro Contrato associado a um ponto de variação, equivalente ao *CommunicationGroupContract*. Trata-se do Contrato Dinâmico *MedicalSensorGroupContract* associado ao ponto de variação *MedicalSensorVP* da Arquitetura da LPS e ao grupo de características *MedicalSensorGroup*.

O Contrato possui dois serviços, sendo cada um deles associado a uma variante do ponto de variação. O primeiro serviço (*GlucometerService*) realiza adaptações na arquitetura para configurar um glucômetro (componente *Glucometer*) de forma que possa enviar os seus dados coletados para o componente *MedicalSensorEvent*. Este, ao receber os dados, notifica o componente mandatório *Persistence* para que sejam então persistidos. Uma configuração similar é realizada pelo serviço *BloodPressureService* para que o componente *MedicalSensorEvent* também receba os dados coletados pelo componente *BloodPressure* e notifique *Persistence* para que a persistência seja feita. Este diagrama é uma arquitetura possível para as características *Glucometer* e *BloodPressure*. Para efeito de simplificação, as características *WeightScale* e *HeartRate* foram suprimidas da arquitetura, assim como a mandatória *MedicalSensor*.

E) Grupo de Características HealthProblemGroup

Para complementar o Contrato *MedicalSensorGroupContract*, a Figura 6.15 mostra um Contrato Dinâmico também associado a um ponto de variação. O Contrato *HealthProblemGroupContract* configura a arquitetura para o grupo de características *HealthProblemGroup*, tendo como ponto de variação o problema de saúde de um paciente, composto das variantes *Diabetes* e *Hypertension*.

Compostos de regras personalizadas para disparar um alarme junto à *SMF*, estes componentes são configurados pelos seus respectivos serviços, *DiabetesService* e *HypertensionService*. Supondo que *Hypertension* seja a variante selecionada para o ponto de variação e o componente *BloodPressure* também seja selecionado (representado no diagrama da Figura 6.14), o serviço *HypertensionService* configura o componente *MedicalSensorEvent* na arquitetura para que receba os dados coletados pelo componente *BloodPressure* e no-

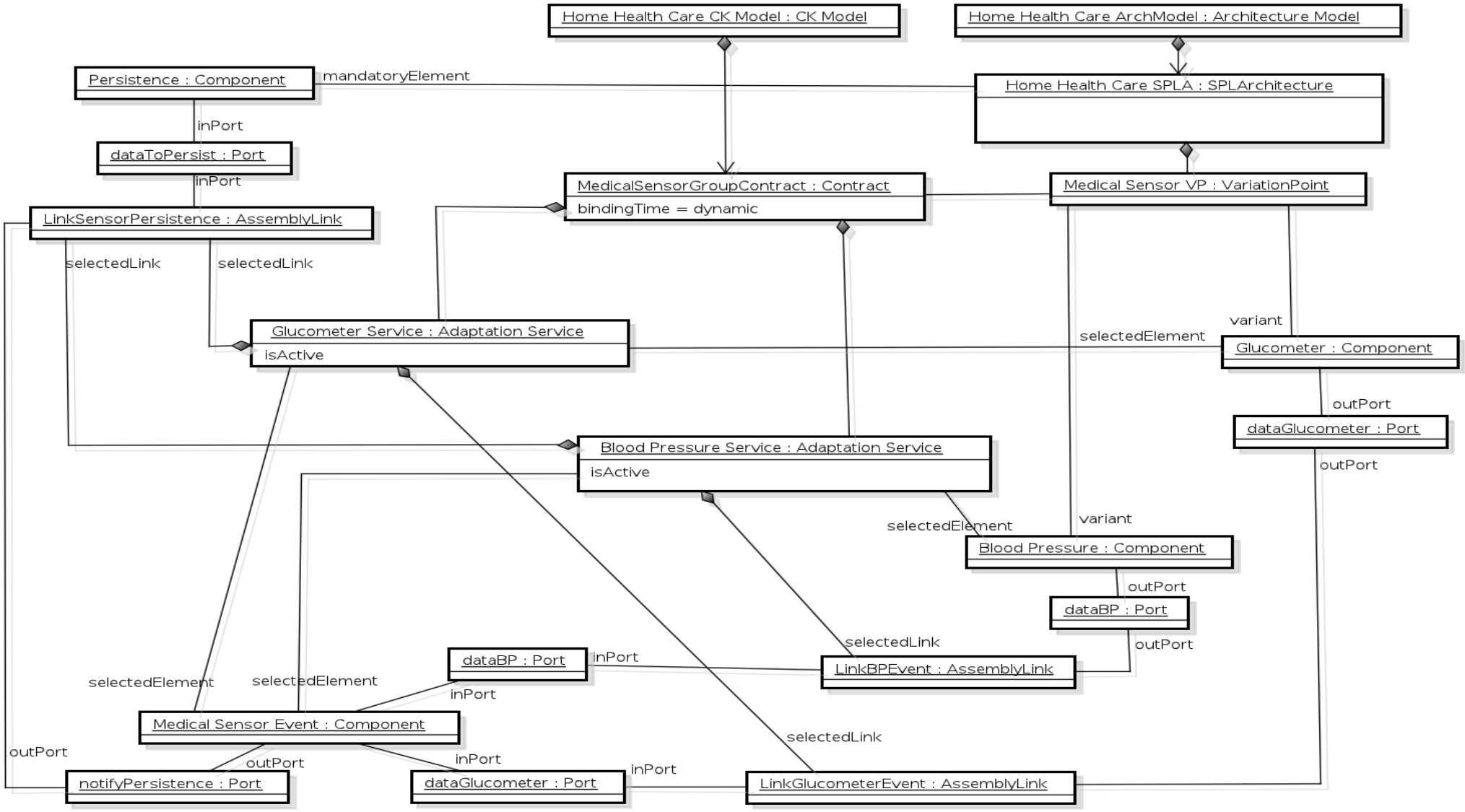


Figura 6.14: Contrato *MedicalSensorGroupContract* representando o grupo de características *MedicalSensorGroup*.

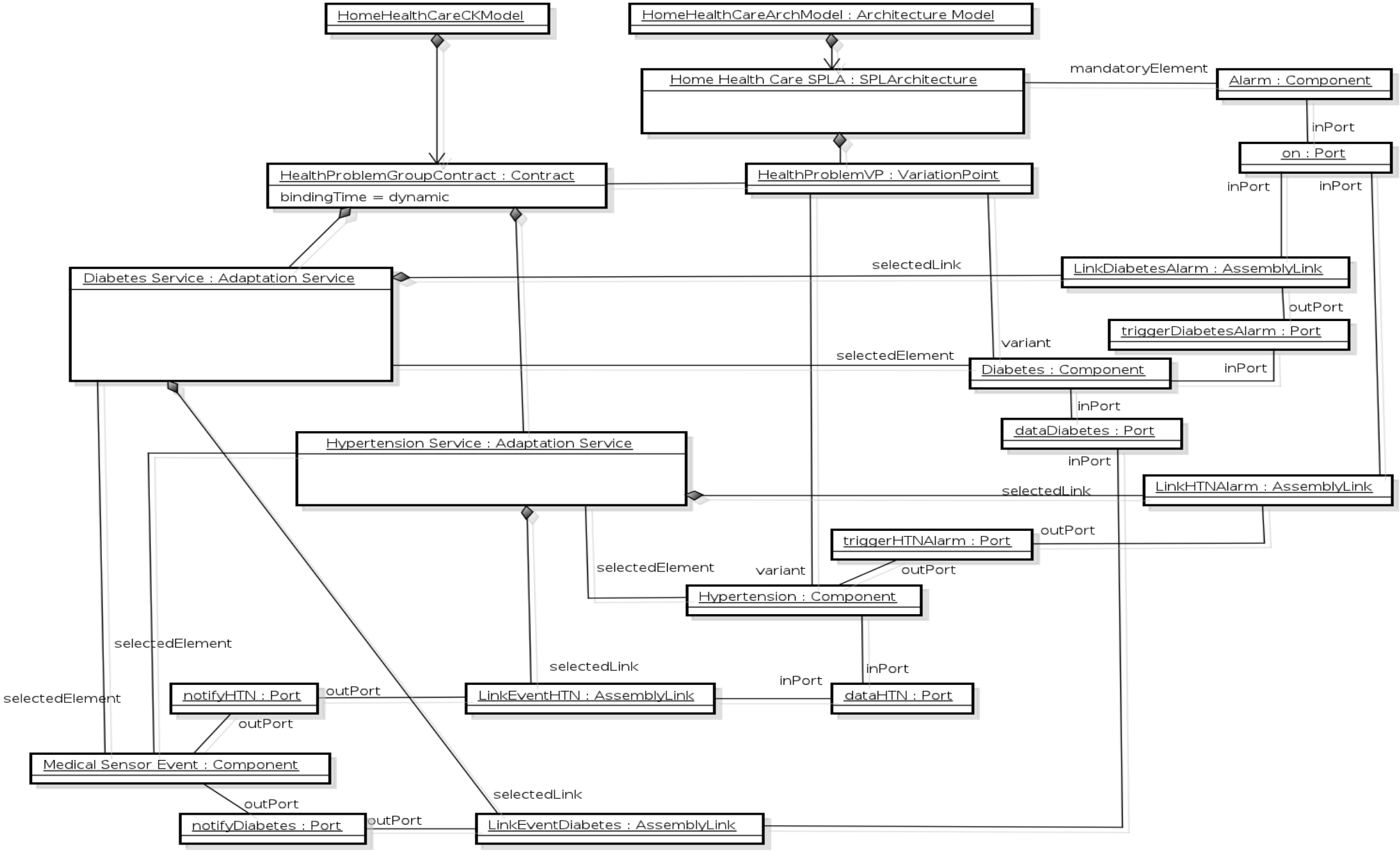


Figura 6.15: Contrato *HealthProblemGroupContract* representando o grupo de características *HealthProblemGroup*.

tifique *Hypertension*, para que este verifique se os dados coletados representam ou não uma situação de anormalidade. Caso seja detectada alguma anormalidade, *Hypertension* envia um alarme de emergência por meio de sua porta de saída. Vale lembrar que a configuração definida pelo serviço *HypertensionService* somente pode ser realizada se a variante *BloodPressure* também for selecionada. Essa restrição de dependência está no Modelo de Características mostrado na Figura 6.1. A Subseção 6.2.3.2 mostra como criar perfis e regras para tratar das restrições de dependência no nível da arquitetura.

A configuração realizada pelo serviço *DiabetesService* é semelhante, envolvendo no entanto os componentes *Glucometer* e *Diabetes*. Para efeito de simplificação, a característica *CardiacInsufficiency* foi suprimida da arquitetura, e também a característica opcional *HealthProblem*. Para representar esta última, pode-se criar um Contrato específico para ela, de forma semelhante ao Contrato construído para a característica opcional *PanicButton* (Figura 6.11). Vale ressaltar que a opcionalidade de *HealthProblem* significa que apenas nos casos onde ela for selecionada, uma ou mais alternativas do grupo podem ser selecionadas.

F) Grupo de Características DisplayDeviceGroup

A Figura 6.16 apresenta o Contrato *DisplayDeviceGroupContract*, com uma configuração de serviços voltada para a seleção de dispositivos que mostrem as mensagens de aviso ao paciente.

As características alternativas disponíveis são, de acordo com o Modelo de Características (Figura 6.1): *TV*, *Tablet*, *MobilePhone* e *OrdinaryDisplay*. Para tornar mais simples o diagrama, apenas os serviços para *TV* e *Tablet* estão modelados: *TVService* e *TabletService*. Os serviços configuram os componentes correspondentes aos dispositivos de forma interligada ao componente *ReminderEvent*, o mesmo apresentado no Contrato *ReminderContract* do diagrama da Figura 6.12. Com isso, os avisos ao paciente são enviados ao componente *ReminderEvent*, o qual notifica, usando as suas portas de saída (*notifyTablet* e *notifyTV*), os dispositivos selecionados de acordo com os respectivos serviços.

6.2.3.2 Regras de Composição

A) Restrições de Dependência

O Modelo de Características apresentado na Figura 6.1 possui um cenário de dependência no qual a característica *Hypertension* requer a *BloodPressure* na derivação de um produto, e a característica *Diabetes* requer a *Glucometer*. A Figura 6.17 mostra perfis

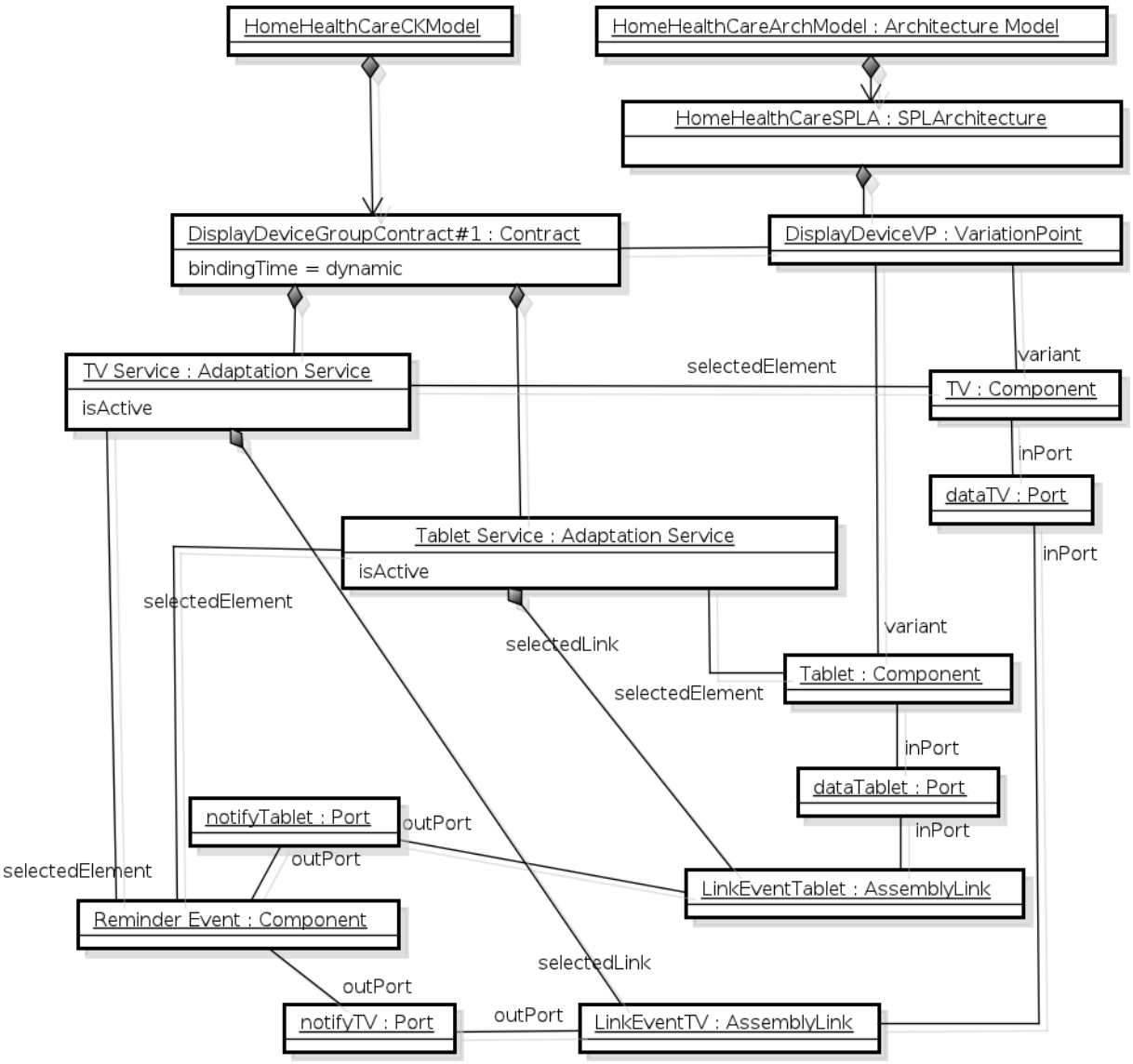


Figura 6.16: Contrato *DisplayDeviceGroupContract#1* representando o grupo de características *DisplayDeviceGroup*.

e regras de composição criados para representar a dependência.

O perfil *HypertensionProfile* contém a regra *HypertensionRule*, a qual está associada com as características *Hypertension* e *BloodPressure* (*relevantFeature*). *DiabetesProfile*, por sua vez, contém a regra *DiabetesRule*, a qual envolve as características *Diabetes* e *Glucometer*.

Para tornar mais simples o Diagrama de Objetos da Figura 6.17 e, ao mesmo tempo, facilitar o seu entendimento, o próprio nome da característica está sendo usado como variável lógica na condição da regra (atributo *condition*), onde o valor *true* significa que a respectiva característica está selecionada, e o valor *false* significa que não está selecionada. A regra *HypertensionRule* define que a seleção de *Hypertension* requer a seleção de *Blood-*

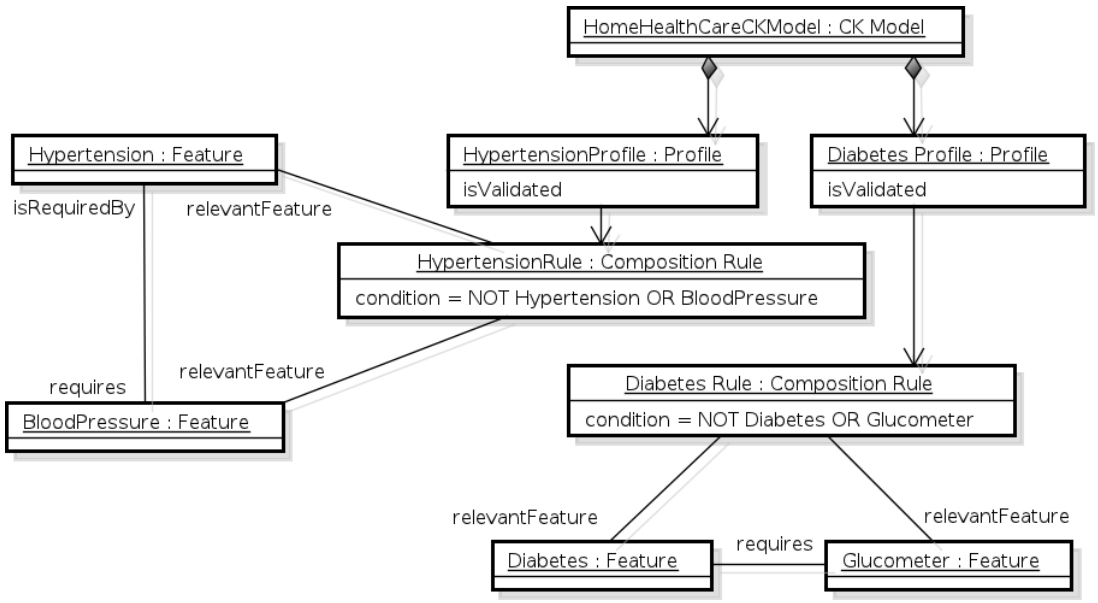


Figura 6.17: Perfis *HypertensionProfile* e *DiabetesProfile* com suas regras de composição.

Pressure, enquanto que a regra *DiabetesRule* define que a seleção de *Diabetes* requer a seleção de *Glucometer*. As regras estão definidas com a expressão *NOT p OR q*, o que equivale a uma implicação lógica na forma $p \Rightarrow q$, indicando que q deve ser *true* sempre que p for *true*.

B) Relações OR e XOR

A Figura 6.18 estende o diagrama da Figura 6.17 definindo o novo perfil *HealthProblemProfile*, com o objetivo de definir a relação OR entre as características *Hypertension* e *Diabetes*. Uma vez que a característica *HealthProblem* está definida como opcional no Modelo de Características da Figura 6.1, três regras foram definidas para validar o perfil. A primeira e a segunda regras garantem que, se *Diabetes* ou *Hypertension* estiverem selecionadas, *HealthProblem* deve estar selecionada. A terceira regra garante que, se *HealthProblem* estiver selecionada, *Diabetes* ou *Hypertension* devem estar selecionadas. Apenas se as três regras forem satisfeitas, ou seja, resultarem em *true*, o perfil *HealthProblemProfile* estará válido (*isValidated=true*).

O diagrama da Figura 6.19, por sua vez, adiciona o perfil *CommunicationProfile*, o qual define uma regra XOR entre as características *GSM* e *ADSL*.

6.2.3.3 Modelagem de Contexto

Além das regras de composição, as características *ADSL* e *GSM* podem ter regras que avaliam o contexto no qual operam. O objetivo é prover algum mecanismo que permita

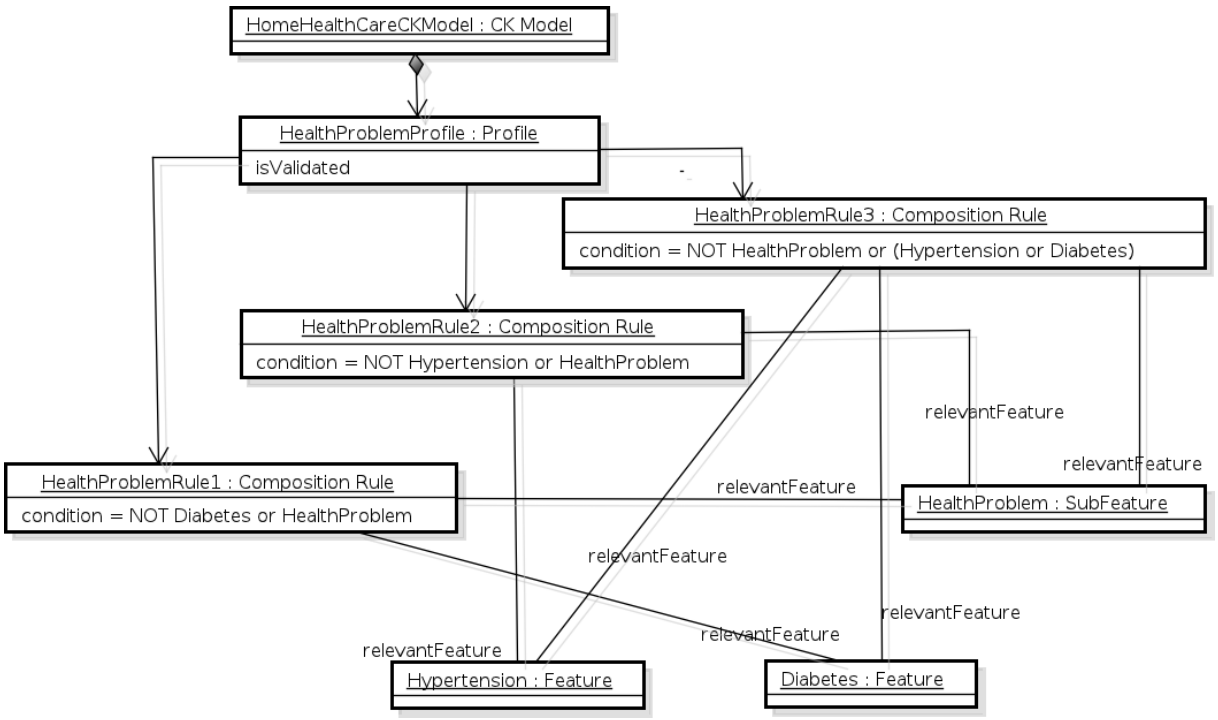


Figura 6.18: Perfil *HealthProblemProfile* com três regras de composição.

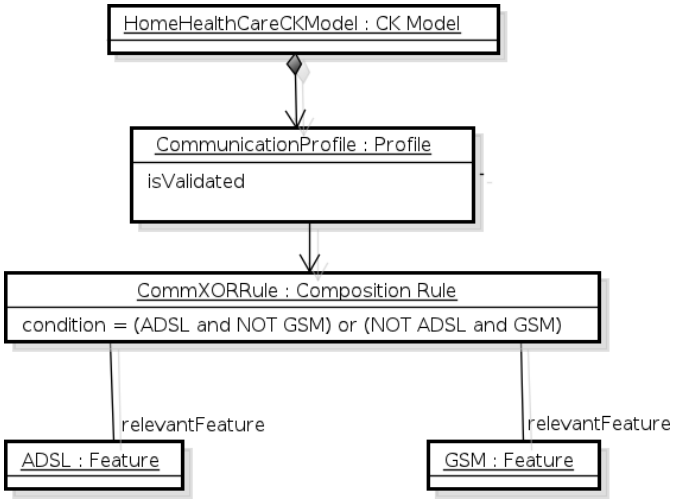


Figura 6.19: Perfil *CommunicationProfile* com uma regra de composição.

adaptar a Arquitetura do Produto de acordo com a disponibilidade operacional de cada uma das tecnologias de comunicação. Nesse sentido, o diagrama da Figura 6.20 apresenta perfis criados para controlar os serviços *GSMService* e *ADSLService* com base em regras de contexto. Estes serviços foram apresentados no Contrato *CommunicationGroupContract* da Figura 6.13,

Na Figura 6.20, *TransportType* é um *ConnectorType* constituído por dois elementos de contexto, *TransportName* e *TransportAvailability*, uma vez que *ConnectorType* é uma subclasse de *ContextualEntity*, como apresentado na Seção 5.2.3.3. Estes elementos de

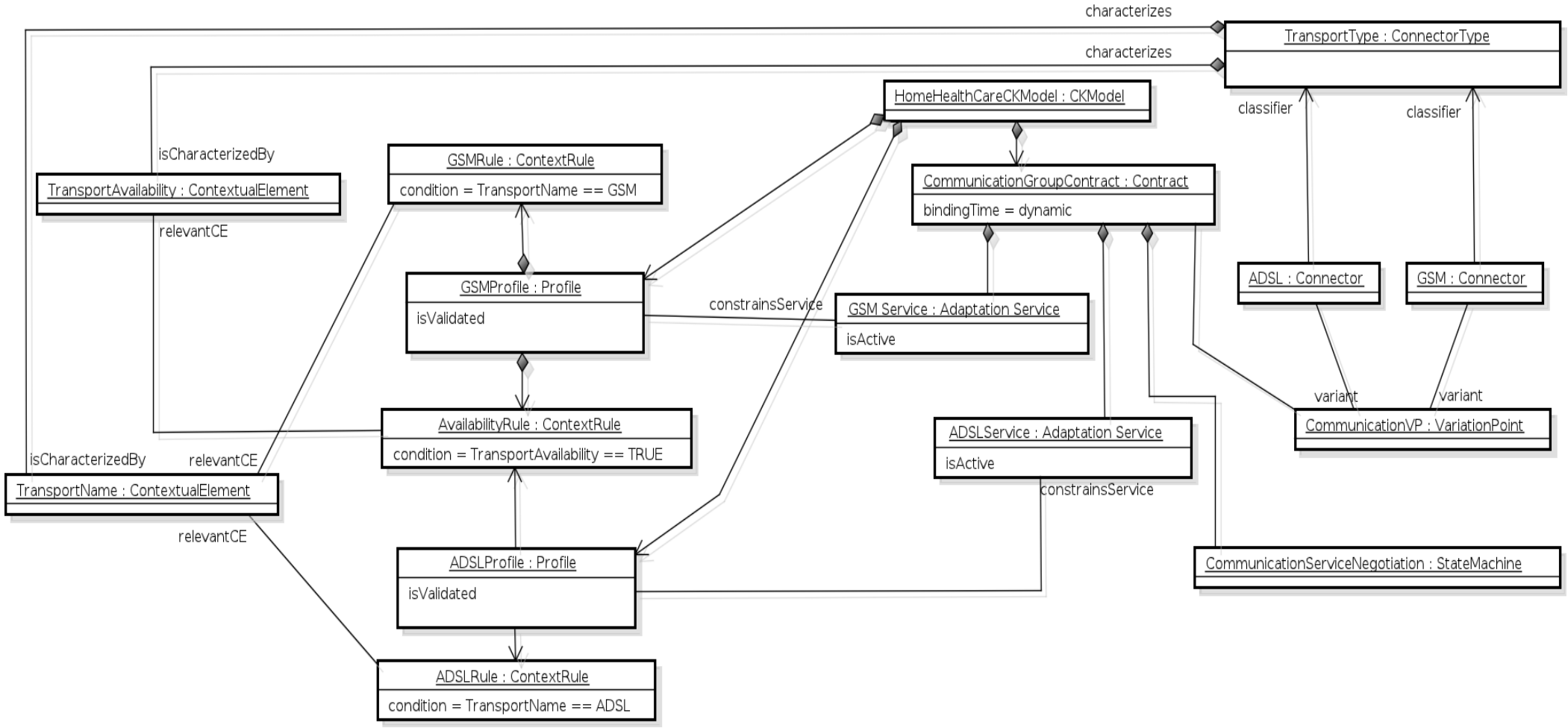


Figura 6.20: Modelagem de contexto para os serviços *GSMService* e *ADSLService*.

contexto mantêm propriedades de contexto dos conectores *ADSL* e *GSM*. De acordo com o diagrama, o serviço *GSMService* somente pode estabelecer a sua configuração se o perfil *GSMProfile* estiver válido (*isValidated=true*), ou seja, se a tecnologia de comunicação GSM (regra *GSMRule*) estiver disponível (regra *AvailabilityRule*). As condições para se estabelecer ADSL são similares, envolvendo o perfil *ADSLProfile* e a regra *ADSLRule*. Vale ressaltar que, para tornar mais simples a representação, as regras estão descritas apenas com os objetos do tipo *ContextualElement* sem o uso de atributos.

Neste modelo, as entidades e elementos de contexto definidos auxiliam a identificar se as respectivas tecnologias de comunicação estão operacionais. Por exemplo, se a GSM tem a sua operação interrompida por conta de alguma falha, o perfil *GSMProfile* se torna inválido fazendo com que o serviço *GSMService* seja interrompido e, por conseguinte, o serviço *ADSLService* seja iniciado. Se ambas estiverem disponíveis para operação, apenas um dos serviços é executado, de acordo com a negociação associada ao Contrato, representada por *CommunicationServiceNegotiation*. Este objeto tem acesso aos serviços por meio do Contrato e estabelece critérios para se selecionar um ou mais deles para execução, conforme descrito nas Seções 4.2.1 e 5.2.3.6. Neste caso, uma máquina de estados (*StateMachine*) é usada para realizar a transição entre os serviços.

Outro grupo de características pode ter regras de contexto definidas. Trata-se do grupo *DisplayDeviceGroup*, já representado no nível da Arquitetura da LPS por meio do ponto de variação *DisplayDeviceVP* (Figura 6.16). O diagrama da Figura 6.21 apresenta um novo Contrato, denominado *DisplayDeviceGroupContract#2*, associado ao ponto de variação e composto do serviço *DisplayDeviceService*, o qual usa informações de contexto com o objetivo de selecionar um dispositivo (entre TVs e *tablets*) para avisar ao paciente sobre suas atividades registradas no plano de cuidados.

O primeiro Contrato, *DisplayDeviceGroupContract#1* apresentado no diagrama da Figura 6.16, possui serviços que interligam um componente *TV* (*TVService*) e um componente *Tablet* (*TabletService*) ao componente *ReminderEvent* por meio de portas de notificação (*notifiyTV* e *notifyTablet*). Assim, tanto a *TV* quanto o *Tablet*, ou mesmo ambos, podem ser usados para avisar o paciente.

O Contrato *DisplayDeviceGroupContract#2*, no entanto, faz com que o aviso seja apresentado no dispositivo que estiver mais próximo do paciente, considerando que podem haver várias TVs e/ou vários *tablets*, ou seja, vários componentes *TV* e/ou vários componentes *Tablet*. Para isso, um modelo contextual é descrito onde as propriedades de contexto dos dispositivos e do paciente são representadas como entidades de contexto

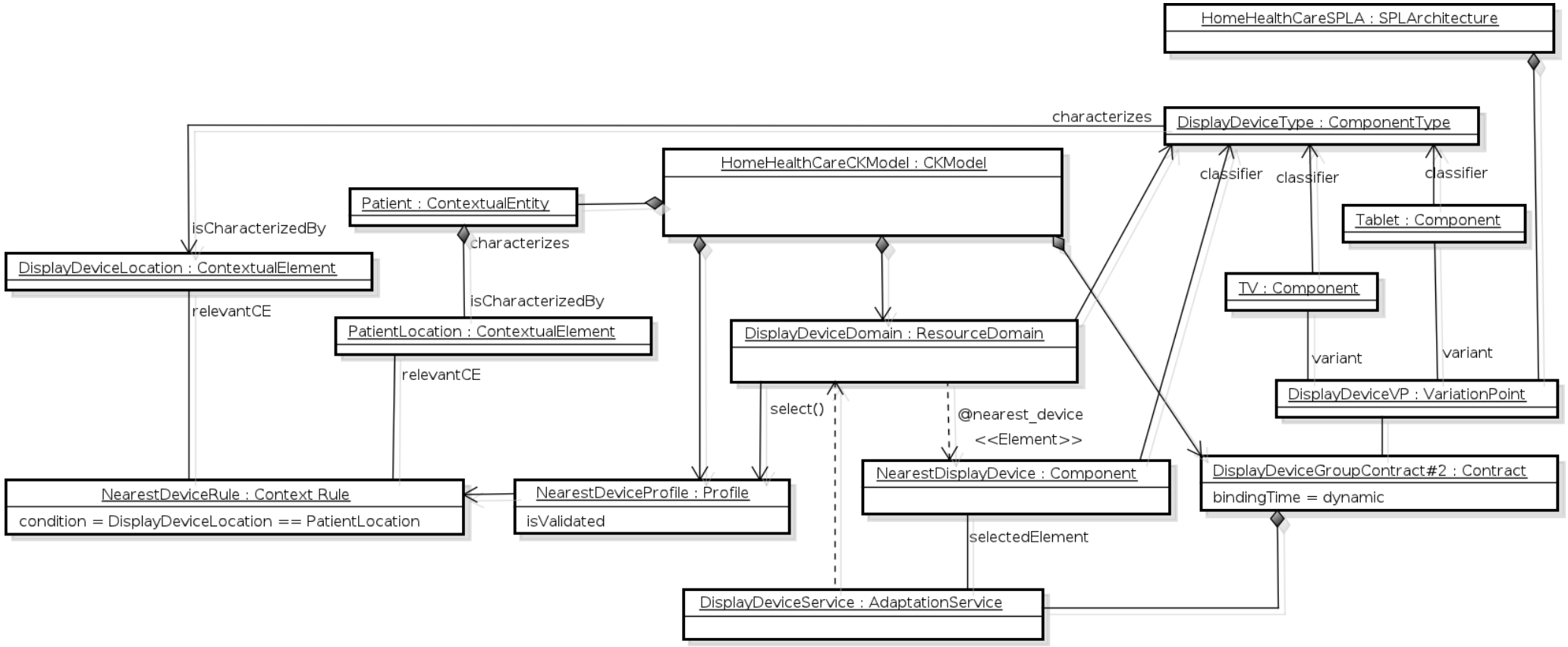


Figura 6.21: Modelagem de contexto para encontrar o dispositivo mais próximo do paciente.

(*DisplayDeviceType* e *Patient*) e como elementos de contexto (*DisplayDeviceLocation* e *PatienLocation*), incluindo a localização dos dispositivos e do paciente. Como no Metamodelo, *ComponentType* é uma subclasse de *ContextualEntity*, *TV* e *Tablet* terão suas propriedades de contexto coletadas do ambiente de execução visando identificar, por meio da regra *NearestDeviceRule*, o dispositivo que está no mesmo ambiente do paciente. Esta regra compõe o perfil *NearestDeviceProfile*, o qual está associado a *DisplayDeviceDomain* (classe *ResourceDomain*), objeto que representa o Domínio de Recursos a serem consultados junto ao Diretório de Recursos (veja Seções 4.2.2 e 5.2.3.5).

O serviço de adaptação *DisplayDeviceService* invoca a operação *select*, a qual busca junto ao Diretório de Recursos a instância do dispositivo que satisfaz a regra, ou seja, aquela que representa o dispositivo que esteja mais próximo do paciente. Como o objeto *DisplayDeviceDomain* está associado ao Tipo *DisplayDeviceType*, uma superclasse de *TVType* e *TabletType*, apenas instâncias destes tipos são retornadas pela operação *select*. O componente *@nearest_device* representa, no Diagrama de Objetos, uma instância retornada pela busca.

Na sequência, o serviço pode então adaptar a arquitetura interligando a porta de entrada do componente retornado à porta de saída do componente *ReminderEvent*, de forma similar ao realizado pelo Contrato *DisplayDeviceGroupContract#1*. A forma com que o Contrato *DisplayDeviceGroupContract#2* foi construída, torna possível modelar a configuração das variabilidades da arquitetura sem a necessidade de nomear as suas instâncias. Isso permite que novas instâncias de dispositivos que se registram no Diretório de Recursos, por exemplo, uma nova TV instalada na cozinha da casa, sejam configuradas na Arquitetura do Produto.

6.3 Derivação e Implantação do Produto

O Metamodelo deve ter condições de gerar durante o processo de derivação de produtos da LPS, a Configuração de Características e a Arquitetura do Produto. Ambos são fundamentais para a representação da adaptação arquitetural dos produtos dinâmicos da LPS. As questões relacionadas à derivação e implantação de produtos são apresentadas a seguir.

6.3.1 *Como gerar a Configuração de Características do produto?*

A partir do Modelo de Características apresentado na Figura 6.1, pode-se derivar configurações de características de produtos. O diagrama da Figura 6.22 traz a Configuração de Características de um produto, representada pela classe *ProductFeatureConfiguration*.

O produto *Product#1* contém as características mandatórias *SMF*, *CarePlan*, *Alarm* e *Persistence*, todas configuradas por meio da associação *mandatoryFeature* da classe *ProductFeatureConfiguration*. O produto contém ainda a característica opcional *PanicButton*, selecionada como resultado da resolução da variabilidade estática, e configurada usando-se a associação *selectedFeature*. Por meio da associação *dynamicFeature* estão configuradas as características dinâmicas *ADSL* e *GSM* que fazem parte do grupo *CommunicationGroup*. Por serem dinâmicas, ambas serão selecionadas apenas durante a execução do produto. Por opção, as demais características que compõem o grupo, *Landline* e *Radio*, não foram escolhidas para fazer parte da Configuração de Características deste produto. Isso permite criar um produto que não irá lidar com essas características, por exemplo, por indisponibilidade técnica de serem implantadas na residência.

A *ADSL*, além de associada à Configuração de Características do produto por meio de *dynamicFeature*, também está associada via *selectedFeature*. Isso significa que, apesar de dinâmica, essa característica deve fazer parte da Arquitetura do Produto já durante a sua derivação. Com isso, a Arquitetura do Produto é derivada com *ADSL*, mas pode ser adaptada em tempo de execução, tendo a característica *GSM* configurada de acordo com o contexto, como já descrito na Subseção 6.2.3.3.

6.3.2 *Como gerar a Arquitetura do Produto?*

O mesmo produto *Product#1* é apresentado nas Figuras 6.23 e 6.24, mas dessa vez com foco na configuração arquitetural.

O diagrama da Figura 6.23 representa parte da Arquitetura do Produto, derivada tendo como base o Núcleo da Arquitetura da LPS (Figura 6.9), o Contrato *PanicButtonContract* (Figura 6.11), e a Configuração de Características (Figura 6.22).

No diagrama, a Arquitetura do Produto (representada pela classe *ProductArchitectureConfiguration*) possui três elementos configurados: *Alarm*, *PanicButton* e *ButtonConnector*. O componente *Alarm* está configurado por ser relacionado à característica mandatória, e por conseguinte, ser um elemento definido no Núcleo da Arquitetura da

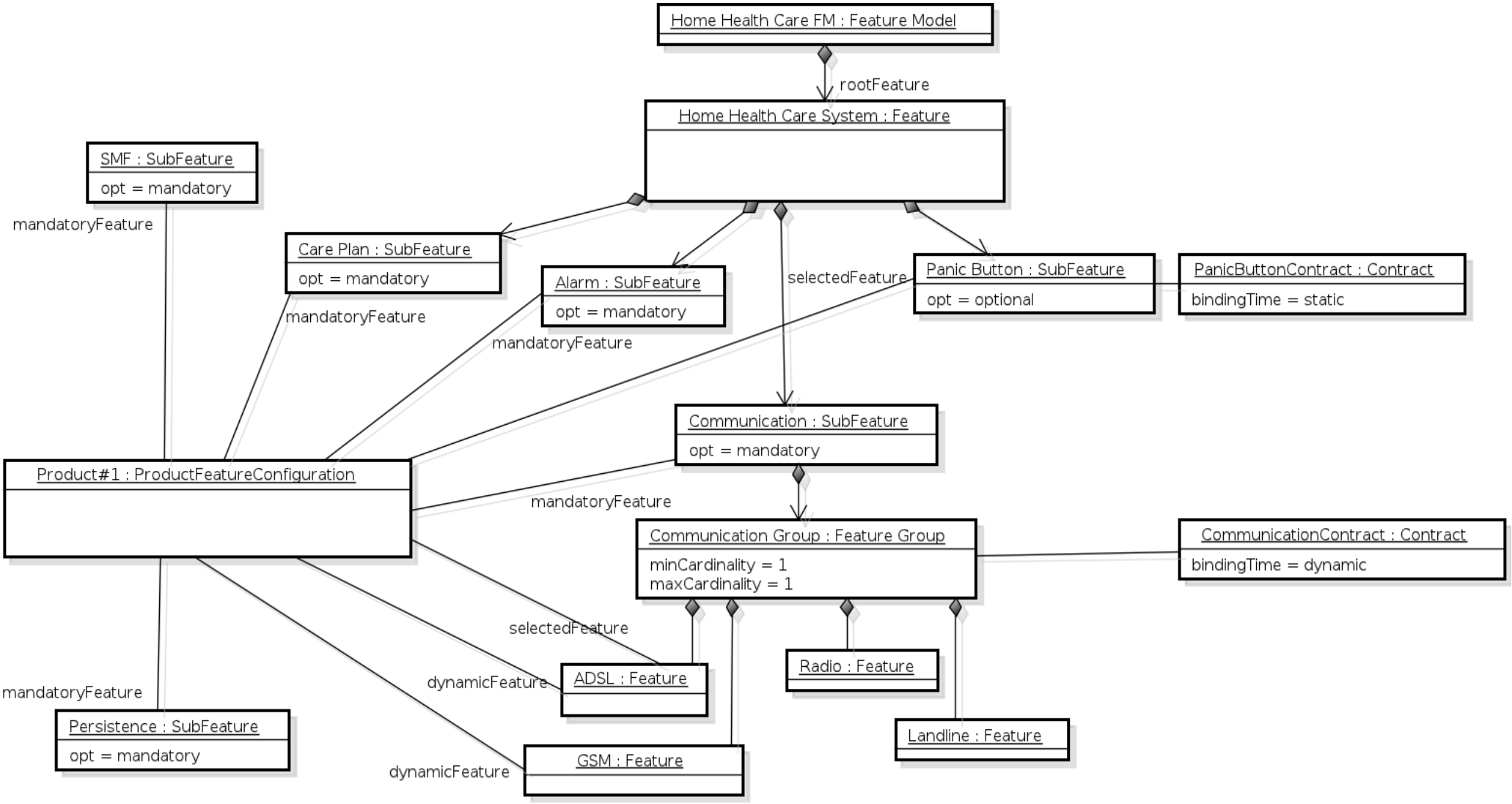


Figura 6.22: Configuração de Características do produto *Product#1*.

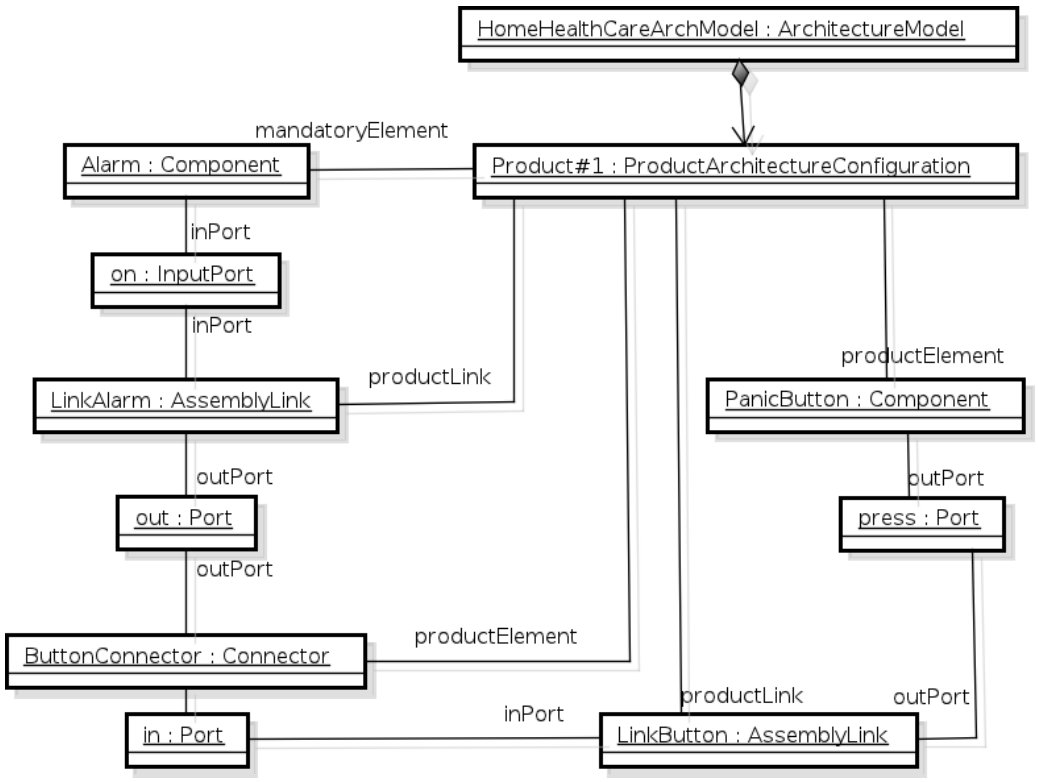


Figura 6.23: Arquitetura do Produto *Product#1* configurada com *PanicButton* e *Alarm*.

LPS por meio da associação *mandatoryElement*. Diferentemente, *PanicButton* e *ButtonConnector* estão configurados pela associação *productElement*, por estarem relacionados à característica opcional *PanicButton*. Neste caso, a configuração foi feita pelo Contrato *PanicButtonContract*, o qual interligou *PanicButton* e *Alarm* por meio do conector *ButtonConnector*, usando as interligações *LinkAlarm* e *LinkButton*, configuradas na arquitetura pela associação *productLink*.

Outra parte da configuração da arquitetura de *Product#1* é mostrada na Figura 6.24. Com o propósito de enviar alarmes para a central de supervisão médica, o componente *Alarm* é interligado ao componente *SMF* via conector *ADSL*.

O conector *ADSL* foi configurado em tempo de derivação pelo serviço *ADSLService* do Contrato *CommunicationGroupContract* (Figura 6.13), pois a característica *ADSL* foi incluída na Configuração de Características por meio da associação *selectedFeature*. Durante a execução, no entanto, a sua arquitetura pode ser adaptada, com a tecnologia de comunicação sendo alterada de ADSL para GSM, e vice-versa.

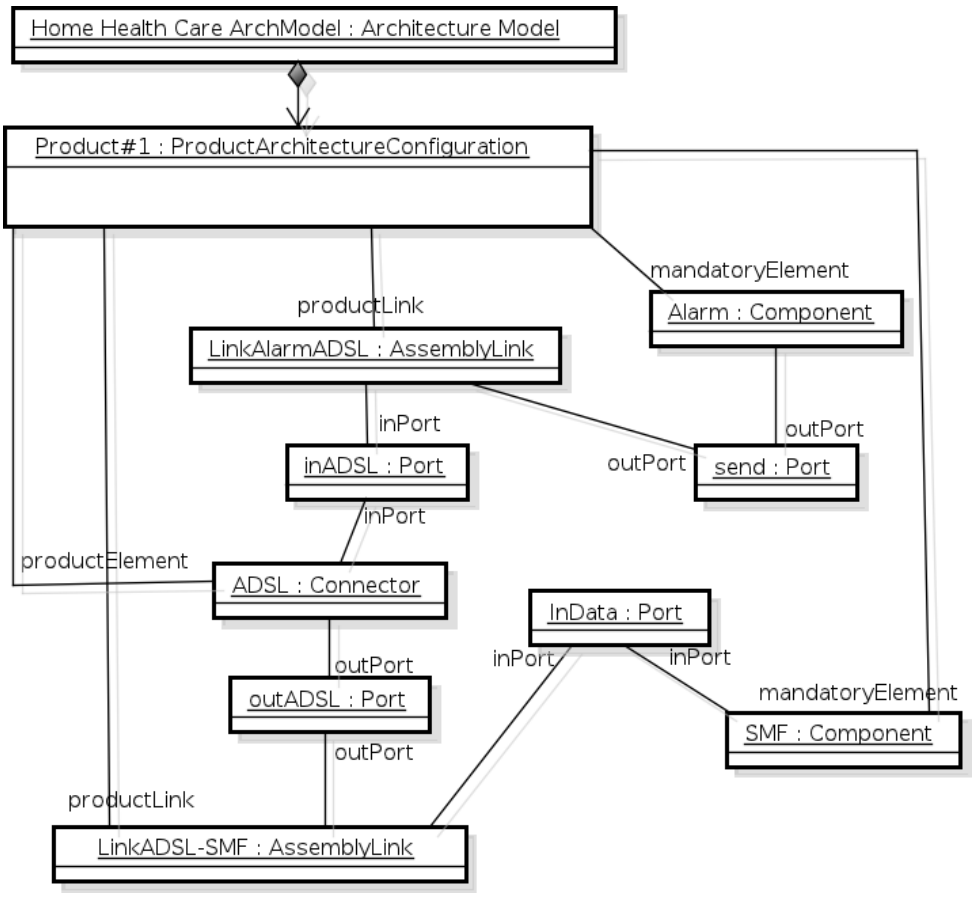


Figura 6.24: Arquitetura do Produto *Product#1* configurada com *ADSL*.

6.3.3 *Como implantar a Arquitetura do Produto?*

A Arquitetura do Produto *Product#1*, representada pelas Figuras 6.23 e 6.24, está pronta para ser implantada e executada. No entanto, como essa arquitetura foi derivada a partir de uma Configuração de Características onde *ADSL* e *GSM* foram selecionadas como *dynamicFeature* (Figura 6.22), o Contrato Dinâmico *CommunicationGroupContract* deve ser implantado junto com a Arquitetura do Produto, para que durante a execução possa realizar adaptações na arquitetura, modificando a tecnologia de comunicação de acordo com o contexto.

6.4 Correspondência Característica-Arquitetura

O Modelo de Configuração, criado a partir do Metamodelo, deve fornecer meios de, a partir de uma característica se alcançar os seus elementos arquiteturais correspondentes, e vice-versa. As questões relacionadas à correspondência característica-arquitetura são apresentadas a seguir.

6.4.1 *Como obter os elementos arquiteturais de uma característica?*

O diagrama da Figura 6.25 mostra que a partir da característica *PanicButton* pode-se alcançar os seus elementos arquiteturais correspondentes, o componente *PanicButton* e o conector *ButtonConnector*. A característica *PanicButton* possui uma associação com o Contrato *PanicButtonContract* (*optionalFeature*) e este possui, em sua composição, o serviço *PanicButtonService*. Partindo então desse serviço e usando a associação *selectedElement*, pode-se obter os elementos arquiteturais que correspondem à característica *PanicButton*.

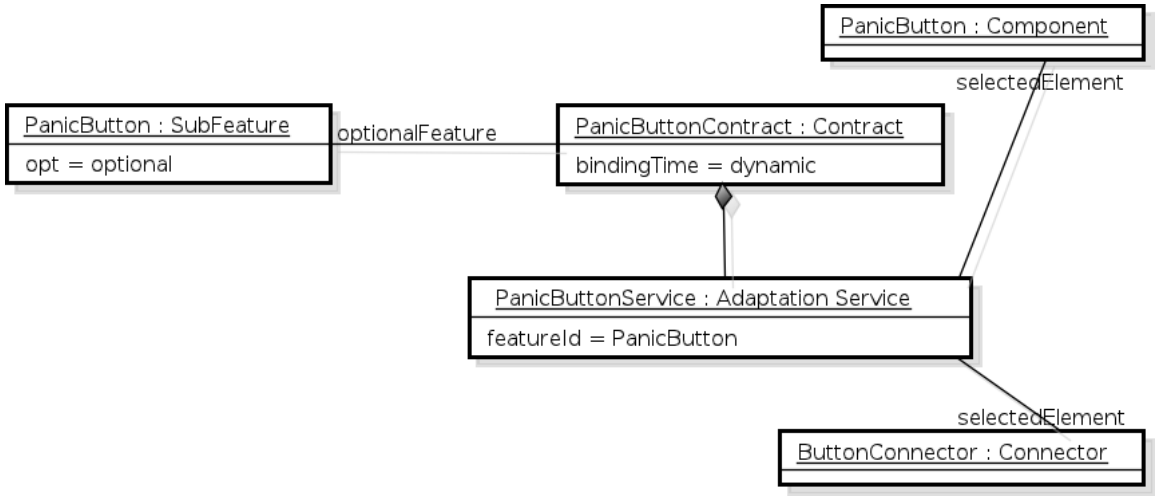


Figura 6.25: Correspondência entre a característica *PanicButton* e elementos arquiteturais.

Em outro cenário, o diagrama da Figura 6.26 mostra como alcançar os componentes correspondentes à característica alternativa *BloodPressure*. O Contrato *MedicalSensorGroupContract* (Figura 6.14) está associado ao grupo de características *MedicalSensorGroup* e possui um serviço que por sua vez está associado à característica *BloodPressure*. A partir daí os respectivos componentes podem ser obtidos por meio do serviço *BloodPressureService* e de sua associação *selectedElement*.

6.4.2 *Como obter as características de um elemento da arquitetura?*

O diagrama da Figura 6.27 mostra que a partir do conector *ADSL*, configurado na arquitetura pelo serviço *ADSLService*, pode-se alcançar a característica alternativa *ADSL* por meio da sua associação com o serviço, o qual compõe o Contrato *CommunicationGroupContract*.

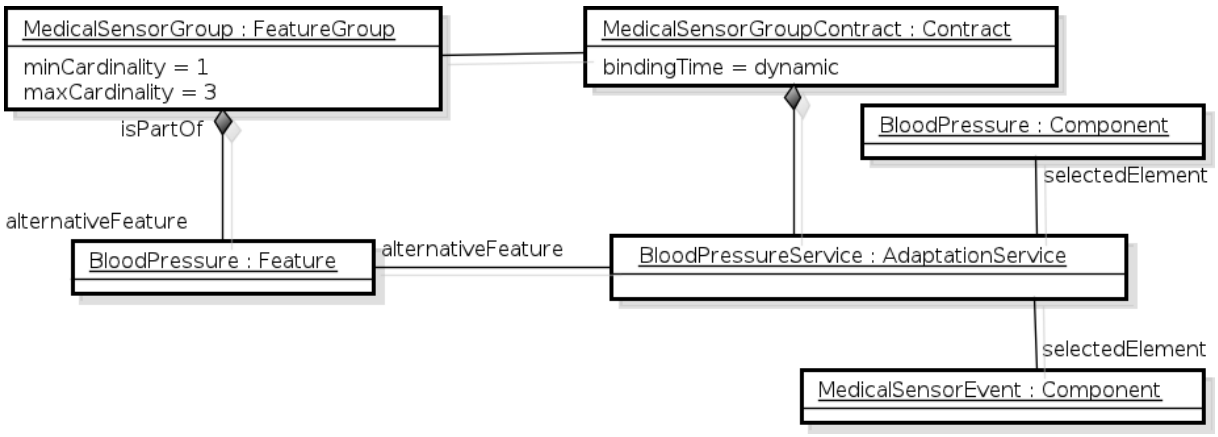


Figura 6.26: Correspondência entre a característica *BloodPressure* e elementos arquiteturais.

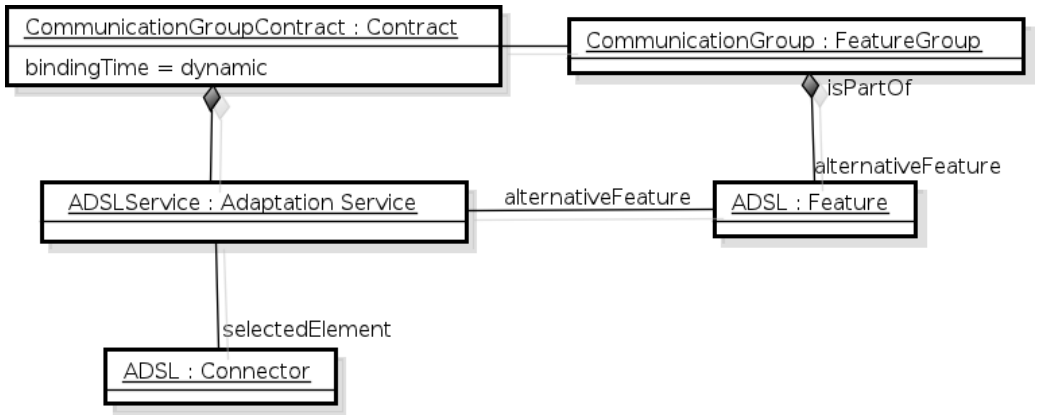


Figura 6.27: Correspondência entre o conector *ADSL* e a característica.

Na Figura 6.28, os componentes *Reminder* e *ReminderEvent* possuem correspondência com a característica *Reminder*. A partir de um dos componentes, a característica pode ser alcançada por meio da associação com o serviço *ReminderService* e, na sequência, pela associação do Contrato *ReminderContract* com a respectiva característica. Este Contrato está no diagrama da Figura 6.12.

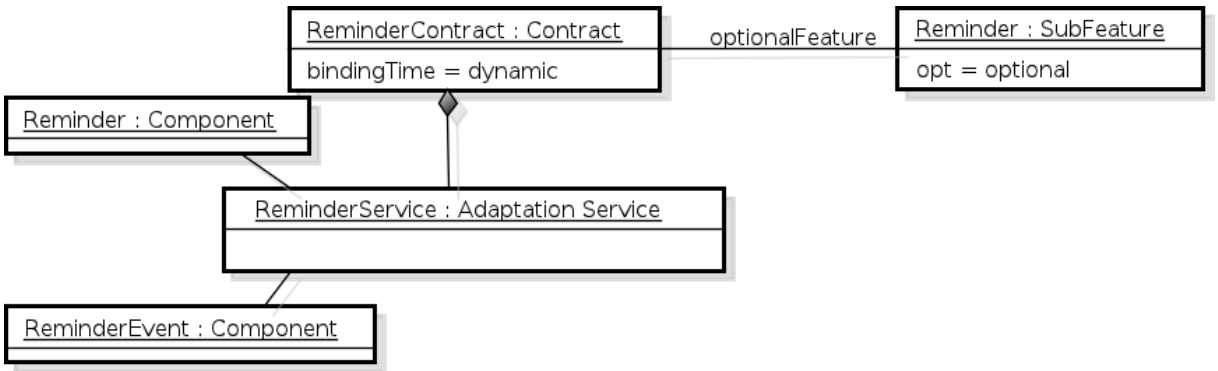


Figura 6.28: Correspondência entre os componentes *Reminder* e *ReminderEvent*, e a característica.

6.5 Adaptação Dinâmica da Arquitetura do Produto

Modificações na Configuração de Características feitas pelo usuário em tempo de execução podem provocar a adaptação da Arquitetura do Produto, adicionando ou removendo os elementos correspondentes à característica dinâmica modificada. Por outro lado, mudanças do contexto de execução do produto podem provocar a adaptação da sua arquitetura e, por conseguinte, modificar a sua Configuração de Características. As questões relacionadas à adaptação dinâmica da Arquitetura do Produto são apresentadas a seguir.

6.5.1 *Como o usuário modifica a configuração de características de um produto?*

A Configuração de Características de um produto é apresentada no Diagrama de Objetos da Figura 6.29.

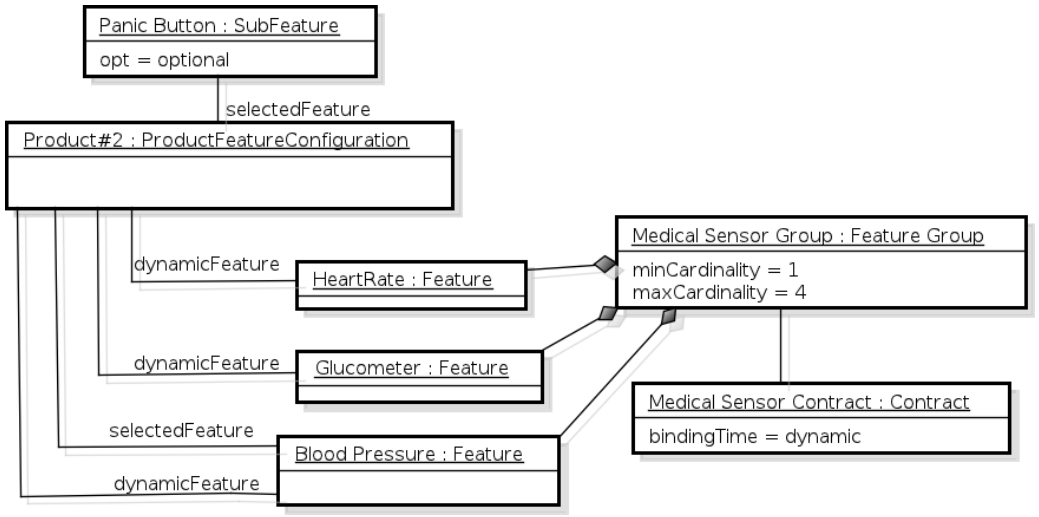


Figura 6.29: Configuração de Características do produto *Product#2* com *PanicButton* e *BloodPressure* selecionadas.

Product#2 possui duas características em sua configuração, *PanicButton* e *BloodPressure*, ambas selecionadas em tempo de derivação e identificadas pela associação *selectedFeature*. O produto está configurado, portanto, com um botão de pânico e um sensor de pressão arterial. As características mandatórias foram suprimidas para simplificar o diagrama.

Uma vez que há características dinâmicas, representadas pela associação *dynamicFeature*, a Configuração de Características do produto *Product#2* pode ser modificada em tempo de execução. Para adicionar outro sensor médico, como, por exemplo, um

glucômetro, a característica dinâmica *Glucometer* deve ser selecionada pelo usuário, acarretando na sua adição à associação *selectedFeature* e à associação *userFeature* do produto. A inserção de *Glucometer* em *userFeature* faz com que ela não fique sujeita a mudanças de contexto detectadas por perfis que estejam associados aos serviços do Contrato Dinâmico. Caso o usuário não queria a característica configurada no produto, esta deve ser removida da associação *selectedFeature* e, neste caso, mantida em *userFeature* para evitar que seja selecionada novamente pelos perfis.

Para que quaisquer modificações sejam feitas, no entanto, as relações OR e XOR e as restrições de dependência devem ser verificadas. Para adicionar ou remover uma das características dinâmicas da Configuração de Características de Product#2, por exemplo, as relações e as restrições definidas nos diagramas das Figuras 6.3 e 6.4 devem ser checadas antes que a modificação seja efetivada.

6.5.2 *Como a arquitetura é adaptada quando o usuário modifica a configuração de características?*

A Figura 6.30 representa a Arquitetura do Produto *Product#2* derivada a partir da Configuração de Características já definida na Figura 6.29. As características *BloodPressure* e *Glucometer* são dinâmicas, no entanto apenas *BloodPressure* foi selecionada em tempo de derivação (associação *selectedFeature*) e configurada junto à arquitetura de *Product#2*. Essa configuração foi realizada pelo serviço *BloodPressureService* do Contrato *MedicalSensorGroupContract*, já apresentado na Figura 6.14.

Se o usuário seleciona em tempo de execução a característica *Glucometer*, o Contrato *MedicalSensorGroupContract* é iniciado e o serviço *GlucometerService* adapta a Arquitetura do Produto estabelecendo ligações entre os componentes *Glucometer* e *MedicalSensorEvent*, como representado na Figura 6.31. Posteriormente, caso o usuário não mais queira a característica *BloodPressure* configurada na arquitetura, ele pode removê-la da Configuração de Características. Com isso, o serviço *BloodPressureService* tem sua execução interrompida, desfazendo as ligações estabelecidas entre os componentes *BloodPressure* e *MedicalSensorEvent*.

Em outro exemplo, a Figura 6.32 traz uma extensão à Configuração de Características do mesmo produto *Product#2*, envolvendo dessa vez os dispositivos *TV* e *Tablet*. No diagrama, a característica alternativa *TV* está selecionada (associação *selectedFeature*), fazendo com que a Arquitetura do Produto tenha o componente *TV* em sua configuração.

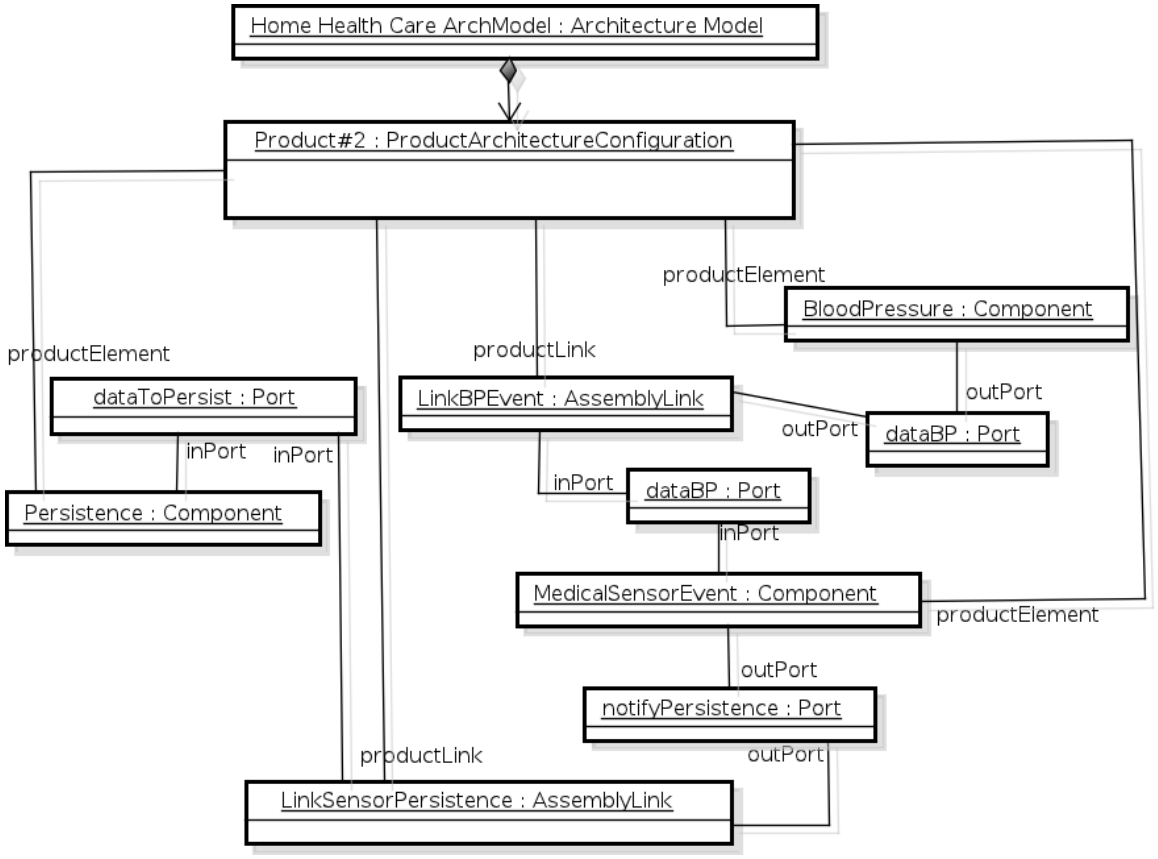


Figura 6.30: Arquitetura do Produto *Product#2* configurada apenas com *BloodPressure*.

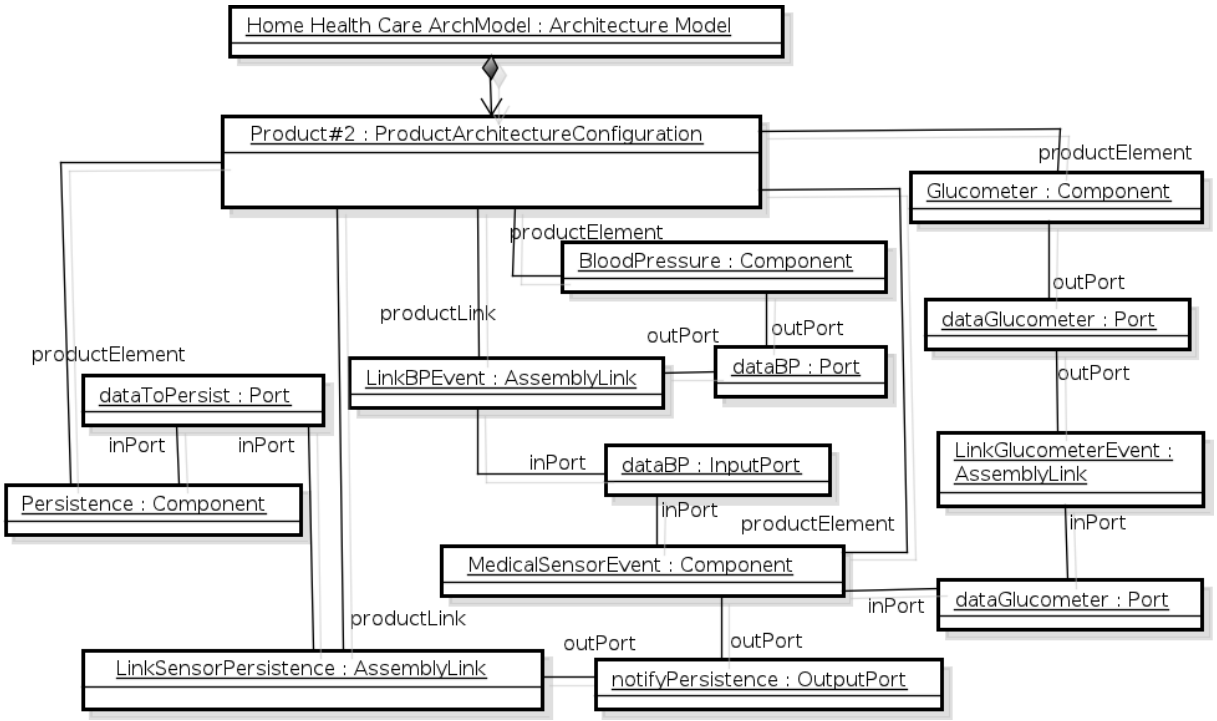


Figura 6.31: Arquitetura do Produto *Product#2* configurada com *BloodPressure* e *Glucometer*.

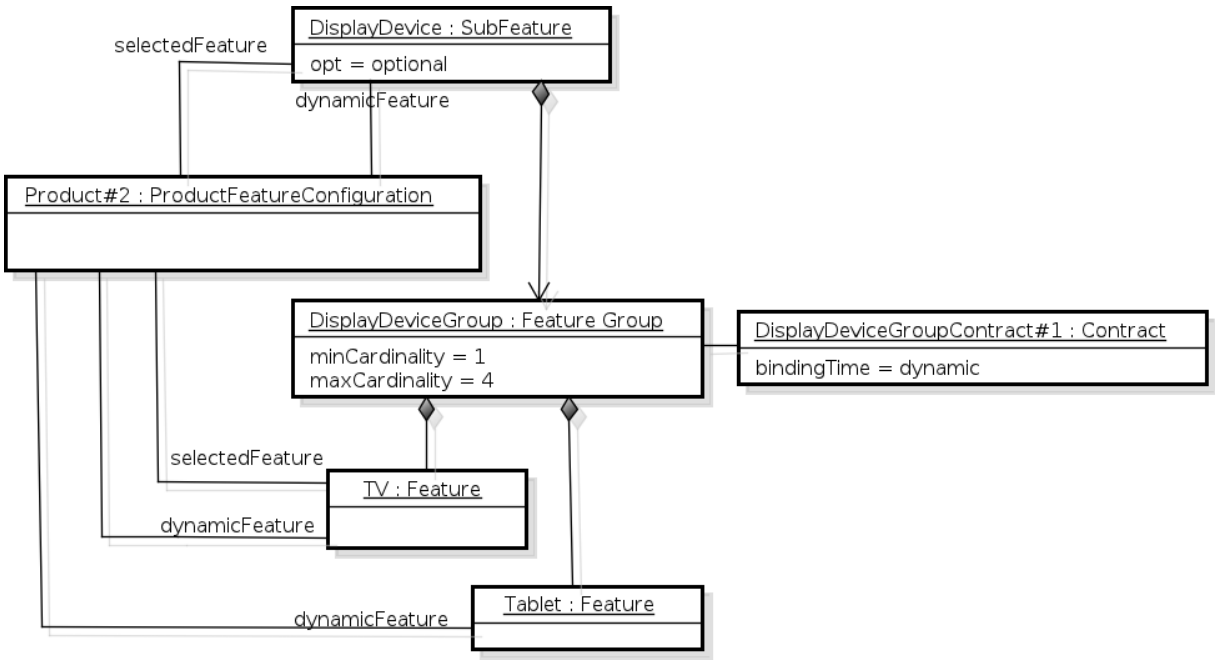


Figura 6.32: Configuração de Características do produto *Product#2* com *TV* selecionada.

A arquitetura de *Product#2*, mostrada na Figura 6.33 foi derivada a partir do Contrato *DisplayDeviceGroupContract#1*, já definido na Figura 6.16, o qual interliga um componente *TV* ao componente *ReminderEvent*. Com isso, a *TV* pode receber notificações de lembrete ao paciente. O Contrato em questão é dinâmico e está associado ao grupo de características *DisplayDeviceGroup*, permitindo portanto que a seleção de outras características alternativas do grupo (por exemplo, *Tablet*) modifique a Arquitetura do Produto em tempo de execução.

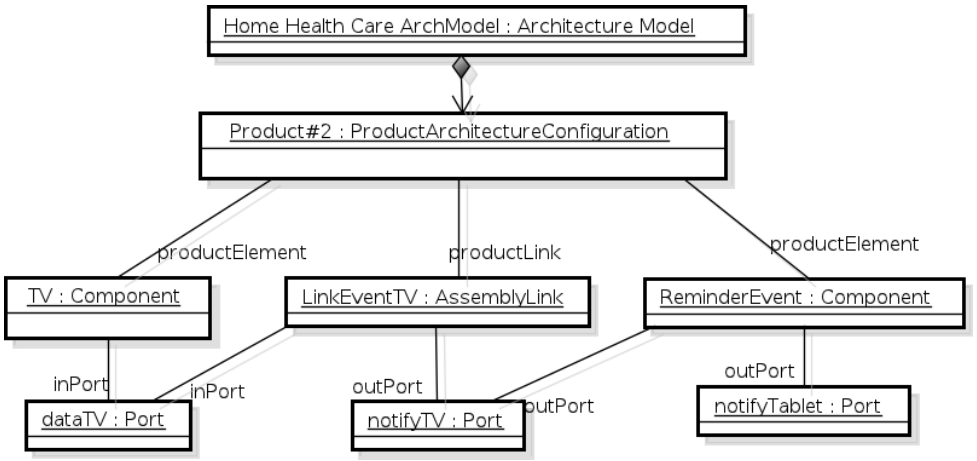


Figura 6.33: Arquitetura do Produto *Product#2* configurada com *TV*.

6.5.3 *Como a arquitetura é adaptada em decorrência de uma mudança do contexto?*

O diagrama da Figura 6.24, já apresentada na Subseção 6.3.2, mostra uma parte da Arquitetura do Produto *Product#1* envolvendo o grupo de características *CommunicationGroup*. A arquitetura está configurada com um conector *ADSL* interligando o componente *Alarm* e o componente *SMF*. Essa configuração foi estabelecida pelo serviço *ADSLService* do Contrato *CommunicationGroupContract*, definido anteriormente no diagrama da Figura 6.13.

O serviço *ADSLService* possui uma associação com o perfil *ADSLProfile* que por sua vez está associado à regra de contexto *AvailabilityRule* (veja a Figura 6.20). De acordo com a regra, o serviço somente pode continuar em execução se a comunicação ADSL estiver disponível para o uso, ou em outros termos, estiver operacional. Portanto, uma falha na comunicação ADSL durante a execução do produto *Product#1*, torna o perfil *ADSLProfile* inválido e provoca a interrupção do serviço *ADSLService*. Nesse caso, o mecanismo de negociação deve selecionar o serviço *GSMService*, o qual, uma vez ativo, realiza adaptações na Arquitetura do Produto a fim de configurar o conector GSM. A arquitetura resultante desse processo está na Figura 6.34.

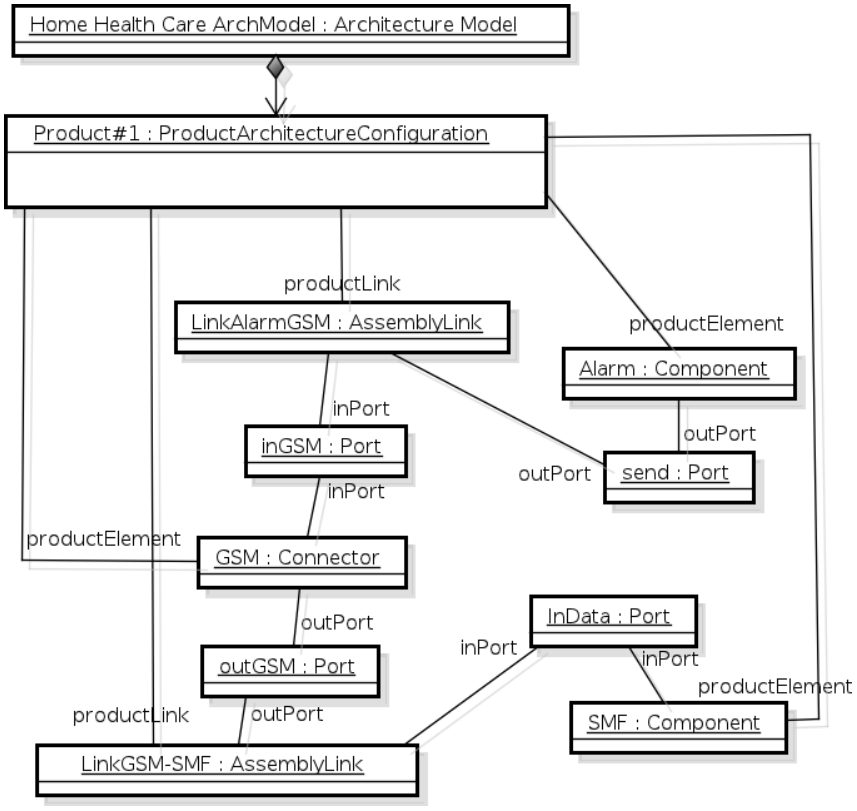


Figura 6.34: Arquitetura do Produto *Product#1* configurada com o conector *GSM*.

Outro Contrato Dinâmico denominado *DisplayDeviceGroupContract#2* (Figura 6.21), também lida com a mudança do contexto. Um modelo contextual e uma regra de contexto denominada *NearestDeviceRule* definem que as notificações ao paciente sejam enviadas para o dispositivo (TVs ou *tablets*) que esteja no mesmo ambiente do paciente, ou em outros termos, para aquele mais próximo dele. O Contrato está modelado para que faça a descoberta do dispositivo e o configure junto à arquitetura de acordo com a sua localização na casa.

Caso o dispositivo mais próximo ao paciente seja um *tablet*, o serviço *DisplayDeviceService* adapta a Arquitetura do Produto (já mostrada na Figura 6.33) estabelecendo uma configuração onde o componente *ReminderEvent* é interligado ao componente *Tablet*, para que este possa receber as notificações. Por outro lado, se uma TV estiver mais próxima do paciente, o serviço estabelece uma configuração onde o componente *TV* é interligado ao componente *ReminderEvent*. Mesmo depois que uma das configurações esteja estabelecida, o serviço pode ser interrompido caso nenhum dos dispositivos continue próximo ao paciente, ou seja, o perfil *NearestDeviceProfile* se torne inválido. Nessa situação, outros Contratos implantados junto com a arquitetura podem manter alguma configuração estabelecida, como, por exemplo, o Contrato *DisplayDeviceGroupContract#1*, mostrado anteriormente na Figura 6.16.

O mecanismo de descoberta dos dispositivos, utilizado pelo serviço *DisplayDeviceService*, depende da modelagem do Domínio de Recursos (veja Seção 5.2.3.5), representado pela classe *ResourceDomain*, e da existência de um Serviço de Descoberta de Recursos (veja Seção 4.2.2). Essa estrutura torna possível adicionar e remover componentes da arquitetura em tempo de execução.

6.5.4 *Como a configuração de características é modificada em decorrência de uma adaptação da arquitetura?*

A parte da Arquitetura do Produto *Product#1* envolvendo as tecnologias de comunicação, pode ser adaptada de acordo com o contexto, como tratado na Subseção 6.5.3. A adaptação da arquitetura implica na modificação da Configuração de Características do produto.

O Contrato *CommunicationGroupContract*, já definido no diagrama da Figura 6.13, guia a adaptação arquitetural por meio de seus serviços *ADSLService* e *GSMService*. O serviço *ADSLService* possui, assim como *GSMService*, uma associação com as características *ADSL* e *GSM*, respectivamente.

Caso o serviço *ADSLService* seja desativado pelo perfil *ADSLProfile*, como descrito no exemplo da Subseção 6.5.3, a característica *ADSL* não mais deve fazer parte da Configuração de Características de *Product#1*, ou seja, deve ser removida de *ProductFeatureConfiguration* por meio de sua associação *selectedFeature*. A desativação de *ADSLService* pode ainda ter como consequência a ativação do serviço *GSMService*, implicando na seleção da característica *GSM* junto à Configuração de Características do produto. Isso é feito adicionando-se a característica *GSM* à associação *selectedFeature*.

6.6 Resumo dos Resultados

A técnica de formulação de questões de competência permitiu avaliar o Metamodelo quanto aos seus objetivos, quais sejam: criar os modelos da LPS, derivar produtos, associar características a elementos arquiteturais e realizar a adaptação dinâmica da Arquitetura do Produto.

Em relação ao primeiro objetivo, modelos foram instanciados para representar o Modelo de Características (*e.g.*, Figura 6.1), Modelo de Arquitetura (*e.g.*, Figura 6.7, Figura 6.8) e Modelo de Configuração, este composto de Contratos (*e.g.*, Figura 6.11), Perfis e Regras (*e.g.*, Figura 6.17, Figura 6.20), Modelo de Contexto (*e.g.*, Figura 6.21) e Negociação.

Como o cenário utilizado se refere às características do SCIADS, os modelos foram criados envolvendo o mesmo conjunto de dados. Ao visualizar a interação dos objetos, percebe-se que o Modelo de Características e o Modelo de Arquitetura não contêm ou conhecem quaisquer informações de configuração (*e.g.*, regras, perfis), as quais são mantidas apenas pelo Modelo de Configuração. Esta situação ratifica a propriedade de independência deste modelo como preconizada na proposta do Metamodelo.

Para tornar mais simples a apresentação dos modelos, estes foram criados e apresentados de forma sucessiva, no entanto, podem também ser criados de forma iterativa e integrada. Em vez de criar o Modelo de Características como um todo e de uma única vez, pode-se, por exemplo: *(i)* definir a característica opcional *PanicButton* (Figura 6.2); *(ii)* definir o componente *PanicButton* e associá-lo à variabilidade *PanicButtonOptional* (Figura 6.10); *(iii)* definir o Contrato *PanicButtonContract* (Figura 6.11) e associá-lo à mesma variabilidade; e, por fim, *(iv)* associar a característica ao Contrato (Figura 6.5).

A derivação de produtos, segundo objetivo do Metamodelo, trata da geração da Configuração de Características (Figura 6.22) e, na sequência, da geração da Arquitetura

do Produto (Figura 6.23). Da mesma forma que ocorre com os modelos da LPS, estes modelos podem ser gerados de forma iterativa, por meio da resolução das variabilidades representadas no Modelo de Características.

A Configuração de Características e a Arquitetura do Produto representam o produto em execução. Para que sejam estimulados e se adaptem de forma sincronizada, deve-se ter uma associação definida entre características e arquitetura (Figura 6.25), o que representa o terceiro objetivo do Metamodelo.

Por fim, uma adaptação ocorre quando a Configuração de Características é modificada pelo usuário (Seção 6.5.1), causando a adaptação da Arquitetura do Produto (Seção 6.5.2). Em outra situação, a Arquitetura do Produto pode ser adaptada em decorrência da mudança do contexto (Seção 6.5.3), o que pode gerar, ainda, a modificação da Configuração de Características (Seção 6.5.4).

6.7 Considerações Finais

Este capítulo apresentou a avaliação do Metamodelo, incluindo diversos cenários de uso contendo características do SCIADS, desenvolvidos na forma de Diagramas de Objetos da UML. Além de possibilitar a avaliação, estes diagramas auxiliam na própria compreensão do funcionamento do Metamodelo, pois permitem detalhar cada interação entre os objetos representados.

A avaliação apresentada com todos os seus modelos e explicações textuais permite concluir que o Metamodelo suporta as tarefas e questões a ele submetidas, e, portanto, apresenta competência frente aos seus objetivos. Além disso, a avaliação contribui com o objetivo desta tese, visto que a integração entre características e elementos arquiteturais, a independência do Modelo de Configuração e os objetivos dos Contratos, podem ser determinados pelos modelos e cenários apresentados.

O próximo capítulo apresenta as conclusões desta tese.

Capítulo 7

Conclusões

Este capítulo apresenta as considerações finais e as principais contribuições desta tese, assim como limitações e trabalhos futuros.

7.1 Considerações Finais

Esta pesquisa teve o seu início quando algumas das ideias relacionadas ao SCIADS já vinham sendo implementadas na forma de protótipos e experimentos [23, 122]. A noção de personalização trouxe para este contexto perspectivas relacionadas ao uso das técnicas de LPS, como discutido ao longo desta tese. Uma das primeiras ações, neste sentido, foi o desenvolvimento do Modelo de Características do SCIADS e o levantamento das suas variabilidades. Ao mesmo tempo, Contratos foram estudados e desenvolvidos para descrever as variabilidades da Arquitetura da LPS. Diante dos resultados obtidos, a pesquisa foi conduzida rumo à definição de metamodelos, com o objetivo de representar uma forma com a qual os Contratos pudessem lidar com a adaptação de arquiteturas no contexto de LPS e aplicações ubíquas. O Metamodelo de Configuração de Variabilidades em LPS foi então desenvolvido, considerando não só o aspecto arquitetural como também o de características, relevante para que a personalização possa ser feita em um nível mais alto de abstração.

A trajetória desta pesquisa foi, em sua maior parte, orientada pelos dois *Pontos* caracterizados na Seção 1.2 do capítulo introdutório desta tese:

1. *As abordagens estudadas não permitem descrever, de uma maneira integrada, variabilidades estáticas e dinâmicas no nível da Arquitetura da*

LPS.

2. *As abordagens que suportam variabilidades dinâmicas não representam um modelo para o CK que seja explícito e independente dos modelos da LPS (características e arquitetura), e que tenha condições de integrar características, arquitetura, variabilidades estáticas e dinâmicas, e os processos de derivação e adaptação dinâmica de arquiteturas de produtos.*

Ambos os pontos foram apresentados e tratados ao longo do texto da tese. O primeiro foi abordado nas seções representadas na Figura 7.1, enquanto o segundo foi abordado nas seções representadas na Figura 7.2.

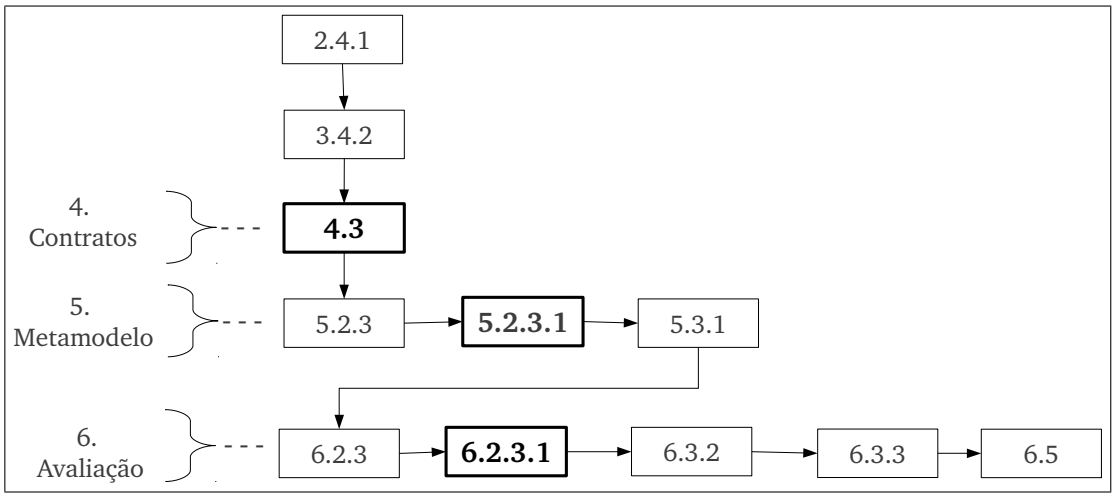


Figura 7.1: Seções da tese que desenvolvem e tratam do *Ponto #1.*

A Figura 7.1 apresenta um destaque para as Seções *4.3*, *5.2.3.1* e *6.2.3.1* do texto, pois são as mais relevantes para o *Ponto #1.* A Seção *4.3* apresenta como Contratos podem descrever variabilidades no nível da Arquitetura da LPS, em especial, variabilidades dinâmicas. A Seção *5.2.3.1*, por sua vez, apresenta como Contratos Estáticos e Contratos Dinâmicos são descritos para representar, respectivamente, variabilidades estáticas e dinâmicas da Arquitetura da LPS. Por fim, a Seção *6.2.3.1* apresenta diversos cenários envolvendo Contratos Estáticos e Dinâmicos definidos na Arquitetura da LPS.

A Figura 7.2 apresenta as seções em que o *Ponto #2* é abordado no texto, separadas em 05 (cinco) grupos: características, arquitetura, modelo para o CK, derivação e adaptação dinâmica.

Essa organização do texto permite concluir que os objetivos específicos da tese, definidos na Seção 1.3 do capítulo introdutório, foram atingidos, assim como o objetivo geral. Os

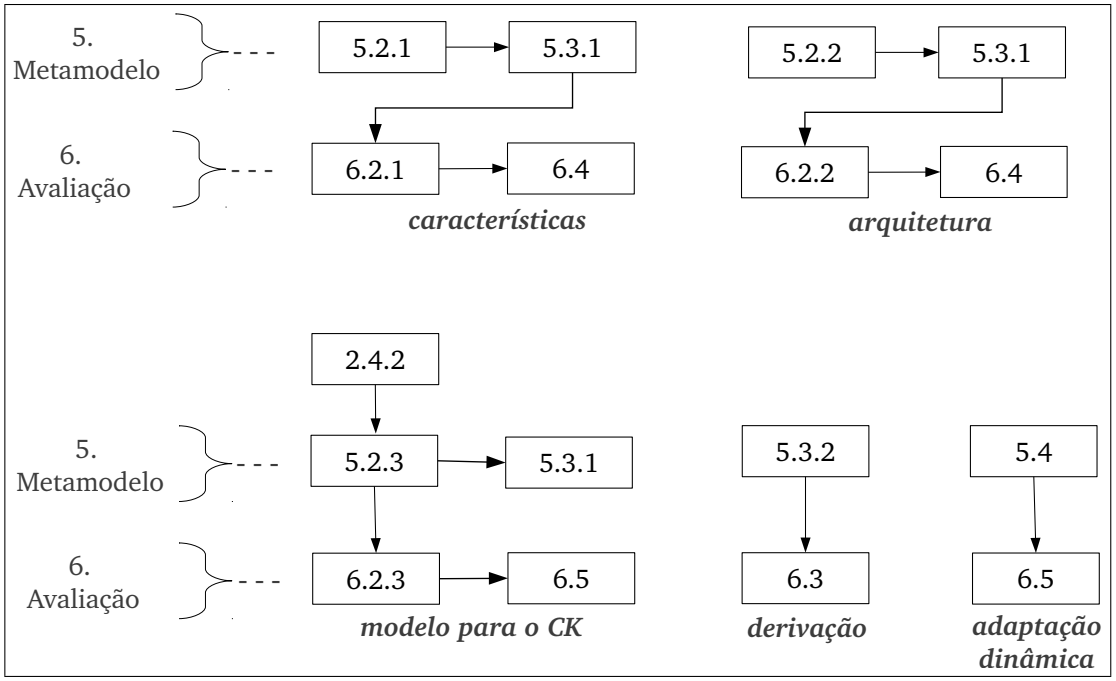


Figura 7.2: Seções da tese que desenvolvem e tratam do *Ponto #2*.

objetivos específicos são listados abaixo, acompanhados das respectivas seções do texto onde são abordados.

1. Mostrar como variabilidades podem ser descritas na forma de Contratos no nível da Arquitetura da LPS – Todas as seções do texto identificadas na Figura 7.1.

2. Definir um metamodelo de características e um metamodelo de Arquiteturas de LPS – Seções *5.2.1* e *5.2.2*, identificadas na Figura 7.2.

3. Definir um metamodelo de configuração integrando características, Arquitetura da LPS e Contratos – Seção *5.2.3* identificada na Figura 7.2.

4. Definir diretrizes para a criação (instanciação) dos modelos da LPS, e para a derivação e adaptação dinâmica de Arquiteturas de Produtos – Seções *5.3.1* e *5.3.2*, identificadas na Figura 7.2.

5. Avaliar o metamodelo por meio de questões de competência – Todas as seções do Capítulo 6, identificadas na Figura 7.2.

7.2 Contribuições

As principais contribuições desta tese são:

- O Metamodelo de Configuração de Variabilidades em LPS, concebido com o propósito de modelar LPSs voltadas para sistemas cuja arquitetura seja adaptável diante de mudanças de contexto e mudanças dos requisitos dos usuários, tais como as aplicações ubíquas. Desenvolvido usando o meta-metamodelo Ecore e o conjunto de ferramentas do EMF, o Metamodelo inclui diretrizes para a instanciação dos modelos da LPS (Características, Arquitetura e Configuração) e para a derivação e adaptação de Arquiteturas de Produtos. Com o Metamodelo pode-se criar os modelos da LPS de uma forma integrada, sendo que no Modelo de Configuração podem ser definidos Contratos para descrever as variabilidades estáticas e dinâmicas na forma de ações de adaptação realizadas nas Arquiteturas dos Produtos, seja para derivá-las, seja para reconfigurá-las em tempo de execução.

- A ampliação do escopo de utilização dos Contratos, permitindo que sejam aplicados na modelagem de Arquiteturas de LPS com o propósito de: *(i)* descrever as variabilidades estáticas e dinâmicas na forma de ações de adaptação arquitetural, e as condições sob as quais tais adaptações devem ser realizadas; *(ii)* associar características a elementos da arquitetura; *(iii)* guiar a derivação e a adaptação dinâmica das Arquiteturas dos Produtos.

- Os resultados em relação à descrição de variabilidades na forma de Contratos no nível da Arquitetura da LPS, onde Contratos são desenvolvidos para descrever variabilidades dinâmicas do SCIADS.

- O Modelo de Características do SCIADS, desenvolvido com base na experiência adquirida no desenvolvimento do protótipo e no conhecimento obtido junto a profissionais da área de saúde, e uma descrição das variabilidades do sistema, com destaque para as questões de personalização e de adaptação dinâmica.

- Os resultados da avaliação do Metamodelo, onde questões de competência são apresentadas juntamente com Diagramas de Objetos da UML, usados para formalizar a execução dos modelos instanciados a partir do Metamodelo. A instanciação é realizada envolvendo dados de cenários do SCIADS.

Em termos de artigos e relatórios, foram produzidos, como mencionado no Capítulo 1, 12 (doze) trabalhos ao longo do desenvolvimento da pesquisa, sendo 10 (dez) artigos [68, 28, 21, 20, 26, 25, 23, 122, 113, 19] e 02 (dois) relatórios técnicos [27, 22]. Dois artigos [20, 26] foram indicados para o prêmio de *Melhor Artigo* de suas respectivas conferências.

7.3 Limitações

O trabalho desenvolvido nesta tese possui algumas limitações.

Com relação aos Contratos, uma limitação que pode ser atribuída a este trabalho é o fato de não ter sido realizada a implementação dos Contratos mostrados no Capítulo 4. A opção por desenvolvê-los de forma mais conceitual, entretanto, foi feita para que se pudesse retratar em um nível mais alto de abstração as questões discutidas, e, por conseguinte, alcançar os objetivos da tese. De qualquer forma, o desenvolvimento foi feito seguindo-se as particularidades da estruturação e das construções da linguagem de Contratos. Isso foi possível devido à forma detalhada com a qual os aspectos de implementação dos Contratos são tratados nos trabalhos anteriores do grupo de pesquisa, inclusive em relação à infraestrutura de suporte e execução, apresentada no Capítulo 4 desta tese.

No contexto do Metamodelo, a definição de regras de composição e regras de contexto expressa no diagrama, não inclui a maneira como elas são formadas. Por exemplo, classes poderiam ser definidas para representar as expressões consequentes e antecedentes das regras, e da mesma forma para cada um dos possíveis conectores lógicos (*e.g.*, NOT, AND, OR). Para as regras de contexto também poderiam haver classes para representar ações e condições. Isso pode ser apontado como uma limitação, no entanto, foi uma opção escolhida ao se considerar que um detalhamento neste nível contribui pouco em direção ao objetivo geral do trabalho e, ao mesmo tempo, traz um aumento significativo na complexidade, o que pode dificultar a compreensão do todo. Há outras partes do Metamodelo em que esta ponderação também foi feita, por exemplo, aquelas relacionadas à negociação, ao domínio e diretório de recursos, e ao modelo de contexto, este último formado por apenas duas classes essenciais à representação do contexto da aplicação.

Por fim, pode-se salientar como limitação do trabalho o fato da avaliação não incluir as diretrizes relacionadas ao Metamodelo. As diretrizes, porém, foram idealizadas para servirem unicamente como um guia voltado ao entendimento de como usar o Metamodelo. Outros trabalhos podem ser desenvolvidos para estender os seus propósitos, incluindo o detalhamento das ações e a sua avaliação.

7.4 Trabalhos Futuros

O trabalho desenvolvido nesta tese, incluindo a avaliação e os resultados obtidos, possibilitou identificar trabalhos que podem ser realizados visando a continuidade da pesquisa. As principais perspectivas em termos de trabalhos futuros são apresentadas a seguir, contemplando possibilidades em quatro dimensões: ampliação dos objetivos, aperfeiçoamento da proposta, aprimoramento das avaliações e aplicação da proposta.

Em termos de ampliação dos objetivos desta tese, uma perspectiva é a integração ao Metamodelo proposto de uma abordagem para a descrição de variabilidades relacionadas a atributos de qualidade em LPS. Ao derivar um produto, a seleção de características pode afetar o nível de atributo de qualidade que se deseja para o produto, uma vez que diferentes produtos de uma LPS podem ter diferentes atributos de qualidade. Por exemplo, determinadas características podem aumentar o desempenho de um produto, mas diminuir a sua confiabilidade, enquanto outras podem influenciar a qualidade do produto de maneira oposta. Uma extensão ao Metamodelo proposto pode ser a incorporação de mecanismos e informações ao Modelo de Configuração que permitam a escolha, durante a derivação, de Contratos especificamente construídos para lidar com determinados níveis de qualidade. Tais Contratos poderiam adaptar a Arquitetura do Produto como um todo em diferentes e variados pontos visando obter os níveis de qualidade requeridos.

Em trabalhos futuros podem ser investigadas, ainda, formas de integrar ao Metamodelo os elementos e as técnicas de Gerência de Configuração para gerenciar as configurações arquiteturais dos produtos geradas em tempo de execução. Com essas técnicas modeladas junto ao Metamodelo, se poderia, entre outros recursos, criar o versionamento das configurações, identificar as diferenças entre as versões, manter um histórico de adaptações, a partir do qual se poderia avançar e retroceder entre as configurações. Além disso, a incorporação destes elementos ao Metamodelo traria a possibilidade de criar os modelos da LPS já associados a um modelo de Gerência de Configuração capaz de apoiar o gerenciamento das Arquiteturas dos Produtos em tempo de execução.

Outra perspectiva está relacionada aos Contratos, os quais possuem, como visto no Capítulo 4, uma estruturação que facilita a verificação da arquitetura de aplicações. Um trabalho futuro diz respeito à investigação no sentido de estender o Metamodelo para que sejam incorporados elementos, regras e restrições que permitam realizar verificações das Arquiteturas dos Produtos da LPS em termos de propriedades de *safety*, as quais vêm sendo bastante investigadas no contexto de sistemas críticos como os sistemas de

assistência à saúde. Esta extensão poderia ser investigada no contexto do próprio Contrato e seus elementos, e ainda do Modelo de Configuração como um todo.

Em relação ao aperfeiçoamento da proposta desta tese, há trabalhos futuros que podem ser realizados a fim de levar melhorias ao próprio Metamodelo. Uma possibilidade está na melhor formalização das restrições, o que pode ser feito empregando-se a linguagem OCL (*Object Constraint Language*), a qual permite expressar, de forma declarativa, as restrições do modelo. Por exemplo, pode ser especificada em OCL a restrição de que um serviço de adaptação deve ser ativado apenas quando o seu perfil associado estiver válido. Outros trabalhos podem ainda refinar alguns dos elementos do Metamodelo, por exemplo, apresentando mais detalhes em termos da formação das regras de composição e de contexto, e da modelagem do contexto e da negociação.

Quanto às avaliações realizadas nesta tese, trabalhos adicionais poderiam ser realizados onde as questões de competência desenvolvidas fossem aplicadas em outros cenários além do próprio SCIADS. Isso reforçaria a relevância dos resultados obtidos. Outra possibilidade refere-se à formulação de novas questões por parte de profissionais que atuam na modelagem de LPS. Estas questões poderiam então ser aplicadas ao Metamodelo no contexto do SCIADS ou de outros cenários que envolvam variabilidades estáticas e dinâmicas. Em outra opção, os participantes também poderiam utilizar o Metamodelo para criar os modelos de uma LPS. Uma ferramenta de apoio seria desenvolvida com o propósito de permitir a instanciação dos modelos a partir do Metamodelo, ou mesmo o editor do EMF poderia ser utilizado. Para visualizar a adaptação dinâmica em execução, um simulador pode ser desenvolvido para manipular os modelos e apresentar as configurações da Arquitetura do Produto em tempo de execução.

Ao finalizar, vale salientar que a proposta desta tese pode ser empregada no contexto de outros trabalhos, o que pode contribuir para que muitos dos trabalhos futuros anteriormente discutidos possam ser realizados. Em termos de aplicações, um exemplo é o projeto *SmartAndroid* [1], uma aplicação ubíqua voltada para a concepção de *smart homes*. Neste projeto, TVs, *tablets*, celulares, eletrodomésticos e outros artefatos são representados por dispositivos *Android*[2]. Os dispositivos se comunicam utilizando, principalmente, um módulo de *software* denominado Agente de Recurso, o qual mantém informações de contexto relativas ao dispositivo (*e.g.*, localização). Há ainda um *middleware* formado por um Serviço de Descoberta e Registro de Recursos e um Serviço de Contexto, o qual gerencia as informações de contexto do ambiente, definidas na forma de entidades, elementos e

[1] *http://www.tempo.uff.br/smartandroid*
[2] *http://www.google.com/android*

regras de contexto [89, 57, 53].

Nesta aplicação, o Metamodelo proposto pode auxiliar na descrição, em um nível mais alto de abstração, da Configuração de Características de cada uma das *smart homes* (*e.g.*, TV, celular, fogão, cama, luminosidade, presença, porta), assim como de suas arquiteturas. Modelos de Configuração podem facilitar a definição de diferentes subsistemas, um requisito do projeto, como, por exemplo, Segurança (*e.g.*, portas, janelas, incêndio, alarmes) e Saúde (*e.g.*, SCIADS). Domínios de Recursos podem lidar com a descoberta de recursos e Contratos guiar a interligação dos elementos arquiteturais conforme as regras de contexto definidas. Com essa estruturação, o Metamodelo de Configuração de Variabilidades em LPS, proposto nesta tese, vai ao encontro dos princípios do projeto, pois favorece a modelagem de diferentes subsistemas em cada *smart home* e a adaptação dinâmica da aplicação às necessidades do usuário final.

APÊNDICE A – Notação BNF de um Contrato

Este apêndice apresenta parte da notação BNF (Backus-Naur Form) da linguagem CBabel referente à descrição de Contratos. Um ou mais serviços e uma cláusula de negociação compõem um Contrato (*SERVICE* e *NEGOTIATION*, linha 02). Um serviço tem na sua estrutura as primitivas que realizam a adaptação da arquitetura (*ADAPTATION*, linha 07) e uma ou mais identificações de perfis (*PROFILENAMES*, linha 13). Os perfis (*PROFILE*, linha 26) contêm em sua estrutura uma ou mais regras construídas a partir da definição de categorias e propriedades (*RULE*, linha 28). Uma categoria (*CATEGORY*, linha 35) contém uma ou mais propriedades com seus tipos (*PROPERTYNAME*, linhas 37 e 38). A cláusula de negociação, por sua vez, define transições entre serviços (*NEGOTIATION*, linha 18).

Segue a sintaxe de um Contrato em notação BNF (adaptada de [42]).

```
1  CONTRACT       := contract { CONTRACTBODY } ID ;
2  CONTRACTBODY   := SERVICE NEGOTIATION
3  ID             := STRING
4
5  SERVICE        := service { SERVICEBODY } SERVICENAME ;
6  SERVICEBODY    := ADAPTATION
7  ADAPTATION     := ADLCOMMAND with PROFILENAMES ;
8  ADLCOMMAND     := INSTANTIATE | REMOVE | LINK
9  INSTANTIATE    := instantiate INSTANCENAME
10 REMOVE         := remove INSTANCENAME
11 LINK           := link INSTANCENAME to INSTANCENAME
12 INSTANCENAME   := ID
13 PROFILENAMES   := PROFILENAMES , PROFILENAME | PROFILENAME
14 PROFILENAME    := ID
15 SERVICENAME    := ID
16 ID             := STRING
17
18 NEGOTIATION    := negotiation { TRANSITION }
```

```
19 TRANSITION      := not SERVICENAME -> SERVICES ; |
20                    SERVICENAME -> SERVICES
21 SERVICES        := SERVICENAME | ( SERVICESNAMES )
22 SERVICESNAMES   := SERVICENAMES , SERVICENAME |
23                    SERVICENAME
24 SERVICENAME     := STRING
25
26 PROFILE         := profile ( PROFILEBODY } ID ;
27 PROFILEBODY     := RULE
28 RULE            := CATEGORYNAME.PROPERTYNAME OPERATOR VALUE ;
29 CATEGORYNAME    := ID
30 PROPERTYNAME    := ID
31 OPERATOR        := < | > | = | == | <= | >=
32 VALUE           := STRING | NUMBER
33 ID              := STRING
34
35 CATEGORY        := category ID { CATEGORYBODY } ;
36 CATEGORYBODY    := STATEMENT
37 STATEMENT       := PROPERTYNAME : TYPE UNIT GUIDANCE ; |
38                    PROPERTYNAME : TYPE GUIDANCE ;
39 PROPERTYNAME    := ID
40 TYPE            := numeric | ENUM
41 UNIT            := ID
42 GUIDANCE        := in | out
43 ENUM            := enum { ENUMERATIONS }
44 ENUMERATIONS    := ENUMERATIONS , ID | ID
45 ID              := STRING
```

APÊNDICE B - Arquivo XMI Ecore do Metamodelo

Este apêndice traz o arquivo XMI Ecore do Metamodelo de Configuração de Variabilidades em LPS. O arquivo foi exportado a partir do próprio Eclipse. A partir deste arquivo pode-se gerar código-fonte em Java das classes do Metamodelo e instanciar modelos.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ecore:EPackage xmi:version="2.0"
3      xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
           instance"
4      xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="sa"
5      nsURI="sa" nsPrefix="sa">
6    <eClassifiers xsi:type="ecore:EClass" name="ModelElement" abstract="true"/>
7    <eClassifiers xsi:type="ecore:EClass" name="SoftwareArchitecture" eSuperTypes="#//
         ModelElement">
8      <eStructuralFeatures xsi:type="ecore:EReference" name="mandatoryElement" lowerBound="
           1"
9          upperBound="-1" eType="#//ArchitectureElement" eOpposite="#//ArchitectureElement/
               arch"/>
10     <eStructuralFeatures xsi:type="ecore:EReference" name="configurationModel" eType="#//
           ConfigurationModel"/>
11     <eStructuralFeatures xsi:type="ecore:EReference" name="mandatoryLink" upperBound="-1"
12         eType="#//Link" containment="true"/>
13   </eClassifiers>
14   <eClassifiers xsi:type="ecore:EClass" name="ElementType" abstract="true" eSuperTypes="
         #//ArchitectureElementType #//ContextualEntity">
15     <eStructuralFeatures xsi:type="ecore:EReference" name="inputPortType" upperBound="-1"
16         eType="#//PortType"/>
17     <eStructuralFeatures xsi:type="ecore:EReference" name="outputPortType" upperBound="-1
           "
18         eType="#//PortType"/>
19   </eClassifiers>
20   <eClassifiers xsi:type="ecore:EClass" name="Link" abstract="true">
21     <eStructuralFeatures xsi:type="ecore:EReference" name="profiles" upperBound="-1"
22         eType="#//Profile" eOpposite="#//Profile/constrainsLink"/>
23     <eStructuralFeatures xsi:type="ecore:EReference" name="services" eType="#//
           AdaptationService"
24         eOpposite="#//AdaptationService/selectedLink"/>
25   </eClassifiers>
26   <eClassifiers xsi:type="ecore:EClass" name="ComponentType" eSuperTypes="#//ElementType"
       >
```

```
27      <eStructuralFeatures xsi:type="ecore:EReference" name="internalArchitecture" eType="
            #//SoftwareArchitecture"/>
28    </eClassifiers>
29    <eClassifiers xsi:type="ecore:EClass" name="ConnectorType" eSuperTypes="#//ElementType"
          />
30    <eClassifiers xsi:type="ecore:EClass" name="PortType" eSuperTypes="#//
          ArchitectureElementType">
31      <eStructuralFeatures xsi:type="ecore:EReference" name="elementTypes" lowerBound="1"
32          upperBound="-1" eType="#//ElementType"/>
33    </eClassifiers>
34    <eClassifiers xsi:type="ecore:EClass" name="Variability" abstract="true">
35      <eStructuralFeatures xsi:type="ecore:EReference" name="spla" lowerBound="1" eType="
            #//SPLArchitecture"
36          eOpposite="#//SPLArchitecture/variabilities"/>
37      <eStructuralFeatures xsi:type="ecore:EReference" name="contract" upperBound="-1"
38          eType="#//Contract" eOpposite="#//Contract/variability"/>
39    </eClassifiers>
40    <eClassifiers xsi:type="ecore:EClass" name="Optionality" eSuperTypes="#//Variability">
41      <eStructuralFeatures xsi:type="ecore:EReference" name="optional" lowerBound="1"
42          upperBound="-1" eType="#//Element" eOpposite="#//Element/optionality"/>
43    </eClassifiers>
44    <eClassifiers xsi:type="ecore:EClass" name="VariationPoint" eSuperTypes="#//Variability
          ">
45      <eStructuralFeatures xsi:type="ecore:EReference" name="variant" lowerBound="2"
46          upperBound="-1" eType="#//Element" eOpposite="#//Element/variationPoints"/>
47    </eClassifiers>
48    <eClassifiers xsi:type="ecore:EClass" name="SPLArchitecture" eSuperTypes="#//
          SoftwareArchitecture">
49      <eStructuralFeatures xsi:type="ecore:EReference" name="variabilities" lowerBound="1"
50          upperBound="-1" eType="#//Variability" containment="true" eOpposite="#//
              Variability/spla"/>
51    </eClassifiers>
52    <eClassifiers xsi:type="ecore:EClass" name="Contract" eSuperTypes="#//CKElement">
53      <eStructuralFeatures xsi:type="ecore:EReference" name="variability" upperBound="-1"
54          eType="#//Variability" eOpposite="#//Variability/contract"/>
55      <eStructuralFeatures xsi:type="ecore:EReference" name="negotiation" eType="#//
            Negotiation"
56          containment="true" eOpposite="#//Negotiation/dynamicContract"/>
57      <eStructuralFeatures xsi:type="ecore:EReference" name="services" lowerBound="1"
58          upperBound="-1" eType="#//AdaptationService" containment="true" eOpposite="#//
              AdaptationService/contract"/>
59      <eStructuralFeatures xsi:type="ecore:EReference" name="featureGroup" eType="#//
            FeatureGroup"
60          eOpposite="#//FeatureGroup/contract"/>
61      <eStructuralFeatures xsi:type="ecore:EReference" name="optionalFeature" eType="#//
            SubFeature"
62          eOpposite="#//SubFeature/contract"/>
63      <eStructuralFeatures xsi:type="ecore:EAttribute" name="bindingTime" eType="#//
            BindingTimeType"/>
64    </eClassifiers>
65    <eClassifiers xsi:type="ecore:EClass" name="Profile" eSuperTypes="#//CKElement">
66      <eStructuralFeatures xsi:type="ecore:EReference" name="constrainsLink" upperBound="-1
            "
67          eType="#//Link" eOpposite="#//Link/profiles"/>
```

```
68    <eStructuralFeatures xsi:type="ecore:EReference" name="constrainsService" upperBound=
          "-1"
69        eType="#//AdaptationService" eOpposite="#//AdaptationService/profiles"/>
70    <eStructuralFeatures xsi:type="ecore:EReference" name="rules" lowerBound="1"
          upperBound="-1"
71        eType="#//Rule"/>
72    <eStructuralFeatures xsi:type="ecore:EAttribute" name="isValidated" eType="
          ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean"/>
73  </eClassifiers>
74  <eClassifiers xsi:type="ecore:EClass" name="ContextualRule" eSuperTypes="#//Rule">
75    <eStructuralFeatures xsi:type="ecore:EReference" name="relevantCE" upperBound="-1"
76        eType="#//ContextualElement" eOpposite="#//ContextualElement/contextualRule"/>
77  </eClassifiers>
78  <eClassifiers xsi:type="ecore:EClass" name="Negotiation" abstract="true">
79    <eStructuralFeatures xsi:type="ecore:EReference" name="dynamicContract" lowerBound="1
          "
80        eType="#//Contract" eOpposite="#//Contract/negotiation"/>
81  </eClassifiers>
82  <eClassifiers xsi:type="ecore:EClass" name="AdaptationService">
83    <eStructuralFeatures xsi:type="ecore:EReference" name="contract" lowerBound="1"
84        eType="#//Contract" eOpposite="#//Contract/services"/>
85    <eStructuralFeatures xsi:type="ecore:EReference" name="selectedLink" lowerBound="1"
86        upperBound="-1" eType="#//Link" containment="true" eOpposite="#//Link/services"/>
87    <eStructuralFeatures xsi:type="ecore:EReference" name="profiles" upperBound="-1"
88        eType="#//Profile" eOpposite="#//Profile/constrainsService"/>
89    <eStructuralFeatures xsi:type="ecore:EReference" name="selectedElement" lowerBound="1
          "
90        upperBound="-1" eType="#//Element" eOpposite="#//Element/service"/>
91    <eStructuralFeatures xsi:type="ecore:EReference" name="alternativeFeature" eType="#//
          Feature"
92        eOpposite="#//Feature/service"/>
93    <eStructuralFeatures xsi:type="ecore:EAttribute" name="isActive" eType="
          ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean"/>
94  </eClassifiers>
95  <eClassifiers xsi:type="ecore:EClass" name="Element" abstract="true" eSuperTypes="#//
        ArchitectureElement">
96    <eStructuralFeatures xsi:type="ecore:EReference" name="optionality" upperBound="-1"
97        eType="#//Optionality" eOpposite="#//Optionality/optional"/>
98    <eStructuralFeatures xsi:type="ecore:EReference" name="variationPoints" upperBound="
          -1"
99        eType="#//VariationPoint" eOpposite="#//VariationPoint/variant"/>
100   <eStructuralFeatures xsi:type="ecore:EReference" name="service" upperBound="-1"
101       eType="#//AdaptationService" eOpposite="#//AdaptationService/selectedElement"/>
102   <eStructuralFeatures xsi:type="ecore:EReference" name="prodArchConfig" upperBound="-1
          "
103       eType="#//ProductArchitectureConfiguration" eOpposite="#//
              ProductArchitectureConfiguration/productElement"/>
104   <eStructuralFeatures xsi:type="ecore:EReference" name="inPort" upperBound="-1"
105       eType="#//Port"/>
106   <eStructuralFeatures xsi:type="ecore:EReference" name="outPort" upperBound="-1"
107       eType="#//Port"/>
108 </eClassifiers>
109 <eClassifiers xsi:type="ecore:EClass" name="Port" abstract="true" eSuperTypes="#//
        ArchitectureElement">
```

```
110     <eStructuralFeatures xsi:type="ecore:EReference" name="classifier" lowerBound="1"
111         eType="#//PortType"/>
112     <eStructuralFeatures xsi:type="ecore:EReference" name="delegationLinkSource" eType="
            #//DelegationLink"
113         eOpposite="#//DelegationLink/source"/>
114     <eStructuralFeatures xsi:type="ecore:EReference" name="delegationLinkTarget" eType="
            #//DelegationLink"
115         eOpposite="#//DelegationLink/target"/>
116     <eStructuralFeatures xsi:type="ecore:EReference" name="inLink" upperBound="-1"
117         eType="#//AssemblyLink" eOpposite="#//AssemblyLink/inPort"/>
118     <eStructuralFeatures xsi:type="ecore:EReference" name="outLink" upperBound="-1"
119         eType="#//AssemblyLink" eOpposite="#//AssemblyLink/outPort"/>
120     <eStructuralFeatures xsi:type="ecore:EReference" name="element" lowerBound="1"
121         eType="#//Element"/>
122   </eClassifiers>
123   <eClassifiers xsi:type="ecore:EClass" name="Connector" eSuperTypes="#//Element">
124     <eStructuralFeatures xsi:type="ecore:EReference" name="classifier" lowerBound="1"
125         eType="#//ConnectorType"/>
126   </eClassifiers>
127   <eClassifiers xsi:type="ecore:EClass" name="Component" eSuperTypes="#//Element">
128     <eStructuralFeatures xsi:type="ecore:EReference" name="classifier" lowerBound="1"
129         eType="#//ComponentType"/>
130   </eClassifiers>
131   <eClassifiers xsi:type="ecore:EClass" name="CKElement" abstract="true" eSuperTypes="#//
          ModelElement"/>
132   <eClassifiers xsi:type="ecore:EClass" name="Root">
133     <eStructuralFeatures xsi:type="ecore:EReference" name="architectureModel" upperBound=
            "-1"
134         eType="#//ArchitectureModel" containment="true"/>
135     <eStructuralFeatures xsi:type="ecore:EReference" name="featureModel" upperBound="-1"
136         eType="#//FeatureModel" containment="true"/>
137     <eStructuralFeatures xsi:type="ecore:EReference" name="configurationModel" upperBound
            ="-1"
138         eType="#//ConfigurationModel" containment="true"/>
139   </eClassifiers>
140   <eClassifiers xsi:type="ecore:EClass" name="ArchitectureElementType" abstract="true"
141       eSuperTypes="#//ModelElement"/>
142   <eClassifiers xsi:type="ecore:EClass" name="ArchitectureElement" abstract="true"
143       eSuperTypes="#//ModelElement">
144     <eStructuralFeatures xsi:type="ecore:EReference" name="arch" upperBound="-1" eType="
            #//SoftwareArchitecture"
145         eOpposite="#//SoftwareArchitecture/mandatoryElement"/>
146   </eClassifiers>
147   <eClassifiers xsi:type="ecore:EClass" name="ArchitectureModel">
148     <eStructuralFeatures xsi:type="ecore:EReference" name="elementType" lowerBound="1"
149         upperBound="-1" eType="#//ArchitectureElementType" containment="true"/>
150     <eStructuralFeatures xsi:type="ecore:EReference" name="sa" lowerBound="1" upperBound=
            "-1"
151         eType="#//SoftwareArchitecture" containment="true"/>
152     <eStructuralFeatures xsi:type="ecore:EReference" name="architectureElement"
            lowerBound="1"
153         upperBound="-1" eType="#//ArchitectureElement" containment="true"/>
154   </eClassifiers>
155   <eClassifiers xsi:type="ecore:EClass" name="DelegationLink" eSuperTypes="#//Link">
```

```
156     <eStructuralFeatures xsi:type="ecore:EReference" name="source" lowerBound="1"
157         eType="#//Port" eOpposite="#//Port/delegationLinkSource"/>
158     <eStructuralFeatures xsi:type="ecore:EReference" name="target" lowerBound="1"
159         eType="#//Port" eOpposite="#//Port/delegationLinkTarget"/>
160   </eClassifiers>
161   <eClassifiers xsi:type="ecore:EClass" name="Feature" eSuperTypes="#//ModelElement">
162     <eStructuralFeatures xsi:type="ecore:EReference" name="isExcludedBy" upperBound="-1"
163         eType="#//Feature" eOpposite="#//Feature/excludes"/>
164     <eStructuralFeatures xsi:type="ecore:EReference" name="excludes" upperBound="-1"
165         eType="#//Feature" eOpposite="#//Feature/isExcludedBy"/>
166     <eStructuralFeatures xsi:type="ecore:EReference" name="isRequiredBy" upperBound="-1"
167         eType="#//Feature" eOpposite="#//Feature/requires"/>
168     <eStructuralFeatures xsi:type="ecore:EReference" name="requires" upperBound="-1"
169         eType="#//Feature" eOpposite="#//Feature/isRequiredBy"/>
170     <eStructuralFeatures xsi:type="ecore:EReference" name="subFeature" upperBound="-1"
171         eType="#//SubFeature" containment="true"/>
172     <eStructuralFeatures xsi:type="ecore:EReference" name="featureGroup" upperBound="-1"
173         eType="#//FeatureGroup" containment="true"/>
174     <eStructuralFeatures xsi:type="ecore:EReference" name="isPartOf" eType="#//
            FeatureGroup"
175         eOpposite="#//FeatureGroup/alternativeFeature"/>
176     <eStructuralFeatures xsi:type="ecore:EReference" name="compositionRule" upperBound="
            -1"
177         eType="#//CompositionRule" eOpposite="#//CompositionRule/relevantFeature"/>
178     <eStructuralFeatures xsi:type="ecore:EReference" name="service" upperBound="-1"
179         eType="#//AdaptationService" eOpposite="#//AdaptationService/alternativeFeature"/
                >
180     <eStructuralFeatures xsi:type="ecore:EReference" name="mandatoryInPFC" upperBound="-1
            "
181         eType="#//ProductFeatureConfiguration" eOpposite="#//ProductFeatureConfiguration/
                mandatoryFeature"/>
182     <eStructuralFeatures xsi:type="ecore:EReference" name="selectedInPFC" upperBound="-1"
183         eType="#//ProductFeatureConfiguration" eOpposite="#//ProductFeatureConfiguration/
                selectedFeature"/>
184     <eStructuralFeatures xsi:type="ecore:EReference" name="dynamicInPFC" upperBound="-1"
185         eType="#//ProductFeatureConfiguration" eOpposite="#//ProductFeatureConfiguration/
                dynamicFeature"/>
186     <eStructuralFeatures xsi:type="ecore:EReference" name="userInPFC" upperBound="-1"
187         eType="#//ProductFeatureConfiguration" eOpposite="#//ProductFeatureConfiguration/
                userFeature"/>
188   </eClassifiers>
189   <eClassifiers xsi:type="ecore:EClass" name="FeatureGroup">
190     <eStructuralFeatures xsi:type="ecore:EReference" name="alternativeFeature" lowerBound
            ="2"
191         upperBound="-1" eType="#//Feature" containment="true" eOpposite="#//Feature/
                isPartOf"/>
192     <eStructuralFeatures xsi:type="ecore:EReference" name="contract" upperBound="-1"
193         eType="#//Contract" eOpposite="#//Contract/featureGroup"/>
194     <eStructuralFeatures xsi:type="ecore:EAttribute" name="minCardinality" eType="
            ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
195     <eStructuralFeatures xsi:type="ecore:EAttribute" name="maxCardinality" eType="
            ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
196   </eClassifiers>
197   <eClassifiers xsi:type="ecore:EClass" name="Rule" abstract="true">
```

```
198     <eStructuralFeatures xsi:type="ecore:EAttribute" name="condition" eType="
            ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
199   </eClassifiers>
200   <eClassifiers xsi:type="ecore:EClass" name="CompositionRule" eSuperTypes="#//Rule">
201     <eStructuralFeatures xsi:type="ecore:EReference" name="relevantFeature" upperBound="
            -1"
202         eType="#//Feature" eOpposite="#//Feature/compositionRule"/>
203   </eClassifiers>
204   <eClassifiers xsi:type="ecore:EClass" name="ContextualEntity" eSuperTypes="#//CKElement
        ">
205     <eStructuralFeatures xsi:type="ecore:EReference" name="isCharacterizedBy" upperBound=
            "-1"
206         eType="#//ContextualElement" containment="true" eOpposite="#//ContextualElement/
                characterizes"/>
207   </eClassifiers>
208   <eClassifiers xsi:type="ecore:EClass" name="ContextualElement">
209     <eStructuralFeatures xsi:type="ecore:EReference" name="characterizes" lowerBound="1"
210         eType="#//ContextualEntity" eOpposite="#//ContextualEntity/isCharacterizedBy"/>
211     <eStructuralFeatures xsi:type="ecore:EReference" name="isComposedBy" upperBound="-1"
212         eType="#//ContextualElement"/>
213     <eStructuralFeatures xsi:type="ecore:EReference" name="contextualRule" upperBound="-1
            "
214         eType="#//ContextualRule" eOpposite="#//ContextualRule/relevantCE"/>
215   </eClassifiers>
216   <eClassifiers xsi:type="ecore:EClass" name="ProductFeatureConfiguration">
217     <eStructuralFeatures xsi:type="ecore:EReference" name="mandatoryFeature" lowerBound="
            1"
218         upperBound="-1" eType="#//Feature" eOpposite="#//Feature/mandatoryInPFC"/>
219     <eStructuralFeatures xsi:type="ecore:EReference" name="selectedFeature" lowerBound="1
            "
220         upperBound="-1" eType="#//Feature" eOpposite="#//Feature/selectedInPFC"/>
221     <eStructuralFeatures xsi:type="ecore:EReference" name="dynamicFeature" upperBound="-1
            "
222         eType="#//Feature" eOpposite="#//Feature/dynamicInPFC"/>
223     <eStructuralFeatures xsi:type="ecore:EReference" name="userFeature" upperBound="-1"
224         eType="#//Feature" eOpposite="#//Feature/userInPFC"/>
225   </eClassifiers>
226   <eClassifiers xsi:type="ecore:EClass" name="ProductArchitectureConfiguration"
          eSuperTypes="#//SoftwareArchitecture">
227     <eStructuralFeatures xsi:type="ecore:EReference" name="productLink" lowerBound="1"
228         upperBound="-1" eType="#//Link"/>
229     <eStructuralFeatures xsi:type="ecore:EReference" name="productElement" lowerBound="1"
230         upperBound="-1" eType="#//Element" eOpposite="#//Element/prodArchConfig"/>
231   </eClassifiers>
232   <eClassifiers xsi:type="ecore:EEnum" name="BindingTimeType">
233     <eLiterals name="static"/>
234     <eLiterals name="dynamic" value="1"/>
235   </eClassifiers>
236   <eClassifiers xsi:type="ecore:EClass" name="ResourceDomain" eSuperTypes="#//CKElement">
237     <eOperations name="select"/>
238     <eStructuralFeatures xsi:type="ecore:EReference" name="elementType" lowerBound="1"
239         upperBound="-1" eType="#//ElementType"/>
240     <eStructuralFeatures xsi:type="ecore:EReference" name="profile" upperBound="-1"
241         eType="#//Profile"/>
```

```
242   </eClassifiers>
243   <eClassifiers xsi:type="ecore:EClass" name="RootFeature" eSuperTypes="#//Feature"/>
244   <eClassifiers xsi:type="ecore:EClass" name="FeatureModel" eSuperTypes="#//ModelElement"
          >
245     <eStructuralFeatures xsi:type="ecore:EReference" name="rootFeature" lowerBound="1"
246         upperBound="-1" eType="#//Feature" containment="true"/>
247   </eClassifiers>
248   <eClassifiers xsi:type="ecore:EClass" name="SubFeature" eSuperTypes="#//Feature">
249     <eStructuralFeatures xsi:type="ecore:EReference" name="contract" upperBound="-1"
250         eType="#//Contract" eOpposite="#//Contract/optionalFeature"/>
251     <eStructuralFeatures xsi:type="ecore:EAttribute" name="opt" eType="#//Optional"/>
252   </eClassifiers>
253   <eClassifiers xsi:type="ecore:EEnum" name="Optional">
254     <eLiterals name="optional" value="1"/>
255     <eLiterals name="mandatory"/>
256   </eClassifiers>
257   <eClassifiers xsi:type="ecore:EClass" name="AssemblyLink" eSuperTypes="#//Link">
258     <eStructuralFeatures xsi:type="ecore:EReference" name="inPort" lowerBound="1"
259         eType="#//Port" eOpposite="#//Port/inLink"/>
260     <eStructuralFeatures xsi:type="ecore:EReference" name="outPort" lowerBound="1"
261         eType="#//Port" eOpposite="#//Port/outLink"/>
262   </eClassifiers>
263   <eClassifiers xsi:type="ecore:EClass" name="UtilityFunction" eSuperTypes="#//
          Negotiation"/>
264   <eClassifiers xsi:type="ecore:EClass" name="StateMachine" eSuperTypes="#//Negotiation"/
          >
265   <eClassifiers xsi:type="ecore:EClass" name="ConfigurationModel">
266     <eStructuralFeatures xsi:type="ecore:EReference" name="ck" lowerBound="1" upperBound=
          "-1"
267         eType="#//CKElement" containment="true"/>
268   </eClassifiers>
269 </ecore:EPackage>
```

Referências

[1] Abowd, G., Dey, A., Brown, P., Davies, N., Smith, M., Steggles, P. Towards a better understanding of context and context-awareness. In *Handheld and Ubiquitous Computing* (1999), Springer, p. 304–307.

[2] Abowd, G., Mynatt, E., Rodden, T. The human experience [of ubiquitous computing]. *Pervasive Computing, IEEE 1*, 1 (2002), 48–57.

[3] Alves, V., Schneider, D., Becker, M., Bencomo, N., Grace, P. Comparative Study of Variability Management in Software Product Lines and Runtime Adaptable Systems. In *VaMoS* (University of Duisburg-Essen, 2007), p. 9–17.

[4] Amft, O., Tröster, G. Recognition of Dietary Activity Events Using On-Body Sensors. *Artificial Intelligence in Medicine 42*, 2 (2008), 121–136.

[5] Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., others. A view of cloud computing. *Communications of the ACM 53*, 4 (2010), 50–58.

[6] Asikainen, T., Männistö, T., Soininen, T. Kumbang: A domain ontology for modelling variability in software product families. *Advanced Engineering Informatics 21*, 1 (2007), 23–40.

[7] Asikainen, T., Soininen, T., Männistö, T. A koala-based approach for modelling and deploying configurable software product families. *Software Product-Family Engineering* (2004), 225–249.

[8] Bachmann, F., Goedicke, M., Leite, J., Nord, R., Pohl, K., Ramesh, B., Vilbig, A. A meta-model for representing variability in product family development. In *Software Product-Family Engineering*. Springer, 2004, p. 66–80.

[9] Batory, D. Feature-oriented programming and the ahead tool suite. In *Proceedings of the 26th International Conference on Software Engineering* (Washington, DC, USA, 2004), ICSE '04, IEEE Computer Society, p. 702–703.

[10] Batory, D. Feature models, grammars, and propositional formulas. In *Software Product Lines*, H. Obbink and K. Pohl, Eds., vol. 3714 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005, p. 7–20.

[11] Benavides, D., Segura, S., Ruiz-Cortés, A. Automated analysis of feature models 20 years later: A literature review. *Information Systems 35*, 6 (2010), 615 – 636.

[12] Bencomo, N., Sawyer, P., Blair, G., Grace, P. Dynamically Adaptive Systems are Product Pines too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems. In *2nd International Workshop on Dynamic Software Product Lines, Limerick, Ireland* (2008).

[13] Beugnard, A., Jézéquel, J.-M., Plouzeau, N., Watkins, D. Making Components Contract Aware. *Computer 32* (July 1999), 38–45.

[14] Blackwell, B. Treatment Adherence. *The British Journal of Psychiatry 129*, 6 (1976), 513–531.

[15] Braga, C., Chalub, F., Sztajnberg, A. A Formal Semantics for a Quality of Service Contract Language. *Electron. Notes Theor. Comput. Sci. 203*, 7 (2009), 103–120.

[16] Cardoso, L., Sztajnberg, A., Loques, O. Self-Adaptive Applications Using ADL Contracts. *Lecture Notes in Computer Science 3996* (2006), 87.

[17] Cardoso, L. X. T. Integração de Serviços de Monitoração e Descoberta de Recursos a um Suporte para Arquiteturas Adaptáveis de Software. Dissertação de Mestrado, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil, 2006.

[18] Carvalho, S. T. *Variabilidades Dinâmicas em um Sistema Computacional Inteligente de Assistência Domiciliar à Saúde: Uma Abordagem Baseada em Linhas de Produto e Contratos Arquiteturais.* PhD thesis, (proposta) Instituto de Computação, Universidade Federal Fluminense, Niterói, Brasil, 2010.

[19] Carvalho, S. T., Clua, E., Loques, O. E-Learning e Jogos Eletrônicos Interativos: Possibilidades para a Educação Médica. In *XIX Simpósio Brasileiro de Informática na Educação (SBIE 2008)* (Fortaleza, Brasil, novembro 2008), p. 279–287.

[20] Carvalho, S. T., Copetti, A., Loques, O. Um Sistema Computacional Inteligente de Assistência Domiciliar à Saúde. In *XII Congresso Brasileiro de Informática em Saúde (CBIS 2010)* (Porto de Galinhas, Brasil, outubro 2010), p. 01–06.

[21] Carvalho, S. T., Copetti, A., Loques, O. Sistema de Computação Ubíqua na Assistência Domiciliar à Saúde. *Journal of Health Informatics 3*, 2 (2011), 51–57.

[22] Carvalho, S. T., Coppeti, A., Sztajnberg, A., Erthal, M., Santos, R., Loques, O. Sistema de Assistência Domiciliar à Saúde Telemonitorada. Relatório Técnico, Laboratório Tempo - Sistemas de Tempo Real e Embarcados, Universidade Federal Fluminense, Niterói, Brasil, março 2009.

[23] Carvalho, S. T., Erthal, M., Mareli, D., Sztajnberg, A., Copetti, A., Loques, O. Monitoramento Remoto de Pacientes em Ambiente Domiciliar. In *XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2010)* (Gramado, RS, Brasil, maio 2010), p. 1005–1012.

[24] Carvalho, S. T., Lisbôa, J., Loques, O. Um Design Pattern para Configuração de Arquiteturas de Software. In *2nd Latin American Conference on Pattern Language of Programming (SugarloafPLOP 2002)* (São Carlos, Brazil, 2002), p. 37–53.

[25] Carvalho, S. T., Loques, O. Arquitetura de Software para Sistemas Pervasivos de Assistência Domiciliar à Saúde. In *X Workshop de Informática Médica (WIM 2010). XXX Congresso da Sociedade Brasileira de Computação (CSBC 2010)* (Belo Horizonte, Brasil, julho 2010), p. 1542–1545.

[26] Carvalho, S. T., Loques, O., Murta, L. Dynamic Variability Management in Product Lines: An Approach Based on Architectural Contracts. In *4th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS 2010). Conference on Software: Theory and Practice (CBSOFT 2010)* (Salvador, Brazil, September 2010), p. 61–69.

[27] Carvalho, S. T., Murta, L., Loques, O. A Contract-based Approach for Managing Dynamic Variability in Software Product Line Architectures. Relatório Técnico, Tempo Laboratory – Real-Time and Embedded Systems, Universidade Federal Fluminense, Niterói, Brazil, March 2011.

[28] Carvalho, S. T., Murta, L., Loques, O. Variabilities as First-Class Elements in Product Line Architectures of Homecare Systems. In *4th International Workshop on Software Engineering in Health Care (SEHC 2012). 34th International Conference on Software Engineering (ICSE 2012)* (Zurich, Switzerland, June 2012), p. 33–39.

[29] Cerqueira, R. C. Uma Proposta de Descrição e Implementação de Contratos para Serviços com Qualidade Diferenciada. Dissertação de Mestrado, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil, 2002.

[30] Cetina, C., Fons, J., Pelechano, V. Applying Software Product Lines to Build Autonomic Pervasive Systems. In *12th International Software Product Line Conference, 2008. SPLC'08* (2008), p. 117–126.

[31] Cetina, C., Giner, P., Fons, J., Pelechano, V. Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes. *Computer 42*, 10 (2009), 37–43.

[32] Cetina, C., Giner, P., Fons, J., Pelechano, V. Using Feature Models for Developing Self-Configuring Smart Homes. In *5th ICAS* (Valencia, Spain, 2009), p. 179–188.

[33] Cetina, C., Haugen, Ø., Zhang, X., Fleurey, F., Pelechano, V. Strategies for variability transformation at run-time. In *Proceedings of the 13th International Software Product Line Conference* (2009), Carnegie Mellon University, p. 61–70.

[34] Chen, L., Ali Babar, M., Ali, N. Variability management in software product lines: a systematic review. In *Proceedings of the 13th International Software Product Line Conference* (2009), Carnegie Mellon University, p. 81–90.

[35] Choudhury, T., Consolvo, S., Harrison, B., Hightower, J., LaMarca, A., LeGrand, L., Rahimi, A., Rea, A., Borriello, G., Hemingway, B., Klasnja, P., Koscher, K., Landay, J. A., Lester, J., Wyatt, D., Haehnel, D. The Mobile Sensing Platform: An Embedded Activity Recognition System. *Pervasive Computing 7*, 2 (April-June 2008), 32–41.

[36] Chung, L., do Prado Leite, J. C. S. On non-functional requirements in software engineering. In *Conceptual modeling: Foundations and applications*. Springer, 2009, p. 363–379.

[37] Chung, W.-Y., Bhardwaj, S., Purwar, A., Lee, D.-S., Myllylae, R. A Fusion Health Monitoring Using ECG and Accelerometer sensors for Elderly Persons at Home. In *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE* (22-26 2007), p. 3818 –3821.

[38] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., Stafford, J. *Documenting Software Architectures: Views and Beyond, Second Edition*. Addison-Wesley, 2010.

[39] Clements, P., Northrop, L. *Software product lines: practices and patterns*. Addison-Wesley Longman Publishing Co., 2001.

[40] Conradi, R., Westfechtel, B. Version Models for Software Configuration Management. *ACM Computing Surveys 30*, 2 (1998), 232–282.

[41] Copetti, A. *Monitoramento Inteligente e Sensível ao Contexto na Assistência Domiciliar Telemonitorada*. PhD thesis, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil, 2010.

[42] Corradi, A. M. Um Framework de Suporte a Requisitos Não-Funcionais para Serviços de Nível Alto. Dissertação de Mestrado, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil, 2005.

[43] Czarnecki, K., Antkiewicz, M., Kim, C., Lau, S., Pietroszek, K. Model-driven software product lines. In *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (2005), ACM, p. 126–127.

[44] Czarnecki, K., Eisenecker, U. W. *Generative programming: methods, tools, and applications*. ACM Press Addison-Wesley Publishing Co., 2000.

[45] Czarnecki, K., Helsen, S., Eisenecker, U. Staged configuration using feature models. *Software Product Lines* (2004), 162–164.

[46] Czarnecki, K., Helsen, S., Eisenecker, U. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice 10*, 1 (2005), 7–29.

[47] Dashofy, E., van der Hoek, A., Taylor, R. A comprehensive approach for the development of modular software architecture description languages. *ACM Trans Softw Eng Methodol 14* (2005), 199–245.

[48] Dey, A. Understanding and using context. *Personal and ubiquitous computing 5*, 1 (2001), 4–7.

[49] Dieckmann, J. Home health care: An historical perspective and overview. *Handbook of home health care administration* (2010), 1.

[50] Dijkstra, E. *Notes on structured programming*. Technological University, Department of Mathematics, 1970.

[51] ElHelw, M., Pansiot, J., McIlwraith, D., Ali, R., Lo, B., L., A. An Integrated Multi-Sensing Framework for Pervasive Healthcare Monitoring. In *3rd International Conference on Pervasive Computing Technologies for Healthcare* (april 2009), p. 1 –7.

[52] Erthal, M. Sistema de Computação Ubíqua para Aplicação em Telessaúde, 2011. Monografia (Bacharelado em Ciência da Computação), Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil.

[53] Erthal, M., Mareli, D., Ferreira, D., Loques, O. Interpretação de Contexto em Ambientes Inteligentes. In *V Simpósio Brasileiro de Computação Ubíqua e Pervasiva (SBCUP 2013). XXXIII Congresso da Sociedade Brasileira de Computação (CSBC 2013)* (Maceió, AL, Brasil, julho 2013).

[54] Eslami, M. Z., van Sinderen, M. J. Flexible Home Care Automation Adapting to the Personal and Evolving Needs and Situations of the Patient. In *3rd International Conference on Pervasive Computing Technologies for Healthcare, 2009. PervasiveHealth 2009, London, UK* (Los Alamitos, CA, March 2009), IEEE, p. 1–2.

[55] Fairbanks, G. *Just Enough Software Architectures: A Risk-Driven Approach*. Marshall & Brainerd, 2010.

[56] Fernandes, P., Werner, C., Teixeira, E. An approach for feature modeling of context-aware software product line. *Journal of Universal Computer Science 17*, 5 (mar 2011), 807–829.

[57] Ferreira, D., Erthal, M., Mareli, D., Loques, O. Uma Interface de Prototipagem para Aplicações Pervasivas. In *XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2013)* (Brasília, DF, Brasil, maio 2013), p. 629–642.

[58] Freire, S., Sztajnberg, A., Copetti, A., Loques, O. Utilizando o Modelo Dual para a Representação e Persistência de Contexto em Aplicações Ubíquas de Telemonitoramento. In *VIII Workshop de Informática Médica. WIM 2008. XXVIII Congresso da Sociedade Brasileira de Computação. CSBC 2008* (Belém, PA, Brasil, julho 2008), p. 252–255.

[59] Frølund, S., Koistinen, J. Quality of services specification in distributed object systems design. *Distributed Systems Engineering Journal 5*, 4, 179–202.

[60] Garg, A., Critchlow, M., Chen, P., van der Westhuizen, C., van der Hoek, A. An Environment for Managing Evolving Product Line Architectures. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on* (2003), p. 358–367.

[61] Gomaa, H. *Designing Software Product Lines with UML: from Use Cases to Pattern-based Software Architectures*. Addison-Wesley, 2005.

[62] Grüninger, M., Fox, M. S. Methodology for the design and evaluation of ontologies. In *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95* (Montreal, CA, 1995).

[63] Hallsteinsen, S., Hinchey, M., Park, S., Schmid, K. Dynamic Software Product Lines. *Computer 41*, 4 (2008), 93–95.

[64] Hallsteinsen, S., Stav, E., Solberg, A., Floch, J. Using Product Line Techniques to Build Adaptive Systems. In *SPLC '06: Proceedings of the 10th International on Software Product Line Conference* (Washington, DC, USA, 2006), IEEE Computer Society, p. 141–150.

[65] Halmans, G., Pohl, K. Communicating the variability of a software-product family to customers. *Software and Systems Modeling 2* (2003), 15–36. 10.1007/s10270-003-0019-9.

[66] Hatcliff, J., King, A., Lee, I., MacDonald, A., Fernando, A., Robkin, M., Vasserman, E., Weininger, S., Goldman, J. Rationale and architecture principles for medical application platforms. In *Cyber-Physical Systems (ICCPS), 2012 IEEE/ACM Third International Conference on* (2012), IEEE, p. 3–12.

[67] Haynes, R., McDonald, H., Garg, A. Helping Patients Follow Prescribed Treatment Clinical Applications, 2002.

[68] Heráclio, D., Carvalho, S. T., Murta, L., Loques, O. Runtime Monitoring and Auditing of Self-Adaptive Systems. In *25th International Conference on Software Engineering and Knowledge Engineering (SEKE 2013)* (Boston, USA, June 2013).

[69] Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report, CMU/SEI-90-TR-21. Relatório Técnico, Software Engineering Institute (Carnegie Mellon), Pittsburgh, PA 15213, 1990.

[70] Kaup, I., Bricola, S. Estratégias para adesão medicamentosa do paciente crônico assistido em domícilio. In *Assistência Domiciliar, Uma proposta interdisciplinar*, A. Yamaguchi, K. Higa-Taniguchi, L. Andrade, S. Bricola, W. Jacob Filho, and M. Martins, Eds. Manole, 2010, p. 252–264.

[71] Kinsella, A. The Home Telehealth Primer. *Telemedicine Information Exchange* (2008).

[72] Koch, S. Home Telehealth-Current State and Future Trends. *International Journal of Medical Informatics 75*, 8 (2006), 565–576.

[73] Kramer, J., Magee, J. The evolving philosophers problem: Dynamic change management. *Software Engineering, IEEE Transactions on 16*, 11 (1990), 1293–1306.

[74] Kramer, J., Magee, J. Exposing the skeleton in the coordination closet. *Coordination Languages and Models 1282* (1997), 18–31.

[75] Kramer, J., Magee, J. Analysing dynamic change in distributed software architectures. In *Software, IEE Proceedings-* (1998), vol. 145, IET, p. 146–154.

[76] Krueger, C. Variation Management for Software Production Lines. *Software Product Lines* (2002), 107–108.

[77] Larson, B., Hatcliff, J., Procter, S., Chalin, P. Requirements specification for apps in medical application platforms. In *Software Engineering in Health Care (SEHC), 2012 4th International Workshop on* (2012), IEEE, p. 26–32.

[78] Leal, D. A. S. Suportando a Adaptação de Aplicações Pervasivas pelo Uso de Funções Utilidade. Dissertação de Mestrado, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil, 2007.

[79] Lee, H., Park, K., Lee, B., Choi, J., R., E. Issues in Data Fusion for Healthcare Monitoring. In *Proceedings of the International Conference on PErvasive Technologies Related to Assistive Environments* (New York, USA, 2008), p. 1–8.

[80] Lee, J., Kang, K. C. A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering. In *10th SPLC* (Washington, DC, USA, 2006), p. 131–140.

[81] Lee, J., Muthig, D. Feature-Oriented Variability Management in Product Line Engineering. *Communications of the ACM 49*, 12 (2006), 59.

[82] Leijdekkers, P., Gay, V., Lawrence, E. Smart Homecare System for Health Tele-monitoring. In *Digital Society, 2007. ICDS '07* (jan. 2007), p. 3 –3.

[83] Loques, O., Sztajnberg, A. Adaptation Issues in Software Architectures of Remote Health Care Systems. In *2nd Workshop on Software Engineering in Health Care - SEHC 2010. 32nd International Conference on Software Engineering* (Cape Town, South Africa, May 2010), p. 24–28.

[84] Loques, O., Sztajnberg, A., Cerqueira, R., Ansaloni, S. A Contract-Based Approach to Describe and Deploy Non-Functional Adaptations in Software Architectures. *Journal of the Brazilian Computer Society 10*, 1 (2004), 5–18.

[85] Malucelli, V. V. BABEL - Construindo Aplicações por Evolução. Dissertação de Mestrado, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, 1996.

[86] Mannion, M. Using first-order logic for product line model validation. In *Software Product Lines*, G. Chastek, Ed., vol. 2379 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2002, p. 149–202.

[87] Mannion, M., Camara, J. Theorem proving for product line model verification. In *Software Product-Family Engineering*, F. van der Linden, Ed., vol. 3014 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2004, p. 211–224.

[88] Mareli, D. Reconhecimento de Atividades em um Sistema Computacional Pervasivo de Assistência Domiciliar à Saúde, 2011. Monografia (Bacharelado em Ciência da Computação), Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil.

[89] Mareli, D., Erthal, M., Ferreira, D., Loques, O. Um Framework de Desenvolvimento de Aplicações Ubíquas em Ambientes Inteligentes. In *XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2013)* (Brasília, DF, Brasil, maio 2013), p. 643–656.

[90] Medvidovic, N., Rosenblum, D., Taylor, R. A language and environment for architecture-based software development and evolution. In *Software Engineering, 1999. Proceedings of the 1999 International Conference on* (1999), IEEE, p. 44–53.

[91] Medvidovic, N., Taylor, R. N. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Trans. Softw. Eng. 26*, 1 (2000), 70–93.

[92] Menezes, A., Cirilo, C., Moraes, J., Souza, W.and Prado, A. Using Archetypes and Domain Specific Languages on Development of Ubiquitous Applications to Pervasive Healthcare. In *23rd International Symposium on Computer-Based Medical Systems* (Perth, Australia, October 2010), p. 395–400.

[93] Meyer, B. Applying'design by contract'. *Computer 25*, 10 (1992), 40–51.

[94] Mion Jr, D., Kohlman Jr, O., Machado, C., Amodeo, C., Gomes, M., Prachedes, J. V Diretrizes Brasileiras de Hipertensão Arterial. *Revista Brasileira de Hipertensão 13* (2006), 256–312.

[95] Moore, B., Dean, D., Gerber, A., Wagenknecht, G., Vanderheyden, P. *Eclipse Development: using the Graphical Editing Framework and the Eclipse Modeling Framework*. IBM Redbook, 2004.

[96] Morin, B., Barais, O., Jezequel, J.-M., Fleurey, F., Solberg, A. Models at Runtime to Support Dynamic Adaptation. *Computer 42*, 10 (oct. 2009), 44 –51.

[97] Morin, B., Barais, O., Nain, G., Jezequel, J.-M. Taming dynamically adaptive systems using models and aspects. In *Proceedings of the 31st International Conference on Software Engineering* (Washington, DC, USA, 2009), ICSE '09, IEEE Computer Society, p. 122–132.

[98] Muthig, D., Atkinson, C. Model-driven product line architectures. *Software product lines* (2002), 79–90.

[99] Northrop, L. M. SEI's Software Product Line Tenets. *IEEE Software 19* (2002), 32–40.

[100] OMG. Meta Object Facility (MOF) Specification version 1.4, abril 2002. Object Management Group (OMG). Document number: formal/2004-04-03.

[101] Oreizy, P., Gorlick, M., Taylor, R., Heimhigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D., Wolf, A. An architecture-based approach to self-adaptive software. *Intelligent Systems and their Applications, IEEE 14*, 3 (may/jun 1999), 54 –62.

[102] Orwat, C., Graefe, A., Faulwasser, T. Towards Pervasive Computing in Health Care - A Literature Review. *BMC Medical Informatics and Decision Making 8*, 1 (2008), 26.

[103] Parnas, D. On the criteria to be used in decomposing systems into modules. *Communications of the ACM 15*, 12 (1972), 1053–1058.

[104] Parnas, D. On the design and development of program families. *Software Engineering, IEEE Transactions on*, 1 (1976), 1–9.

[105] Parra, C., Blanc, X., Duchien, L. Context awareness for dynamic service-oriented product lines. In *Proceedings of the 13th International Software Product Line Conference* (2009), Carnegie Mellon University, p. 131–140.

[106] Parra, C., Cleve, A., Blanc, X., Duchien, L. Feature-based composition of software architectures. *Software Architecture* (2010), 230–245.

[107] Petrucci, V., Loques, O. Suporte a Adaptação Dinâmica de Aplicações usando Funções de Utilidade. In *Workshop on Pervasive an Ubiquitous Computing* (Gramado, Brasil, 2007), p. 1–6.

[108] Piniewski, B., Muskens, J., Estevez, L., Carroll, R., Cnossem, R. Empowering Healthcare Patients with Smart Technology. *Computer 43*, 7 (2010), 27–34.

[109] Pohl, K., Böckle, G., van Der Linden, F. *Software product line engineering: foundations, principles, and techniques.* Springer-Verlag New York Inc, 2005.

[110] Rademaker, A., Braga, C., Sztajnberg, A. A rewriting semantics for a software architecture description language. *Electronic Notes in Theoretical Computer Science 130* (2005), 345–377.

[111] Ranganathan, A., Chetan, S., Al-Muhtadi, J., Campbell, R. H., Mickunas, M. D. Olympus: A high-level programming model for pervasive computing environments. *Pervasive Computing and Communications, IEEE International Conference on 0* (2005), 7–16.

[112] Riebisch, M., Böllert, K., Streitferdt, D., Philippow, I. Extending feature diagrams with uml multiplicities. In *Proceedings of the 6th Conference on Integrated Design & Process Technology (IDPT 2002).*

[113] Rodrigues, A., Gomes, I., Bezerra, L., Sztajnberg, A., Carvalho, S. T., Copetti, A., Loques, O. Using Discovery and Monitoring Services to Support Context-Aware Remote Assisted Living Applications. In *Computational Science and Engineering, 2009. CSE '09. International Conference on* (aug. 2009), vol. 2, p. 1092 –1097.

[114] Rodrigues, A. L. B., Bezerra, L. N., Sztajnberg, A., Loques, O. Self-Adaptation of Fault Tolerance Requirements Using Contracts. *Computational Science and Engineering, IEEE International Conference on 2* (2009), 245–253.

[115] Santos, A., Koskimies, K., Lopes, A. A model-driven approach to variability management in product-line engineering. *Nordic Journal of Computing 13*, 3 (2006), 196.

[116] Satyananda, T., Lee, D., Kang, S., Hashmi, S. Identifying traceability between feature model and software architecture in software product line using formal concept analysis. In *Computational Science and its Applications, 2007. ICCSA 2007. International Conference on* (2007), IEEE, p. 380–388.

[117] Schmidt, D. Model-Driven Engineering. *IEEE Computer 29*, 2 (February 2006), 25–31.

[118] Sinnema, M., Deelstra, S., Nijhuis, J., Bosch, J. COVAMOF: A Framework for Modeling Variability in Software Product Families. In *3rd SPLC* (2004), p. 197–213.

[119] Stachura, M., Khasanshina, E. Telehomecare and remote monitoring: An outcomes overview. Relatório Técnico, The Advanced Medical Technology Association, 2007.

[120] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E. *EMF: Eclipse Modeling Framework, Second Edition*. Addison-Wesley Professional, 2008.

[121] Svahnberg, M., van Gurp, J., Bosch, J. A Taxonomy of Variability Realization Techniques. *Software: Practice and Experience 35*, 8 (2005), 705–754.

[122] Sztajnberg, A., Rodrigues, A., Bezerra, L., Loques, O., Copetti, A., Carvalho, S. T. Applying Context-Aware Techniques to Design Remote Assisted Living Applications. *International Journal of Functional Informatics and Personalised Medicine 2*, 4 (2009), 358–378.

[123] Taylor, R., Medvidovic, N., Dashofy, E. *Software Architecture: Foundations, Theory, and Practice*. John Wiley and Sons, 2010.

[124] Trask, B., Roman, A. Leveraging model driven engineering in software product line architectures. In *Proceedings of the 16th International Software Product Line Conference - Volume 2* (New York, NY, USA, 2012), SPLC '12, ACM, p. 271–273.

[125] Trinidad, P., Cortés, A. R., Peña, J., Benavides, D. Mapping Features Models onto Component Models to Build Dynamic Software Product Lines. In *Workshop on DSPL* (Kyoto, Japan, 2007), p. 51–56.

[126] Trujillo, S., Batory, D., Diaz, O. Feature oriented model driven development: A case study for portlets. In *Proceedings of the 29th international conference on Software Engineering* (Washington, DC, USA, 2007), ICSE '07, IEEE Computer Society, p. 44–53.

[127] Uschold, M., Gruninger, M., Uschold, M., Gruninger, M. Ontologies: Principles, methods and applications. *Knowledge Engineering Review 11* (1996), 93–136.

[128] van der Hoek, A. Design-time product line architectures for any-time variability. *Science of computer programming 53*, 3 (2004), 285–304.

[129] van Gurp, J., Bosch, J., Svahnberg, M. On the Notion of Variability in Software Product Lines. In *WICSA '01: Proceedings of the Working IEEE/IFIP Conference on Software Architecture* (Washington, DC, USA, 2001), IEEE Computer Society, p. 45.

[130] van Ommering, R., van der Linden, F., Kramer, J., Magee, J. The Koala Component Model for Consumer Electronics Software. *Computer 33* (2000), 78–85.

[131] Varshney, U. Pervasive Healthcare. *Computer 36*, 12 (2003), 138–140.

[132] Vieira, V., Tedesco, P., Salgado, A. C. Designing context-sensitive systems: An integrated approach. *Expert Systems with Applications 38*, 2 (2011), 1119 – 1138.

[133] Webber, D., Gomaa, H. Modeling variability in software product lines with the variation point model. *Science of Computer Programming 53*, 3 (2004), 305–331.

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

[135] White, J., Schmidt, D., Wuchner, E., Nechypurenko, A. Automating product-line variant selection for mobile devices. In *Software Product Line Conference, 2007. SPLC 2007. 11th International* (2007), IEEE, p. 129–140.

[136] Wood, A., Stankovic, J., Virone, G., Selavo, L., He, Z., Cao, Q., Doan, T., Wu, Y., F. L., Stoleru, R. Context-Aware Wireless Sensor Networks for Assisted-Living and Residential Monitoring. *IEEE Network 22*, 4 (2008), 26–33.

[137] Zhang, J., Cheng, B. H. C. Model-based development of dynamically adaptive software. In *Proceedings of the 28th international conference on Software engineering* (New York, NY, USA, 2006), ICSE '06, ACM, p. 371–380.

[138] Zhang, R., Liu, L. Security models and requirements for healthcare application clouds. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on* (2010), IEEE, p. 268–275.