Universidade Federal do Amazonas
Instituto de Computação
Programa de Pós-Graduação em Informática

Arquitetura de apoio à integração de Linha de Produto de Software e Engenharia Dirigida a Modelos

Daniella Rodrigues Bezerra

Manaus - Amazonas
2016

Universidade Federal do Amazonas
Instituto de Computação

Daniella Rodrigues Bezerra

Arquitetura de apoio à integração de Linha de Produto de Software e Engenharia Dirigida a Modelos

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Informática do Instituto de Computação da UFAM, como parte dos requisitos necessários à obtenção do título de Doutora em Informática. Área de concentração: **Engenharia de Software**.

Orientador: Prof. Raimundo da Silva Barreto, Dr.

Manaus - Amazonas

2016

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Bezerra, Daniella Rodrigues
B574a Arquitetura de apoio à integração de Linha de Produto de Software e Engenharia Dirigida a Modelos / Daniella Rodrigues Bezerra. 2016
204 f.: il. color; 31 cm.

Orientador: Raimundo da Silva Barreto
Tese (Doutorado em Informática) - Universidade Federal do Amazonas.

1. Modelagem Arquitetural. 2. Variabilidade. 3. Linha de Produto de Software. 4. Engenharia Dirigida a Modelos. 5. Linguagem de Domínio Específico. I. Barreto, Raimundo da Silva II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Arquitetura de Apoio à Integração de Linha de Produto de Software e Engenharia Dirigida a Modelos"

DANIELLA RODRIGUES BEZERRA

Tese de Doutorado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Raimundo da Silva Barreto - PRESIDENTE

Prof. Arilo Claudio Dias Neto - MEMBRO INTERNO

Profa. Rosiane de Freitas Rodrigues - MEMBRO INTERNO

Prof. José Luiz de Souza Pio - MEMBRO EXTERNO

Profa. Elisa Yumi Nakagawa - MEMBRO EXTERNO

Manaus, 04 de Março de 2016

Ao meu pai e à minha mãe por plantarem a semente. À vida pelo solo fértil. Em especial ao meu marido e a todos da minha família pelo incentivo à realização deste trabalho.

Agradecimentos

A Deus, por me mostrar que o divino está na ciência e que a ciência está no divino;
Ao meu orientador, Prof. Barreto por ter me conduzido ao longo do doutorado, pela confiança e amizade que sem as quais não seria possível o desenvolvimento deste trabalho com tamanha satisfação;
Ao meu supervisor de doutorado sanduíche, Prof. Bernd Fischer pelas contribuições;
Aos professores do INPA, Niro Higushi e Adriano Nogueira por apoiarem o projeto e compartilharem conhecimento sobre o domínio de manejo florestal;
À Profa. Rosiane Rodrigues, pela orientação, estímulos e contribuições;
Ao Prof. Arilo Dias-Neto, por sua visão cirúrgica;
Aos meus irmãos acadêmicos Odette, Herbert e Rawlinson pelo companheirismo e bons momentos de convivência, dentro e fora do laboratório;
Aos amigos Margaret, Arnold, Peter, Clive, Lavinia, Mary e Malcolm pelos bons momentos na Inglaterra;
Aos amigos Clara, Preta, Robert, Bart, Meg, Coração, Teko e Camael pelos momentos de descontração;
À Elienai e equipe da secretaria pelo apoio diário;
À CAPES pela oportunidade e concessão da bolsa de estudos;
À UFAM e a todos os professores pela experiência de vida que o doutorado me proporcionou.

As leis da natureza nada mais são que pensamentos matemáticos de Deus.

Johannes Kepler

Que vocês sejam grandes empreendedores. Se empreenderem, não tenham medo de falhar. Se falharem, não tenham medo de chorar. Se chorarem, repensem a sua vida, mas não desistam. Dêem sempre uma nova chance a si mesmos.

Augusto Cury

Resumo

Essa tese investiga o problema de como integrar do ponto de vista arquitetural, modelos conceituais e de projeto e manter as diferenças necessárias para viabilizar os vários níveis de abstrações compatíveis com o tamanho, complexidade e domínio das famílias de software. O ponto de partida à obtenção de uma solução para esse problema é investigar a integração entre linha de produto de software (SPL) e engenharia guiada a modelos (MDE) por individualmente fornecerem um arcabouço para esse propósito; porém, pouco investigado para famílias de linguagens de domínio específico (DSL) como domínio de aplicação. Sendo assim, esse trabalho tem como objetivo apresentar um método de suporte à modelagem arquitetural de famílias de software com base em MDE composto por dois elementos: (i) o esquema de integração entre as engenharias e (ii) o *framework* arquitetural. O esquema elicita como a integração efetivamente ocorre e destaca as possibilidades de aplicação. Já o *framework* arquitetural complementa esse esquema de integração e ajuda a projetar visões arquiteturais e a integração entre as mesmas. Para conduzir a pesquisa, foi utilizada a metodologia ARDev (*Agile Research Development*) que tem como propósito auxiliar a gestão e produção dos artefatos de pesquisa. O método proposto é avaliado por meio de provas de conceito com aplicação em DSL's no domínio de dinâmica florestal e estimativa de carbono, selecionados por apresentarem requisitos e processos coerentes com a complexidade de observação do método e avaliação das implicações do mesmo sobre uma aplicação prática. Os resultados obtidos indicam que a integração entre as engenharias favorece a estruturação e adaptação da arquitetura. Já o *framework* ajuda o projetista a escolher, desenvolver e testar rapidamente abordagens usadas para a integração entre as engenharias e recuperar informações arquiteturais adotadas para um domínio.

Palavras-chave: Modelagem Arquitetural, Variabilidade, Linha de Produto de Software Dirigido por Modelo e Linguagem de Domínio Específico.

Abstract

This thesis investigates the problem of how integrate the architectural point of view, conceptual and design models to maintain the necessary differences enable various levels of abstractions compatible with the size, complexity and the field of software families. The starting point to obtain a problem solution is to investigate the integration of software product line (SPL) and model-driven engineering (MDE) by individually provide support for this purpose; but little investigated for a family of domain specific languages as application domain. Thus, this work aims to present a support method to architectural modelling software families based on MDE compound two components: (i) integration scheme between the engineering and (ii) the framework architectural. The scheme elicits the integration actually takes place and highlights the possibilities of application. But the architectural framework complements this integration scheme and helps design architectural views and integration between them. To conduct the research, ARDev methodology (Agile Research Development) was used which aims to assist the management and production of research artifacts. The proposed method is assessed by proofs of concept in forest dynamics domain and estimate carbon DSL's, selected by presenting requirements and processes consistent with the complexity of observation and evaluation method of the implications thereof on a practical application. The results indicate that the integration between engineering favors the structuring and adaptation of architecture and the framework helps the designer to choose, quickly develop and test approaches used for integration between engineering and architectural retrieve information taken to a domain.

Key words: Architectural Modelling, Variability, Model-Driven Software Product Line Domain Specific Language.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

ADL:	*Architecture Description Language*
ARDev:	*Agile Research Development*
CIM:	*Computational Independent Model*
CORBA:	*Common Object Request Broker Architecture*
COTS:	*Commercial off the Shelf*
DCOM:	*Distributed Component Object Model*
DSL:	*Domain Specific Language*
DSQL:	*Domain Specific Query Language*
DSSA:	*Domain Specific Software Architecture*
FODA:	*Feature-Oriented Domain Analysis*
GPS:	*Global Positioning System*
SQL:	*Structured Query Language*
HQL:	*Hibernate Query Language*
MDE:	*Model-Driven Engineering*
MD-SPL:	*Model-Driven Software Product Line*
MVC:	*Model - View - Control*
ODM:	*Organization Domain Modeling*
ORM:	*Object-Relational Mapping*
OWL-DL:	*Web Ontology Language - Description Logic*
PIM:	*Platform Independent Model*
PPL:	*Permanent Parcel Language*
PSM:	*Platform Specific Model*
RFID:	*Radio-Frequency IDentification*
RPC:	*Remote Procedure Call*
SPL:	*Software Product Line*
SOA:	*Service Oriented Architecture*
SQL:	*Structured Query Language*
TSP:	*Travel Salesman Problem*
UML:	*Unified Modeling Language*
XML:	*eXtensible Markup Language*

Capítulo 1

Introdução

Esse capítulo está dividido em oito seções. A Seção 1.1 apresenta conceitos, destacados no texto em *"itálico"*, essenciais para o entendimento da tese e procura responder as seguintes perguntas: (i) Como é uma modelagem arquitetural sem método? (ii) Por que um arquiteto de software precisa de um método que ofereça suporte à modelagem arquitetural? O problema investigado nessa tese também é caracterizado nessa seção. A Seção 1.2 apresenta a motivação e justificativa para realização da pesquisa. As questões de pesquisa são discutidas na Seção 1.3. O objetivo geral e os específicos são apresentados na Seção 1.4. A Seção 1.5 apresenta metodologia científica usada na condução da pesquisa. A Seção 1.6 apresenta uma introdução do método proposto. As contribuições do trabalho são apontadas na Seção 1.7. Por fim, a Seção 1.8 apresenta como essa tese está organizada.

1.1 Contexto

Construir software por meio do reúso em larga escala e gerenciar simultaneamente a diversidade do domínio de aplicação em função da minimização do custo é um processo de alta complexidade que necessita de um método adequado e minucioso [7].

Com o intuito de superar esse desafio, surgiu o paradigma de desenvolvimento SPL [8, 9, 10], capaz de contribuir com a customização, minimização do tempo de lançamento/disponibilização do software para o mercado consumidor, redução do custo de desenvolvimento e na qualidade final do produto.

SPL é uma área que vem sendo naturalmente impulsionada em função do incremento da complexidade e da variabilidade dos softwares, alinhado a demanda

contínua por produtos e serviços customizados, principalmente em domínios de amplo crescimento econômico e com forte demanda tecnológica tais como: o bancário, de comunicação, transportes e medicina [10].

Mas para entender esse fenômeno de propulsão é necessário compreender que a *variabilidade* [4, 11, 12] está diretamente associada ao atual cenário mercadológico e científico de desenvolvimento de software, pois além de ter características endógenas, comportando diferentes funcionalidades associadas especificamente ao software, há também a variabilidade exógena que concentra-se em preparar os softwares para serem executados em meios diferentes tais como: hardwares, *display* variados, dispositivos de entrada, preferências do usuário, sistemas operacionais e até de localização, ou seja, para sistemas globais como jogos, dependendo de onde o usuário acessar, o sistema deve ter a capacidade de se autoconfigurar com propriedades e características voltadas para determinadas regiões geográficas.

Diante de tanta diversidade, não é sustentável desenvolver software em larga escala sem ter uma *plataforma comum* de apoio que concentre a *similaridade* [13, 14]. No contexto da engenharia de SPL, a similaridade é tida como uma propriedade capaz de caracterizar famílias de software, ou seja, o reúso.

São partes integrantes de SPL duas formas de construção do reúso: a engenharia de domínio e engenharia de aplicação [10, 14, 15], ambas fundamentadas pela *teoria de domínio* [16] que denota/mapeia estruturas em funções parciais com o intuito de representar, por exemplo um dado comportamento. Sendo assim, o reúso em SPL é um conjunto sistemático de processos, técnicas e ferramentas para desenvolver software com base no reaproveitamento de projetos anteriores ou parte dos mesmos como: especificações, módulos, processos, componentes, arquitetura, casos de teste, etc.

A *engenharia de domínio* [17, 18], ou desenvolvimento "para o reúso", foca na sistemática criação de modelos conceituais do domínio, arquitetura e implementação, com ênfase para o reúso.

A *engenharia de aplicação* [19, 20], ou desenvolvimento "com reúso", é voltada para o produto final, com base nos requisitos identificados pela engenharia de domínio e consiste em reusar plataformas existentes com o objetivo de atender a variabilidade de diferentes aplicações.

O elemento central que sustenta uma família de software é a arquitetura capaz de viabilizar a estruturação de componentes, a visualização externa de propriedades, a relação entre os mesmos e serve de base para a qualidade e longevidade do software

[5, 21].

Basicamente, a *arquitetura* representa um conjunto de decisões prévias para a implementação do software e que pode ter um grande impacto sobre o sucesso ou não do projeto [22]. É comum as decisões serem documentadas e não como elas foram tomadas, recaindo sobre o conhecimento tácito do arquiteto a responsabilidade de resolver conflitos como: performance versus segurança e custo do desenvolvimento atual versus futuro.

> *É assim que uma modelagem arquitetural sem método ocorre. Basicamente ela fica na cabeça do arquiteto que a criou, não pode ser replicada, auditada e acaba comprometendo a qualidade final do projeto pois as escolhas arquiteturais não são documentadas explicitamente e sim implicitamente presentes nos modelos que os arquitetos constroem, supostamente no campo da racionalidade e gerando dúvidas sobre a abordagem.*
> *É por isso que os arquitetos necessitam de um método claro, objetivo para a modelagem arquitetural e que funcione como um guia decisivo sobre o projeto. Os clientes também beneficiam-se quando há um método factível. Eles podem ter um entendimento mais claro das possíveis mudanças de comportamento do sistema e ter garantias quanto a cobertura das necessidades de negócio especificadas.*

Durante a modelagem arquitetural, os requisitos funcionais bem como a similaridade e variabilidade são usados para identificar, caracterizar e fundamentar a família de software. *Modelos de projetos* usualmente representam abstrações que levam em consideração detalhes da implementação. Já os *modelos arquiteturais* oferecem tipicamente, perspectivas no nível conceitual. Uma perspectiva comum inclui a organização das funcionalidades em subsistemas e componentes (chamada visão lógica) [7].

O primeiro problema em aberto para o qual busca-se solução que apresente comprovada melhoria é fazer esses dois modelos integrarem-se de forma robusta, como se fossem um só, mas ao mesmo tempo, manter as diferenças necessárias para viabilizar os vários níveis de abstrações compatíveis com o tamanho, complexidade e domínio da família de software [7, 23, 24]. A integridade e compatibilidade entre esses modelos devem ser garantidas e preservadas ao longo da linha de produto, de forma que se o modelo conceitual mudar, tenha o reflexo esperado no modelo de projeto e vice-versa. Da mesma forma, caso os modelos sejam reusados em uma outra linha de produto, o impacto da adoção deve ser previsto e passível de adaptações.

Apesar de não ser o foco de investigação dessa tese, associado ao primeiro, há um **segundo problema** coexistente que é o rastreamento das decisões tomadas tanto a nível de modelo de projeto como a nível de modelos conceituais [25, 7, 21, 26]. Decisões boas ou ruins têm impacto direto sobre a qualidade, a variabilidade, e a longevidade de implementação da família de software. Se essas decisões forem difíceis de serem identificadas, principalmente em caso negativo, podem resultar em correção de erros de arquitetura a nível de implementação, resultando em um custo elevado de tempo e recursos [7, 22].

Um completo caos pode se estabelecer caso a arquitetura e todos os elementos de projetos forem usados pelos demais produtos da família que por sua vez, podem sofrer centenas de variações. Todos esses fatores de criticidade e impacto, tanto para o projeto quanto para os arquitetos, fortalecem a justificativa de realização dessa pesquisa no sentido de contribuir com a modelagem arquitetural de famílias de software capaz de oferecer uma evolução intra e inter-familiar.

Devido a natureza do problema, abordagens de solução denotacional e de mapeamento entre modelos podem oferecer resultados com mais ou menos eficiência dependendo da comparação entre o desempenho das abordagens. O estado da arte revela uma concentração de estudos voltados para utilização de MDE como meio para resolver a lacuna existente na modelagem arquitetural de famílias de software, mais especificamente entre o modelo conceitual e de projeto.

Os modelos são o principal artefato de desenvolvimento em MDE. Os de plataforma independente (em inglês, *Platform-Independent Models* - PIM) definem a estrutura e o comportamento do software e os modelos de plataforma específica (em inglês, *Platform-Specific Models* - PSM) estão mais próximos da execução. Por meio dessa infraestrutura é possível manipular modelos de forma uni ou bidirecioanal (por exemplo, PIM $->$ PSM ou PIM $<->$ PSM) já que o mapeamento entre eles é garantido via metamodelos.

MDE promove o desenvolvimento de modelos em diferentes níveis de abstração, onde os modelos são transformados do alto para o baixo nível com o intuito de serem interpretados ou processados por meio de técnicas generativas de código [27, 28].

A *programação generativa* é uma abordagem de desenvolvimento de métodos e ferramentas que direciona a geração automática de código a partir de especificações em alto nível. Isso significa que parte ou a totalidade de artefatos produzidos durante o ciclo de vida do software é gerada automaticamente [29].

Basicamente, a geração automática de código ocorre por dois métodos elemen-

tares: a tradução e/ou a transformação. A partir de um código fonte de entrada, a *tradução* [3] gera um código de saída preservando a equivalência semântica entre eles, modificando apenas a sintaxe do código de saída. Por exemplo, a estrutura de controle *switch* na linguagem C deve ser traduzida como *case* para um código de saída na linguagem Pascal.

Já a *transformação* [30, 31], a partir de um código de entrada, modifica a semântica e a sintaxe do código de saída em função de alguma propriedade desejada. Um exemplo clássico de transformação é a geração automática de código anotado, onde a anotação incrementa a semântica do código original. Um outro exemplo relevante é a transformação de metamodelo para modelo que ocorre em MDE.

As DSLs [32, 33] são linguagens que fazem uso de ambas as técnicas e podem ser implementadas com base em um vasto conjunto de abordagens tais como interpretadores, compiladores, preprocessadores, de forma embarcada, compiladores/interpretadores extensíveis, de forma híbrida, etc. A disciplina de DSLs é ponto de interseção entre várias outras áreas de pesquisa que a utilizam como arcabouço, tais como: linguagem de programação, MDE, programação generativa, programação orientada a linguagens e SPL [34].

Mesmo sendo uma área de pesquisa bem estabelecida e amplamente discutida, ao levar esse arcabouço de arquitetura de software para o contexto de SPL [35, 10], mais especificamente DSL como nicho de família de software, há muito o que avançar. No contexto de DSL [27], existe uma demanda por parsers genéricos, construção de arvores de sintaxe abstrata, análises de fluxo de controle, fluxo de dados, análises customizadas, transformações e *front-end* de linguagens legadas (HTML, XLM, lógica de primeira ordem, lógica temporal, etc.).

Diretamente, SPL e MDE contribuem para a redução de custo e agilidade no processo de desenvolvimento e manutenção de parsers, analisadores e tradutores/-transformadores não apenas entre linguagens da mesma família, mas entre dialetos (SQL para SGBD Oracle, MySQL, Postgre) e entre famílias diferentes de linguagens (paradigmas como orientado a objeto versus objeto relacional). Secundariamente, após a linguagem criada, é necessário definir um *framework* para criar aplicações que atendam a variabilidade do ambiente de produção como sistemas operacionais, hardware, banco de dados, etc. O *framework* e o ambiente de produção serão artefatos reusados por meio das várias linhas de produto [36, 37].

Apesar de SPL e MDE terem surgido em paralelo e com linhas isoladas de de-

senvolvimento, desde o ano de 2005, com a publicação de Czarnecki *et al.* [38], é possível observar na comunidade científica uma crescente discussão entre os pontos de interseção das áreas dando origem a uma fusão chamada, em inglês, *Model-Driven Software Product Line* (MD-SPL).

O estado da arte concentra vários trabalhos que investigam a integração de SPL e MDE por meio de práticas pontuais para obter otimização de processos [39, 40, 41, 42, 43]. No geral, esses trabalhos são, por exemplo, de SPL com adoção de práticas MDE, ou o inverso, focam nas abordagens MDE e integram técnicas SPL. Essa abordagem é muito útil quando o problema a ser investigado exige uma fragmentação das abordagens. Por outro lado, fragmentações podem dificultar a gestão dos projetos e isso desencadeou um esforço na comunidade científica por uma abordagem controlada e sistemática de integração que é foco de MD-SPL.

A figura 1.1 apresenta um mapa dos conceitos relacionados ao problema discutido, a conexão entre os mesmos como forma de instanciar uma representação do escopo de investigação dessa tese e fornece fundamentos para a próxima seção que aborda uma análise dos trabalhos relacionados.

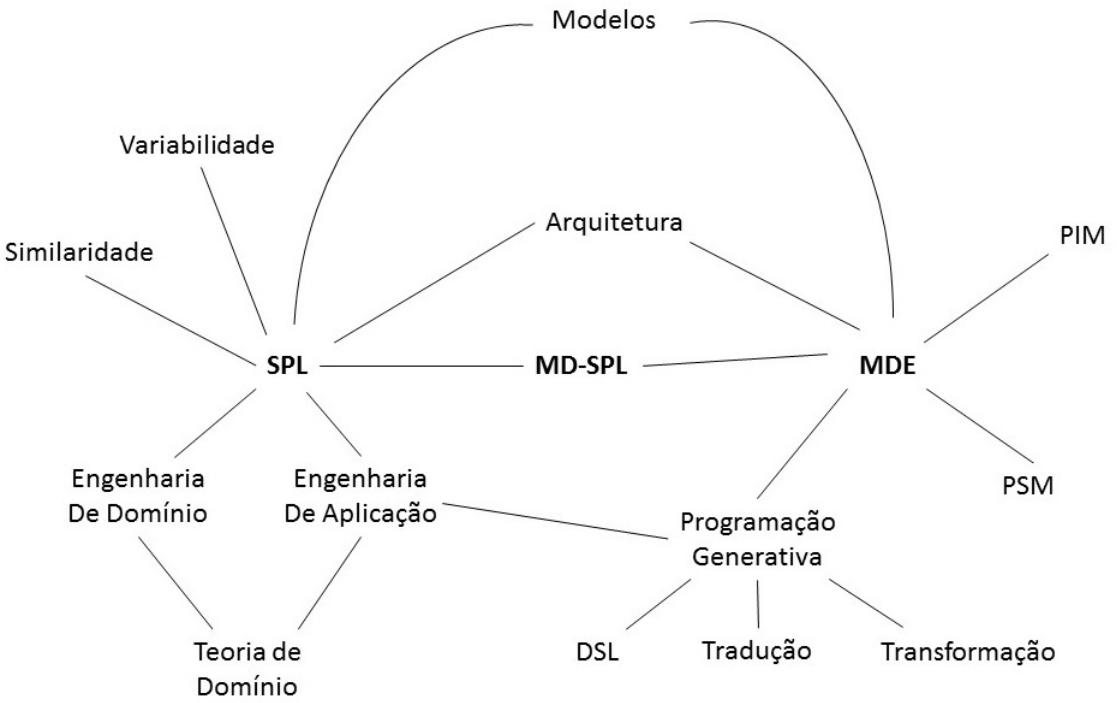


Figura 1.1: Mapa de Conceitos associados à tese.

Estudos comparativos entre SPL e MDE [44, 45, 43, 46] revelam que uma das diferenças entre os dois paradigmas é que enquanto para MDE o conhecimento sobre

reúso é representado como *modelo e suas várias formas de transformação para o desenvolvimento de software*, para SPL, o conhecimento sobre reúso é usado com o intuito de produzir uma *família de software em um dado domínio*.

Com base nas evidências coletadas na revisão da literatura, discutido no Apêndice A, as engenharias (SPL+MDE) se complementam e fornecem à engenharia de software, dentre outras abstrações, a de agregação (família de software) e desdobramento (modelo e suas várias transformações ou configurações), o que resultou na reflexão das questões de pesquisa, discutidas na Seção 1.3.

1.2 Motivação e Justificativa

A motivação para realização desse trabalho está no fato de contribuir com dois grupos: (i) a comunidade de MDE e SPL, interessada em criar/evoluir modelos computacionais que representem abstrações do mundo real, com o objetivo de entender como modelos, metamodelos e métodos podem ser otimizados, e (ii) a comunidade de arquitetura e reúso de software, com interesse em organizar melhor camadas de distribuição de serviços e obter componentes mais eficientes que atendam a demanda ao longo de todo o ciclo de vida do software.

A justificativa para a realização desse trabalho está alinhada com a demanda crescente pela definição e melhoria contínua de processos de software, com foco na qualidade e, consequentemente, aumentar a previsibilidade de que abordagens são mais adequadas, ou não, em um dado cenário de projeto, diminuir prazos, custos e minimizar riscos.

1.3 Questões de Pesquisa

A literatura técnica concentra várias abordagens de modelagem para C&V, porém o que se observa é a carência de uma abordagem que auxilie os projetistas a documentar a forma de integração entre as engenharias e a variabilidade no espaço do problema integrada a similaridade no espaço da solução[1] permitindo identificar, representar, delimitar e escolher mecanismos de implementação. Essa percepção deu origem às seguintes questões de pesquisa investigadas nessa tese:

[1] Os termos *espaço do problema* e *espaço da solução* foram introduzidos por Czrneck e Eisenecker [47]. Espaço do problema diz respeito as especificações do sistema estabelecidas durante a análise do domínio e engenharia de requisitos. Já o espaço da solução refere-se ao sistema concreto gerado a partir das fases de projeto e implementação [11].

Questão de Pesquisa 1. *A variabilidade da modelagem arquitetural influencia na eficiência e eficácia das provas de conceito?*

Questão de Pesquisa 2. *Como o framework arquitetural proposto pode auxiliar o reúso de software?*

Questão de Pesquisa 3. *Como a integração entre as engenharias (SPL+MDE) pode influenciar no desenvolvimento de sistemas mais eficientes e incrementais?*

Questão de Pesquisa 4. *Qual o impacto da variabilidade da modelagem arquitetural sobre o método proposto?*

1.4 Objetivos

O objetivo geral desse trabalho é propor um método de suporte à modelagem arquitetural de famílias de software com base em engenharia dirigida por modelos que ofereça suporte ao processo de documentação e comunicação das escolhas arquiteturais feitas pelos projetistas.

Em linhas gerais, esse objetivo geral pode ser decomposto nos seguintes objetivos específicos:

1. Esquematizar a modelagem arquitetural de famílias de software dirigidas por modelos;
2. Formalizar um framework arquitetural que siga o esquema de integração entre MDE e SPL;
3. Especializar o método proposto para o domínio de famílias de DSLs;
4. Demonstrar o método proposto via provas de conceito;
5. Apontar vantagens e desvantagens sobre o método proposto; e
6. Extrair lições aprendidas sobre as aplicações práticas.

1.5 Metodologia de Pesquisa

A Figura 1.2 sumariza a metodologia percorrida ao longo da pesquisa com base em oito etapas:

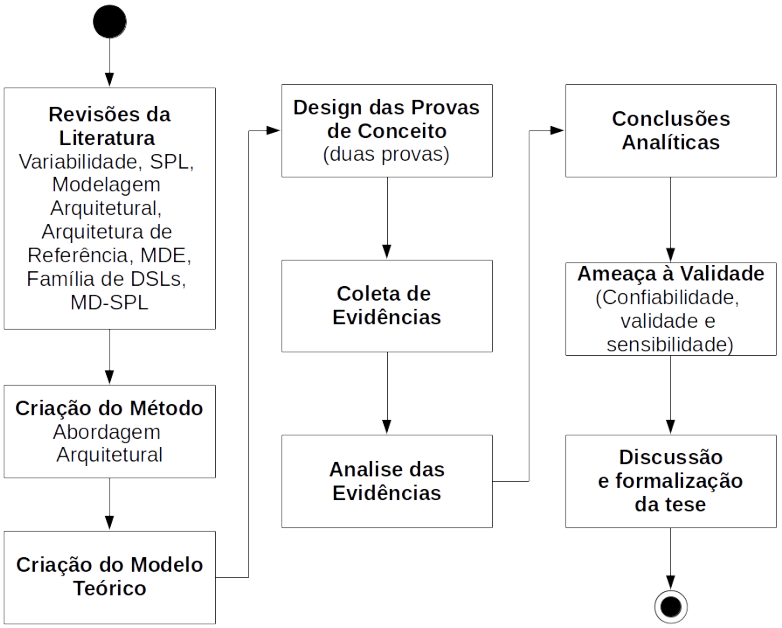


Figura 1.2: Método de Pesquisa.

❶ **Revisões da Literatura.** Abordagens existentes foram identificadas bem como desafios associados.

❷ **Criação do modelo teórico.** Baseado nas questões de pesquisa, o modelo teórico representa conceitos relacionados a pesquisa e seus respectivos relacionamentos. A Figura 1.3 apresenta as variáveis latentes e observáveis com destaque para o link entre as mesmas e as questões de pesquisa. As variáveis latentes descrevem conceitos teóricos que não podem ser diretamente mensurados por serem abstratos, ou por serem novos a ponto de não disporem de um método adequado de avaliação. As variáveis observáveis definem formas de mensuração das variáveis latentes e tem múltiplos indicadores empíricos. A Integração *SPL + MDE*, o *Framework Arquitetural* e as *DSLs* representam a *causa* e são destacadas como variáveis independentes. a *Variabilidade da Modelagem Arquitetural* representa o *efeito*, ou seja, uma variável dependente. A relação causal entre variáveis independentes/dependentes permitem a observação da relação causa/efeito e da definição de proposições, que podem ser testadas apenas pela avaliação das relações entre as variáveis observáveis.

O método, discutido no Capítulo 5, oferece suporte a sistemas cujos *requisitos funcionais* contemplem a transformação e/ou configuração de modelos e dentre os *requisitos não-funcionais*, tenha a presença da variabilidade seja no espaço do problema ou da solução e o reúso da similaridade, de preferência, com base em uma família de software estabelecida. A quantidade de provas de conceito foi definida com

base na necessidade de investigar prioritariamente os requisitos funcionais. Por isso, foram definidas duas provas de conceitos: a CarbonQL que evidencia a relação entre os conceitos do ponto de vista da *transformação* de modelos e a PPL que evidencia a relação entre os conceitos do ponto de vista da *configuração* entre modelos.

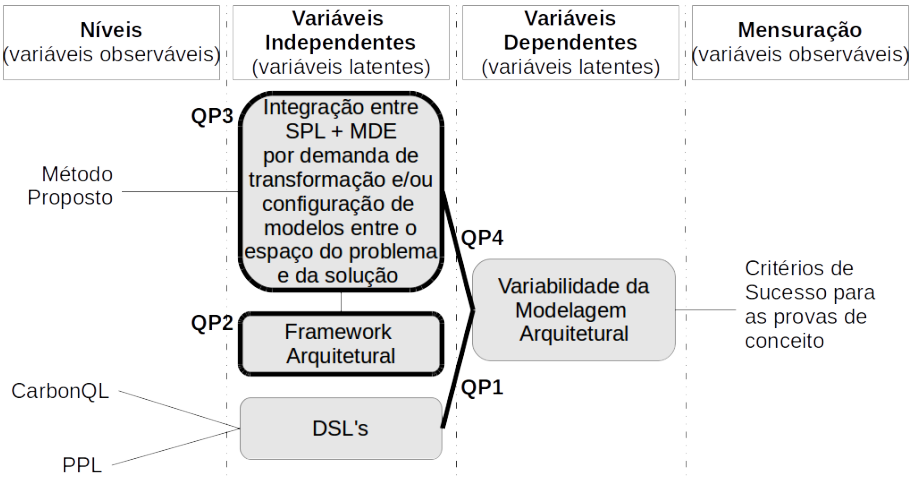


Figura 1.3: Modelo Teórico e as questões de pesquisa.

❸ **Provas de Conceito.** Foram definidas duas provas de conceitos: a modelagem arquitetural da família CarbonQL, que evidencia uma aplicação prática do método a partir da *transformação* de modelos enquanto funcionalidade essencial e a modelagem arquitetural da família PPL que evidencia a *configuração* entre modelos enquanto principal funcionalidade.

❹ **Coleta e Análise de Evidências.** Dados foram coletados sobre as provas de conceito com base nos critérios empíricos de análise definidos.

❺ **Delineamento das conclusões.** De posse dos resultados obtidos, foi traçada uma conclusão sobre o modelo teórico definido e a identificação de novas oportunidades de pesquisa.

❻ **Ameaças à validade.** O viés do pesquisador é certamente uma ameaça à validade, ainda mais quando o resultado da pesquisa é a análise do pesquisador sobre os dados. Partindo do princípio que a pesquisa deve ser reprodutível por outros pesquisadores, as ameaças foram identificadas a partir da análise do método de pesquisa (se o mesmo realmente fornece informações sobre a variável conceitual definida no modelo teórico) e da *sensibilidade* (mudança dos dados em função da mudanças na variável conceitual).

❼ **Discussão.** Conduzidas com base nas questões de pesquisa.

As etapas macro discutidas acima foram realizadas com base na metodologia

ARDev (*Agile Research Development*) [1] que deu suporte à execução de cada uma delas por meio do seu arcabouço de gestão fundamentado em em três fases incrementais e iterativas: (i) concepção, (ii) construção e (iii) avaliação do framework proposto, conforme Figura 1.4.

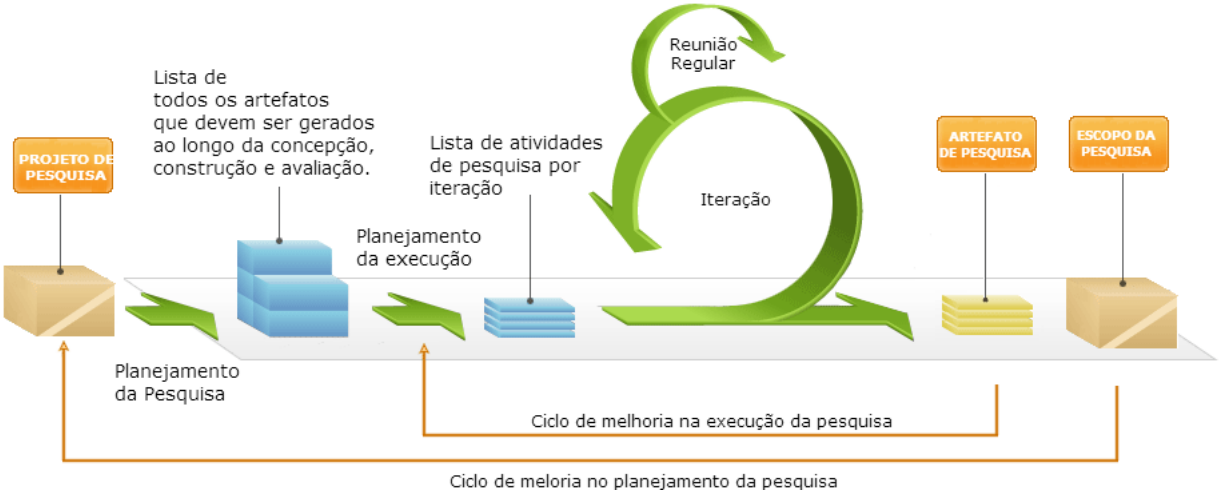


Figura 1.4: Projeto da pesquisa usando ARDev [1].

A fase de concepção (ou pré-produção) foi marcada pela identificação das arquiteturas de referência, bem como a identificação de suas características e componentes. Foi feita a compilação de um conjunto de requisitos relevantes do ponto de vista arquitetural para direcionar a nova arquitetura.

A proposta do framework arquitetural se concretizou por meio de um processo incremental seguindo as etapas de: (1) análise, (2) projeto, (3) implementação e (4) avaliação prática.

A revisão da literatura técnica foi o artefato de entrada para a fase de construção, gerando como saída a proposta de um framework, parte integrante do método de integração entre linha de produto de software e engenharia guiada por modelos, que gradativamente foi refinado com as várias iterações da fase de construção, caracterizando um desenvolvimento orientado pela concepção.

A fase de avaliação (ou pós-produção) compreende (1) planejamento das provas de conceito, (2) execução, (3) avaliação e (4) análise e interpretação dos resultados. A arquitetura em si, gerada pelas fases de concepção e construção, é a entrada para a fase de avaliação. A interpretação dos resultados e novas descobertas representam a saída e respectivas contribuições científicas desse trabalho.

1.6 O Método Proposto

O método de integração entre as engenharias e elicitação da variabilidade da modelagem arquitetural é baseado em dois elementos centrais: (i) o esquema de integração entre as engenharias (SPL+MDE) e (ii) o framework arquitetural.

A Figura 1.5 representa um esquema conceitual do método por meio de uma representação que integra notações de diagrama de blocos ou camadas (no topo) e diagrama de fluxos. As setas representam as possíveis relações entre os elementos de cada engenharia e ilustra o processo de integração entre as engenharias. As fases da engenharia de domínio e o respectivo artefato produzido por elas servem como entrada para as fases da engenharia de aplicação que contempla um refluxo para atualizar os artefatos gerados pela engenharia de domínio, caso novos requisitos sejam identificados. Outros dois refluxos de customização estão presentes respectivamente nas fases de *análise de projeto* e *integração e teste*. A MDE integra o processo dando vazão aos modelos produzidos nas etapas de engenharia de domínio/aplicação, de forma que eles sejam transformados como parte integrante da solução.

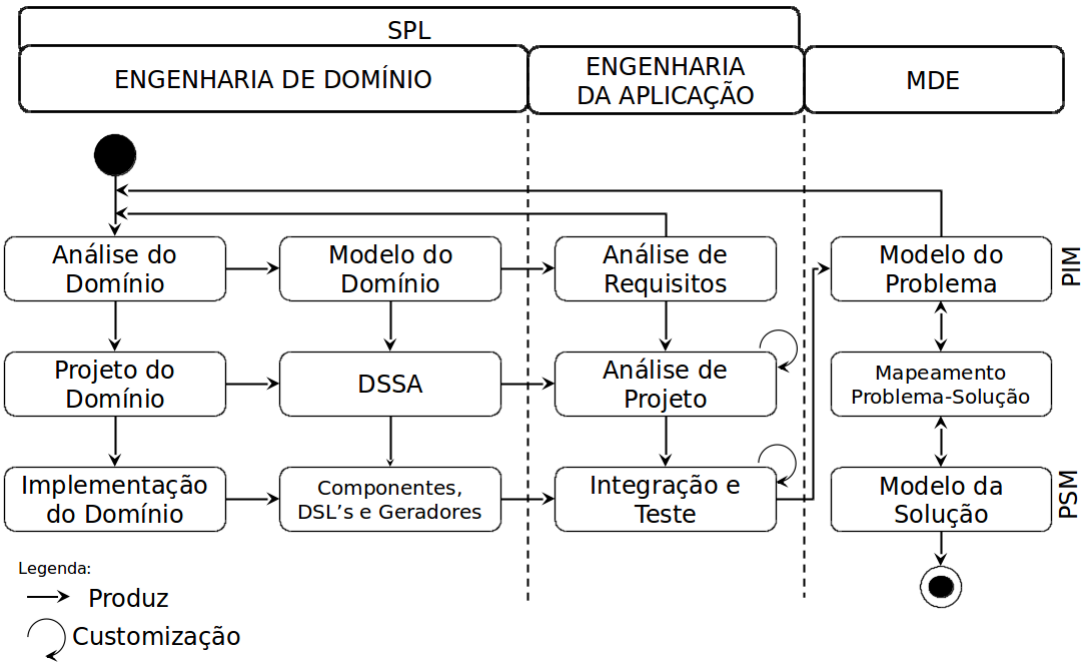


Figura 1.5: Integração entre engenharias de domínio, aplicação e MDE.

Com o intuito de exemplificar essa integração, suponha que na camada da solução PSM (*Platform Specific Model*) tenha um *framework* que dê suporte a uma família de software com o requisito comum de mapeamento objeto-relacional. Na camada

do problema, é possível criar uma linguagem de domínio específico que contemple a **variabilidade** do domínio específico, tema de investigação da *Prova de Conceito-1* (CarbonQL), discutida no Capítulo 7 dessa tese, na qual o domínio específico é representado pela dinâmica florestal, área em que os engenheiros florestais desejam ter autonomia na elaboração de questões relativas à observação de dados da floresta coletados por meio de inventários florestais e armazenados em banco de dados científico.

Mas para isso, é necessário ter uma linguagem de consulta de domínio específico (*Domain-Specific Query Language* - DSQL) que dê suporte aos questionamentos dos engenheiros florestais e que contemple estruturas que facilitem as consultas. Por meio da arquitetura proposta, consultas elaboradas na DSQL (que contempla a variabilidade) podem ser traduzidas para uma linguagem intermediária de mapeamento objeto-relacional (foco denotacional da engenharia de domínio/aplicação). O *framework* objeto-relacional, recebe essa consulta e a transforma mais uma vez em baixo nível, para um modelo (código em SQL) em nível de execução (foco denotacional de MDE), caracterizando aqui a interface de similaridade e viabilizando o reúso de software e todo o arcabouço otimizado para esse propósito, preparado para evitar retrabalho.

Perceba que se o modelo da solução (a nível de *framework* - PSM) evoluir, o impacto a nível de modelo do problema PIM (*Platform Independent Model*) é refletido na análise do domínio que, por sua vez, pode produzir um novo modelo do domínio e, em "efeito cascata", enquadrar todos os artefatos ao novo cenário. O mesmo acontece caso a modificação seja no espaço do problema, a nível de modelo do domínio. Eis o ganho do esquema de integração proposto diante dos demais que não esclarecem como se dá o ciclo de vida dos componentes internos.

Visando facilitar a integração entre as engenharias, o *framework* arquitetural proposto oferece um conjunto de procedimentos comportamentais e estruturais que correspondem às diferentes fases de integração. Além disso, a flexibilidade provida por essa divisão de fases permite que as implementações de cada uma delas sejam substituídas, estendidas ou reutilizadas em novas combinações. Assim, por meio do uso do *framework* arquitetural, e possível escolher, desenvolver e testar rapidamente muitas abordagens usadas para a integração entre as engenharias e recuperar informações arquiteturais adotadas para um domínio.

A Figura 1.6 ilustra duas linhas de produtos de software, as DSL's CarbonQL e PPL, bem como outras "N linguagens" que demandem do mesmo nível de varia-

bilidade entre versões e o processo de geração das mesmas via o método proposto. A ideia é reaproveitar artefatos na camada de projeto arquitetural e código fonte. No entanto, vários outros artefatos podem ser evoluídos e ter controle sobre a sua variabilidade inter e intra linha de produto.

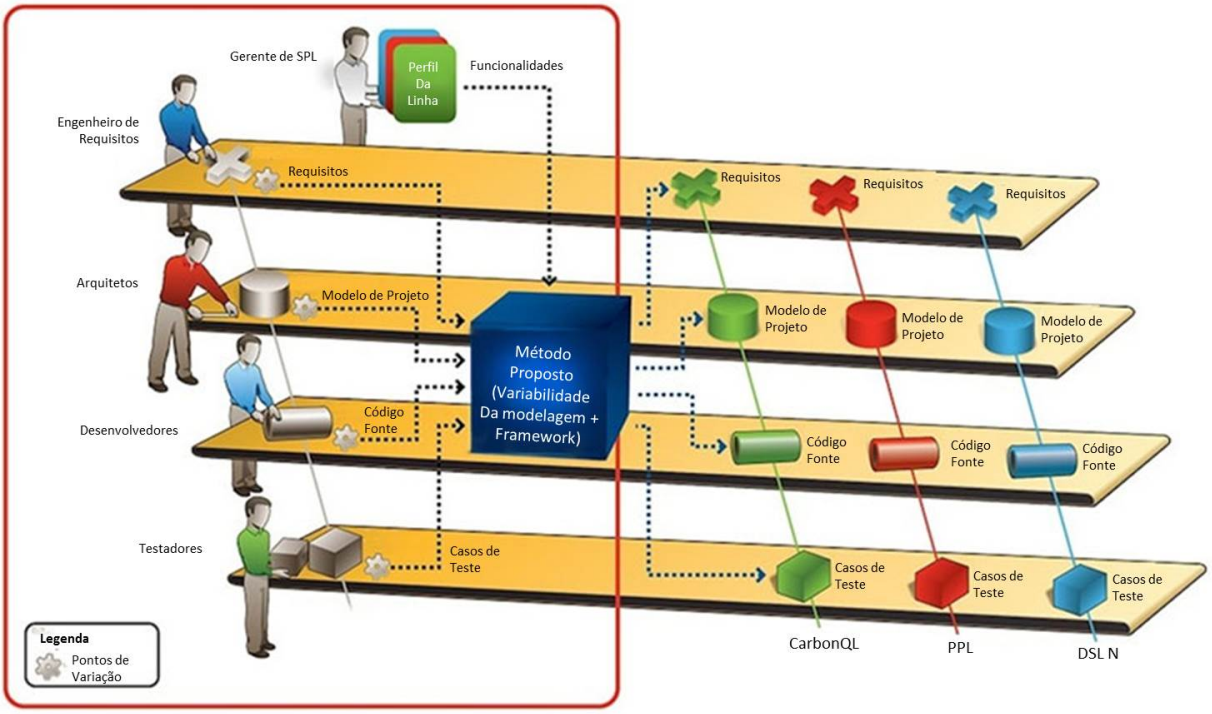


Figura 1.6: Método proposto e as aplicações do mesmo em linhas de produto de DSL's (CarbonQL e PPL).

1.7 Contribuições

O esquema de integração entre as engenharias e o framework arquitetural compõem a principal contribuição dessa tese. No entanto, vale destacar as seguintes contribuições complementares:

1. **Análise da variabilidade da modelagem arquitetural:** A percepção endógena e exógena esclarece a margem de evolução da arquitetura e auxilia na escolha pelo reúso, modificação e avaliação de arquiteturas e se o sistema resultante condiz com arquitetura.

2. **Protótipo da CarbonQL:** uma linguagem de consulta de domínio específico (*Domain-Specific Query Language* - DSQL) com foco na transformação, componentização e reúso;

3. **Protótipo da PPL:** *Permanent Parcel Language*, uma linguagem de especificação de parcelas permanentes e do planejamento para a realização do inventário florestal com foco na transformação, configuração, componentização e reúso;

4. **A Carbontology:** uma ontologia formalmente definida em OWL-DL, cujo escopo é a dinâmica florestal e estimativa de carbono;

5. **Metodologia ARDev:** *Agile Research Development*, uma metodologia para identificar atividades, planejar ciclos de pesquisa, respondendo muito mais a mudanças do que simplesmente seguir um planejamento e com o objetivo de organizar a comunicação entre os pesquisadores. A ARDev ajuda a entender fatores que afetam a produtividade da pesquisa e consequentemente, pode ajudar a concentrar esforço de gerenciamento onde realmente é necessário.

1.8 Organização da Tese

Essa tese está organizada em três partes.

A Parte I, Fundamentação Teórica, corresponde ao estudo realizado e concentra as referências mais relevantes para esse trabalho.

- **Capítulo 2: Modelos Computacionais e Modelagem.** Aborda conceitos relacionados à configuração dinâmica dos modelos. Esse tópico é particularmente relevante para o contexto desse trabalho, pois está relacionado com as questões de pesquisa.

- **Capítulo 3: Arquitetura de Famílias de Software e a Modelagem da Variabilidade.** Enuncia o potencial da arquitetura de software e um conjunto de boas práticas para garantir a qualidade da formalização. Princípios e benefícios sobre a modelagem da variabilidade, bem como dificuldades associadas ao reúso de software também são discutidos.

A Parte II, Descrição Metodológica, trata da orientação metodológica utilizada para a condução desse trabalho.

- **Capítulo 4: Instrumental de Pesquisa.** Destaca os métodos, técnicas e ferramentas utilizadas.

A Parte III, Apresentação, Análise e Interpretação dos Resultados, empacota o estudo para a repetição e esclarece vantagens e desvantagens do mesmo.

- **Capítulo 5: Método Proposto.** Apresenta o método de suporte à modelagem arquitetural de famílias de software com base em engenharia dirigida por modelos baseado em dois elementos centrais: (i) o esquema de integração entre as engenharias (SPLE+MDE) e (ii) o framework arquitetural.

- **Capítulo 6: Introdução às Provas de Conceito.** Apresenta detalhes sobre o domínio das provas e informações preliminares, essenciais ao entendimento dos dois capítulos subsequentes.

- **Capítulo 7: Prova de Conceito 1 - Linguagem CarbonQL.** Apresenta a execução sistemática do método proposto, a modelagem arquitetural para a família da CarbonQL e aspectos de implementação com ênfase para as vantagens e desvantagens observadas na prática.

- **Capítulo 8: Prova de Conceito 2 - Linguagem PPL.** Apresenta a execução sistemática do método proposto, a modelagem arquitetural para a família da PPL e aspectos de implementação com ênfase para as vantagens e desvantagens observadas na prática.

- **Capítulo 9: Avaliação.** O método proposto é avaliado por meio das provas de conceito. Ao total são oito critérios de avaliação, classificados dentro de quatro dimensões: (i) integração entre as engenharias, (ii) *framework* arquitetural, (iii) variabilidade da modelagem arquitetural e (iv) qualidade do projeto arquitetural.

- **Capítulo 10: Trabalhos Relacionados.** Apresenta um estudo comparativo entre os trabalhos existentes e essa tese. Basicamente eles estão agrupados por métodos de modelagem arquitetural e métodos de suporte à decisão arquitetural.

- **Capítulo 11: Discussão e Conclusões.** Apresenta as conclusões do trabalho, discussão sobre a contribuição inovadora do trabalho, ameaças à validade, limitações e trabalho futuro, seguido pelas Referências.

- **Apêndice A: Revisão da Literatura.** Apresenta o método de revisão e os resultados obtidos.
- **Apêndice B: A Carbontology.** Apresenta a Carbontology, base de especificação do domínio utilizada pela CarbonQL.

Parte I

Fundamentação Teórica

Capítulo 2

Modelos Computacionais e Modelagem

Esse capítulo responde às seguintes perguntas: Quais são as características relevantes e a relação entre modelo e modelagem de forma a dar suporte evolutivo para uma teoria? Como se dá a mútua relação de benefícios entre a teoria de domínio e a engenharia de linguagens? Qual é a motivação e contribuição da engenharia de domínio na evolução do processo produtivo de software, de artesanal para industrial? Como a teoria de domínio influencia a engenharia de aplicação? De que forma a tradução e transformação de modelos podem ser combinadas? Quais são os principais espaços técnicos que MD-SPL oferece e em qual deles o método proposto se enquadra?

Para tanto, esse capítulo está organizado em sete seções. Inicialmente a Seção 2.1 destaca características relacionadas aos modelos computacionais bem como fatores motivantes para se conduzir uma modelagem sem esquecer dos riscos inerentes a tal processo.

Conforme a Figura 2.1, as demais seções seguem uma organização piramidal, no sentido que certos conceitos são mais teóricos e formam a base para a construção e evolução de conceitos mais emergentes que fundamentam essa tese. A teoria de domínio é discutida na Seção 2.2. A Seção 2.3 aborda SPL. As engenharias de domínio e de aplicação, partes integrantes de SPL e explicadas pela teoria de domínio, fornecem um campo de estudo prático para a transformação e tradução de modelos, discutidas respectivamente na Seção 2.4. A Seção 2.5 aborda MDE, um paradigma de construção e manipulação de modelos, capaz de minimizar a complexidade relacionada à modelagem computacional. A Seção 2.6 aborda MD-SPL.

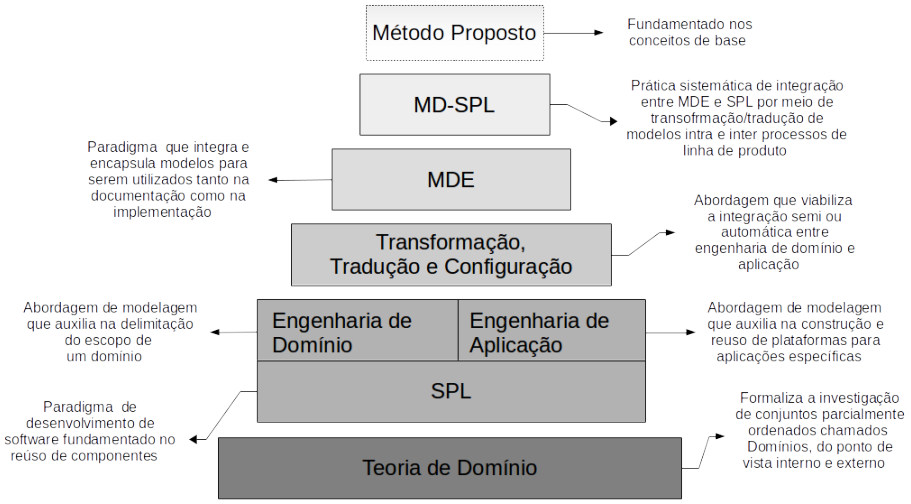


Figura 2.1: Estrutura do Capítulo 2.

Finalmente, a Seção 2.7 apresenta algumas considerações.

2.1 Conceitos e motivações

Modelos implementam teorias, fornecem provas de suficiência e respondem perguntas sobre as teorias. Dessa forma, um dos questionamentos mais comuns atrelados aos modelos é se alguns fenômenos podem emergir, ou serem explicados, por tais princípios. Além disso, eles podem ajudar a compreender processos, mesmo quando a teoria não é detalhada o suficiente para se construir um modelo [48].

Também representam uma forma de obter, dentre outros benefícios, a elicitação de características e a previsão de comportamentos. A constatação desse fato pode ser analisada por meio de estudos de observação e validação nas quais pesquisadores fazem uso de modelos computacionais para descrever, implementar, testar e consequentemente obter experimentos sobre teorias, capazes de fornecer entendimento para áreas de conhecimento que não estão diretamente relacionadas à teoria original [49].

Uma outra forte característica é que os modelos não têm necessariamente que capturar os fenômenos ou processos como eles são. Modelos computacionais estão abertos a uma gama muito maior de experiências. Eles podem também ser expostos a entradas não naturais, ou manipuladas de uma maneira que não pode ser feito, por exemplo, com seres humanos [50]. Isso os torna uma ferramenta ideal para investigar

fenômenos que nem sempre podem ser estudados empiricamente em ambiente real.

As aplicações práticas ou experimentos comportamentais dos modelos são mutuamente informativos. Estudos comportamentais avaliam previsões e fornecem uma métrica para comparar modelos. Schlesinger e McMurray [48] comentam que para uma modelo ter sucesso ele deve oferecer uma explicação em linguagem científica, sem prender-se a jargões de modelagem e enfatiza que modelos podem construir níveis de análise, por exemplo, genes-neurônios, neurônios-cérebro, cérebro-comportamento e de volta novamente formando um ciclo de análise.

Sendo assim, os modelos computacionais podem desempenhar um papel crítico no desenvolvimento de explicações coerentes que abrangem vários níveis de análise. Lane e Gobet [51] argumentam que a concepção de um modelo computacional útil e aplicável restringe-se a um domínio específico. Modelos mais abstratos são usados para responder questões mais teóricas, exercendo assim um papel crucial como parte do processo científico-dedutivo, ou seja, quando bem sucedidos, os modelos fornecem uma prova de demonstração que a teoria é plausível ou possível, e talvez o mais importante, fornecem parâmetros que ajudam a rever teorias. Ao mesmo tempo, os modelos computacionais não apenas reproduzem descobertas centrais de experimentos críticos, mas também geram novas previsões e sugerem novos testes [49].

Uma das contribuições mais importantes dos modelos computacionais é conduzir a teoria em uma nova direção e ajudar na análise de consequências não óbvias das mesmas. Assim como os modelos computacionais tornam-se incrivelmente poderosos e complexos, é igualmente importante que eles tornem-se acessíveis e compreensíveis, para que possam efetivamente contribuir com a interconexão entre novos modelos e teorias [49]. É exatamente nesse ponto que entra o papel da modelagem.

Modelagem é a atividade de construir modelos tendo como motivação fundamental a capacidade de oferecer ao projetista uma liberdade relativa aos mecanismos de integração entre o modelo e o seu ambiente de aplicação, viabilizando dessa forma a observação das consequências, por vezes não óbvias, de tais escolhas feitas pelo projetista [14, 51].

O esforço desempenhado por pesquisadores que desenvolvem modelos para definir explicitamente construções teóricas e as relações causais entre as mesmas também representa uma motivação para o uso da modelagem como uma ferramenta de investigação. Nem sempre é possível identificar com antecedência o impacto de certos compromissos teóricos, particularmente quando se desenvolve ou se faz uso de

múltiplos mecanismos de interação. Em alguns casos, a única maneira de compreender as suas implicações é formalizá-los matematicamente [49].

Daniluk [2] comenta sobre um possível fluxo de formalização, ilustrado na Figura 2.2 que inicia com a modelagem matemática. Após o tratamento numérico é possível criar um algoritmo. A atividade de implementação produz um modelo computacional implementado que permitirá a visualização de resultados obtidos com o processamento de tal modelo passíveis de análise, interpretação e aplicação prática.

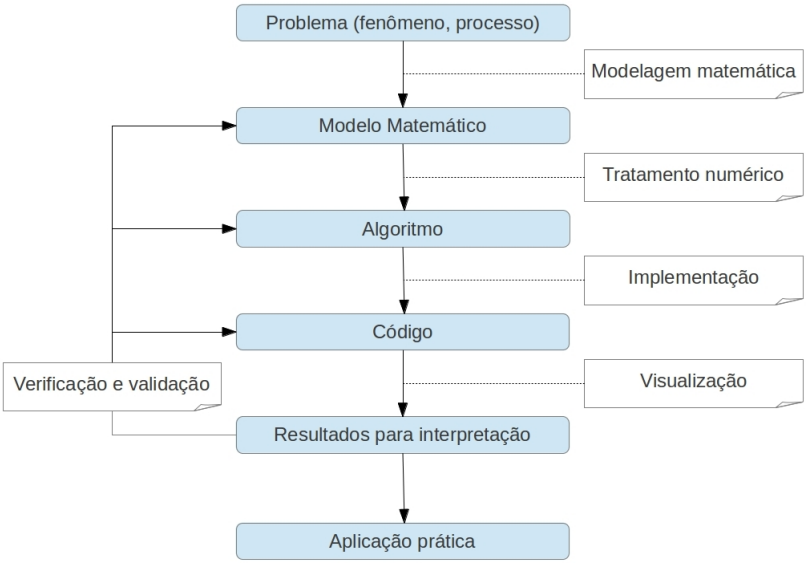


Figura 2.2: Tradicional abordagem entre a modelagem matemática e computacional, adaptada de Daniluk [2].

Todas essas características e fatores motivantes não querem dizer que a modelagem computacional é isenta de riscos. Além das habilidades técnicas, a modelagem exige do pesquisador uma habilidade teórica complexa.

Do ponto vista das contribuições da modelagem a uma determinada teoria, uma das mais relevantes é viabilizar que, no início, a construção do modelo computacional comece com uma estrutura relativamente simples e se torne progressivamente elaborada por meio de um processo iterativo de projeto, teste e refinamento. A modelagem está no seu melhor caso quando nos permite entender a gama de possíveis mecanismos que podem dar origem a um comportamento e porque.

2.2 Teoria de Domínio

A teoria de domínio foi originalmente proposta por Scott [52] cuja preocupação era demonstrar como a teoria pode ser baseada em representações elementares, logicamente construída e evoluída para observar propriedades de modelos computacionais tais como: se o modelo é conciso, não ambíguo, aberto à análise matemática, verificável automaticamente, executável, legível, etc. Estudos na área de linguagens de programação abriram portas para uma geração de investigações da teoria de domínio, particularmente do ponto de vista da semântica denotacional, que visa atribuir significado às construções das linguagens de programação ou especificação [53]. Nessa abordagem, os significados são modelados por objetos matemáticos, geralmente funções semânticas que denotam/mapeiam estruturas da linguagem para funções parciais que representam o comportamento dos programas, dando origem ao nome da abordagem chamada *denotacional*.

De acordo com Abramsky e Jung [54], os estudos em linguagens de programação fortalecem e confirmam a teoria de domínio em dois aspectos:

- **Investigação Individual de Domínio:** propriedades como convergência, aproximação e topologia são analisadas;
- **Investigação Coletiva de Domínio:** técnicas de comparação de domínios são analisadas juntamente com construções finitas (produto cartesiano, etc.), construções infinitas (limites, colimites [1], etc.), domínios universais, bem como características axiomáticas e sintáticas.

Sendo assim, a teoria de domínio representa para essa tese o elemento piramidal base, capaz de fundamentar e sistematizar implicações que gerem novas observações e análise lógica para o domínio selecionado. A partir dessa teoria, surgiram duas abordagens que se complementam. A primeira permite compreender o contexto do problema, chamada engenharia de domínio. A segunda permite compreender o espaço da solução para o problema, chamada engenharia de aplicação. Ambas foram incorporadas por SPL, tema de discussão da próxima seção.

[1] Cone e cocone são noções abstratas usadas para definir limite e colimite segundo a teoria das categorias, que por sua vez, embasa o conceito de morfismo e suas derivações.

2.3 Linha de Produto de Software (SPL)

Linha de Produto de Software, em inglês, *Software Product Line* (SPL), vem se consolidando como uma alternativa eficiente em termos de custo, qualidade e tempo de desenvolvimento de software por basear-se em componentes que podem ser reutilizados [24, 9]. Uma Linha de Produto de Software consiste em uma abordagem sistemática de planejamento, reúso e rotinas automáticas para a construção de sistemas com menor esforço intra e inter famílias de software.

Nesse contexto, o software é projetado como um conjunto de componentes que integram-se para o reúso, pois os componentes facilitam a sua própria inserção, substituição ou remoção no software. Dessa forma, o reúso de componente torna-se mais eficaz para construção de famílias de software [55]. A manutenção/evolução, também é beneficiada já que torna-se mais fácil manipular o software por unidade (componente) com base em requisitos ou características do sistema [56]. De forma análoga o teste de software também pode ser otimizado para atender grupos de componentes [57].

O reúso é uma estratégia importante para agilizar o processo de desenvolvimento, mas exige planejamento constante. De uma forma geral, o reúso pode ser realizado de três formas: (i) total, onde o software é reutilizado na íntegra; (ii) parcial, onde partes do software são reusadas, como componentes; e (iii) reúso de artefatos, como requisitos, casos de teste, arquitetura, scripts, etc. Também pode ser organizado sob duas abordagens: reúso externo, para construção de outros sistemas; e reúso interno, para construção do próprio software como uma nova versão evolutiva [58].

Apesar de estar muito associado a SPL, vale a pena ressaltar que é possível ter reúso sem SPL, mas não é possível ter SPL sem reúso. O reúso tradicional ocorre por meio de repositórios que armazenam bibliotecas, componentes e algoritmos, em sua grande maioria, não sistematizados para facilitar o reúso em larga escala. Isso leva a equipe de desenvolvimento a gastar tempo e esforço em buscas *ad-hoc* por fragmentos de software que podem ser reusados.

Já no contexto SPL, os versionadores[2] tendem a ser customizados para a instanciação de produtos por meio de mecanismos de similaridade e variabilidade [59]. O ponto de partida é fazer com que os desenvolvedores consigam gerar um produto rapidamente a partir do reúso por meio do mapeamento da similaridade com um pro-

[2]Os versionadores, popularmente conhecidos como sistemas de controle de versão, são responsáveis em gerenciar o repositório, bem como todos os artefatos de projeto. São exemplos de versionadores *open source*: CVS, Git, SVN, Mercurial, etc.

duto existente, ficando apenas a variabilidade para ser efetivamente implementada. Caso uma manutenção seja necessária, a mesma é feita em componentes centrais e replicada para todos os produtos da mesma família [60].

O processo de desenvolvimento SPL ocorre em duas etapas que são: engenharia de domínio e engenharia de aplicação. Ambas serão discutidas nas próximas seções com enfoque denotacional entre as engenharias.

2.3.1 Engenharia de Domínio: uma abordagem para representação do problema

De acordo com Siy e Mockus [61], a engenharia de software tradicional fornece um arcabouço para projetos e processos de desenvolvimento de software enquanto produto individualmente customizável, "peça única", feito sob encomenda, possivelmente reflexo de uma época que o software era muito dependente do hardware. Com o surgimento de empresas de software interessadas em elevar o processo produtivo para uma escala industrial, o conceito de "linha de produção" para o software ganhou força juntamente com questionamentos sobre a viabilidade de tal prática adaptada para esse contexto. Seria realmente possível conciliar as similaridades de uma dada família de software com as solicitações de personalização feitas pelo cliente? Quais são as melhores práticas de reúso de software? Como reduzir o "time-to-market"? Como o ciclo de vida do software pode ser prolongado?

Os métodos tradicionais de projeto e desenvolvimento não oferecem formalismos para se extrair vantagens desse cenário desafiador. Como resultado, os desenvolvedores fazem uso de práticas informais de reúso de projeto, código e outros artefatos, basicamente para adaptá-los a novos requisitos. Isso pode fragilizar muito o software e resultar em uma complexa manutenção já que os componentes reusados não foram concebidos para o reúso.

Como uma forma de auxiliar tal processo, a engenharia de domínio direciona uma sistemática criação de modelos conceituais do domínio, arquitetura e implementação, com ênfase para o reúso. De acordo com Bjørner [15], além do propósito de construir e desenvolver técnicas, ferramentas e princípios para a criação de modelos, a engenharia de domínio também se concentra na pesquisa e evolução dos mesmos de forma que possam representar adequadamente o domínio.

Para conduzir o processo de engenharia de domínio, são necessárias ao menos três fases [17, 62]:

- Análise: o escopo do domínio é definido, requisitos são coletados, analisados bem como a identificação de elementos-chave para a modularização, similaridade, individualidade, combinação e solução de possíveis conflitos.
- Projeto: o modelo do domínio é integrado a uma arquitetura adaptável para que o mesmo seja reusado por aplicações específicas.
- Implementação: criação de um mecanismo conceitual e componentização capaz de viabilizar a aplicação/criação de artefatos reusáveis.

A Figura 2.3 apresenta essas três fases internas da engenharia de domínio. Os produtos dessas três fases são modelos de domínio, modelos de projeto e arquitetura de software de domínio específico (em inglês, *domain-specific software architecture* - DSSA) e modelo de componentes. A motivação para executar as atividades da engenharia de domínio é atingir um incremento ou avanço significativo na produtividade e confiabilidade no contexto do domínio selecionado, uma vez que tais atividades envolvem reúso, representação do conhecimento e validação. Artefatos de domínio podem ser reusados no contexto de uma aplicação particular (aplicando técnicas de adaptação, se necessário). Eles representam o conhecimento adquirido no domínio e podem servir como validação de aplicações.

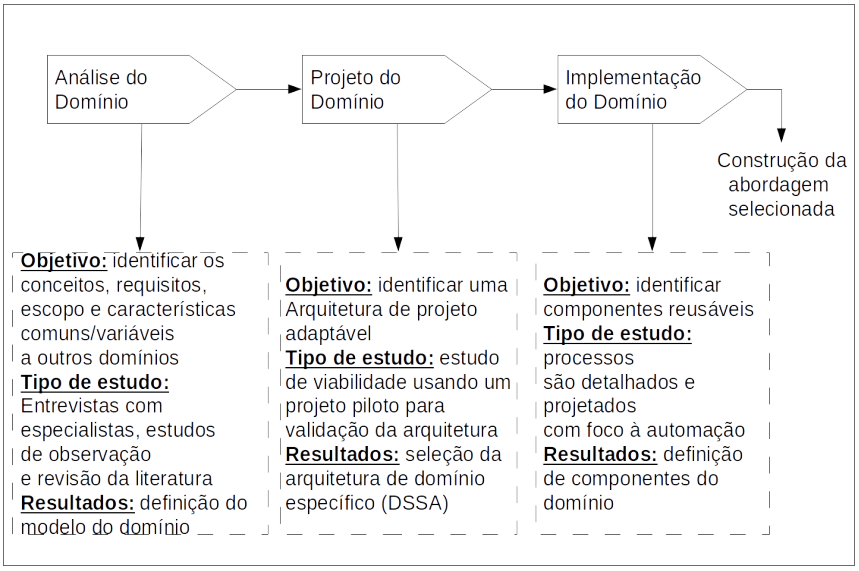


Figura 2.3: Fases da engenharia de domínio.

Existem várias metodologias de engenharia de domínio, dentre as de base, consideradas como metodologias-mãe mais populares temos: ODM (*Organizational Do-*

main Modeling) e FODA (*Feature-Oriented Domain Analysis*) [63] (consultar Asmaa *et al.* [64] para visualizar a evolução das metodologias). Essa tese não tem como objetivo seguir uma metodologia específica e sim abstrair as fases gerais da engenharia de domínio, identificar e adotar boas práticas dentro de um contexto de desenvolvimento guiado por modelos.

2.3.2 Engenharia de Aplicação: uma abordagem para instanciação da solução

A engenharia de aplicação baseia-se nos resultados obtidos na engenharia de domínio para produzir software [61]. De fato, uma importante etapa que faz a engenharia de domínio ser reprodutível e verificável é um claro mapeamento entre as saídas da análise do domínio e as entradas necessárias para se construir geradores de aplicações. Lembrando que denotar/mapear são as palavras que sintetizam a teoria de domínio, que por sua vez, fundamenta a conexão entre a engenharia de domínio e a de aplicação.

De acordo com Frakes e Kang [65], a programação generativa tem forte aderência com a engenharia de aplicação por implementar características relevantes do domínio tendo como base um gerador de aplicação de domínio específico. O gerador de código traduz a especificação do sistema de uma linguagem de origem para uma linguagem de destino de forma automática ou semiautomática, considerando possíveis intervenções feitas pelo programador. A programação generativa fortalece a engenharia de aplicação por meio da teoria de meta-compiladores, também conhecidos como geradores de aplicações. Observe que o gerador assume o papel de elemento denotacional, caracterizando assim, a representação e sistematização da teoria de domínio.

A Figura 2.4 apresenta três processos genéricos oriundos da engenharia de domínio que produzem como resultado a aplicação. Optou-se em apresentar como exemplo a aplicação resultante para o contexto dessa tese, mas vale ressaltar que é necessário consultar o Capítulo 7 para um maior aprofundamento desse processo. O modelo do domínio é instanciado pela Carbontology e CarbonQL, a DSSA é instanciada por camadas de serviços e ações. Já o modelo de componentes do domínio é de integração entre o modelo desenvolvido e o ambiente externo, ou seja, a conexão com um *framework* de mapeamento objeto-relacional, resultando assim em uma nova aplicação e tecnologia reusável.

A engenharia de domínio e a engenharia de aplicação são abordagens que se complementam. No entanto, para essa complementação ocorrer com fluidez, é ne-

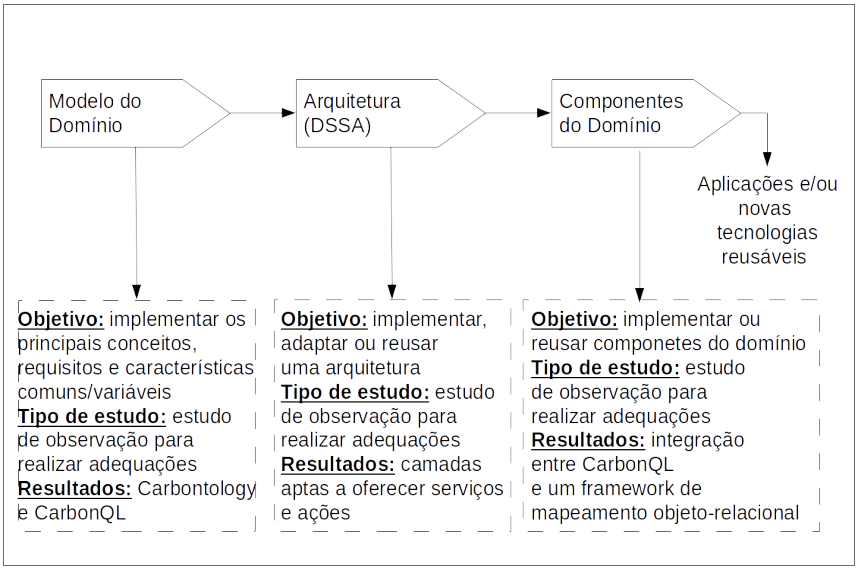


Figura 2.4: Fases da engenharia de aplicação.

cessário um passo intermediário de conexão entre as duas abordagens que pode ser por tradução e/ou transformação. A seguir, as duas abordagens são discutidas.

2.4 Tradução e transformação de modelos

Partindo do princípio que uma linguagem computacional é um meta-modelo capaz de criar conforme o seu suporte sintático/semântico, modelos gráficos ou textuais; a exemplo, programas (código fonte), esquemas, diagramas, especificações, etc. Um tradutor mapeia cada sentença[3] de entrada de um modelo para, a princípio, uma sentença de saída[4] correspondente, mantendo o compromisso semântico que o modelo de entrada tem equivalência com o modelo de saída. Segue o mesmo princípio da tradução linguística presente nos dicionários, por exemplo, uma palavra em português será representada por uma ou mais palavras sintaticamente diferentes em alemão, no entanto, a equivalência semântica entre as palavras deve ser mantida.

A Figura 2.5 ilustra o fluxo básico de um tradutor. A partir dos caracteres de entrada, a análise léxica identifica os *tokens* (símbolos pertencentes a um vocabulário específico). O *parser*, por sua vez, reconhece sentenças estruturadas sobre o resultado fornecido pelo analisador léxico e nesse momento, o trabalho elementar de um

[3]Tecnicamente, uma sentença de entrada representa uma sequência completa implicitamente seguida por um símbolo terminal.

[4]A sentença de saída pode variar em função do tipo de transformação, tais como: um-para-um, um-para-muitos, muitos-para-um, muitos-para-muitos [66].

tradutor é executar ações que gerem uma saída.

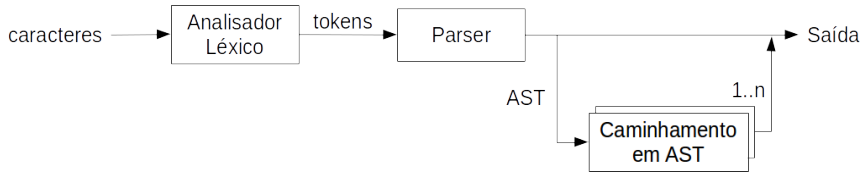


Figura 2.5: Fluxo de tradução, adaptado de [3].

No entanto, existem tradutores com um grau a mais de complexidade cujo objetivo é usar o *parser* para construir uma árvore de sintaxe abstrata (em inglês, *abstract syntax tree* - AST), usadas em geral para computar outras estruturas de dados e informações necessárias para etapas seguintes, como a geração automática de código [3].

Quando não há um compromisso de equivalência entre o modelo de entrada/saída e sim de modificação semântica seja pela adição, remoção, mesclagem, fragmentação do modelo ou qualquer outra ação que tenha como finalidade modificar características ou propriedades, estamos diante de um processo de *transformação de modelos*, que por sua vez pode se basear em regras declarativas ou operacionais. Também pode seguir características unidirecionais, bidirecionais, internas, externas, implícitas, explicitas, temporizadas ou incrementais [67] [66]. A exemplo, suponha que um dado código fonte de um sistema desenvolvido em Java passe por um processo automático de teste. Esse mesmo código pode receber comentários no código automaticamente, de forma que os mesmos sejam úteis única e exclusivamente para a ferramenta de teste. Sendo assim, há uma transformação sintática e principalmente semântica do modelo por meio da adição de propriedades de teste.

A Figura 2.6, adaptada de Ehrig *et al.* [68] ilustra o fluxo básico da transformação de modelos. O artefato de entrada é mapeado[5] para um grafo de entrada, com base em uma sintaxe abstrata[6]. Esse grafo de entrada é processado pelo transformador de grafo, que por sua vez integra as regras de transformação. Ao final desse processamento, um grafo de saída é gerado na sintaxe concreta[7] e posteriormente pode ser mapeado em um artefato de saída. Observe que tanto na entrada como na

[5]O mapeamento pode ser entendido como um processo de "tradução", pois a equivalência semântica entre o artefato de entrada e o grafo de entrada deve ser mantida.

[6]Sintaxe abstrata está, por exemplo, para nodo-aresta. É uma forma de descrição universal para que a especificidade possa ser simplesmente atribuída, mapeada diretamente e facilmente compreendida.

[7]Sintaxe concreta está, por exemplo, para estado-transição de Redes de Petri. A sintaxe concreta faz parte da linguagem-alvo "de compreensão não-universal" e sim de propósito específico.

saída, o processo de transformação pode conter um processo de tradução, ou seja, dependendo do objetivo, transformação e tradução podem se concatenar e combinar independente da ordem (transformação-tradução, tradução-transformação). É nesse ponto que as engenharias de domínio e aplicação fazem uso dessas abordagem como ponto de integração entre os artefatos gerados.

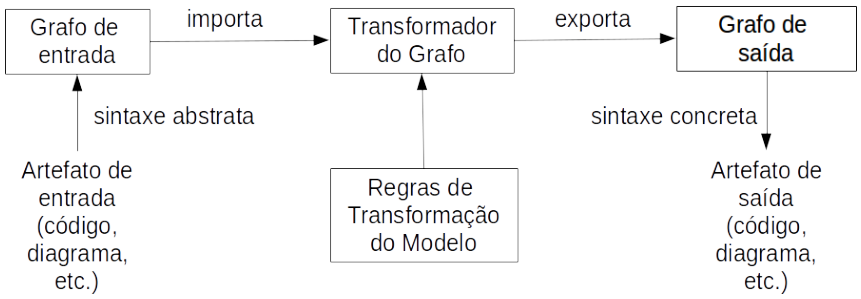


Figura 2.6: Fluxo de transformação.

As aplicações mais comuns da transformação de modelos estão presentes: na geração de código a partir de um modelo, síntese e engenharia reversa entre modelos em diferentes níveis de abstração, geração de instância de meta-modelos, consultas e atualizações simples de modelos, migração de modelos para adaptar os meta-modelos envolvidos, composição de modelos e sincronização dos mesmos [30].

Syriani e Gray [69] defendem a adoção dos padrões de projeto como uma boa prática para garantir a qualidade da transformação de modelos que podem se diferenciar, basicamente em duas categorias: padrões de propósito geral e de domínio específico.

A transformação de modelos é amplamente utilizada como artefato de software com aplicação variando de protótipos a aplicações industriais de larga escala. São requisitos para a transformação de modelos: (i) promover uma solução eficiente para o problema a ser resolvido, (ii) demonstrar alta qualidade de projeto e integração com outros sistemas e tecnologias e (iii) manter sistematicamente o método de transformação desenvolvido de forma a garantir uma continuidade.

Tanto o processo de tradução como o de transformação seguem uma fundamentação teórica robusta e o grande benefício dessa formalização está na facilidade de compreensão, documentação, comunicação e repetição do método. A próxima seção aborda a engenharia guiada por modelos como abordagem que encapsula os conceitos de base, discutidos nas seções anteriores, oferece um conhecimento estruturado e totalmente voltado para a compreensão dos modelos pela ótica da arquitetura de manipulação dos mesmos.

2.5 Engenharia Dirigida por Modelo (MDE)

Um dos grandes desafios que a Engenharia de Software enfrenta é: como construir softwares confiáveis, flexíveis e que se integrem com diferentes tipos de sistemas? Vários métodos são apontados como resposta a essa pergunta. Um deles é engenharia guiada por modelos (em inglês, *model-driven engineering* - MDE).

MDE procura contribuir com seus exemplos de heterogeneidade por meio da automação de modelos entre linguagens de modelagem de forma que modelos sejam transformados do alto para o baixo nível, resultando em um modelo passível de execução, seja por geração de código ou pela interpretação de modelos.

Sendo assim, MDE emerge como um paradigma promissor em engenharia de software por enfatizar o uso de modelos não apenas para fins de documentação e comunicação, mas como artefatos que podem ser transformados em outros modelos [70, 71]. MDE promove o uso de modelos em vários níveis de abstração como artefatos para a especificação de sistemas bem como a transformações automáticas do modelo do usuário em software.

A ideia central desse método inicia com modelos computacionais em alto nível de abstração (em inglês, *Computational Independent Model* - CIM). Em seguida, o modelo é submetido a um processo de transformação que reduz o nível de abstração para um modelo independente da plataforma computacional usada (*Plataform Independent Model* - PIM) que representa a solução em nível de projeto para os requisitos do CIM. O PIM pode ser transformado em um ou mais modelos específicos para uma ou mais plataformas tecnológicas desejadas (*Plataform Specific Model* - PSM) que estará pronto para ser refinado ou diretamente usado [72, 73, 74].

Kent [75] sugere UML (*Unified Modeling Language*) como linguagem de modelagem PIM de forma que os modelos gerados possam ser transformado de forma semi ou automática para código. Linguagens expressivas como UML oferecem um bom suporte para modelagem de sistemas até um certo ponto de vista, facilitando a comunicação entre múltiplas aplicações. No entanto, sabe-se que quanto mais expressiva é a linguagem de modelagem, mais difícil é o tratamento semântico da mesma. A complexidade de linguagens como UML se reflete em seus metamodelos. A identificação de dependência entre conceitos fica comprometida, dificultando o entendimento e o uso efetivo por parte dos desenvolvedores que necessitam manipulá-los e identificar se o metamodelo captura todas as dependências requeridas.

No entanto, de acordo com Hutchinson *et al.* [71] os resultados de sua pesquisa empírica mostram que os usuários MDE utilizam várias linguagens de modelagem.

Aproximadamente 85% dos entrevistados fazem uso de UML e aproximadamente 40% usam uma DSL de projeto próprio. Entretanto, há uma ambivalência significativa sobre o equilíbrio entre a complexidade da UML. 43% dos participantes acham que UML é muito complexa comparado com 32% que discordam. Enquanto que 23% são neutros. Essa visão inconclusiva indica que a discussão sobre linguagens de modelagem alternativas é bem-vinda, bem como entender que tipo de linguagem de modelagem pode representar melhor certos tipos de modelos.

O estudo de Torchiano *et al.* [76], baseado em uma pesquisa de opinião com profissionais da indústria italiana de software, destaca um mapeamento de relevâncias, benefícios e problemas relacionados com a engenharia guiada por modelos. Do ponto de vista da relevância, os entrevistados destacaram que MDE, guardadas as proporções da empresa e os critérios de análise, é relevante por oferecer suporte à geração de código, interpretação e transformação de modelos. Já os benefícios de se utilizar MDE que obtiveram maior destaque pelos entrevistados foram: suporte ao projeto e melhoria na documentação. Quanto aos problemas identificados, os mais relevantes foram: (i) MDE exige muito esforço da equipe, (ii) não é usável o suficiente, (iii) exige muitas competências. O item (i) e (iii) estão de certa forma relacionados pois se um desenvolvedor possui muitas habilidades, está sujeito a acumular responsabilidades e possivelmente sobrecarregar-se, demandando assim muito esforço da equipe, cada vez mais com número reduzido de integrantes onde poucos fazem muito. O uso total e a praticidade (tópico ii) estão relacionados ao nível de maturidade do método. Como MDE é emergente, muito ainda se tem a fazer para que o método chegue à maturidade, o que é bom pois há margem para pesquisa e consequentemente a comunidade espera por contribuições.

MDE engloba várias tecnologias e integra vários nichos de conhecimento que viabilizam a transição entre os vários níveis como de PIM para PSM e vice versa. Favre [77] cita como exemplo:

- *Grammarware* - Engenharia de linguagens;
- *Documentware* - XML;
- *Dataware* - SQL;
- *Modelware* - UML;

Em cada uma dessas áreas, modelo, metamodelo e transformação são conceitos que podem sofrer variações, por exemplo, o que é chamado de metamodelo em

modelware, corresponde a "schema" em *documentware*, que por sua vez corresponde a uma "gramática" em *grammarware* [77].

2.6 Linha de Produto de Software Dirigido por Modelo (MD-SPL)

Em seções anteriores, foi discutido isoladamente SPL e MDE. Essa seção tem como objetivo esclarecer o que é a abordagem MD-SPL e procura responder em que exatamente essa abordagem se diferencia de uma simples integração isolada entre SPL e MDE. Para começar, é possível definir um método SPL que englobe uma ou outra técnica pontual de MDE. É exatamente isso que MD-SPL não é. A proposta dessa abordagem é se consolidar como uma prática sistemática e automática que integre as duas engenharias de domínio e aplicação por meio de transformação/tradução de modelos intra e inter processo de linha de produto.

Deng *et al.* [78] iniciaram essa discussão motivados pela arquitetura de SPL. De acordo com os autores, existem muitos desafios associados ao desenvolvimento de uma arquitetura de linha de produto de software em larga escala e para minimizar tais desafios, os autores defendem a integração de técnicas de desenvolvimento dirigido por modelos juntamente com *frameworks* de componentes. Tal defesa é sustentada pela aplicação prática do método no domínio de sistemas embarcados de tempo real. Os autores usaram (i) metamodelagem e interpretadores de modelos para criar uma DSL juntamente com (ii) a análise da variabilidade e similaridade para criar um *framework* de componentes. O propósito da DSL é ajudar a automatizar atividades repetitivas que podem ser simplificadas nos próximos produtos. Já o *framework* de componentes tem o propósito de encapsular padrões em uma plataforma de reúso.

Correa *et al.* [79] comentam que MD-SPL envolve uma complexa relação de dependência entre diferentes artefatos de desenvolvimento. O processo de desenvolvimento de software dirigido por modelos soma-se a SPL para fornecer transformações de modelos e automatizar tarefas repetitivas relacionadas a criação de artefatos. Em MD-SPL o processo de transformação pode ser usado durante a engenharia de domínio para gerar elementos centrais com base na variabilidade e similaridade identificadas. As transformações também podem ser usadas no nível da engenharia de aplicação para gerar produtos específicos.

Buchmann *et al.* [80] aplica uma abordagem MD-SPL no domínio de gestão de

configuração de software que compreende basicamente o controle de versão definido a partir de um modelo. Ao comparar versões do mesmo software, é possível que haja uma substancial diferença entre elas. Por outro lado, todas elas compartilham das mesmas propriedades como revisão, variantes, etc. Mas quando esses modelos de versão estão implicitamente contidos no código do programa, reusá-lo para desenvolver um novo sistema de gestão de configuração de software torna-se trabalhoso e muito complexo. Para resolver tal problema, os autores criaram um método capaz de integrar o modelo de *features* com o modelo de domínio.

Zhang *et al.* [81] comentam que em MD-SPL, produtos são derivados de modelos, ou seja, o código fonte do produto final é gerado a partir de modelos e que o ciclo de vida de um desenvolvimento MD-SPL tem três fases: (i) a identificação da variabilidade e similaridade (engenharia de domínio), (ii) a especificação e (iii) a implementação da variabilidade. A maioria das pesquisas focam apenas na especificação e implementação e acabam deixando de lado a identificação da variabilidade. No contexto dessa tese, as três etapas são seguidas e especificadas.

MD-SPL é uma área de pesquisa emergente, com impacto sobre a produtividade de linha de produtos e aponta vários caminhos de evolução para a transformação de modelos. O estado da arte indica problemas interessantes de serem investigados, e o estado da prática aponta uma carência muito grande de ferramentas que materializem e facilitem a utilização dessa abordagem sistêmica proposta.

Kulkarni [82] investiga a lacuna entre os modelos produzidos por SPL e usa técnicas dirigidas por modelo para suprir a demanda por adaptabilidade entre camadas de *processos de negócio* e *plataforma tecnológica*. O autor reúne uma experiência prática de observação do problema no domínio de aplicações industriais ao longo de quinze anos e defende a união entre SPL e MDE, com base nos resultados práticos obtidos. Kulkarni resolve o problema por meio de uma arquitetura para extensão e configuração de famílias de software. A contribuição de Kulkarni encontra-se em alto nível, ideal para quem sabe exatamente como integrar SPL e MDE. Enquanto que o método proposto nessa tese abrange a parte de *integração entre modelos arquiteturais* a nível conceitual e de projeto, formaliza *funcionalidade, decisões de projeto, arquitetura e plataforma tecnológica* no domínio de famílias de DSL's.

Carlos *et al.* [83] discute sobre a integração entre *Service Oriented Architecture* (SOA), SPL e questões relacionadas a variabilidade. Os autores comentam que a abordagem sugerida por eles envolve uma série de desafios relacionados à *modula-*

ridade e reúso arquitetural. Essas discussões somam-se a essa tese no sentido de avaliar os resultados.

Buchmann *et al.* [80] apresentam um sistema para gestão de configuração fundamentado nos princípios de MD-SPL. Os autores comentam que a partir da definição de modelos, o reúso pode ser conduzido a nível de modelagem. Sendo assim, a codificação manual é totalmente dispensável. Para comprovar esse princípio, eles desenvolveram um método chamado FORM que na verdade transforma/anota *features* em modelo de *features*. O trabalho proposto pelos autores investiga a lacuna entre os modelos produzidos por SPL e usa técnicas dirigidas por modelo tais como *mapeamento e configuração*. Com base no escopo dessa tese, a técnica de mapeamento entre modelos PIM - PSM é abordada. Já a configuração está fora do escopo, conforme apresentado na Figura 1.1.

Cuong e Yongjie [84] apresentam uma abordagem prática que baseia-se em uma arquitetura de SPL inicial e a deriva juntamente com a geração automática de código para instâncias de produtos customizados da mesma família. Essa estratégia é centrada na arquitetura, auxilia no gerenciamento da variabilidade de requisitos, combina geração de código e anotação de código baseado na arquitetura. Essa tese assemelha-se ao de Cuong e Yongjie por apresentar uma abordagem prática que gere código de produtos customizados, mas diferencia-se na generalização por abranger a modelagem arquitetural do ponto de vista inter e intra famílias de software.

Alsawalqah *et al.* [85] propuseram um método para encontrar plataformas otimizadas de produtos de software a partir de características informadas pelo usuário. O destaque para o trabalho de Alsawalqah *et al.* está no fato de ter sido formalizado matematicamente como um problema de otimização para maximizar o ciclo de vida da família de software e registrar o total de similaridades, objetivos e necessidades do usuário, que pode ser muito útil ao arquiteto na tomada de decisão por qual plataforma utilizar. Apesar de focar na mesma classe de contribuição e ver o problema com uma possível caracterização de otimização, essa tese optou em usar um outro ferramental científico e tecnológico para investigá-lo, mais voltado para o formalismo denotacional de MD-SPL.

Seguindo a mesma linha de Alsawalqah *et al.*, o trabalho de Mariani *et al.* [21] defende que os estilos arquiteturais ajudam a melhorar a arquitetura de SPL nos quesitos flexibilidade, extensão e manutenção. Mariani *et al.* usa a pesquisa operacional para preservar o estilo arquitetural durante o processo de otimização multiobjetivo do projeto arquitetural de SPL. O trabalho de Mariani *et al.* está desconectado

da visão dirigida por modelos, mas assemelha-se a essa tese do ponto de vista da investigação arquitetural.

2.7 Considerações Finais

Esse capítulo consolida uma percepção holística sobre modelo computacional e modelagem, trazendo a tona uma conexão entre conceitos por vezes discutidos de forma isolada. A Figura 2.1 representa o escopo e propósito do capítulo com início na compreensão de como modelo e modelagem podem contribuir com o processo evolutivo de uma teoria como a de "domínio".

Primeiramente, a engenharia de linguagens computacionais vem gradativamente fornecendo provas de suficiência para a teoria de domínio. A cada novo paradigma ou a cada nova linguagem que surge, novos processos e testes ajudam a rever a teoria em si, contribuindo dessa forma para a sua evolução. Se abstrairmos uma linguagem computacional como um modelo, a modelagem permite que o modelo comece com uma estrutura relativamente simples e se torne progressivamente elaborado por meio de um processo iterativo de projeto, teste e refinamento. Por outro lado, a teoria de domínio oferece fundamentação e formalização robusta para os princípios tanto do modelo como da modelagem, caracterizando assim uma mútua relação de benefícios.

A linha de produto de software é um paradigma de desenvolvimento composto internamente pelas engenharias de domínio e aplicação, ambas fundamentadas pela teoria de domínio. A engenharia de domínio tem uma reconhecida contribuição na evolução do processos produtivo de software por investigar e indicar boas práticas de como conciliar as similaridades de uma dada família de software com as solicitações de personalização feitas pelo cliente, agregando ao processo o reúso e viabilizando assim a produção do software de forma análoga à industrial em escala de "linha de produção". Já a engenharia de aplicação usa os artefatos gerados pela engenharia de domínio para instanciar um software voltado para o espaço da solução.

Sendo MDE uma abordagem de modelagem que abrange a manipulação (seja pela tradução ou transformação) de modelos em vários níveis de abstração, incluindo o espaço do problema e da solução, é possível dizer que MDE contém os princípios de engenharia de domínio e aplicação, e que há uma convergência entre essas áreas de modelagem, por vezes, tratadas na literatura de forma desconexa.

Diante da possibilidade de se obter benefícios significativos entre a integração de SPL e MDE, surgiu MD-SPL com a fusão entre esses dois paradigmas. MD-SPL

contribui com uma abordagem sistemática com impacto sobre a produtividade de linha de produtos e aponta vários caminhos de evolução para a transformação de modelos.

Toda essa fundamentação, oferece uma base para especificar que o método de suporte à modelagem arquitetural proposto parte da engenharia de domínio/aplicação e sofre um processo de integração com o espaço técnico e boas práticas sugeridas por MDE para resolver conforme a proposta sistemática de MD-SPL demandam de conexão que vai do conceitual ao espaço técnico de *grammarware*, fonte de discussão do próximo capítulo.

Capítulo 3

Arquitetura de Famílias de Software e a Modelagem da Variabilidade

Esse capítulo complementa a fundamentação teórica e aborda conceitos relacionados à arquitetura de famílias de software, a modelagem da variabilidade e a gestão de decisões arquiteturais. Para tanto, esse capítulo está organizado em cinco seções. Inicialmente, a Seção 3.1 aborda o reúso de software, conceito amplamente discutido e associado às práticas adotadas nessa tese, também envolve desafios que precisam ser adequadamente mapeados para serem mitigados.

Conforme a Figura 3.1, as demais seções seguem uma organização piramidal, com abordagem de discussão da base para o topo representando uma análise do conceitual ao emergente, capaz de sustentar essa tese. O processo de desenvolvimento de uma arquitetura SPL e práticas associadas, bem como artefatos gerados no processo de desenvolvimento da arquitetura são discutidos na Seção 3.2.

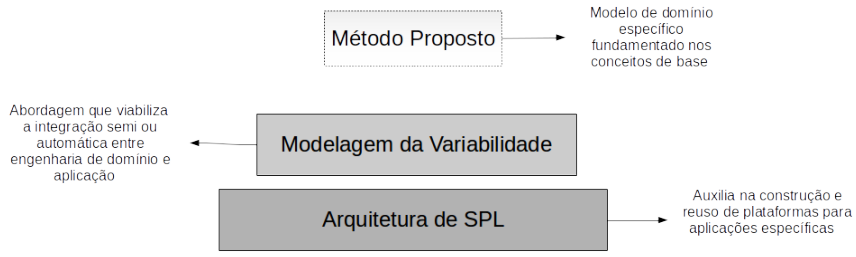


Figura 3.1: Estrutura do Capítulo 3.

A modelagem da variabilidade, conceito-chave nessa tese, é discutida na Seção

3.3. Para finalizar, são apresentadas algumas considerações finais na Seção 3.4.

3.1 Conceitos

O reúso é uma disciplina investigada em diversas áreas do conhecimento, como engenharia mecânica e eletrônica. Essas experiências adquiridas em outras áreas revelam que investir no reúso agrega confiança no produto, redução nos riscos associados aos processos (de produção, distribuição, manutenção, etc.), uso eficiente dos especialistas e conformidade com padrões [86, 87].

No domínio de SPL, as investigações sobre o reúso são especialmente motivadas pela demanda de custos reduzidos ao longo do processo de desenvolvimento, aumento da qualidade e entregas mais rápidas [88, 89]. Muitas dessas demandas são conflitantes entre si no sentido que ao atingir uma se diminui a outra. O desafio que se coloca é atingir o equilíbrio entre essas demandas e minimizar as perdas. O reúso também envolve dois processos correlatos: o desenvolvimento de componentes reusáveis e o desenvolvimento de sistemas com reúso de componentes [90].

Três atividades básicas estão associadas à obtenção do reúso [91, 92]:

- Seleção: identificar artefatos candidatos ao reúso. Essa atividade pode ser manual ou automática dependendo do nível de maturidade do projeto;
- Avaliação: identificar se o artefato está pronto para o reúso ou se necessita de adaptações bem como o esforço e custo associado para tal;
- Adaptação: o artefato criado para o reúso dispõe de parâmetros, arquivos de configuração ou mesmo, estrutura interna capaz de permitir a variabilidade da sua aplicação. Uma boa documentação para o reúso é essencial pois irá guiar os desenvolvedores e projetistas a extrair vantagens de customização e um melhor aproveitamento do tempo gasto na adaptação;

Tanto SPL quanto MDE possuem foco no reúso de software. Essas engenharias encapsulam informações e conceitos reutilizáveis em diferentes níveis de abstração na forma de artefatos, produtos e processos, para que os mesmos sejam facilmente adaptados em diferentes contextos, plataformas e ambientes.

Mas para que tudo isso seja realmente viável, é necessário ter uma arquitetura que dê suporte às construções a ponto de detalhar como efetivamente esse reúso pode ser atingido, sistematicamente realizado e melhorado.

No contexto dessa tese, a arquitetura de software pode ser compreendida como elemento fundamental de um sistema capaz de incorporar seus componentes[1], a conexão entre os mesmos, o ambiente no qual estão inseridos e princípios que direcionem os projetos e a evolução do sistema [93, 94, 95].

O processo de desenvolvimento de uma arquitetura de software pode atingir diferentes níveis de formalização do conhecimento, que podem ser combinados entre si, tais como [96]:

- *Conhecimento Tácito/implícito* - Em muitos casos, parte da arquitetura de software não é explicitamente especificada ou modelada por questões de custo e esforço, mas faz parte de um conhecimento tácito de domínio dos arquitetos.

- *Conhecimento Documentado* - Assim como na engenharia civil que a arquitetura contribui com múltiplas perspectivas de análise e projeto da obra (como planta baixa, alta, hidráulica, elétrica, etc.), na arquitetura de software não é diferente. Múltiplas visões ou perspectivas, não necessariamente completas ou precisas, podem ser definidas ao software baseadas em esquemas visuais/-modelos, apenas com a importante missão de passar a ideia, tais como: visão conceitual, física, lógica, de processo, de projeto, de desenvolvimento, etc.

- *Conhecimento Formalizado* - É a incorporação de um plano para a construção, especificado por meio de uma linguagem formal como a *Architecture Description Language* (ADL) [97], que dependendo do propósito, pode envolver a compilação, execução ou a tradução de modelos.

Sendo assim, diagramas UML podem documentar a arquitetura sob uma visão, no entanto, a percepção da aplicação da arquitetura em diferentes contextos depende do conhecimento tácito do arquiteto em compreender e interpretar os diagramas UML.

Garlan e Sousa [98] comentam que independentemente do nível de formalização da arquitetura, as seguintes práticas devem ser observadas:

- *No contexto de uma arquitetura, práticas informais não são adequadas.* Muitas arquiteturas são descritas de forma ambígua, inconsistente e ineficiente. As vezes, isso acontece em função de não deixarem claro qual a natureza da visão arquitetural que está sendo descrita, ou sobre o significado dos elementos da visão.

[1]Um componente é uma parte modular de um sistema. Ele define o seu comportamento em termo de interfaces requeridas e providas.

- *Arquitetura não é uma estrutura de programação baixo nível.* Um dos requisitos mais importantes relativo a documentação da arquitetura é elucidar a estrutura do software e isso só é possível via abstração.

- *Múltiplas visões arquiteturais são necessárias.* Diferentes modelos são necessários para cobrir a faixa de requisitos de uma arquitetura. Algumas documentações falham por combinarem muitos requisitos em uma simples descrição.

- *Boas arquiteturas fazem uso de uma arquitetura de referência.* Uma arquitetura de referência pode fornecer desde um vocabulário até o mapeamento de limitações/restrições. Isso permite um crescimento sobre um conhecimento prévio.

- *Muitos projetos arquiteturais são na verdade frameworks.* A princípio, uma arquitetura representa um sistema em particular (como uma cardinalidade 1..1). Entretanto, arquiteturas podem ser projetadas para cobrir sistemas que se diferem entre si (como uma cardinalidade 1..N). Uma chave de sucesso para documentar esse tipo de arquitetura é deixar claro que a variabilidade é aceitável.

A próxima seção aborda como especificar uma arquitetura de forma que outros possam usá-la com sucesso, mantê-la e construir sistemas a partir da arquitetura. Para tal, é de fundamental importância que as visões arquiteturais e o processo de escolha das mesmas com base na característica do projeto sejam detalhadas.

3.2 Projeto Arquitetural

Diversas experiências são relatadas na literatura técnica sobre processos, guias e melhores prática para o desenvolvimento de arquitetura de software, bem como desafios também. Hofmeister *et al.* [99] comentam que um desses desafios é manter o controle intelectual ao longo do projeto, a partir de definições resultantes de conflitos e consensos entre a equipe técnica e os interessados pelo projeto, principalmente ao longo do tempo.

Mesmo com tanta diversidade de processos e domínios de aplicação, existem algo em comum no desenvolvimento de uma arquitetura de software. Foi com esse

objetivo, após analisar cinco métodos[2] que Hofmeister *et al.* [99] chegaram a um método geral composto por:

- Análise da Arquitetura: concentra os requisitos da arquitetura;
- Síntese da Arquitetura: contempla especificações da arquitetura candidata com base nos requisitos especificados;
- Avaliação da Arquitetura: garante que as decisões arquiteturais estão em conformidade com o esperado.

Devido a complexidade do projeto arquitetural, as atividades associadas a essa fase não são executadas sequencialmente, mas sim de uma forma iterativa e incremental e em vários níveis de granularidade, conforme uma sequência imprevisível até que a arquitetura esteja completa e validada [99].

De acordo com Taylor *et al.* [100], O processo de desenvolvimento de uma arquitetura comporta as fases de análise, projeto, implementação, teste e evolução que podem ser desenvolvidas de forma iterativa e incremental.

A fase de **análise de uma arquitetura** contempla: a definição de metas, o escopo, comprometimento arquitetural, nível de formalidade dos modelos da arquitetura, o tipo em si da análise, o nível de automação e quem são os interessados ou as pessoas que vão se beneficiar da análise. Cada uma dessas etapas pode ser detalhada da seguinte forma:

- Metas: as arquiteturas tendem a se comprometer com a completude, consistência, compatibilidade e corretude. A completude pode ser interna ou externa. A completude interna tem haver com a notação da modelagem e se a mesma atende todos os componentes/conectores que precisam ser representados. Já a completude externa verifica se a arquitetura suporta a representação de todos os requisitos do sistema. A consistência avalia se nenhum componente/conector contradiz o outro. A compatibilidade garante que os modelos arquiteturais seguem um estilo, uma arquitetura de referência e padrões. Por último, a corretude garante que a arquitetura dará o suporte esperado ao sistema e a implementação do sistema seguirá a arquitetura.

[2]Os métodos de desenvolvimento de arquiteturas analisados por Hofmeister *et al.* [99] foram: *Attibute-Driven Design* (ADD), *Siemens'4 Views* (S4V), *Rational Unified Process 4+1 Views* (RUP 4+1), *Business Architecture Process and Organization* (BAPO) e *Architectural Separation of Concerns* (ASC).

- Escopo: define os diferentes níveis de abstração da arquitetura (componente-conector, sistema-subsistema, fluxo/estrutura/propriedade de dados, processamento, interação, configuração, etc.)
- Comprometimento: pode ser definido do ponto de vista estrutural, comportamental, de interação e não-funcional.
- Formalidade: modelos informais, semiformais e formais.
- Tipo de Análise: estática, dinâmica e guiada a cenários.
- Nível de Automação: manual, automático ou parcialmente automático.
- Usuários: podem ser outros arquitetos, desenvolvedores, gerentes, clientes ou distribuidores do software resultante.

A fase de **projeto de uma arquitetura** compreende a relação entre *elementos estruturais* do software, estilos arquiteturais e padrões de projeto que podem ser usados para suprir requisitos definidos para o sistema e restrições que afetam a forma como a arquitetura pode ser implementada [101].

Toda arquitetura nasce para atender um software específico ou uma família de softwares. O arquiteto precisa ter a percepção do domínio (requisitos não funcionais do software e/ou família e o contexto de aplicação da arquitetura) juntamente com as necessidades do cliente (requisitos funcionais) até para decidir por uma arquitetura de referência. Com base nessas premissas, o arquiteto tem em mãos os elementos essenciais para criar várias visões arquiteturais que contemplem requisitos funcionais, não funcionais e contexto de aplicação.

Essa fase é muito crítica, pois na literatura existem vastos relatos de projetos de arquitetura que resultaram em produtos deficientes, que não representam adequadamente os requisitos, não é adaptável às mudanças de requisitos ao longo do tempo, não é reusável e ainda têm comportamentos imprevisíveis. Projetos de arquitetura concebidos com base nas boas práticas reduzem riscos associados à produção do software, reduzem os custos de manutenção e teste, contribuindo assim para um software de qualidade.

A fase de **desenvolvimento** concentra-se em transformar os artefatos gerados no projeto, como modelos (definidos em UML ou ADLs) em artefatos derivados, como o código.

Tabela 3.1: Modelo de maturidade para arquitetura de SPL [5].

Dimensão	Sequência	Atividade
Projeto arquitetural	1	Engenharia do domínio
	2	Gerenciamento de requisitos e modelagem
	3	Análise arquitetural e avaliação
Gerenciamento SPL	4	Gerenciamento da similaridade
	5	Gerenciamento da variabilidade
Documentação	6	Gerenciamento dos artefatos de arquitetura

O **teste da arquitetura de software** a nível de projeto é incipiente. Algumas abordagens estão a nível de ADLs [102], outras mapeiam as características identificadas pelo projeto (definidas com base em notação visuais) para métodos formais [103, 104].

Já a **evolução** da qualidade arquitetural está em um nível de maturidade mais elevado e ocupa posição de destaque nas discussões que envolvem iterações subsequentes à arquitetura, mas basicamente essas discussões se concentram em como elicitar e refinar requisitos de qualidade com base em: experiências, cenários e métricas [105].

Ahmed e Capretz [5] estabeleceu um modelo de maturidade de processo arquitetural para SPL, com o intuito de dar suporte ao desenvolvimento de SPL dentro das organizações. Esse modelo de maturidade passa, do mais baixo para o mais alto nível de maturidade, pelas escalas de: (1) base de produto configurável, (2) estabelecimento de uma família de software, (3) estabelecimento de uma plataforma de software, (4) infraestrutura padronizada e (5) desenvolvimento de produto independente. Do ponto de vista da configuração do modelo de maturidade, ele segue as dimensões e atividades conforme a Tabela 3.1.

Decisões de projeto devem ser corretamente documentadas e registradas de forma a garantir uma identificação eficiente e correta recuperação de mudanças que podem afetar criticamente o sistema ao longo do tempo.

A gestão de decisão arquitetural vem se destacando como item de primeira necessidade e que não pode ser negligenciado ao longo o processo de construção arquitetural com o risco de causar desordem no projeto e inviabilidades técnicas.

3.3 Modelagem da Variabilidade

O termo variabilidade não significa apenas as diferenças entre sistemas, mas também sua capacidade em ser customizado ou configurado de forma eficiente para uso em um contexto particular [11, 13].

De acordo com Berger *et al.* [12], os relatos dessa customização ocorrem, em sua grande maioria, sobre componentes de software, código fonte, requisitos, arquitetura, projeto, plataforma, casos de teste, bibliotecas, documentação, DSLs, planos de *release*, modelos de especificação e representação do conhecimento.

Uma informação interessante revelada pelo estudo de Berger *et al.* é que os projetistas usam os *modelos de feature*[3] como notação mais frequente para representar a variabilidade, mas também são usados os modelos de decisão, notações UML, linguagens de descrição de arquitetura (ADLs) e abordagens não formais, como planilhas, descrição textual, arquivos de propriedade, entre outros. Múltiplas notações também são combinadas e usadas ao mesmo tempo, possivelmente por não ter uma notação robusta o suficiente para comportar de forma integrada as principais necessidades de modelagem.

Do modelo de *feature*, é possível obter: os recursos entregáveis ao usuário, grupos de requisitos e a configuração de produtos. A estrutura original do modelo de *features* [106] comporta primitivas de modelagem bem simples como: relacionamento estrutural (composição, generalização/especialização), opcionalidade e dependências mútuas (inclusão, exclusão). Além disso, a lógica de seleção entre *features* opcionais e mandatórias devem ser explicadas de forma descritiva.

O Linux kernel gerencia suas variabilidades por meio da definição de configurações (o modelo Kconfig), que contempla um conjunto de *features* e suas interdependências. O processo de configuração resulta em uma seleção válida de *features*, usadas no código fonte com a adição do prefixo `CONFIG` [107].

Sendo assim, a variabilidade pode ser vista como um atributo de qualidade, assim como manutenibilidade ou usabilidade, que deve ser gerenciado com base em três abordagens elementares [108, 109, 110]: (i) proativa, quando a similaridade é identificada primeiro e dá origem a uma família de software antes de qualquer produto ser derivado por meio da identificação da variabilidade; (ii) reativa, quando um produto é derivado a partir de uma família de software e (iii) extrativa, quando produtos existentes passam por uma reengenharia para se enquadrar a uma família

[3] *Features* são conceitos ou características de um domínio, como serviços, operações e funções. Nessa tese é usado o termo em inglês por uma questão de conformidade com a literatura.

de software.

A análise do ciclo de vida do sistema abre margem para uma dimensão espacial, de forma que *features* podem variar ao longo do tempo. Apesar de não ser o foco dessa tese, é importante destacar esse tópico emergente de pesquisa onde essa característica é investigada sob a ótica de que a variabilidade pode sofrer influência das dimensões e restrições temporais [111, 112].

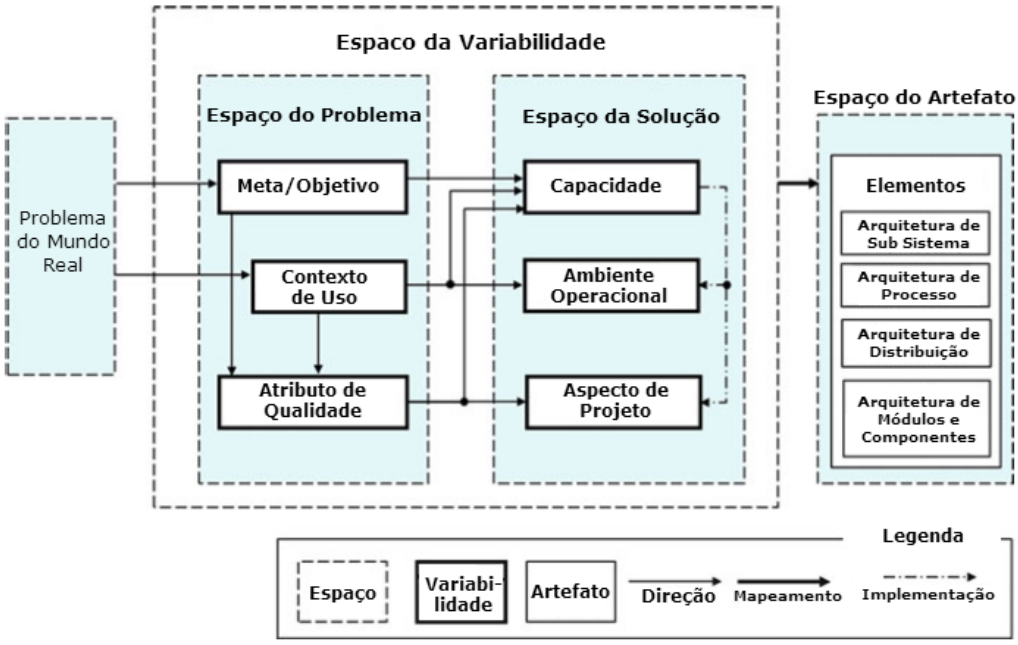


Figura 3.2: Espaço da Variabilidade [4].

A variabilidade pode ser modelada de várias formas e motivada pela necessidade de gerar produtos de software para diferentes contextos de aplicação. Em alto nível, a modelagem da variabilidade inicia pela separação entre o espaço do problema e da solução, conforme ilustrado na Figura 3.2.

No espaço do problema, são tipicamente modeladas as metas/objetivos, que, por sua vez, necessitam ter seus respectivos atributos de qualidade e os contextos de uso/aplicação do produto. No espaço da solução, a variabilidade é tipicamente modelada para atender as dimensões funcionais (como recursos e serviços), dimensões do ambiente operacional (como sistemas operacionais e plataformas) e as dimensões de projeto.

Os mesmos recursos, por uma decisão de projeto, podem ser implementados de várias formas e podem ter características de qualidade diferentes. A composição

dessas dimensões são exploradas na modelagem de variabilidade.

A variabilidade modelada para essas dimensões é materializada como arquitetura de software, componentes, pontos de variação e variantes. Mecanismos de implementação como herança, template, framework, macro e geradores de código podem ser usados para implementar pontos de variação e variantes.

Artefatos aplicados em vários contextos necessitam de vários recursos e/ou diferentes atributos de qualidade.

De acordo com Kang [4], a modelagem da variabilidade traz os seguintes benefícios:

- Configuração do produto;
- Especificação de requisitos;
- Derivação de produtos, projetos e arquitetura;
- Manutenção e estimativa de custo para novas features;
- Planejamento do desenvolvimento e evolução do software.

Por outro lado, os problemas associados à modelagem da variabilidade reportados com mais frequência são:

- Gestão de dependências: um significante esforço é gasto para resolver dependências até efetivamente derivar sistemas. Isso revela que os modelos de variabilidade são frágeis e devem ser controlados por uma equipe pequena para que as decisões não fujam do controle;
- Evolução dos modelos de variabilidade: que envolve a decisão por um padrão correto e tomada de decisão por quais variabilidade deixam de ser necessárias, envolvendo muito trabalho manual e engenharia de regras para garantir a verificação de consistência e propagação de valores;
- Processo de configuração: como resolver conflitos.

Boa parte desses desafios podem ser mitigados por meio do uso de modularização e encapsulamento de conceitos, e rastreamento automático por exemplo, para resolver conflitos de configuração.

3.4 Considerações Finais

A arquitetura de software é elemento mandatório para se criar um arcabouço estruturado que permita gerar incrementalmente sistemas mais eficientes. Ao falar de reúso e a integração entre as engenharias, é essencial discutir o suporte arquitetural para essa abordagem.

Em função do modelo proposto, esse capítulo pontua o que é arquitetura no contexto desse trabalho e explora o processo de desenvolvimento da arquitetura, o mesmo adotado por *frameworks* que se diferenciam de uma arquitetura específica por aceitarem a variabilidade. Um outro conceito de destaque é a modelagem da variabilidade e suas várias vertentes. Para finalizar a discussão, foram apresentadas algumas problemáticas associadas ao reúso de software.

Parte II

Descrição Metodológica

Capítulo 4

Instrumental de Pesquisa

O objetivo desse capítulo é apresentar o instrumental de pesquisa adotado para o desenvolvimento dessa tese. A Seção 4.1 define as fronteiras da pesquisa. A Seção 4.2 define a metodologia de pesquisa e detalha cada uma das fases, subfases e ferramentas utilizadas. O critério de avaliação do trabalho é apresentado na Seção 4.3. Na Seção 4.4 são discutidas as considerações finais.

4.1 Escopo da Pesquisa

SPL representa um conjunto de produtos que compartilham *features*, ou seja, funcionalidades visíveis ao usuário que podem assumir natureza mandatória ou não.

A arquitetura é o artefato de fundamental importância para a engenharia de SPL, pois concentra a variabilidade e a similaridade de uma linha de produto de software. Associado a essa importância está a complexidade de realizar tal atividade devido ser uma atividade que envolve intensamente os arquitetos nas decisões de projeto, resoluções de conflitos e propagação de falhas mediante modificações arquiteturais.

O grande desafio da modelagem arquitetural em linha de produto de software dirigido por modelo está na integração entre os modelos arquiteturais e de projeto. Uma vez integrados, um problema decorrente é administrar as mudanças, seja a nível de modelo arquitetural ou de projeto e garantir a rastreabilidade das decisões e os possíveis impactos nas famílias de software.

Nesse contexto, o mapeamento da variabilidade tem impacto direto sobre a administração do domínio ou aos membros de uma família de software. É uma atividade que exige conhecimento especializado, envolve um elevado nível de abstração, além da definição de artefatos formais e consistentes. Quando o domínio em questão

é abrangente, certas particularidades só podem ser compreendidas durante o processo de desenvolvimento, contribuindo para um acréscimo de esforço na criação de modelos de domínio.

Petersen *et al.* [113] extraíram por meio de um mapeamento sistemático, as principais categorias em que se encontram os trabalhos que investigam a variabilidade por contribuição e tipo de pesquisa. Essas mesmas categorias foram utilizadas nos eixos estruturantes da Figura 4.1 para contextualizar e delimitar o escopo dessa pesquisa. SPL é representado nessa figura pela interseção entre a engenharia de domínio e aplicação (ED,EA).

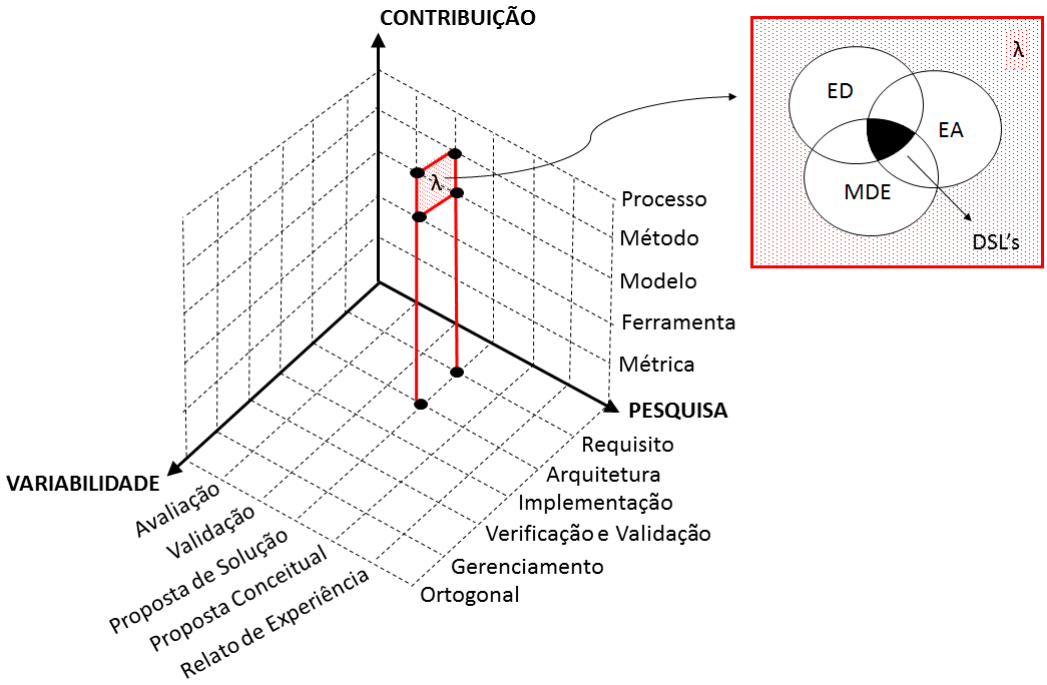


Figura 4.1: Escopo da pesquisa.

O *eixo x* destaca a forma como os pesquisadores relatam suas pesquisas que são: pesquisa de avaliação, de validação, proposta de solução, proposta conceitual e relato de experiência. Com base nessa classificação, esse trabalho se destaca como uma proposta de solução para o problema de integração entre modelos arquiteturais e de projeto. administrando a rastreabilidade das decisões arquiteturais e os possíveis impactos nas famílias de software.

O *eixo y* destaca uma classificação para os tipos de contribuições encontrados na literatura que são: processos, métodos, modelos, ferramentas e métricas. A contribuição desse trabalho é um método de suporte à modelagem arquitetural de linha de produto de software dirigido por modelo.

O *eixo z* compreende: a variabilidade de requisitos, de arquitetura, de implementação, de verificação e validação, gerenciamento da variabilidade e variabilidade ortogonal. Esse trabalho investigou a variabilidade do ponto de vista dos requisitos já que os funcionais e não funcionais tem impacto direto na concepção de elementos reusáveis. Também investigou a variabilidade do ponto de vista da modelagem arquitetura que integra requisitos funcionais, não funcionais e o contexto da aplicação.

No contexto do plano 3D da Figura 4.1, foi feito um refinamento do ponto de vista do arcabouço utilizado que é a integração sistemática entre SPL e MDE, ou seja, MD-SPL e famílias de DSL como domínio de aplicação do método. Essa pesquisa teve como motivação a necessidade de produzir conhecimento a partir da aplicação de seus resultados, contribuindo para fins práticos, caracterizando-a como pesquisa aplicada.

A condução da pesquisa foi realizada no Grupo de Interesse em Sistemas Embarcados e Engenharia de Software (GISE - UFAM), em regime de cooperação com o Laboratório de Manejo Florestal (LMF - INPA) e o Grupo de Sistemas Eletrônicos e Software da Universidade de Southampton - Inglaterra. Essa integração proporcionou maior proximidade com especialistas, validação, socialização de resultados e ajudou a orientar a pesquisa em uma direção promissora.

4.2 Métodos, Técnicas e Ferramentas Utilizadas

A Figura 4.2 sumariza a metodologia percorrida ao longo da pesquisa com base em oito etapas:

❶ **Revisões da Literatura.** Abordagens existentes foram identificadas para SPL+MDE, bem como trabalhos correlatos e problemas em aberto na área de modelagem arquitetural em MD-SPL.

❷ **Criação do método de suporte à modelagem arquitetural.** Com o objetivo de obter uma nova solução que integre modelos arquiteturais e de projeto a partir de MD-SPL. O método, oferece suporte intra e inter famílias de software por meio da transformação e/ou configuração de modelos, auxilia na administração da variabilidade seja no espaço do problema ou da solução e o reúso da similaridade.

Com base na identificação das arquiteturas existentes, bem como a identificação de suas características e componentes foi feita a compilação de um conjunto de requisitos relevantes do ponto de vista arquitetural para direcionar a arquitetura da família de software em questão.

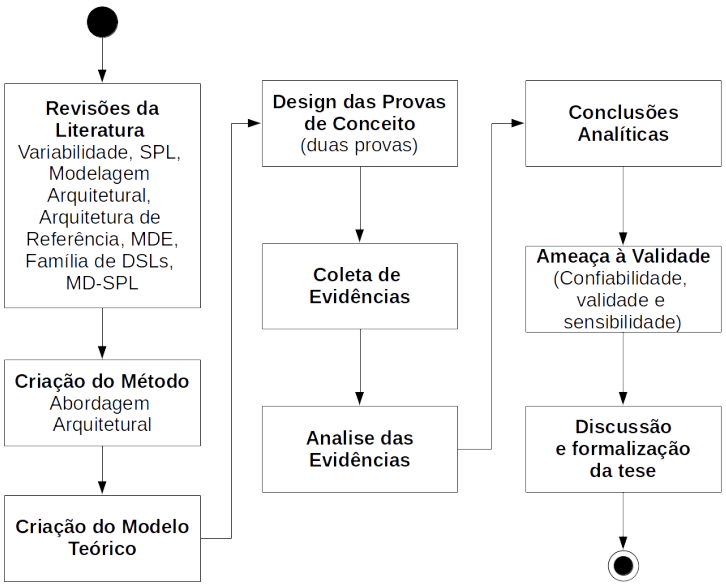


Figura 4.2: Método de pesquisa.

❸ **Criação do modelo teórico.** Representa artefatos de pesquisa e seus respectivos relacionamentos. A Figura 4.3 apresenta as variáveis latentes e observáveis. As variáveis latentes descrevem conceitos teóricos que não podem ser diretamente mensurados por serem abstratos, ou por serem novos a ponto de não disporem de um método adequado de avaliação. As variáveis observáveis definem formas de mensuração das variáveis latentes e têm múltiplos indicadores empíricos. A Integração de modelos arquiteturais e de projeto via MD-SPL, o *Framework Arquitetural* e as *DSLs* representam a *causa* e são destacadas como variáveis independentes. A *Modelagem Arquitetural* representa o *efeito*, ou seja, uma variável dependente. A relação causal entre variáveis independentes/dependentes permitem a observação da relação causa/efeito e da definição de proposições, que podem ser testadas apenas pela avaliação das relações entre as variáveis observáveis.

❸ **Provas de Conceito.** Foram definidas duas provas de conceitos: a modelagem arquitetural da família CarbonQL, que evidencia uma aplicação prática do método a partir da *transformação* de modelos enquanto funcionalidade essencial e a modelagem arquitetural da família PPL que evidencia a *configuração* entre modelos enquanto principal funcionalidade.

❹ **Coleta e Análise de Evidências.** Dados foram coletados sobre as provas de conceito com base nos critérios empíricos de análise definidos.

❺ **Delineamento das conclusões.** De posse dos resultados obtidos, foi traçada uma conclusão sobre o modelo teórico definido e a identificação de novas oportuni-

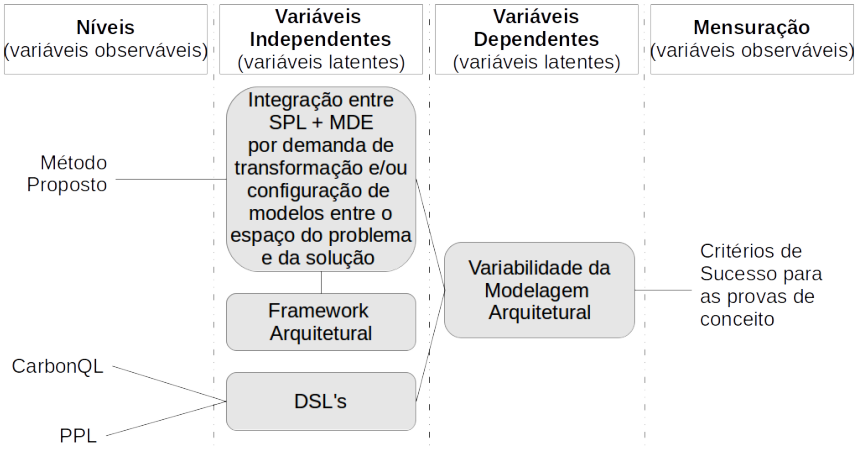


Figura 4.3: Modelo Teórico.

dades de pesquisa.

❻ **Ameaças à validade.** O viés do pesquisador é certamente uma ameaça à validade, Ainda mais quando o resultado da pesquisa é a análise do pesquisador sobre os dados. Partindo do princípio que a pesquisa deve ser reprodutível por outros pesquisadores, as ameaças foram identificadas a partir da análise do método de pesquisa (se o mesmo realmente fornece informações sobre a variável conceitual definida no modelo teórico) e da *sensibilidade* (mudança dos dados em função da mudanças na variável conceitual).

❼ **Discussão.** Conduzidas com base nas questões de pesquisa.

Essas etapas do método de pesquisa foram conduzidas de forma iterativa e incremental, através da metodologia ARDev (*Agile Research Development*) [1]. A metodologia oferece um arcabouço sistemático de gestão dos artefatos de pesquisa, principalmente às mudanças nos artefatos (princípio ágil), promovendo comunicação constante entre os pesquisadores e deixando claro a todos os envolvidos as decisões de projeto, com o objetivo de atingir entregas frequentes de artefatos de pesquisa.

Conforme a Figura 4.4, a pesquisa seguiu as três fases iterativas e incrementais da ARDev que são: (i) concepção, (ii) construção e (iii) avaliação.

A fase de *concepção* compreendeu a definição da pesquisa. Esse processo incluiu o planejamento que gerou o projeto de pesquisa e o projeto das provas de conceito como artefato. Em geral, as decisões eram tomadas após discussões realizadas em reuniões regulares que ocorriam sempre após os ciclos de iteração. Essas reuniões tinham como foco a revisão dos artefatos de pesquisa gerados versus planejados, a retrospectiva do que foi aprendido ao longo da iteração de pesquisa, com destaque para sucessos, insucessos, lições aprendidas e planejamento para os próximos ciclos

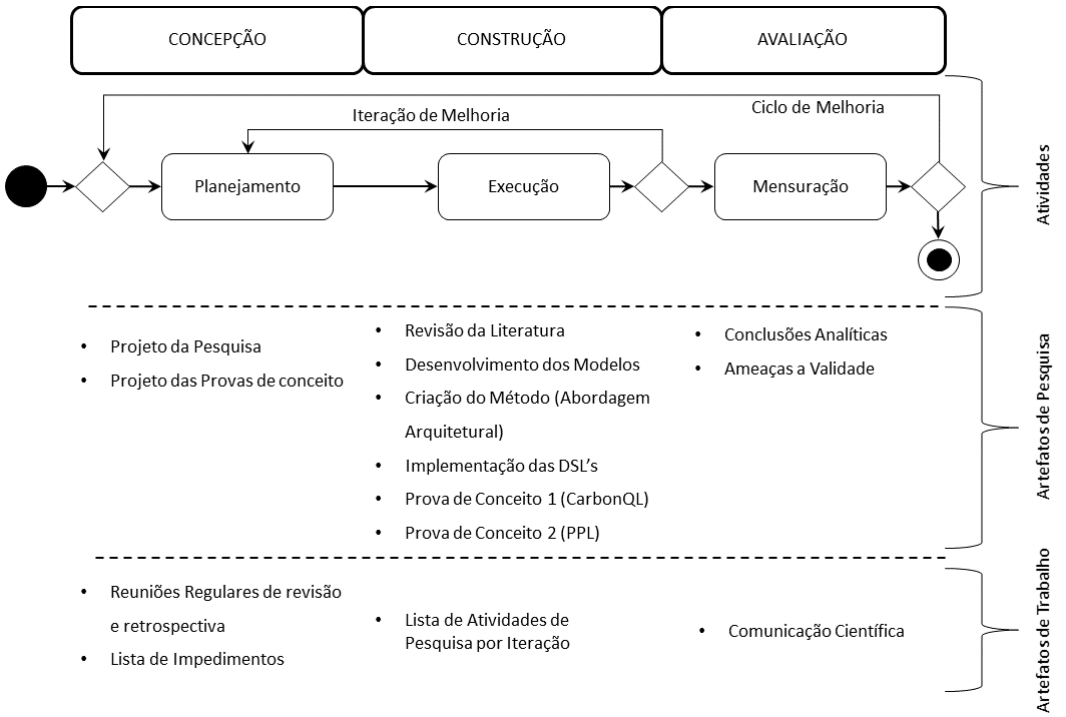


Figura 4.4: Projeto da pesquisa usando ARDev.

e ajustes no direcionamento da pesquisa.

A abordagem proposta concretizou-se por meio da fase de *construção*. A revisão da literatura deu suporte à identificação dos problemas em aberto. Através dos vários ciclos de iteração foi possível definir as fronteiras de investigação do problema com base no refinamento das questões de pesquisa. A identificação do estado da arte permitiu a concepção do método que compreende uma abordagem MD-SPL para integração entre os modelos arquiteturais e de projeto, especificação do framework arquitetural. As atividades de execução eram sempre seguidas de um retorno diário das atividades de pesquisa realizadas com base na atualização do andamento via ferramenta *online* Yodiz[1].

Alguns projetos piloto foram definidos com o intuito de avaliar possíveis tecnologias de suporte à execução da pesquisa, como as seguintes:

- Flex e Bison: viabilizam uma rápida prototipagem de compiladores e interpretadores. Essas ferramentas foram utilizadas na primeira versão da PPL.
- Prefuse: framework de modelagem, visualização e interação de grafos, tabelas e árvores em Java. Com estrutura de dados otimizada, o Prefuse gerencia mo-

[1] www.yodiz.com

dos de visualização, técnicas de codificação visual, animação, busca dinâmica e conexão com banco de dados. Esse framework foi testado, avaliado e selecionado como trabalho futuro para compor a apresentação gráfica de resultados nas duas DSL's (CarbonQL e PPL).

- OpenMath: é uma biblioteca e tipos abstratos definidos para a descrição lógica de fórmulas ou objetos matemáticos, que pode ser usado em conjunto com a linguagem MathML. A OpenMath possui dicionários projetados para serem compatíveis com um pequeno conjunto de conceitos matemáticos onde são definidos símbolos, nomes, descrições e regras. Por exemplo: o dicionário de símbolos aritméticos (+, -, *, /) e dimensão (área, volume, velocidade, etc). Essa biblioteca foi testada, avaliada e selecionada como trabalho futuro para enriquecer um metamodelo que pode ser incorporado na CarbonQL para dar suporte a especificação de equações nas consultas.

Esses projetos pilotos de avaliação tecnológica foram construídos com base nas seguintes etapas:

1. Análise: os requisitos previamente identificados foram refinados e as características, estruturas e comportamentos do projeto piloto foram descritos em nível conceitual;

2. Projeto: nessa etapa, o projeto piloto foi descrito em um nível mais concreto, métodos, linguagens de programação, ferramentas e *frameworks* foram avaliados e selecionados para que a fase seguinte fosse realizada.

3. Implementação: os artefatos do projeto piloto foram efetivamente implementados com o objetivo de testar o suporte tecnológico e amadurecer decisões de projeto.

4. Teste: As tecnologias foram testadas e avaliadas.

Na fase de *avaliação* as provas de conceito foram avaliadas. Para que as provas de conceito ofereçam evidências, elas devem ser adequadamente planejadas e organizadas com base nas seguintes etapas:

1. Planejamento: descreve os objetivos, o foco da qualidade, o ponto de vista e contexto. Como resultado, a fase de planejamento fornece a direção geral da prova de conceito e o seu escopo;

2. Projeto: implementa a fundação da prova de conceito. Nessa fase o contexto é selecionado e os critérios de sucesso são formulados. O resultado dessa fase apresenta a prova de conceito totalmente elaborada e pronta para execução.

3. Coleta de Dados e Mensuração: Ocorre sobre a execução da prova de conceito.

4. Análise e interpretação: oferecem as conclusões sobre a prova de conceito. Nessa fase, os aspectos mais importantes são: explicar os resultados considerando os critérios de sucesso e interpretar corretamente os resultados negativos.

A repetição é um princípio muito importante que deve ser considerado ao propor um método, pois os resultados devem ser reproduzidos por outros pesquisadores. Tal repetição é importante porque implica que as variáveis imprevistas não estão afetando os resultados.

4.3 Critério de Avaliação

O método, discutido no Capítulo 5, oferece suporte aos sistemas cujos *requisitos funcionais* contemplem a transformação e/ou configuração de modelos e dentre os *requisitos não-funcionais*, tenha a presença da variabilidade seja no espaço do problema ou da solução e o reúso da similaridade com base em uma família de software estabelecida.

Sendo assim, o critério de avaliação segue oito métricas que procuram avaliar o nível (alto, médio ou nulo) em que as metas do projeto arquitetural foram atingidas e da abordagem selecionada para isso.

- CS_01 (suporte a requisitos): pontua o nível em que a integração entre as engenharias (SPL+MDE) oferece suporte ao estabelecimento de requisitos funcionais e não funcionais de uma família de software;

- CS_02 (etapas de integração): elicita se as principais etapas do projeto arquitetural foram seguidas;

- CS_03 (variabilidade arquitetural): aponta se o *framework* arquitetural proposto oferece suporte à variabilidade da família de software;

- CS_04 (automação): aponta o nível de geração automática dos modelos necessários à construção, manutenção ou evolução das famílias de software;

- CS_05 (variabilidade exógena): elicita o nível de suporte ao reúso da modelagem arquitetural;
- CS_06 (variabilidade endógena): elicita o nível de suporte à construção incremental e portabilidade;
- CS_07 (comunicação do projeto): pontua se houve melhoria na comunicação do projeto arquitetural
- CS_08 (flexibilidade à mudanças arquiteturais): destaca se houve incremento na identificação/flexibilidade à mudanças arquiteturais.

4.4 Considerações Finais

Esse capítulo apresentou o instrumental de pesquisa. O escopo do estudo foi definido do ponto de vista do tipo da pesquisa, do nicho de contribuição e do tipo de investigação da variabilidade.

O uso da metodologia ARDev agregou produtividade na realização da pesquisa e as provas de conceito foram projetadas para examinarem a previsão teórica de encontro à realidade.

Com a repetição do estudo, conhecimentos adicionais podem ser gerados a respeito dos conceitos estudados e se os resultados são iguais ou diferentes dos obtidos nas provas de conceito. De qualquer maneira, o aumento das repetições traz o aumento do aprendizado dos conceitos investigados e, também, a calibração das características do modelo proposto.

Parte III

Apresentação, Análise e Interpretação dos Resultados

Capítulo 5

O Método Proposto

Esse capítulo tem como objetivo apresentar o método arquitetural proposto baseado em dois elementos centrais: (i) a integração entre as engenharias (SPL+MDE) e (ii) o framework arquitetural.

Para tanto, esse capítulo está organizado em sete seções. Inicialmente a Seção 5.1 esclarece a nomenclatura adotada. Isso se faz necessário já que os termos adotados são sensíveis ao contexto. A Seção 5.2 apresenta uma contextualização.

Conforme a Figura 5.1, as demais seções seguem uma lógica piramidal. Da base ao topo, o método é documentado por sua fundamentação, especificação e evolução.

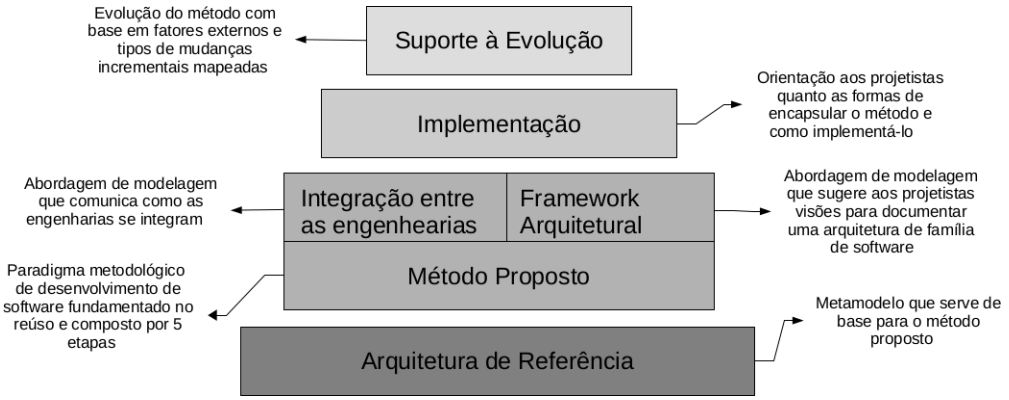


Figura 5.1: Estrutura do Capítulo 5.

A Seção 5.3 apresenta a arquitetura de referência ou metamodelo utilizado como base para a especificação do método. A Seção 5.4 apresenta o método e seus elementos centrais. A Seção 5.5 trata das particularidades de implementação do método. O suporte à evolução é discutido na Seção 5.6. Para finalizar, a Seção 5.7 aborda as considerações finais.

5.1 Nomenclatura Adotada

Em especial nesse capítulo, são adotados vários termos sensíveis ao contexto, que devido as várias definições adotadas na literatura técnica, podem surgir diversas interpretações. Para eliminar qualquer tipo de ambiguidade, os mesmos são definidos conforme a seguir:

- **Método:** conjunto de passos ou etapas seguidas para se chegar a um fim. Nesse caso, o fim almejado é a solução do **primeiro problema** discutido no Capítulo 1, Seção 1.1, que resume-se na dificuldade em integrar abstrações que levam em consideração detalhes da implementação (modelos de projeto) e as abstrações que oferecem perspectivas no nível conceitual (modelos arquiteturais), ou seja, link denotacional entre ambas.

- **Arquitetura:** é a técnica de projetar a disposição das partes ou elementos que compõem um software.

- **Arquitetura de apoio à integração das engenharias:** é a técnica de projetar a disposição integrada das partes de cada engenharia para produzir famílias de software.

- **Framework:** é um conjunto de conceitos usados de forma estruturada para auxiliar na resolução um problema de um domínio específico.

- **Framework Arquitetural:** é um conjunto de conceitos usados de forma estruturada para auxiliar na resolução de um problema arquitetural específico. No caso, o problema arquitetural abordado é a comunicação das decisões tomadas tanto a nível de modelo de projeto como a nível de modelos conceituais. Se essas decisões forem difíceis de serem identificadas, podem resultar em correção de erros de arquitetura a nível de implementação, resultando em um custo elevado de tempo e recursos. O framework arquitetural oferece aos projetistas um conjunto de conceitos que podem ser usados para comunicar as decisões tomadas, como visões de projetos, integração entre as visões, metadados, etc.

5.2 Contextualização

O processo de desenvolvimento baseado em reúso é acompanhado por uma série de desafios atrelados a criação ou a customização de software. Basicamente os

métodos que auxiliam esse processo encaixam-se em 3 categorias: paradigma, assembly e extensão.

Métodos baseados em *paradigma* são fundamentados em metamodelagem, e produzidos por instanciação, abstração ou adaptação de um metamodelo existente. Na abordagem *assembly*, os métodos são construídos por meio do reúso de outros métodos para compor um novo ou para melhorar um existente. Os baseados em padrões de extensão atendem a propósitos específicos de reúso, as vezes limitados a uma única família de software.

No caso dessa tese, o método proposto tem uma arquitetura de referência, discutida na próxima seção, que ajuda a elicitar a abordagem denotacional entre as engenharias, a variabilidade do domínio investigado, caracterizando-o como *paradigma*, pois permite a instanciação, abstração e/ou adaptação da integração entre as engenharias (SPL+MDE) e do framework arquitetural.

5.3 Arquitetura de Referência

A identificação de uma arquitetura de referência permite visualizar [114, 115]:

- O entendimento do que atualmente está funcionando e como é realizado;
- A possibilidade de desenvolvimento de novas arquiteturas com foco na variabilidade;
- O que mais pode funcionar, qual o tempo e custo demandado para realizar;
- As melhores práticas e soluções a partir de projetos passados.

A arquitetura de referência selecionada foi a proposta por Perovich *et al.* [116], devido ao nível de especificação da análise, projeto e indicação de aplicação. Essa arquitetura viabilizou a derivação de um framework arquitetural composto por visões pelas quais é possível extrair pontos de interoperabilidade e portabilidade entre famílias de software.

Perovich *et al.* propuseram o uso de *features* para modularizar as decisões arquiteturais e codificá-las como um modelo de transformação que conecta o fragmento da arquitetura com as *features* correspondentes, partindo do princípio que *features* orientam o desenvolvimento de arquiteturas, obtendo assim, uma relação explícita.

Os autores destacam que a abordagem deles para a definição de arquiteturas é independente de um tipo particular. Eles usam apenas uma visão arquitetural

baseada em componente-conector e destacam que outras visões de representação podem ser usadas conforme a necessidade.

A restrição identificada é o pouco detalhamento da arquitetura e de possíveis passos para aplicar na prática. No entanto, os autores descrevem como uma arquitetura pode ser gerada, detalham a linguagem de transformação e um metamodelo para tal. A indicação de uma ferramenta que automatize o processo é relevante, pode ser a qualquer momento implementada conforme a demanda, mas alguns ponto elementares encontram-se em aberto, por exemplo, os autores comentam que aplicaram técnicas MDE para sistematizar os estágios da ED com o intuito de automatizar os estágios da EA. Só essa descrição não é suficiente para os interessados reproduzirem os passos.

Foi primeiramente com base nessa dificuldade e em segundo lugar, pensando nos projetistas que queiram reproduzir os passos, que surgiu o método proposto detalhado na próxima seção.

5.4 Método Arquitetural

O método arquitetural pode ser aplicado ao longo do ciclo de vida e reúso de software. Requer dados que caracterizem o projeto, envolve informações sobre a relação entre produto(s)/processo(s) e é composto por cinco etapas:

- **Etapa 1: Observação do contexto arquitetural e elaboração da situação-problema.** Para a etapa inicial do método, é necessário realizar um levantamento do domínio associado ao contexto arquitetural, que consiste na observação das relações de variabilidade e similaridade, do ponto de vista endógeno e exógeno; qual a demanda exata pelo reuso de software; se existem elementos denotacionais automáticos, semiautomáticos ou manuais e a quantidade de erros gerados em função disso; se há uma família de software estruturada ou se essa estruturação é desejada e finalmente se o interesse comercial e/ou tecnológico representa um fator motivante para a evolução da relação situação-problema. Esse levantamento pode ser conduzido de forma empírica, com base em várias fontes, por exemplo: *survey*, estudo de campo, estudo de caso, *benchmarking*, documentação de análise, projeto, desenvolvimento e teste. Uma vez mapeada as demandas de rastreabilidade e registros críticos quanto a tomada de decisão arquitetural e nível de reúso de software, é possível traçar escolhas para a definição de visões arquiteturais.

- **Etapa 2 - Planejamento Arquitetural.** Nessa etapa, são identificadas e definidas as visões arquiteturais que vão ajudar a esclarecer e comunicar decisões de projetos, por exemplo, como ED e EA serão aplicadas e/ou integradas com MDE, se o processo denotacional ocorre no sentido PIM-to-PSM ou PSM-to-PIM, se o processo denotacional é baseado em transformação ou tradução de modelos, etc. Essa forma de organização, ajuda os arquitetos de software e todos os envolvidos a evoluir possíveis famílias de software e ajuda na resolução da situação-problema. O método proposto sugere ao menos quatro visões: conceitual, de módulo, de uso e de código. De forma não restritiva, esse conjunto de visões respalda decisões de projeto e permite a comunicação entre projetistas. Alguns domínios vão exigir a diversificação de visões, que devem ser adicionadas com base em uma relação custo-benefício sobre o projeto.

- **Etapa 3 - Aplicação das Visões.** As visões tem uma forma própria de documentação baseada em três informações: (i) os elementos, relacionamentos e propriedades que a formam, (ii) quem são os interessados que podem fazer uso da visão e (iii) quais são as questões respondidas por meio das informações contidas na visão. Os diagramas e artefatos de documentação arquitetural elaborados pelo projetista devem estar sempre em conformidade com essas três especificações. Esse inclusive pode ser um critério de validação e verificação da visão e dos modelos que os ajudam a documentá-la. Os modelos, por sua vez, podem ser individualmente examinados do ponto de vista da consistência e corretude interna. Um teste nos modelos pode revelar componentes, especificações incompletas, padrões indesejáveis, *deadlocks*, falha de segurança, etc. Basicamente o resultado dessa etapa é a perspectiva endógena da visão.

- **Etapa 4 - Relação entre as visões.** Uma vez construída a perspectiva endógena da visão, o projetista deve elaborar a perspectiva exógena formada pela relação. Essa etapa permite identificar propriedades entre as visões como as que influenciam as demais e as que tem maior nível de criticidade complementariedade, associação, especialização, generalização, oposição, etc.

- **Etapa 5 - Análise da modelagem arquitetural.** A modelagem arquitetural é usada para comunicar mudanças, melhorias, erros, acertos e demais decisões de projeto que terão impacto na qualidade e *time-to-market* do software. A análise da modelagem arquitetural pode ser conduzida com base em

critérios objetivos ou subjetivos, por meio de indicadores quantitativos e/ou qualitativos. Tende a ser incremental e quanto mais se executa, melhores resultados são obtidos. Ao final da condução do método, os seguintes resultados são previstos: escolhas arquiteturais visíveis a todos da equipe e incremento na comunicação.

A Figura 5.2 apresenta no fluxo da esquerda as etapas previstas pelo método e no fluxo da direita, o respectivo detalhamento de cada etapa, com destaque para: (i) a integração entre as engenharias e (ii) o framework arquitetural.

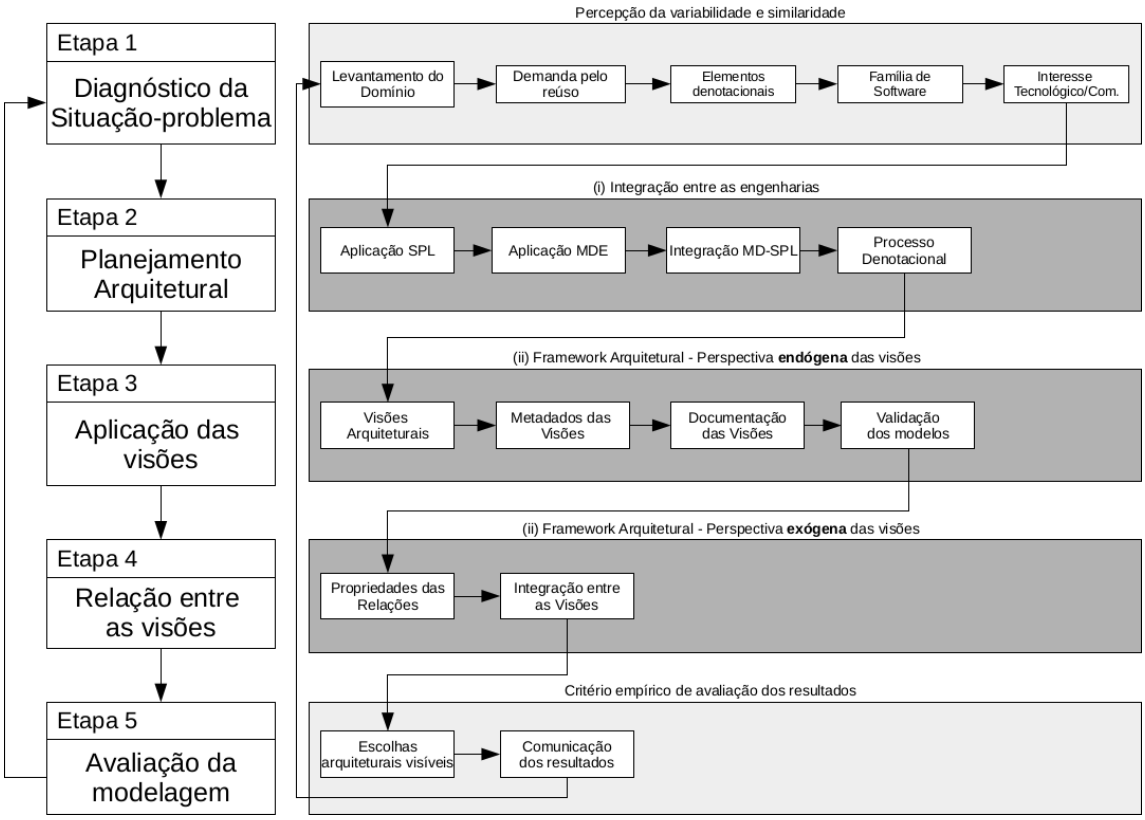


Figura 5.2: O metodo arquitetural.

O esquema de integração entre as engenharias fica evidente com a condução da etapa 2 e o framework arquitetural se consolida com a condução das etapas 3 e 4. A aplicação do método, bem como a visualização dos resultados gerados em cada uma dessas etapas podem ser observadas por meio das provas de conceito nos Capítulos 7 e 8.

5.4.1 Integração entre as engenharias

A Figura 5.3, brevemente discutida na Seção 1.6, representa a técnica de projetar a disposição integrada das partes de cada engenharia para produzir famílias de software. Ocorre que boa parte das aplicações práticas de MD-SPL não esclarecem como as engenharias integraram-se, quais etapas de cada engenharia foram adotadas e como elas efetivamente relacionam-se. A arquitetura de integração pode ser utilizada pelos projetistas para que seja possível visualizar quais das etapas previstas por cada engenharia já foi ou não implementada, quais relacionamentos foram estabelecidos e se a forma escolhida condiz com o mapeamento feito. Isso permite que o projetista eleve o nível de maturidade arquitetural, por meio da documentação das decisões de projeto e passe a trabalhar em um nível mais gerenciado e definido, permitindo socializar entre a equipe, clientes e evitando que escolhas indesejadas sejam identificadas pós-implementação, teste, implantação ou até mesmo manutenção do software.

Vale ressaltar que a arquitetura de integração pode evoluir de um fluxo conceitual para um fluxo automático mediante a construção de API's (*Application Programming Interface*) e ferramentas que otimizem o trabalho feito hoje de forma estruturada porém manual.

Por outro lado, a simples adoção de uma ferramenta ou técnica não é suficiente para que os benefícios da reutilização sejam plenos. É necessário adotar processos bem definidos de reúso que ao invés de desenvolver software considerando apenas os requisitos de um único sistema, considera-se aos requisitos de um conjunto de sistemas similares.

Para isso, basicamente as fases da engenharia de domínio e os respectivos artefato produzidos por elas servem como entrada para as fases da engenharia de aplicação que contempla um refluxo para atualizar os artefatos gerados pela engenharia de domínio, caso novos requisitos sejam identificados. Outros dois refluxos de customização estão presentes respectivamente nas fases de *análise de projeto*, *integração* e *teste*. A MDE contribui dando vazão aos modelos produzidos nas etapas de engenharia de domínio e aplicação, de forma que eles sejam transformados ou configurados para integrarem-se ao modelo da solução.

Como exemplo, suponha que na camada da solução PSM (*Platform Specific Model*) tenha uma API que dê suporte a uma família de software com o requisito comum (similaridade) de mapeamento objeto-relacional. Na camada do problema, é possível criar uma linguagem de domínio específico que ofereça suporte à variabilidade do

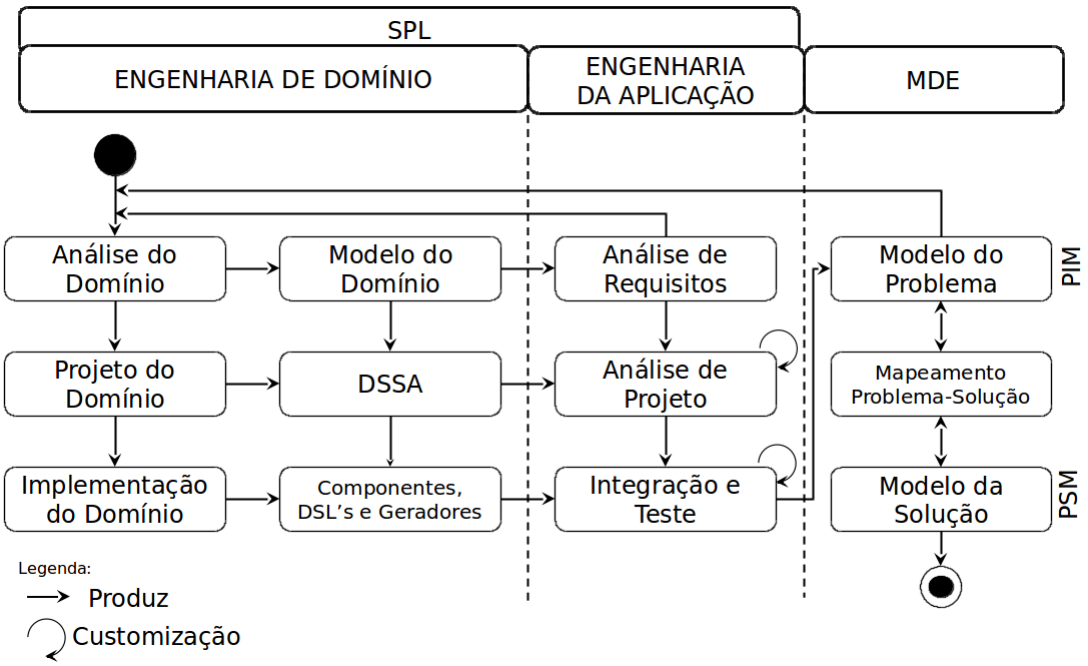


Figura 5.3: Método de integração entre engenharia de Domínio, Aplicação e MDE.

domínio específico.

Mas para isso, é necessário ter uma camada intermediária denotacional, caracterizando aqui a interface de similaridade, viabilizando o reúso de software e todo o arcabouço otimizado para evitar retrabalho.

Perceba que se o modelo da solução PSM evoluir, o impacto a nível de modelo do problema PIM (*Platform Independent Model*) é refletido na análise do domínio que, por sua vez, pode produzir um novo modelo do domínio e, em "efeito cascata", enquadrar todos os artefatos ao novo cenário. O mesmo acontece caso a modificação seja no espaço do problema, a nível de modelo do domínio. Eis o ganho da arquitetura de integração. Ela ajuda na compreensão de como as engenharias se integram no contexto de um projeto, na identificação do ciclo de vida dos componentes internos e ajudam a comunicar decisões e abordagens de projeto.

Abrangência da Integração

Isoladamente a ED e EA podem ser seguidas na íntegra ou parcialmente aplicadas de acordo com a necessidade do projeto. No contexto da arquitetura de integração, a abordagem só faz sentido e justifica-se nos casos em que o projeto arquitetural demanda de uma abordagem denotacional de reúso, capaz de suportar MD-SPL.

Observe que na Figura 5.3 a relação entre as camadas MDE são bidirecionais. O

caminho tradicional é partir do modelo do problema e obter o modelo da solução. No entanto, o inverso também é possível por meio da engenharia reversa.

Análise do Domínio

Um passo importante para se aplicar com sucesso a arquitetura de integração é entender o domínio do problema, os requisitos funcionais e não-funcionais e os ganhos que espera-se ter com a adoção da mesma.

Existe uma classe de problemas que claramente demanda por transformação e/ou configuração de modelos, atrelada ao mapeamento de similaridades e variabilidades. Se essas características relacionarem-se a ponto de haver associações ou dependência, o método pode contribuir como guia à obtenção de uma solução fundamentada no reúso para o problema.

Assim como qualquer outro ponto de *trade off* em um projeto, as escolhas devem ser muito bem analisadas e justificadas pois, dependendo da complexidade do projeto, essa abordagem de integração pode não compensar devido o custo e habilidades exigidas por parte dos projetistas e desenvolvedores.

5.4.2 Framework Arquitetural

Visando documentar as decisões arquiteturais, o framework oferece um conjunto de conceitos, procedimentos comportamentais e estruturais para os modelos de projeto. Assim, através do uso do framework, os projetistas podem escolher, desenvolver e testar rapidamente abordagens usadas para a integração entre as engenharias e recuperar informações arquiteturais adotadas para um domínio. Para lidar com essas variações, o framework permite ao usuário elaborar a modelagem arquitetural seja do ponto de vista endógeno ou exógeno.

A Figura 5.4 ilustra que, várias arquiteturas (A_1, A_2, A_n) pode ser concebidas para atender seus respectivos sistemas (S_1, S_2, S_n). Essa capacidade de atender várias arquiteturas a partir de um mesmo framework, é definida por essa tese como *variabilidade da modelagem arquitetural* do ponto de vista *exógeno*, pois é externo ao framework e percebido apenas no nível da aplicação do mesmo e evidenciado a partir das várias arquiteturas derivadas para atender domínios e requisitos distintos, porém dentro de um escopo.

Já a variabilidade da modelagem arquitetural *endógena* ocorre do ponto de vista interno de uma arquitetura, ou seja, quanto mais visões são usadas para documentar uma arquitetura mais variabilidade endógena ela tem. Esse equilíbrio da quantidade

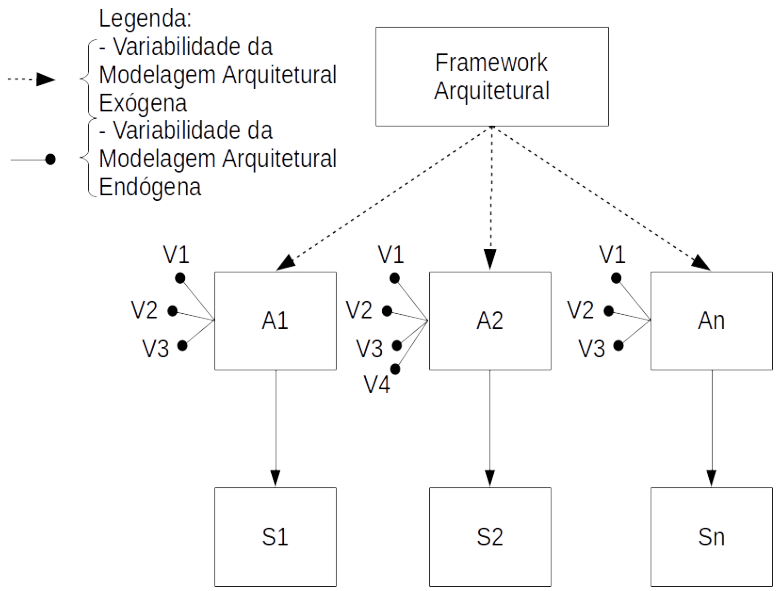


Figura 5.4: Modelagem da Variabilidade Arquitetural e o Framework.

e quais visões usar para uma mesma arquitetura é um problema combinatorial que envolve variáveis diversas, como a análise custo benefício de se gerar mais uma visão. Enquanto essa relação for positiva, vale a pena documentar. No momento que se tornar neutra ou equilibrada, estamos diante do ponto de parada. Se ela se tornar negativa, existe documentação em excesso.

Sendo assim, o projetista de uma família de software pode criar um esquema semelhante ao da Figura 5.4 para documentar que sistemas dentro de uma mesma família necessitam de uma arquitetura específica que diferencia-se das demais por conta de critérios endógenos e exógenos explícitos.

Para exemplificar, a Figura 5.5 apresenta uma instância dessas variabilidades em função do framework arquitetural, usando como base as provas de conceito, discutidas nos Capítulos 7 e 8. A variabilidade exógena é diretamente representada pela relação entre o framework e as arquiteturas. Já a variabilidade endógena no caso da *Prova de Conceito 1*, ocorre pela visão conceitual, de camadas, de módulo, de uso, ontológica e de código. Essas visões detalham a mesma arquitetura, porém com perspectivas diferentes com o intuito de fornecer o conjunto ótimo de informações necessárias para que a mesma seja compreendida, reusada, melhorada, etc. As visões também tem uma conexão entre si que ajuda a documentar melhor a arquitetura. No caso das provas de conceito, essas relações serão discutidas nos Capítulos 7 e 8, Seções 7.1.3 e 8.1.3.

Por fim, a forma na qual a arquitetura será representada pode variar a depender

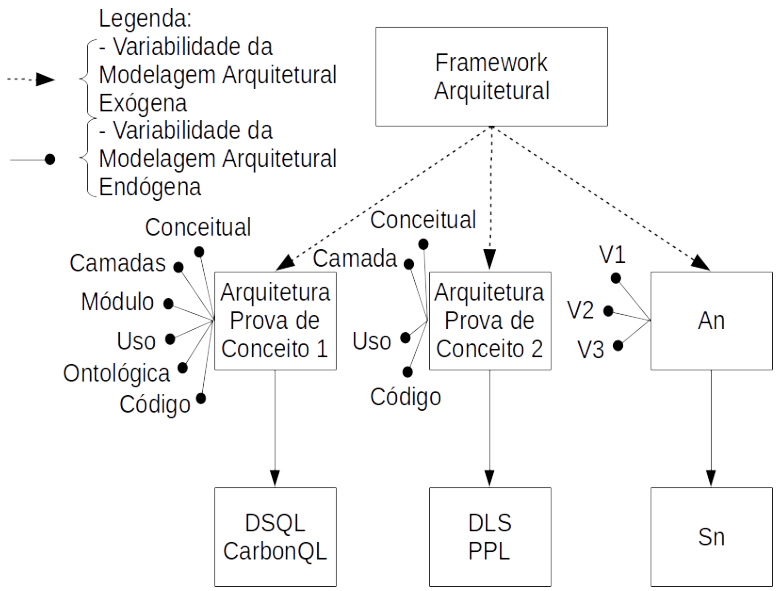


Figura 5.5: Instâncias da variabilidade da modelagem arquitetural.

do grau e do tipo de informações que pretende-se documentar, comunicar e implementar, variações estas que, como será visto a seguir, também serão suportadas pelo framework.

Metadados das Visões

As visões arquiteturais elicitam diferentes atributos de qualidade e em diferentes níveis, por isso, a escolha das visões depende da informação que deseja-se documentar/comunicar e das análises que serão feitas a partir das mesmas, tais como: delegar uma implementação para os desenvolvedores, uma especificação para geração automática de código, ponto de partida para a compreensão do sistema ou um modelo para o planejamento do projeto.

Um detalhe que com frequência é posto de lado por aqueles que propõe arquiteturas é a documentação da visão. As boas práticas arquiteturais sugerem que mais importante que apresentar a visão em si é documentá-la, indicando o seu propósito. Esse framework sugere a documentação conforme metadados indicados na Tabela 5.1.

Visões

Como o propósito desse framework arquitetural é dar suporte a integração das Engenharias (SPL+MDE), foi feito um estudo de observação, variabilidade e simi-

Tabela 5.1: Síntese de um conjunto de metadados para especificação e documentação de uma visão

Nome da Visão: **Tipo da Visão:**
1. Elementos, relacionamentos e propriedades
Tipos de elementos: classes Tipos de relação: agregação Tipos de propriedades: particionamento
2. Quem são os interessados que podem fazer uso da visão?
3. Questões respondidas por meio das informações contidas nessa visão?

laridade sobre os requisitos funcionais e não-funcionais dos sistemas que demandam por tal integração e qual seria o conjunto de visões para esse propósito. Foram identificadas ao menos quatro visões. São elas: conceitual, de módulo, de uso e de código, detalhadas a seguir.

- **Conceitual:** Voltada para todos os interessados, essa visão destaca as funcionalidades e requisitos de ambiente da arquitetura e representa uma visão lógica da mesma. Nessa visão as abstrações chaves são decompostas a partir do domínio do problema, ideal para contextualizar as engenharias, evidenciar porque a integração é relevante e como será feita dentro do contexto do problema;

- **Módulo:** Voltada para a equipe técnica de desenvolvimento, destaca performance, escalabilidade, mostra o sistema em termo de unidade de desenvolvimento e a dependência entre elas. Evidencia manutenção/extensão, facilidade de compreensão, substituição, reúso e portabilidade;

- **Uso:** O propósito desta visão é a extração rápida de subconjuntos e habilitar a construção incremental do sistema;

- **Código:** É a visão física voltada para destacar a topologia do sistema, código-fonte, etc.

A Figura 5.6 apresenta uma perspectiva onde cada bloco representa uma visão e seus respectivos elementos que podem ser usados para documentá-la.

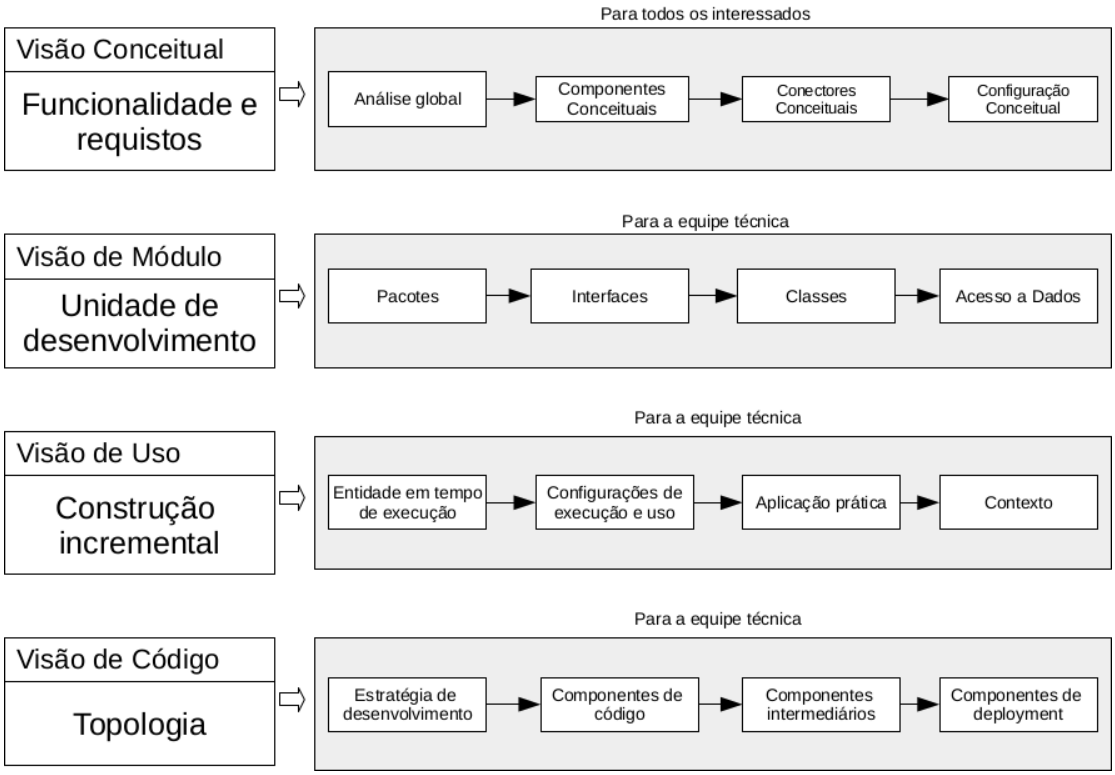


Figura 5.6: As visões previstas pelo framework arquitetural.

Relação entre as Visões

Uma vez que todas as visões de uma arquitetura descrever o mesmo sistema, é esperado que elas tenham muito em comum. Sendo assim, qual a relação entre elas? De que forma essa relação pode ajudar na documentação da integração entre as engenharias?

Entender a relação entre as visões é de fundamental importância para simultaneamente entender como a arquitetura funciona de forma coerente e conceitualmente unificada. Ser claro sobre a relação, fornecendo mapeamentos entre as visões é a chave para aumentar a compreensão, diminuir confusões, mensurar o impacto da modelagem da variabilidade em função do método proposto e do sistema demandante.

5.5 Estratégias de Implementação

Uma vez adotada a arquitetura é hora de tomar decisões técnicas sobre como os pontos de integração e artefatos gerados pelas engenharias serão implementados,

tais como:

- Técnicas Generativas, por exemplo, parsers;
- Frameworks de software - códigos agrupados em pacotes;
- Middleware - CORBA, DCOM, RPC, etc.
- Técnicas baseadas em reúso - COTS, open-source, etc;
- Escrever todo o código manualmente ou usar geração automática de código a partir de modelos, etc.;

5.6 Suporte à Evolução do Método

O principal objetivo de qualquer método é oferecer uma solução compatível com os requisitos, que tenha flexibilidade suficiente para comporta mudanças futuras ou novos requisitos resultantes de sua evolução ao longo do tempo. Sendo assim,

O método proposto é completamente independente do domínio do sistema a ser gerado e pode ser usado parcialmente, considerando apenas um dos dois elementos (arquitetura ou framework), ou em sua totalidade, no caso, os dois elementos juntos.

O suporte para a evolução significa mudanças incrementais enquanto os fatores externos mudam. Para a evolução do método proposto, é necessário identificar o tipo de mudança conforme abaixo:

- **Capturar mudanças.** Em concordância com as necessidades práticas e requisitos explícitos.
- **Representar mudanças.** Identificar e representar mudanças conforme um *template*.
- **Significado para as mudanças.** Verificar o efeito das mudanças no método, para diagnosticar e manter a consistência após a aplicação das mudanças.
- **Formalização de mudanças.** Informar outros pesquisadores sobre as consequências de todas as necessidades de mudanças, aplicar todas as mudanças e indicar as mudanças realizadas.
- **Propagação de mudanças.** Garantir a consistência da dependência dos artefatos após a atualização do método.

- **Validação de mudanças.** Justificar mudanças realizadas e refazê-las de acordo com a necessidade.

5.7 Considerações Finais

Esse capítulo apresentou o método proposto para variabilidade da modelagem arquitetural com base em dois elementos: (i) integração entre as engenharias e (ii) o framework arquitetural.

A arquitetura de integração entre as engenharias tem como objetivo contribuir com uma percepção de aplicação prática para o que muitas vezes é discutido a nível conceitual, deixando muitas dúvidas de como os elementos e processos efetivamente integram-se. Já o framework arquitetural contribui com visões, orientação de integração entre as mesmas e os respectivos metadados.

Vale ressaltar que a arquitetura de integração entre as engenharias e o framework arquitetural foram mapeados pelo modelo teórico como variáveis latentes, ou seja, descrevem conceitos teóricos que não podem ser diretamente mensurados por serem abstratos e por serem novos a ponto de não disporem de um método adequado de avaliação.

Por isso, a aplicação prática do método proposto foi mapeada como uma variável de observação, por meio de indicadores empíricos. Os próximos dois capítulos demonstram a viabilidade da aplicação do método por meio de provas de conceito.

Como trabalho futuro, pode ser feito um estudo escalável para observar a relação da variabilidade arquitetural sobre várias famílias de software, seus ciclos de vida e contribuir com evidências para MD-SPL e ajudar a formalizar um arcabouço que viabilize a construção incremental de sistemas mais eficientes.

Capítulo 6

Introdução às Provas de Conceito

O objetivo desse capítulo é apresentar as provas de conceito. Basicamente o capítulo está dividido em quatro seções. São elas: Seção 6.1 apresenta o critério de seleção das provas. O domínio das provas é apresentado na Seção 6.2. As métricas de avaliação, traduzidas na forma de critérios de sucesso são apresentadas na Seção 6.3. Para finalizar, as considerações finais são apresentadas na Seção 6.4.

6.1 Provas de Conceito

A prova de conceito sintetiza uma solução que atenda os principais requisitos de arquitetura, serve como evidência do método proposto a outros pesquisadores que tenham interesse em aplicá-lo ou modificá-lo para obter vantagens adicionais, avaliar riscos de adoção, etc.

As provas discutidas nesse capítulo foram construídas com base nas seguintes etapas:

- **Preliminar - Definição da abordagem.** Foi selecionado o *protótipo estrutural* como método de construção das provas de conceito por explorar questões arquiteturais, tecnológicas, os respectivos resultados práticos obtidos pela combinação de ambas e por revelar critérios como estabilidade e/ou desempenho do método proposto. Os critérios de sucesso das provas de conceito foram definidos com o intuito de avaliar o resultado obtido;

- **Condução - Selecionar recursos e tecnologias.** Essa etapa será discutida em cada prova (CarbonQL e PPL) pois cada uma tem o seu respectivo escopo

e requisitos. O processo de decisão por qual recurso utilizar ocorreu com base nos requisitos e

- **Avaliação.** Esta etapa se baseia na análise entre os critérios de sucesso e os resultados obtidos.

Quando a prova de conceito é adotada como método de evidência, uma questão que sempre vem à tona é *quantas provas são necessárias para se atingir satisfatoriamente a demonstração desejada*. Um consenso estabelecido orienta que a quantidade de provas exercite os conceitos-chave.

No contexto dessa tese, a partir das questões de pesquisa, foi criado um modelo teórico que representa os conceitos investigados e a relação entre eles e derivada desse modelo duas provas de conceito que representam duas famílias de software com as seguintes características em comum: ambas são DSLs e estão dentro do mesmo domínio específico (engenharia florestal).

No entanto, se diferenciam quanto aos requisitos funcionais. A Prova-1 envolve transformação e tradução de modelos. Já a Prova-2 tem como requisitos funcionais a transformação, tradução e configuração de modelos.

6.2 Domínio das Provas de Conceito

A análise é o primeiro estágio para se definir um conjunto de requisitos para o domínio, com base em três passos: (i) a definição do escopo, (ii) análise das *features* do domínio e (iii) definição do modelo de *features*.

A definição do escopo foi feita por meio do estudo e análise da documentação existente tais como os guias do Painel Intergovernamental sobre Mudança Climática (*Intergovernmental Panel on Climate Change* - IPCC) [117, 118] e de entrevistas com engenheiros florestais, para identificação de objetivos e atuação no processo. Três domínios-alvo foram identificados conforme a Figura 6.1.

Domínio da Gestão de Informação Florestal, que inclui três subdomínios de gestão: (1.1) Inventário Florestal, (1.2) Produção Florestal e (1.3) Logística. Domínio de Conhecimento sobre o Carbono Florestal, que inclui os seguintes subdomínios: (2.1) Representação ontológica e (2.2) Modelos matemáticos. Domínio de Questionamentos/Consultas dos Usuários, que inclui como subdomínio (3.1) Consultas sobre Informação Florestal.

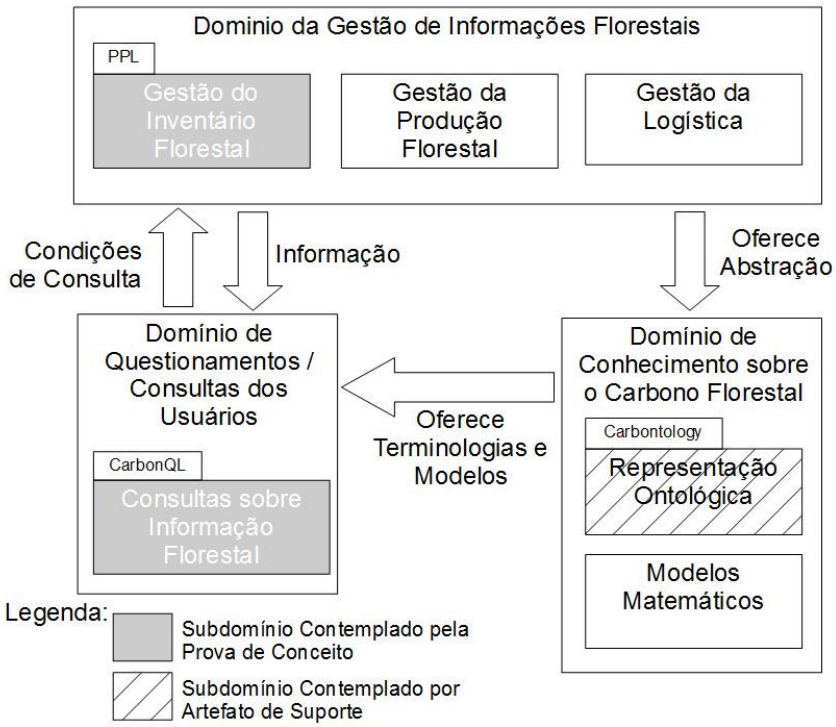


Figura 6.1: Fronteiras dos Domínios.

O domínio de *Gerenciamento da Informação Florestal* pode ser observado mais facilmente no contexto da indústria e pesquisa devido estar relacionado com observações concretas da floresta. O domínio do *Carbono Florestal* é um pouco mais abstrato que o primeiro por se basear em modelos matemáticos de estimativa. O domínio de *Consultas dos Usuários* reflete uma necessidade, onde questionamentos são feitos sob condições de consulta relacionadas ao Gerenciamento da Informação Florestal.

Desses três domínios representados na Figura 6.1, apenas os subdomínios que estão em destaque foram diretamente investigados e contemplados pelas provas de conceito. Vale ressaltar que o subdomínio de *Representação Ontológica*, apesar de estar em destaque, não está diretamente ligado a nenhum processo de transformação ou configuração. No entanto, foi investigado com o intuito de oferecer suporte à etapa de análise da linguagem CarbonQL, por isso, será discutido como uma de suas visões.

Uma premissa para o desenvolvimento de componentes reusáveis é a identificação da variabilidade e da similaridade de *features*. A Tabela 6.1 sintetiza a identificação das mesmas para cada subdomínio.

A variabilidade e similaridade dessas *features* são representadas na Figura 6.2

Tabela 6.1: Similaridade e variabilidade dos requisitos nos subdomínios.

Subdomínio Descrição	Gerenciamento do Inventário Florestal Gerenciamento do que é interessante para se observar ciclicamente na floresta.
Similaridade	O processo de gerenciamento do inventário florestal é composto por: planejamento, coleta de dados e análise.
Variabilidade	(i) O método de coleta dos dados é variável. Pode ser realizado de forma manual ou automática. (ii) As tecnologias de coleta são variáveis, por exemplo, identificação por radiofrequência, redes sensores, etc.
Subdomínio Descrição	Gestão da Produção Florestal Gestão de como obter uma exploração sustentável dos recursos florestais.
Similaridade	O processo de gestão da produção florestal é composto por: planejamento, produção, extração de recursos (madeireiros e não-madeireiros), coleta de dados, análise e reflorestamento (caso a extração seja madeireira).
Variabilidade	(i) Os métodos e tecnologias de produção dependem se a floresta é nativa ou plantada (ii) Métodos e tecnologias de extração devem minimizar os possíveis impactos ambientais. (iii) Métodos e tecnologias de reflorestamento dependem se a floresta é nativa ou plantada.
Subdomínio Descrição	Representação Ontológica Representação do conhecimento da correlação entre a dinâmica florestal e o balanço de carbono. Isso inclui como e quando estimar o carbono, conceitos, relação entre conceitos, etc.
Similaridade	O processo de representação ontológica é composto por: análise, projeto, desenvolvimento, validação e verificação.
Variabilidade	(i) Métodos e tecnologias dependem se a representação ontológica será formal (baseada em lógica de primeira ordem ou outra variante) ou informal (baseada em gráficos como mapas conceituais. A mesma variação acontece para o desenvolvimento e validação/verificação.
Subdomínio Descrição	Modelos Matemáticos Representação não destrutiva da estimativa do carbono
Similaridade	O processo é composto por: análise do domínio específico, criação do modelo (caso ele não exista), experimentação do modelo, validação/verificação
Variabilidade	A definição de constantes e coeficientes são extraídos a partir do método destrutivo
Subdomínio Descrição	Consulta de Informações Florestais Extração de informações derivadas de um modelo sensível ao contexto
Similaridade	As consultas seguem condições definidas
Variabilidade	O resultado das consultas mudam conforme parâmetros e paradigmas de bases de dados (objeto, relacional, etc.).

por meio de um modelo, que organiza hierarquicamente as *features* mandatórias, opcionais e alternativas, distribuídas em três camadas: serviço, funcional e ação.

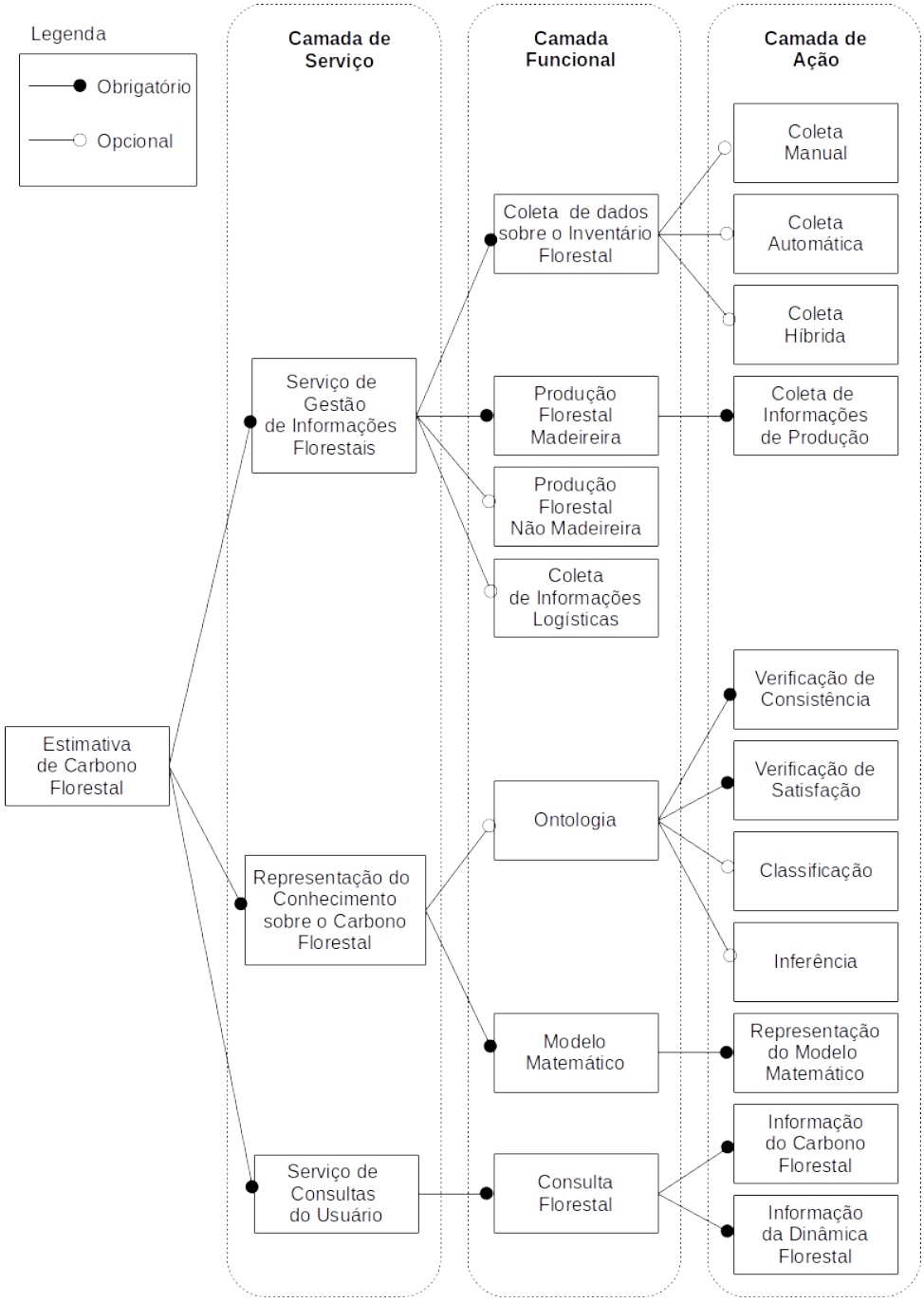


Figura 6.2: Modelo de *Features*.

6.3 Critérios de Sucesso

Com base nas características das provas de conceito, combinado ao método de observação das variáveis, a Tabela 6.2 apresenta os critérios de sucesso, que podem ser atendidos em sua totalidade, parcialmente ou podem não ser atendidos.

Os critérios estão definidos por dimensões de análise. A *dimensão-1* contempla a integração entre a engenharia de domínio, engenharia de aplicação e a engenharia guiada por modelos. As categorias de capacidade e consistência integram essa dimensão com o objetivo de indicar se a abordagem cobre satisfatoriamente os requisitos e se todas as etapas previstas pela integração foram aplicadas. Dependendo da necessidade, as etapas podem ser empregadas na íntegra ou parcialmente. Independente do tipo de aplicação, quanto mais usada, mais tende a ser relevante para o domínio.

O *framework* arquitetural é parte integrante do método proposto, por isso, o mesmo é objeto de análise da *dimensão-2*. É importante que a prova de conceito revele a efetividade e produtividade do ponto de vista da aplicação do *framework*.

A *dimensão-3* examina a variabilidade da modelagem arquitetural do ponto de vista exógeno e endógeno. As categorias de eficiência e eficácia integram essa dimensão com o intuito de identificar se essa abordagem está sendo feita da forma correta e gerando os resultados esperados.

A *dimensão-4* aborda o produto final obtido a partir das provas de conceito e analisa, sobre as categorias da clareza e flexibilidade, os ganhos sobre a utilização do método proposto.

A prova de conceito falha quando algum de seus critérios de sucesso não é atendido. No entanto, mesmo em caso de falha, cabe uma análise para diagnosticar se não houve algum problema de condução da prova. Critérios de sucesso atendidos parcialmente dão uma indicação de onde a prova de conceito pode melhorar e consequentemente, se isso reflete de alguma forma nas variáveis latentes e observáveis.

6.4 Considerações Finais

As provas de conceito definidas sintetizam uma soluções para um domínio real, prático, desafiador, com vários requisitos interessantes de serem observados do ponto de vista arquitetural e evolutivo, incluindo a análise da similaridade e variabilidade. As provas servem como evidência do método proposto a outros pesquisadores que tenham interesse em aplicá-lo ou modificá-lo para obter vantagens adicionais, avaliar riscos de adoção, etc.

Além do mais, elas permitem a compreensão do que era para ser feito e o que era esperado, facilitando a visualização das dimensões do planejamento do projeto e evitando retrabalho. Trouxe velocidade à implementação do projeto, pois foi possível

Tabela 6.2: Critérios de Sucesso das Provas de Conceito

Dimensão 1 - Integração entre as Engenharias		
Índice	Categoria	Critério de Sucesso
CS_01	Capacidade	A integração atende os requisitos funcionais e não-funcionais?
CS_02	Consistência	Todas as etapas previstas pela integração foram executadas?
Dimensão 2 - Framework Arquitetural		
Índice	Categoria	Critério de Sucesso
CS_03	Efetividade	A documentação sugerida pelo *framework* atende a necessidade de identificação da variabilidade arquitetural?
CS_04	Produtividade	Os modelos gerados são usados de forma automática na geração de outros modelos?
Dimensão 3 - Variabilidade da Modelagem Arquitetural		
Índice	Categoria	Critério de Sucesso
CS_05	Eficiência	A identificação da variabilidade exógena da modelagem arquitetural, entre o *framework* e a arquitetura, orientou o reúso?
CS_06	Eficácia	A identificação da variabilidade endógena da modelagem arquitetural contribuiu para a construção incremental e demonstração da portabilidade da DSL?
Dimensão 4 - Qualidade do Projeto Final		
Índice	Categoria	Critério de Sucesso
CS_07	Clareza	Houve melhoria na comunicação do projeto?
CS_08	Flexibilidade	Houve incremento na identificação/flexibilidade à mudanças arquiteturais?

prever os riscos e mitigá-los.

As ameaças que podem afetar o resultado das provas seguem três categorias: confiabilidade, validade e sensibilidade. A coleta de dados sobre as visões de especificação das provas, as características, as particularidades e conexões entre as mesmas ajuda a mitigar riscos relacionados a confiabilidade das evidências. A definição dos critérios de sucesso para as provas ajuda a mitigar riscos relacionados a validade, eliminando a possibilidade de incluir uma prova que não atenda os critérios. O modelo teórico e a definição das variáveis latentes e observáveis ajuda a mitigar riscos relacionados a sensibilidade do estudo.

Capítulo 7

Prova de Conceito 1: Família CarbonQL

O objetivo desse capítulo é apresentar a primeira prova de conceito para o método proposto, que é o protótipo estrutural da linguagem de domínio específico CarbonQL, relacionada ao domínio florestal. Para tanto, foi adotada a seguinte organização: a Seção 7.1 contextualiza a execução da prova, com destaque para os critérios de sucesso sob os quais as prova é avaliada e a definição das fronteiras dos domínios.

A Seção 7.1 aborda a CarbonQL, com destaque para o projeto que comporta as visões: conceitual, de camadas, de módulo, de uso e ontológica. Associada a estratégia de implementação, tem a visão de código que orienta a construção incremental e evolutiva da linguagem. A escolha por essas visões, bem como a conexão entre elas, é discutida na Seção 7.1.3 e a aplicação da linguagem é discutida na Seção 7.1.4.

7.1 CarbonQL: uma DSQL para análise da dinâmica florestal

A dinâmica florestal descreve as mudanças estruturais que ocorrem na floresta ao longo do tempo, como crescimento e mortalidade das árvores, mudanças na biomassa[1] e emissão/sequestro de carbono. No entanto, monitorar a dinâmica florestal não é uma tarefa fácil, em particular, a biomassa e o balanço de carbono, devido

[1] A biomassa é definida como a quantidade (expressa em unidade de massa) de material vegetal por área/unidade de observação da floresta.

a complexidade temporal e espacial associada as variáveis do domínio. A floresta é um sistema natural dinâmico com processos internos, vulnerável a fatores externos como perturbações por mudanças climáticas ou causadas por atividades humanas, como queimadas, desmatamentos, etc.

Sendo assim, a dinâmica florestal representa uma instância complexa do mundo real e tem o ciclo do carbono como um dos principais processos a serem investigados basicamente de duas formas: (i) estimativa via modelos matemáticos, representados por equações alométricas[2] ou via (ii) estudo direto, também conhecido como método destrutivo, que requer o corte e a pesagem da biomassa de uma determinada área. Geralmente o estudo direto antecede o indireto (modelo matemático), exatamente para obtê-lo.

O processo (i) inicia com o inventário florestal para a observação da dinâmica florestal. A partir do inventário, é traçado um plano de observação que contemple a subdivisão da área a ser observada em lotes, definida pelos engenheiros florestais como áreas de parcelas permanentes (para observação a curto e médio prazo) ou contínuas (para observação a longo prazo). Nesse processo, dados são coletados das árvores, como o diâmetro a ≈ 1.3 m do nível do solo, área basal, fuste, espécie da árvore, etc. Em seguida, é necessário estimar a biomassa e o carbono armazenado usando um modelo matemático.

Esses modelos são em geral de domínio específico, ou seja, as equações usadas para estimar o carbono na floresta tropical não podem ser usadas diretamente em uma floresta temperada, já que as espécies de árvores são diferentes e o coeficiente alométrico muda. Particularmente para essa prova de conceito, apenas a variabilidade dos modelos matemáticos são investigados pois os mesmos são passíveis de representação e análise computacional.

Para exemplificar essa variabilidade, o estudo de Araújo *et al.* [6] compara métodos de estimativa de biomassa e carbono a partir de estudos prévios com o objetivo de estabelecer uma equação matemática apropriada para a estimativa em uma área específica da floresta Amazônica. Nesse estudo, quatorze equações foram identificadas para a mesma área conforme Tabela 7.1.

Isso demonstra a diversidade de modelos matemáticos desenvolvidos para uma mesma região, onde os engenheiros florestais precisam lidar, muitas vezes, com um conjunto de equações para analisar o cenário. A grande diversidade desses modelos

[2]Alometria é o estudo de mudanças relativas a proporção de um atributo em função de outro durante o crescimento orgânico. Um exemplo de relação alométrica é a massa do esqueleto e a massa do corpo.

Tabela 7.1: Equações e coeficientes de regressão (Araújo *et. al.* [6], tabela 1). FW - *fresh weight* - é o peso fresco (kg), D - *diameter* - diâmetro à altura do peito (cm), H - *tree hight* - altura total da árvore (m), ρ densidade média da madeira (g cm^{-3}), M teor médio de umidade por unidade de biomassa fresca, α, β, γ e Φ coeficiente de regressão, e R^2 coeficiente de determinação.

Equações	Coeficientes				
	α	β	γ	Φ	R^2
$FW_1 = \alpha D^\beta$	4.06	1.76	—	—	0.87
$FW_2 = \alpha + \beta D + \gamma D^2$	400.32	39.99	0.97	—	0.87
$FW_3 = \alpha + \beta D + \gamma (D^2) H$	318.09	38.12	0.03	—	0.90
$FW_4 = \alpha + \beta D + \gamma (D^2) + \Phi (D^2) H$	350.69	55.50	2.25	0.09	0.92
$FW_5 = \alpha + \beta (D^2) + \gamma (D^2) H$	175.83	0.79	0.07	—	0.88
$FW_6 = \alpha + \beta D + \gamma H$	533.26	130.22	51.00	—	0.82
$FW_7 = \alpha (D^\beta) H^\gamma$	0.026	1.529	1.747	—	0.92
$FW_8 = (\alpha D^\beta)/(1 - M)$	0.465	2.202	—	—	0.90
$FW_9 = (\alpha D^2)/(1 - M)$	1.120	—	—	—	0.94
$FW_{10} = \{exp[\alpha + \beta ln(D^2)]\}/(1 - M)$	-1.966	1.242	—	—	0.97
$FW_{11} = \{exp[\alpha + \beta ln(H \rho D^2)]\}/(1 - M)$	-2.904	0.993	—	—	0.99
$FW_{12} = \{exp[\alpha + \beta ln(\rho D^2)]\}/(1 - M)$	-0.906	1.177	—	—	0.99
$FW_{13} = \{exp[\alpha + \beta ln(H D^2)]\}/(1 - M)$	-3.843	1.035	—	—	0.97
$FW_{14} = \{exp[\alpha + \beta ln(D^2) \gamma ln(\rho)]\}/(1 - M)$	-1.020	1.185	1.071	—	0.99

matemáticos e o fato de que novos modelos são elaborados a todo momento, levanta a discussão dentro da comunidade de engenheiros florestais sobre a capacidade de repetição, auditoria e efetividade dos modelos. Por isso, é importante que os engenheiros tenham autonomia na elaboração de consultas sobre os dados coletados em campo e a manipulação das equações de forma que possam extrair informações, obter respostas aos questionamentos e consequentemente entender melhor o domínio.

Com base nessa demanda real, surgiu a CarbonQL, uma linguagem de consulta de domínio específico (*Domain-Specific Query Language* - DSQL) com foco na transformação, componentização e reúso. Baseada em conceitos relevantes e *features* do domínio, a CarbonQL segue o método proposto no Capítulo 5 e é apresentada como prova de conceito do tipo *protótipo estrutural*.

7.1.1 Projeto

Em meio a um escopo abrangente, essa prova de conceito encontra-se no subdomínio de *Consulta de Informações Florestais* e visa compreender a adequação do modelo proposto ao subdomínio e o impacto do mesmo sobre a variabilidade da modelagem arquitetural conforme requisitos apresentados na Tabela 7.2.

Tabela 7.2: Requisitos da CarbonQL.

Prova de Conceito: CarbonQL	
Arquitetura de Referência: Framework do Capítulo 5	
Requisitos funcionais:	Requisitos não-funcionais:
Realizar consultas em base de dados relacional	Variabilidade no espaço do problema
A linguagem de consulta deve ser formal e textual	Confiabilidade
Ser como SQL, porém com um escopo mais simples. Ao invés de conter as quatro cláusulas (SELECT, INSERT, UPDATE e DELETE), implementar apenas uma cláusula equivalente a SELECT. A intenção dos usuários não é manipular os dados e as estruturas do banco e sim obter respostas aos seus questionamentos científicos	Reusabilidade no espaço da solução
Dar suporte à especificação dos modelos matemáticos (equações) nas consultas	Usabilidade

Visão Conceitual

Essa visão tem como objetivo apresentar de forma integrada três dimensões:

- A instanciação do problema no contexto do método proposto;
- Como se dá a integração entre ED/EA e MDE no contexto do problema;
- Onde efetivamente ocorre a variabilidade e a similaridade (passível de reúso por meio de uma família de software).

A Figura 7.1 apresenta a camada identificado como “zero”, que representa o modelo específico de plataforma mais próximo do nível de execução e restrito aos modelos relacionais pois no final de toda a cadeia de processamento, o código gerado será em SQL. Em seguida, a camada “um” é a intermediária, responsável pelo mapeamento objeto-relacional. A camada de mais alto nível, identificada como “dois” representa o modelo independente de plataforma onde estão efetivamente representados os conceitos da dinâmica florestal e estimativa do carbono contidos na CarbonQL que permite aos especialistas do domínio criarem consultas focando apenas no espaço do problema sem ter que se preocupar com o espaço da solução ou a camada de nível “zero”.

A Tabela 7.3 sintetiza o propósito da visão conceitual. Todas as etapas previstas na engenharia de domínio e engenharia de aplicação foram realizadas. Por ter uma representação direta a nível de domínio específico, a engenharia de aplicação entra em destaque, para indicar a equivalência entre os artefatos previstos e os gerados tanto no espaço do problema como no espaço da solução. O mesmo acontece com a MDE.

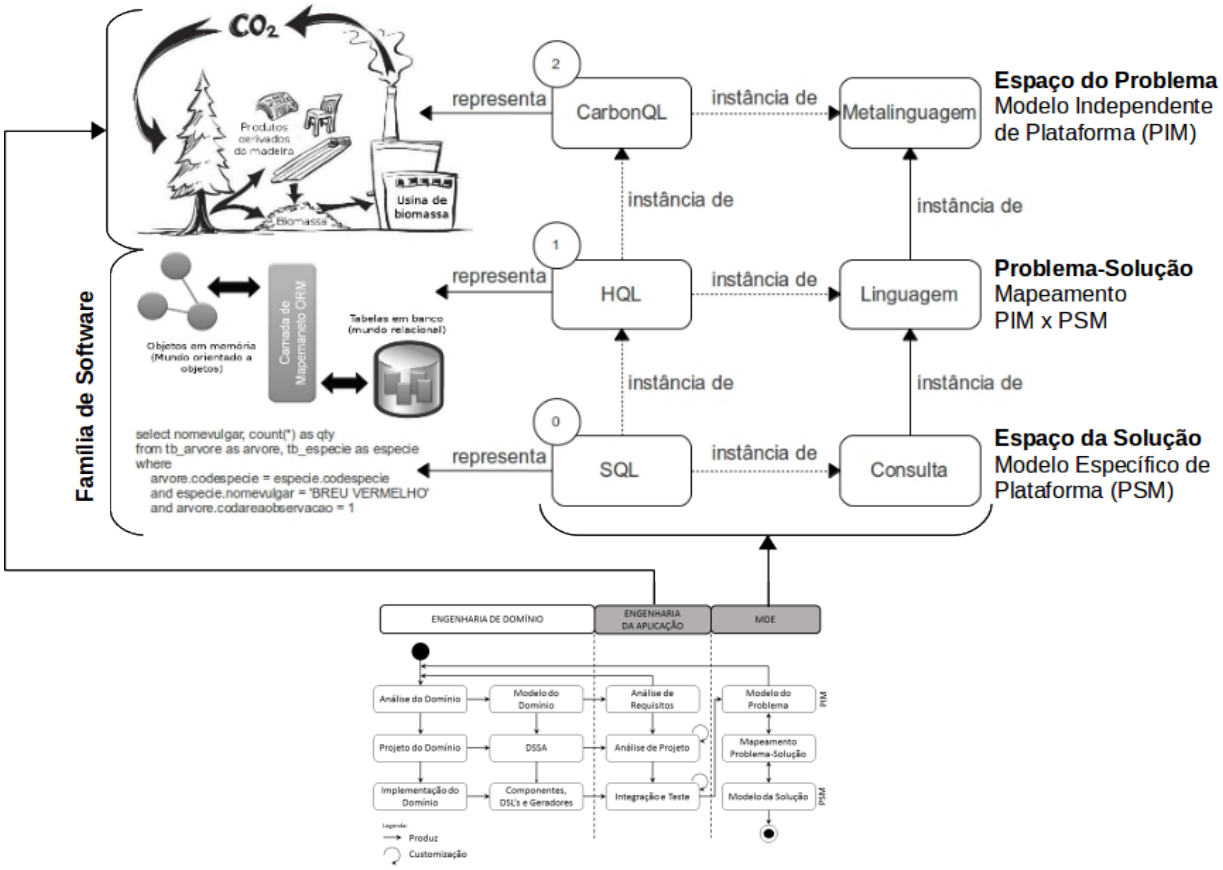


Figura 7.1: Visão Conceitual - Prova de Conceito CarbonQL.

Tabela 7.3: Síntese da Visão Conceitual - CarbonQL

Nome da Visão: Conceitual
1. Elementos, relacionamentos e propriedades
Tipos de elementos: modelos Tipos de relação: instância de, representa Tipos de propriedades: coesão, associação
2. Quem são os interessados que podem fazer uso da visão?
Testadores, mantenedores, arquitetos, analistas e engenheiros de linguagens
3. Questões respondidas por meio das informações contidas nessa visão?
- A correlação entre os modelos derivados a partir da arquitetura de referência podem ser facilmente visualizados? - A correlação entre o modelo de domínio específico gerado e o espaço do problema foi explicitamente representada? - A correlação entre o modelo de transição entre a solução e o problema foi explicitamente representado? - É possível visualizar a família de software?

Visão de Camadas

Essa visão tem como objetivo apresentar as camadas da arquitetura para o domínio específico. A Figura 7.2 ilustra as várias camadas e relação entre elas. A

partir da observação florestal, uma camada tecnológica logo acima oferece suporte via hardware para a coleta automática dos dados em campo com base em RFID, redes sensores e/ou GPS. Sendo assim, a camada de hardware *usa* o ambiente real para coletar dados, que por sua vez, são persistidos em bases de dados. A relação entre a camada de hardware e dados é bidirecional, pois ambas estão *autorizadas-a-usar* uma a outra em função da troca de mensagens de sucesso ou não tanto para a persistência como para a coleta de dados.

A camada de negócio contempla os serviços de gerenciamento e faz a interface entre a camada do usuário e a de transmissão de dados. As três linearmente, estão *autorizadas-a-usar* a camada vizinha em função das trocas de mensagens de sucesso ou não para as requisições solicitadas.

A Tabela 7.4 sintetiza o propósito da visão de camadas. Para esse contexto, busca-se *alta coesão*, de forma que as camadas tenham uma responsabilidade bem definida e *baixo acoplamento* para que as dependências entre camadas sejam minimizadas, de forma que a modificação em uma não tenha impacto nas demais.

Uma outra vantagem dessa visão é a fácil comunicação das decisões de projeto entre a equipe de desenvolvimento e demais interessados, facilitando uma rápida validação do que foi idealizado.

Tabela 7.4: Síntese da Visão de Camadas - CarbonQL

Nome da Visão: Camadas
1. Elementos, relacionamentos e propriedades
Tipos de elementos: camadas Tipos de relação: autorizado-a-usar (unidirecional ou bidirecional) Tipos de propriedades: coesão, acoplamento
2. Quem são os interessados que podem fazer uso da visão?
Testadores, mantenedores, arquitetos, analistas, engenheiros de linguagens e usuários
3. Questões respondidas por meio das informações contidas nessa visão?
- Está claro o tipo de associação entre as camadas? - As camadas são coesas? - Qual o nível de acoplamento entre as camadas? Se alguma for removida, as demais funcionam? - É possível visualizar a correlação entre as camadas e as que foram estabelecidas no modelo de *feature* (serviço, funcional e ação)?

Visão de Módulo

Essa visão tem como objetivo apresentar os componentes que oferecem o serviço de suporte à manipulação e persistência no banco de dados (responsabilidade alheia à CarbonQL).

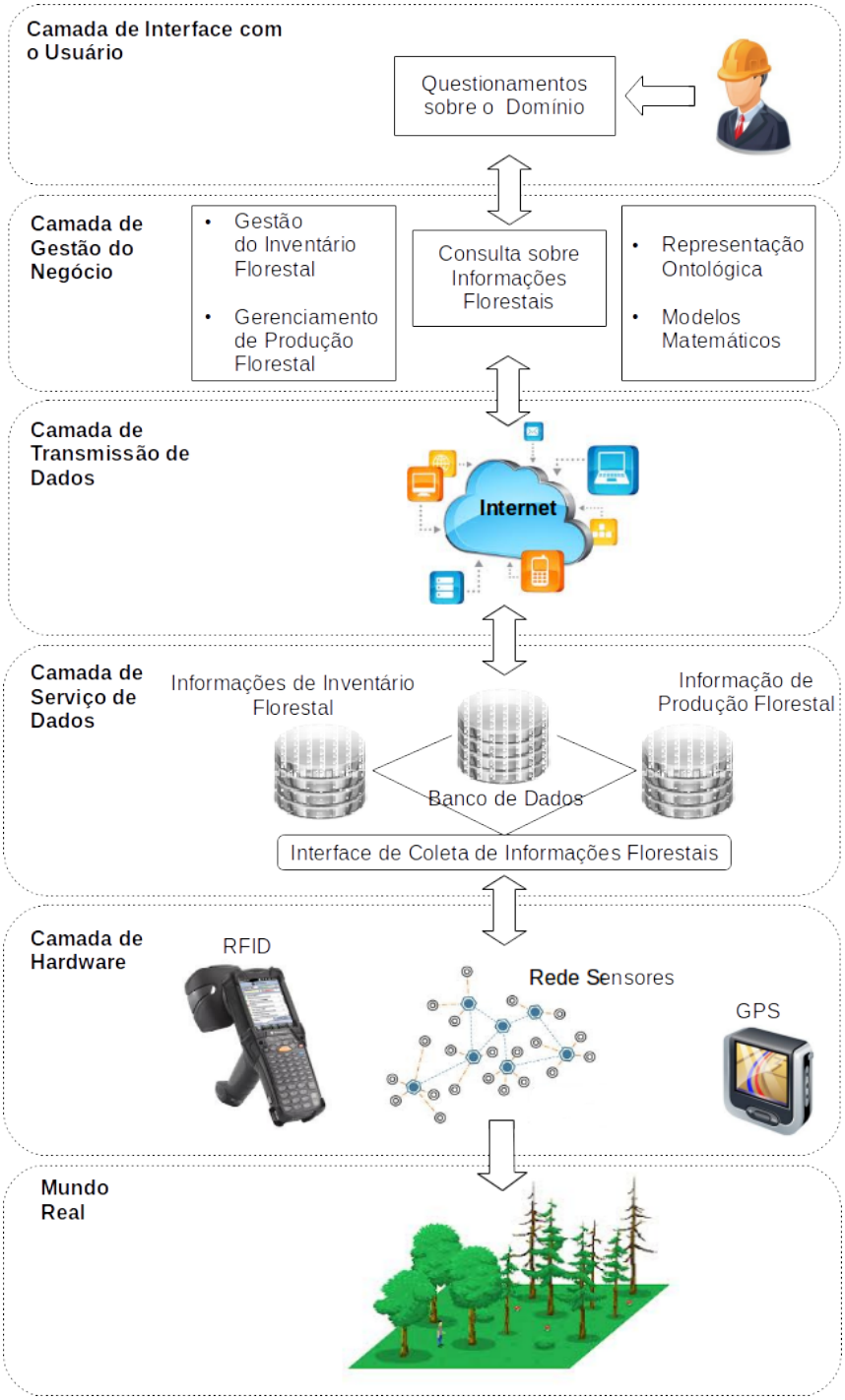


Figura 7.2: Visão de Camadas - Prova de Conceito CarbonQL

Via diagrama de disposição física, a Figura 7.3, destaca a distribuição e os componentes através dos "nós de processamento". Observe que o *framework* ORM usa o componente de banco de dados e a infraestrutura de persistência para processar

o suporte aos modelos mapeados. Cada tabela no banco de dados se torna uma classe, com seus atributos e métodos *getters/setters* para garantir a identidade dos objetos e ter acesso aos dados, ou seja, relacionar facilmente registros com objetos e garantir que não existem duplicadas impróprias.

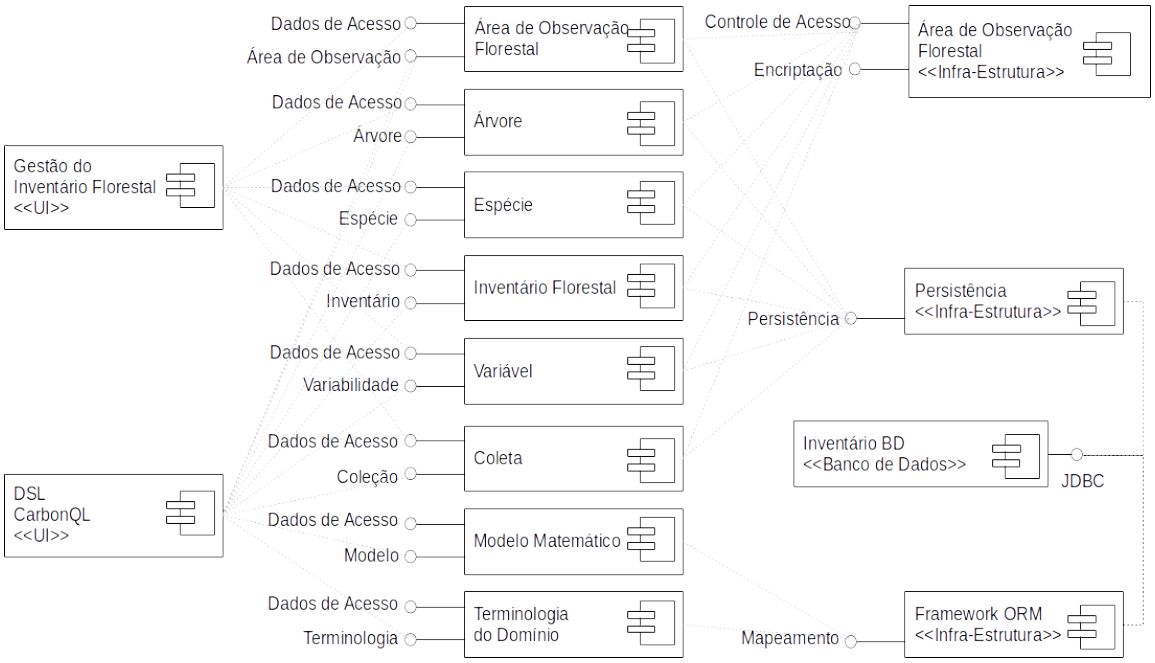


Figura 7.3: Visão de Módulo - Prova de Conceito CarbonQL.

A Tabela 7.5 sintetiza o propósito da visão de módulo. Essa visão é de fundamental importância para a equipe de desenvolvimento, pois permite comunicar os pontos concentradores de processos, que no caso, é a *persistência* e o *controle de acesso*. Da mesma forma, existem os pontos demandantes de processos como a interface da CarbonQL e a interface do sistema de gestão do inventário florestal, responsável pelos CRUD's (acrônimo de *Create, Read, Update* e *Delete*) do domínio.

Os fatores arquiteturais que podem comprometer o desempenho dos processos é uma baixa coesão. Se os componentes assumirem responsabilidades indevidas, termos uma sobrecarga e comprometimento dos processos. A Figura 7.12(c) apresenta, no contexto da ferramenta de implementação, os componentes e relacionamentos previstos nessa visão.

A visão de uso nos permite levantar hipóteses como: o que acontece se a especificação de consulta mudar e exigir um novo formalismo, terminologia do domínio ou o próprio domínio mudar? O que acontece se o código de consulta não for mais para um *framework* ORM e sim para um outro paradigma como, por exemplo, totalmente

Tabela 7.5: Síntese da Visão de Módulo - CarbonQL

Nome da Visão: Módulo
1. Elementos, relacionamentos e propriedades
Tipos de elementos: componentes Tipos de relação: associação Tipos de propriedades: coesão
2. Quem são os interessados que podem fazer uso da visão?
Testadores, mantenedores, arquitetos, analistas, engenheiros de linguagens
3. Questões respondidas por meio das informações contidas nessa visão?
- Quais são os processos críticos? - Que fatores na arquitetura podem comprometer o desempenho dos processos? - Que processos podem ser classificados como prioritários? - Que processos podem ser particionados para melhorar o desempenho?

orientado a objeto? Com as devidas modificações, a arquitetura tem condições de comportar tal mudanças ao longo do tempo?

Elaborar uma visão de uso, exige do arquiteto uma percepção de passado, presente e futuro. Essa percepção pode fazer toda diferença entre o sucesso e o fracasso de um sistema.

Visão de Uso

O propósito dessa visão é a extração rápida de subconjuntos e habilitar a construção incremental da DSQL. Para isso, a Figura 7.4 apresenta um diagrama de contexto que ilustra a composição interna da CarbonQL, o que é esperado como modelo de entrada e saída e como ambos são estruturados.

A Tabela 7.6 sintetiza o propósito da visão de uso.

Tabela 7.6: Síntese da Visão de Uso - CarbonQL

Nome da Visão: Uso
1. Elementos, relacionamentos e propriedades
Tipos de elementos: modelos Tipos de relação: especificado-em, conforme, instância-de e entrada-para Tipos de propriedades: coesão, acoplamento
2. Quem são os interessados que podem fazer uso da visão?
Testadores, mantenedores, arquitetos, analistas, engenheiros de linguagens
3. Questões respondidas por meio das informações contidas nessa visão?
- Está claro o tipo de associação entre os modelos? - Os modelos são coesos? - O nível de acoplamento entre os modelos é satisfatório? - A visão permite identificar os elementos evolutivos?

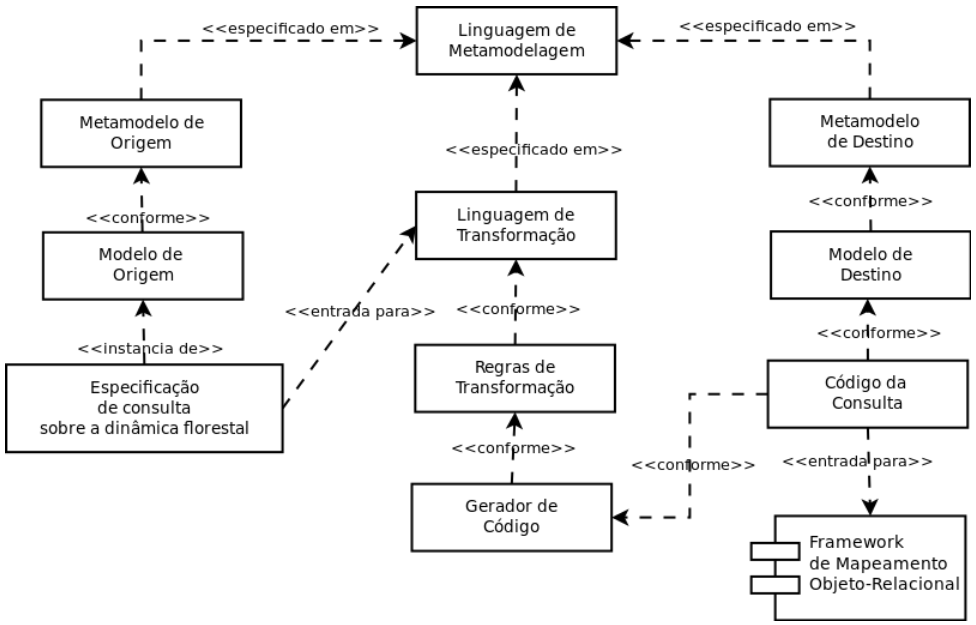


Figura 7.4: Visão de Uso - Prova de Conceito CarbonQL.

Visão Ontológica

O projeto de uma DSL envolve uma análise profunda do domínio, o que acarreta tempo e esforço significativo ao processo de desenvolvimento. De acordo com um estudo desenvolvido por Tairas *et al.* [119], a utilização de ontologias na fase de análise de uma DSL pode reduzir significativamente o custo do projeto por meio da representação da terminologia do domínio, axiomas e validação formal do modelo construído por meio de raciocinadores e isso foi o que motivou o desenvolvimento da Carbontology, cujo escopo é a dinâmica florestal e estimativa de carbono.

Essa ontologia, formalmente definida em OWL DL[3], contempla ao total, 46 classes nomeadas, dentre elas, temos a *biomassa, bioma, tipos de florestas, ciclo de carbono, métodos de estimativa de carbono, métodos de estimativa de biomassa, coeficientes, variáveis,* etc.

Uma característica relevante de DL está na capacidade de representar uma base de conhecimento com distinção clara entre o domínio do problema (TBox) e um problema particular (ABox).

A TBox *(Terminological Box)*, contém o conhecimento do domínio na forma de

[3]Lógica de descrição (DL) é uma variante decidível da Lógica de Primeira Ordem. Esta lógica envolve uma família de linguagens que podem ser usadas para representar uma terminologia. Basicamente, essa lógica descreve o domínio por intermédio de predicados unários (conceitos atômicos), predicados binários (regras atômicas), construtores e axiomas.

terminologia, usada para construir declarações que descrevem propriedades gerais sobre os conceitos. Já a ABox *(Assertional Box)* contém o conhecimento específico relacionado a instâncias das classes. Essa distinção permite especificar várias ABoxes para uma TBox, oferecendo a possibilidade de enriquecer a representação do domínio em questão.

Todos os termos que compõem a TBox e ABox da Carbontology foram especificados em inglês. Sendo assim, o identificador da classe nomeada que representa o balanço de carbono é especificado como `<Class IRI=‘‘#CarbonBalance’’/>` que é uma instância de `owl:Class`. No nome das classes e propriedades, adotou-se o padrão de concatenação de palavras com iniciais maiúsculas, sem o uso de conectores como hífen ou ponto, bem como o singular como padrão para o nome dos conceitos.

Várias instâncias podem ser especificadas a partir de uma classe nomeada. No caso da classe *CarbonBalance*, todo balanço de carbono será positivo, neutro ou negativo. Sendo assim, é possível especificar essas instâncias como uma declaração do tipo:

```
<ClassAssertion>
        <Class IRI="#CarbonBalance"/>
        <NamedIndividual IRI="#Negative"/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI="#CarbonBalance"/>
        <NamedIndividual IRI="#Neutral"/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI="#CarbonBalance"/>
        <NamedIndividual IRI="#Positive"/>
</ClassAssertion>
```

Figura 7.5: Instâncias da classe *CarbonBalance*.

Optou-se pela definição de domínio e contradomínio das propriedades que podem ser, basicamente, de dois tipos: (i) propriedades objeto P_O e (ii) propriedades de tipos de dados P_D.

As propriedades P_O relacionam uma instância a outra. Já as propriedades P_D relacionam uma instância a um valor do tipo de dados XML Schema. As restrições de propriedades são: `owl:allValuesFrom, owl:someValuesFrom, owl:hasValue` e as especificações de cardinalidade são: `owl:cardinality, owl:maxCardinality, owl:minCardinality`.

As expressões lógicas a seguir descrevem uma propriedade de tipo de dados P_D `hasName` cujo domínio é a classe `CarbonEstimationMethod` e o contradomínio é o tipo primitivo `xsd:string`. Significa que todo método de estimativa de carbono

tem um nome de identificação.

$$\texttt{hasName} \in \texttt{P}_\texttt{D}$$

$$\top \sqsubseteq \forall\, \texttt{hasName} \,.\, \texttt{CarbonEstimationMethod}\ (\texttt{CarbonEstimationMethod} \neq \bot)$$

$$\top \sqsubseteq \forall \begin{pmatrix} \texttt{hasName}^- \,.\, \texttt{xsd:string} \\ (\texttt{xsd:string} \neq \bot) \end{pmatrix}$$

Essa mesma descrição é especificada em OWL DL conforme Figura 7.6

```
<SubClassOf>
        <Class IRI="#CarbonEstimationMethod"/>
        <DataSomeValuesFrom>
            <DataProperty IRI="#hasName"/>
            <Datatype abbreviatedIRI="xsd:string"/>
        </DataSomeValuesFrom>
</SubClassOf>
```

Figura 7.6: Propriedade de tipo de dados da classe *CarbonEstimationMethod*.

As parcelas permanentes representam unidades de amostra localizadas em áreas florestais demarcadas e observadas que podem ser de forma contínua visando conhecer o comportamento das espécies florestais e seus processos dinâmicos de crescimento e mortalidade a curto, médio e longo prazo. A expressão lógica a seguir define que instâncias da classe `PermanentParcel` relacionam-se através da propriedade P_O `locatedIn` com pelo menos uma instância da classe `Forest`.

$$\texttt{PermanentParcel} \sqsubseteq \exists\, \texttt{locatedIn} \,.\, \texttt{Forest}$$

Desta forma, `PermanentParcel` é subclasse da classe anônima que contém instâncias que se relacionam, via `locatedIn`, com pelo menos uma instâncias da classe `Forest`. O código em OWL que representa essa definição lógica pode ser observado na Figura 7.7.

```
<SubClassOf>
        <Class IRI="#PermanentParcel"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="#locatedIn"/>
            <Class IRI="#Forest"/>
        </ObjectSomeValuesFrom>
</SubClassOf>
```

Figura 7.7: Propriedade objeto *locatedIn* e uma classe anônima.

A biomassa é definida como a quantidade, expressa em unidades de massa, de

material vegetal por área em uma floresta. Em geral, os componentes da biomassa estimados são: biomassa viva acima do nível do solo, composta por árvores e arbustos (não considerando as raízes); biomassa viva abaixo do nível do solo, composta por raízes; e a biomassa morta, formada pela serrapilheira[4] e troncos caídos. A soma de todos os componentes considerados fornece a biomassa total.

Sendo assim, a biomassa viva pode ser especializada em duas classes disjuntas por não compartilharem instâncias: acima (*Aboveground*) ou abaixo do nível do solo (*Belowground*) conforme demonstrado na Figura 7.8.

```
1 <DisjointClasses>
2         <Class IRI="#Aboveground"/>
3         <Class IRI="#Belowgraund"/>
4 </DisjointClasses>
```

Figura 7.8: Classes disjuntas.

A Tabela 7.7 sintetiza o propósito da visão ontológica. Essa visão permite visualizar a possibilidade de reúso da especificação não só a nível de compreensão dos requisitos para a implementação da DSQL, mas também a nível de projeto interno, de forma que a Carbontology seja incorporada no modelo de origem, de transformação ou modelo de destino e guie o usuário na elaboração de consultas, como um dicionário de dados.

Tabela 7.7: Síntese da Visão Ontológica - CarbonQL

Nome da Visão: Ontológica
1. Elementos, relacionamentos e propriedades
Tipos de elementos: classes nomeadas e não-nomeadas Tipos de relação: todo-parte, é-um, junção, disjunção, etc. Tipos de propriedades: coesão, consistência, completude, objeto, de dados, etc.
2. Quem são os interessados que podem fazer uso da visão?
Arquitetos, analistas, engenheiros de linguagens
3. Questões respondidas por meio das informações contidas nessa visão?
- Qual é a terminologia do domínio? - Qual é o nível de detalhamento da terminologia? - Em quais elementos arquiteturais essa terminologia pode ser reusada? - A representação é consistente?

[4]Serrapilheira é a camada formada pelo acúmulo de matéria orgânica morta em diferentes estágios de decomposição que reveste superficialmente o solo.

7.1.2 Estratégia de Implementação

A Figura 7.9 sintetiza a estratégia de implementação para a linguagem CarbonQL sendo que cada elemento destacado por linhas tracejadas demandaram implementação e os demais foram meramente reusados.

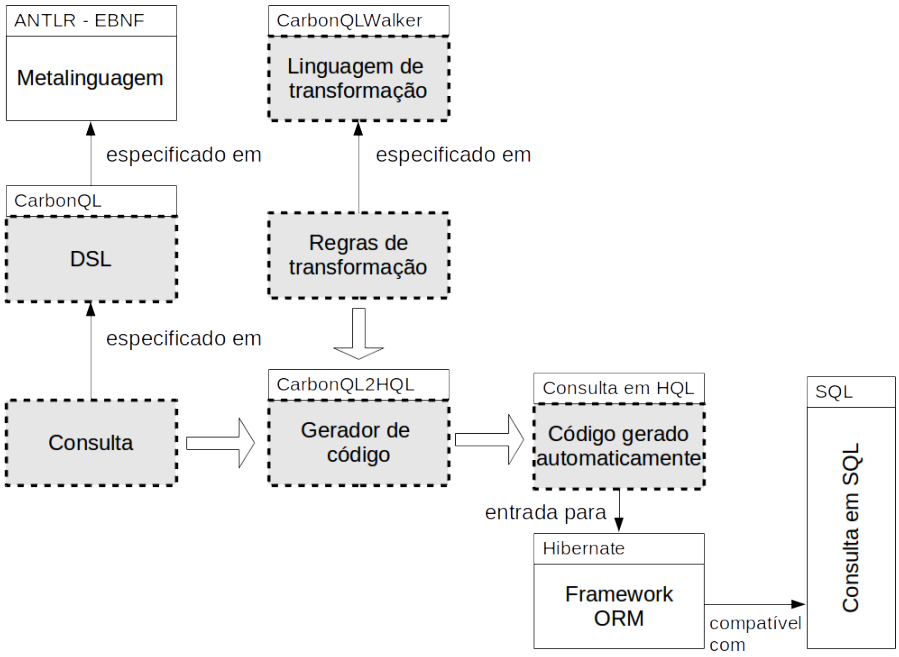


Figura 7.9: Escolhas tecnológicas para a implementação da CarbonQL.

Para que uma dada consulta elaborada na linguagem CarbonQL seja efetivamente executada no nível de um banco de dados relacional, é necessário definir um gerador de código que traduza a consulta de CarbonQL para SQL (*Structured Query Language*). No entanto, as linguagens são de paradigmas diferentes. A linguagem CarbonQL foi projetada para atender as necessidades de consultas dos engenheiros florestais na área de dinâmica florestal e estimativa de carbono. Devido essa proximidade com o mundo real, podemos fazer uma analogia e associar a CarbonQL com o paradigma orientado a objeto (OO) onde os conceitos são equivalentes à classes e as instâncias desses conceitos são objetos com seus respectivos atributos e comportamentos. Como exemplo temos as árvores (classe) que possuem diâmetro, altura (atributos) e no contexto da dinâmica florestal, nascem e morrem (comportamento). Indivíduos (instâncias) que fazem parte desse conjunto árvore são inventariados pelos engenheiros florestais em áreas de parcelas provisórias ou permanentes onde os engenheiros coletam dados e observam o comportamento.

Já o paradigma relacional está em uma camada diferente de expressividade com

foco para as tabelas, os relacionamentos e a normalização. Para unir essa camada de mais alto nível acessível aos engenheiros florestais com essa camada de mais baixo nível, próxima da execução computacional de consultas é necessário realizar um mapeamento.

Existem várias pesquisas e métodos nessa área de mapeamento objeto-relacional e vários *frameworks* que implementam e modularizam tais métodos. No contexto dessa prova de conceito, optou-se pela utilização do Hibernate [120] por ser *open source* e consequentemente capaz de suportar adaptações, compatível com a plataforma de desenvolvimento da CarbonQL, possui uma extensa documentação e é mantido por uma comunidade ativa.

O Hibernate possui uma linguagem nativa chamada HQL (*Hibernate Query Language*) [121] similar, porém com o propósito de ser mais simples que SQL do ponto de vista de seus usuários já que ela foi projetada para atender desenvolvedores e analistas de software que buscam maior produtividade na fase de análise, implementação, teste e manutenção, concentrando-se apenas nas regras de negócio sem ter que se envolver necessariamente com detalhes de arquitetura e projeto de Banco de Dados.

Já SQL visa atender a necessidade dos arquitetos e administradores de Banco de Dados que a usam de forma otimizada para extrair o melhor desempenho possível do Banco de Dados. Dessa forma, cada equipe pode fragmentar o desenvolvimento de sistemas complexos em partes menores e se concentrar na solução do problema de acordo com a sua especialidade.

Sendo assim, o gerador de código **CarbonQL2HQL**, destacado na Figura 7.9, necessita de regras claras de transformação que são definidas com base na linguagem de transformação **CarbonQLWalker**. O resultado desse processo é um código gerado automaticamente para a linguagem HQL e em seguida submetido ao *framework* Hibernate de mapeamento objeto-relacional (em inglês, *object-relational mapping* - ORM) que é responsável em gerenciar a transição das consultas de um nível abstrato para um nível executável em SQL. Dessa forma, o Hibernate encapsula o modelo específico de plataforma (PSM) enquanto que a CarbonQL encapsula o modelo independente de plataforma (PIM).

O ANTLR [3, 122] foi escolhido por disponibilizar um ambiente otimizado para o desenvolvimento e o ANTLRWorks como ambiente de desenvolvimento complementar por agrupa ferramentas como editor, interpretador, rastreamento de código, visualização gráfica de caminhos ambíguos, regras gramaticais na forma de autômato

finito determinístico (em inglês, *Deterministic Finite Automaton* - DFA).

Visão de Código

A Figura 7.10 ilustra o projeto do compilador CarbonQL com quatro etapas distintas: a construção da gramática livre de contexto, o analisador léxico, sintático e semântico.

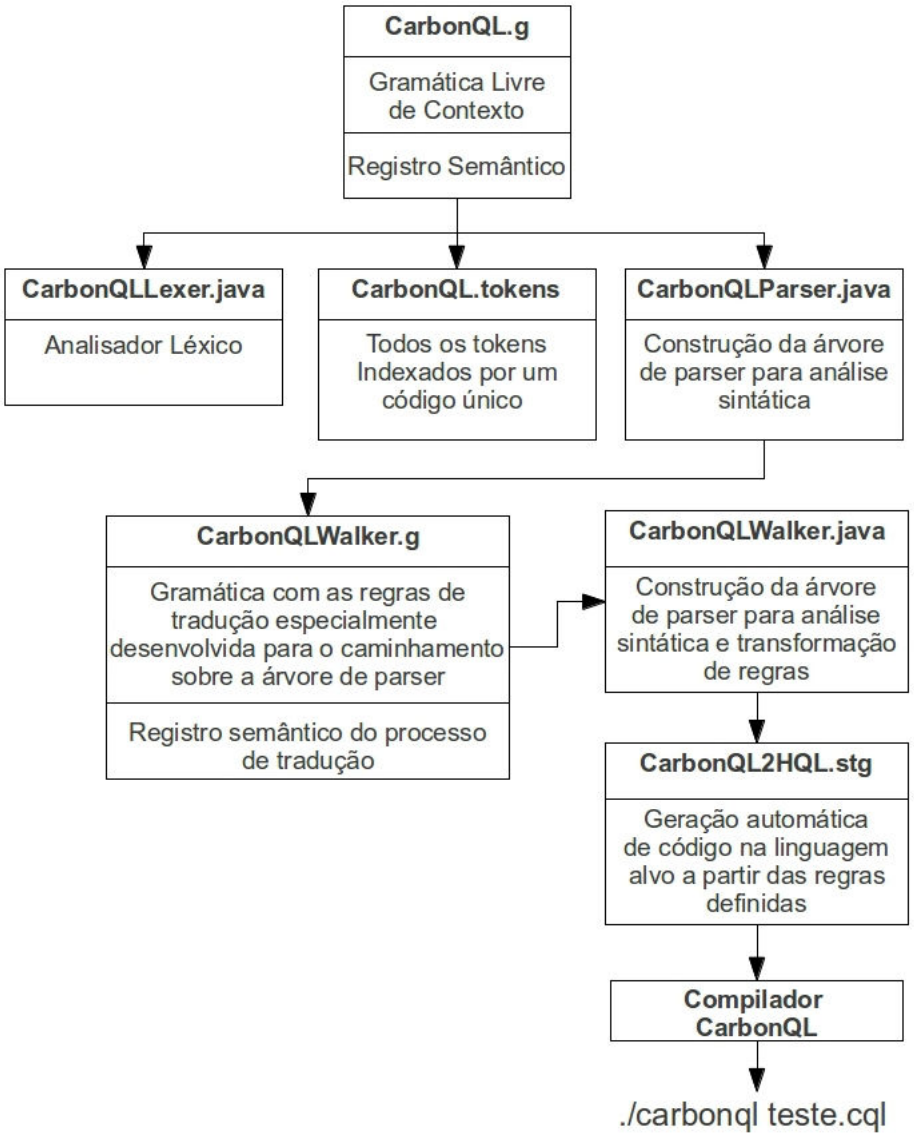


Figura 7.10: Projeto do compilador CarbonQL.

O primeiro passo para a construção da CarbonQL iniciou com a definição da gramática livre de contexto. Embora existam diferentes notações para se especificar

uma linguagem, a Forma Normal de Backus - BNF, tem se popularizado devido a sua facilidade de compreensão. ANTLR usa a versão estendida chamada EBNF.

A EBNF é uma metalinguagem, ou seja, uma linguagem usada para especificar outras linguagens. A formalização da sintaxe da linguagem é feita através da especificação das regras de produção, ou regras de substituição, onde cada símbolo da metalinguagem é associada uma ou mais cadeias de símbolos, indicando as diversas possibilidades de substituição. Um fragmento do código que implementa a gramática CarbonQL é apresentado na Figura 7.11. As principais regras estão destacadas na cor azul. A interpretação dessas regras é apresentada na Tabela 7.8.

Tabela 7.8: Interpretação das principais regras da gramática CarbonQL.

Regras	Interpretação
resultClause	$\{rc \mid rc$ é a clausula de resultado esperado$\}$. Elemento obrigatório para uma consulta em CarbonQL, rc contempla alguns operadores e funções de agregação como COUNT, MAX, MIN, SUM e AVG que permitem formatar os resultados da consulta como esperado.
conditionClause	$\{cc \mid cc$ é a clausula de condição$\}$. Elemento opcional para uma consulta em CarbonQL, cc contempla a definição de parâmetros lógicos de associação do tipo $A :: B \mid A\ has - relation - with\ B$, equivalente a um *join* em SQL.
solutionModifier	$\{sm \mid sm$ é a clausula de modificação da solução$\}$. Elemento opcional para uma consulta em CarbonQL, sm é usado para formatar a seleção através de agrupamento e/ou ordenação.
IDENTIFIER	$\{id \mid id : (a..z\|A..Z\|\_)(a..z\|A..Z\|\_\|0..9)*\}$. id é um *token* terminal que representa a terminologia do domínio. Com o intuito de restringir o universo de consulta para o domínio específico, mediante a implementação, o usuário pode utilizar termos contidos na Carbontology. Caso o termo usado não faça parte, o usuário pode usar o termo de sua preferência t_p com a devida associação ao termo ontológico contido na Carbontology t_o

Como mostrado na Figura 7.10, a gramática em EBNF contida no arquivo **CarbonQL.g** é o artefato de entrada para que o ANTLR gere o arquivo **CarbonQL.tokens** com os *tokens* que representam os símbolos terminais da gramática (ver Figura 7.12(a)).

Do ponto de vista da implementação do compilador, o analisador léxico atua como uma interface entre o código de uma consulta e o analisador sintático, conver-

```
CarbonQL_query : queryElements SEMICOLON EOF ;
queryElements : resultClause conditionClause? solutionModifier? ;
//////////////// RESULT CLAUSE
resultClause : OPEN_BRACE resultTerm CLOSE_BRACE ;
resultTerm : newExpression ( COMMA newExpression )* ;
newExpression : (targetObject | expression) aliasedExpression? ;
aliasedExpression : AS IDENTIFIER ;
targetObject : IDENTIFIER targetProperty? ;
targetProperty : DOT IDENTIFIER ;
expression : logicalOrExpression ;
logicalOrExpression
  : logicalAndExpression ( OR logicalAndExpression )* ;
logicalAndExpression
  : relationalExpression ( AND relationalExpression )* ;
relationalExpression
  : additiveExpression
      ((LESS additiveExpression)
      |(GREATER additiveExpression)
      |(LESS_EQUAL additiveExpression)
      |(GREATER_EQUAL additiveExpression)
      |(IN additiveExpression)
      |(NOT IN additiveExpression)
      |(BETWEEN betweenList)
      |(NOT BETWEEN betweenList)
      |(LIKE targetObject))* ;
betweenList : targetObject ( AND targetObject ) ;
additiveExpression
  : multiplyExpression
      (( PLUS multiplyExpression )
        |( MINUS multiplyExpression ))* ;
multiplyExpression
  : unaryExpression
      (( ASTERISK unaryExpression
      |( DIVIDE unaryExpression
      |( PERCENT unaryExpression )* ;
unaryExpression
  : quantifiedExpression | constant | aggregate | INTEGER ;
quantifiedExpression
  : SOME targetObject
  | EXISTS targetObject
  | ALL targetObject
  | ANY targetObject ;
constant : NULL | TRUE | FALSE ;
aggregate
  : SUM OPEN_BRACE (additiveExpression | targetObject)  CLOSE_BRACE
  | AVG OPEN_BRACE (additiveExpression | targetObject)  CLOSE_BRACE
  | MAX OPEN_BRACE (additiveExpression | targetObject)  CLOSE_BRACE
  | MIN OPEN_BRACE (additiveExpression | targetObject)  CLOSE_BRACE
  | COUNT OPEN_BRACE ( ASTERISK | ((DISTINCT | ALL)? targetObject )) CLOSE_BRACE ;
//////////////// CONDITION CLAUSE
conditionClause : OPEN_SQUARE_BRACKET conditionList CLOSE_SQUARE_BRACKET ;
conditionList : conditionTerm ( COMMA conditionTerm)* ;
conditionTerm : targetObject ( HAS_RELATIONSHIP | HAS_INSTANCE) range ;
range : targetObject | value | INTEGER ;
value : ASPAS IDENTIFIER* ASPAS ;
//////////////// SOLUTION MODIFIER
solutionModifier : groupClause orderClause ;
groupClause : GROUP BY targetObject ( COMMA targetObject )* ;
orderClause : ORDER BY orderElement ( COMMA orderElement )* ;
orderElement : targetObject ( ascendingOrDescending )? ;
ascendingOrDescending : ( ASC | ASCENDING ) | ( DESC | DESCENDING) ;
```

Figura 7.11: Fragmento do código que implementa a gramática CarbonQL.

tendo a sequência de caracteres que constituem a consulta na sequência de *tokens* que o analisador sintático validará. O analisador léxico do compilador tem como função percorrer o código da consulta da esquerda para a direita, agrupando os símbolos de cada item léxico.

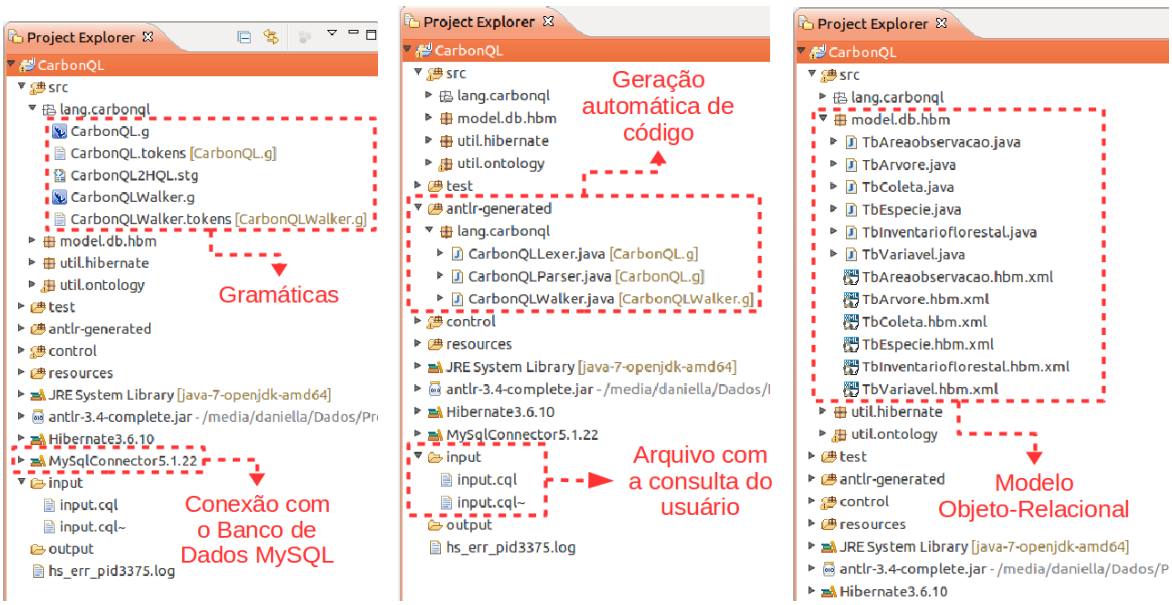
A análise da sequência com que os *tokens* presentes no código da consulta se apresentam, a partir da qual se efetua a síntese da árvore sintática. Na Figura 7.10 o arquivo **CarbonQLParser.java** contém a especificação dos *tokens* e os blocos de regras que compõem a análise sintática.

O analisador sintático processará a sequência de *tokens*, proveniente do código da consulta, extraídos pelo analisador léxico **CarbonQLLexer.java** (ver Figura 7.12(b)). A partir dessa sequência o analisador sintático efetua a verificação. A análise sintática cuida exclusivamente das sentenças da linguagem, e procura, com base na gramática, validar a estrutura das mesmas.

O principal objetivo do analisador semântico é o de captar o significado das ações a serem tomadas com base no código da consulta. Ele compõe a terceira etapa de análise do compilador e refere-se a tradução do código CarbonQL para uma especificação em HQL, normalmente validada pela análise semântica. A semântica de uma sentença é o significado por ela assumido dentro do texto analisado, enquanto que a semântica da linguagem é a interpretação que se pode atribuir ao conjunto de todas as sentenças. A fase da análise semântica, porém, é de difícil formalização, exigindo notações complexas.

Devido a esse fato, a maioria das DSLs adotam especificações mais simplificadas, com bases informais. A principal função da análise semântica é criar, a partir do código fonte, uma interpretação desse texto, expressa em alguma notação adequada que é, geralmente, uma linguagem intermediária do compilador. Essa operação é realizada com base nas informações existentes nas tabelas construídas pelos outros analisadores como a tabela de palavras reservadas.

Após a especificação da gramática livre de contexto, do analisador léxico, sintático e semântico, o compilador CarbonQL está pronto para receber códigos com extensão *.cql e validá-los (ver Figura 7.12(b)). Caso não existam erros, o compilador apresentará ao final uma mensagem informando que a compilação foi realizada com sucesso. Caso contrário, o compilador CarbonQL apresenta uma mensagem de erro genérica e não aponta a linha do código fonte ou o tipo de erro identificado. Esse tratamento de erro é um dos pontos que serão aprimorados em versões futuras do compilador CarbonQL.



(a) Código das gramáticas. (b) Geração automática de código. (c) Modelo objeto-relacional.

Figura 7.12: Detalhes de implementação da CarbonQL com base na exploração do projeto via ambiente de desenvolvimento Eclipse.

As palavras reservadas que formam a sintaxe da CarbonQL estão definidos em inglês devido ser o idioma comum no meio científico. Como mostrado na Tabela 7.8, toda consulta definida em CarbonQL deve conter no mínimo um bloco obrigatório e dois opcionais.

Na clausula de condição existem dois operadores lógicos de associação que são de fundamental importância. O operador de junção classe-classe "::" é usado para indicar o relacionamento entre dois conceitos ou classes do domínio. Já o operador de junção classe-instância ":<" é usado para indicar que determinada classe possuí um elemento específico.

A Tabela 7.9 sintetiza o propósito da visão de código. Os elementos apresentam estrutura coesa e estão de acordo com as visões de uso, ontológica e de camadas.

7.1.3 Relação entre as Visões

A Figura 7.13 apresenta a relação entre as visões da CarbonQL. Tudo começa pela *visão conceitual* que oferece um panorama de como o problema pode ser solucionado pelo método proposto. Essa visão contribui com diagramas que instanciam os vários domínios, elicita a variabilidade e similaridade e se há uma família de software

Tabela 7.9: Síntese da Visão de Código - CarbonQL

Nome da Visão: Código
1. Elementos, relacionamentos e propriedades
Tipos de elementos: Classes e gramáticas Tipos de relação: associação Tipos de propriedades: coesão
2. Quem são os interessados que podem fazer uso da visão?
Testadores, mantenedores, arquitetos, analistas, engenheiros de linguagens
3. Questões respondidas por meio das informações contidas nessa visão?
- Qual a relação entre as classes? - Todos os elementos (classes e gramáticas) estão de acordo com a arquitetura? - Os elementos apresentam estrutura coesa? - Os elementos tecnológicos de suporte a implementação foram definidos?

para o contexto.

A partir dessa modelagem entra a *visão de camadas* que se apoia na especificação definida pela conceitual e complementa a percepção do domínio do ponto de vista funcional, elicitando serviços complementares que podem se conectar via relação de *uso* direcional ou bidirecional.

A *visão de módulo* se apoia na especificação definida pela visão de camadas para identificar possíveis módulos/componentes cada vez mais em uma granularidade menor, se aproximando de uma realidade implementacional que virá pela frente.

A *visão de uso*, se apoia na visão conceitual para direcionar o desenvolvimento, devidamente fortalecido pela *visão ontológica* que oferece a terminologia, com destaque para conceitos-chaves do domínio.

A *visão de código* foi a última a ser definida e integra todas as outras de forma direta ou indireta. A partir dessas visões é possível compreender como a arquitetura efetivamente funciona, implementar ou evoluir a implementação da linguagem.

7.1.4 Aplicação da CarbonQL

Toda consulta especificada na linguagem CarbonQL é resultado de um questionamento lógico. A Tabela 7.10 exemplifica alguns questionamentos dos especialistas identificados na fase de análise. As perguntas 3, 4, e 5 envolvem modelos matemáticos usados por engenheiros florestais do INPA (Instituto Nacional de Pesquisas da Amazônia) para estimativa de carbono, volume e valor comercial de uma dada espécie.

De forma mais detalhada, a Figura 7.14 demonstra como uma consulta especificada em CarbonQL é traduzida para uma consulta HQL com base em três passos:

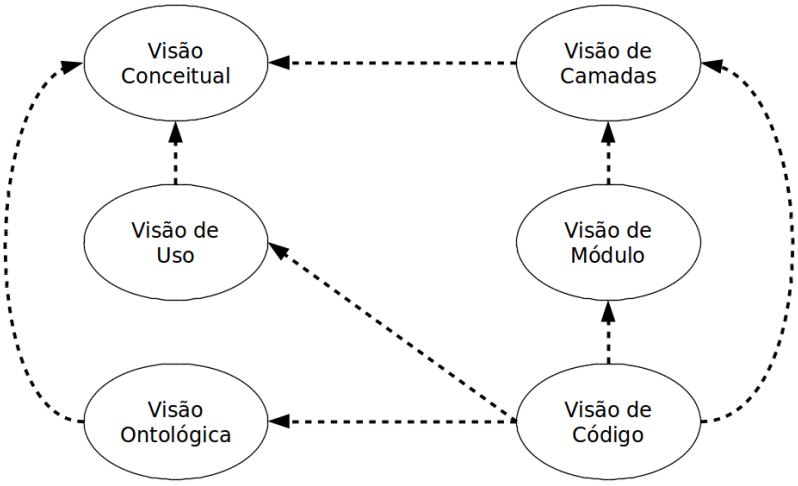


Figura 7.13: Relação entre as visões adotadas na prova de conceito CarbonQL.

(i) a consulta em CarbonQL é transformada usando o método de tradução de sintaxe direta em uma árvore de sintaxe abstrata (*Abstract Syntax Tree* - AST); (ii) A AST-CarbonQL é transformada em uma AST-HQL; (iii) um código HQL é gerado a partir da AST-HQL.

O Hibernate, por sua vez, segue os mesmos passos: cosulta HQL → AST-HQL de entrada → AST-SQL de saída → consulta SQL. Após esse processo, a consulta SQL é executada e o resultado pode ser visualizado.

Esse caminho de transformação, código de entrada → AST de entrada → AST de saída → código de saída, é conhecido como um *padrão de criação* do tipo transformação de código-para-código [3].

Como a CarbonQL ainda não dispõe de uma interface gráfica, foi utilizado o *console* do próprio ambiente de desenvolvimento, o Eclipse, para visualizar o processo de transformação. A Figura 7.15 apresenta uma consulta em CarbonQL e a AST de entrada, juntamente com a consulta HQL gerada automaticamente.

Em seguida, na Figura 7.16 é possível visualizar a validação do mapeamento objeto-relacional para que a consulta seja traduzida em SQL, executada, para que finalmente o resultado da consulta seja apresentado.

A Figura 7.17 apresenta a consulta SQL resultante. Vale ressaltar que a CarbonQL foi desenvolvida como protótipo estrutural a nível de prova de conceito, com o objetivo de validar a viabilidade arquitetural, a eficiência do método e o reúso efetivo.

Mas a intenção é que incrementalmente esse protótipo evolua para uma versão produtiva da linguagem. Das três sub-árvores de regras, somente duas, a *result-*

Tabela 7.10: Consultas de interesse dos especialistas.

ID	Consultas
Pergunta 1	Quantas árvores de determinada espécie existem em uma dada parcela permanente?
Consulta CarbonQL para a Pergunta 1	`(count (specie.name)) [tree grows-in 1, tree::specie, specie.popularname:< 'BREU VERMELHO'] ;`
Pergunta 2	Quantas árvores de uma dada espécie e parcela permanente morreram de acordo com os dados do inventário de um determinado ano?
Consulta CarbonQL para a Pergunta 2	`(count (specie.name as qty)) [tree status died, tree grows-in 2, inventoryyear :< 2009, collection::tree, collection::inventory, tree::specie ] group by specie.name order by qty ;`
Pergunta 3	Qual é a estimativa de carbono para uma dada parcela permanente ao longo dos inventários florestais?
Consulta CarbonQL para a Pergunta 3	`(sum(carbontot)) [ (CONVERT((2.7179*pow(treedap,1.8774)*0.584)*0.485, decimal(10,2)) as carbontot ) [ collection::forestinventory, collection::tree, variable::collection, tree::specie, tree.codpermanentparcel :< 1 ] group by inventoryyear, tree, popularname, treedap ] group by inventoryyear ;`
Pergunta 4	Qual a estimativa de volume das árvores para uma dada espécie e parcela permanente?
Consulta CarbonQL para a Pergunta 4	`(sum(volumetot)) [(CONVERT(0.001176*pow(treedap,1.99868), decimal(10,2)) as volumetot) [collection::forestinventory, collection::tree, variable::collection, tree::specie, tree.codpermanentparcel :< 1, specie.popularname :< 'ABIURANA'] group by inventoryyear, tree, popularname, treedap] group by inventoryyear ;`
Pergunta 5	Qual a estimativa do valor comercial das árvores de uma determinada espécie e parcela permanente ao longo dos inventários florestais?
Consulta CarbonQL para a Pergunta 5	`(sum(marketvalue)) [(CONVERT(1200*(0.001176*pow(treedap,1.99868)), decimal(10,2)) as marketvalue) [collection::forestinventory, collection::tree, variable::collection, tree::specie, tree.codpermanentparcel :< 2, specie.popularname :< 'ANGELIM PEDRA'] group by inventoryyear, tree, popularname, treedap] group by inventoryyear ;`

Clause e a *solutionModifier*, estão com regras de transformação implementadas para HQL, o que permite chegar a nível de execução em SQL. Do ponto de vista da prova de conceito, essas *features* implementadas foram suficientes para validar requisitos, arquitetura, e aplicação no domínio específico.

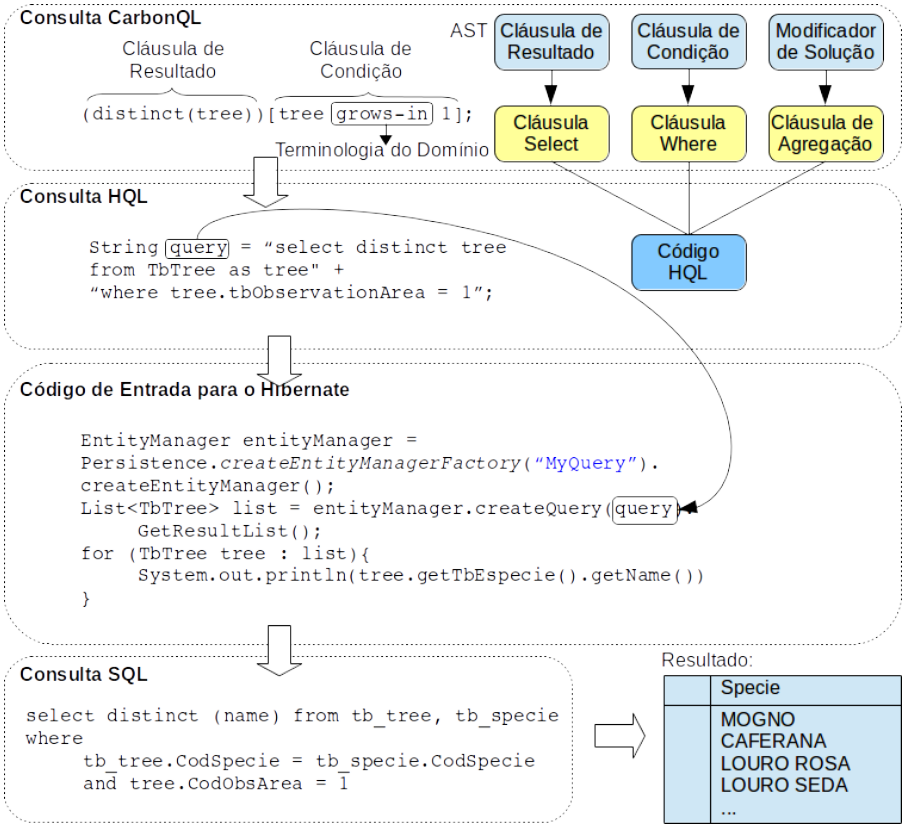


Figura 7.14: Transformação de uma consulta CarbonQL em SQL.

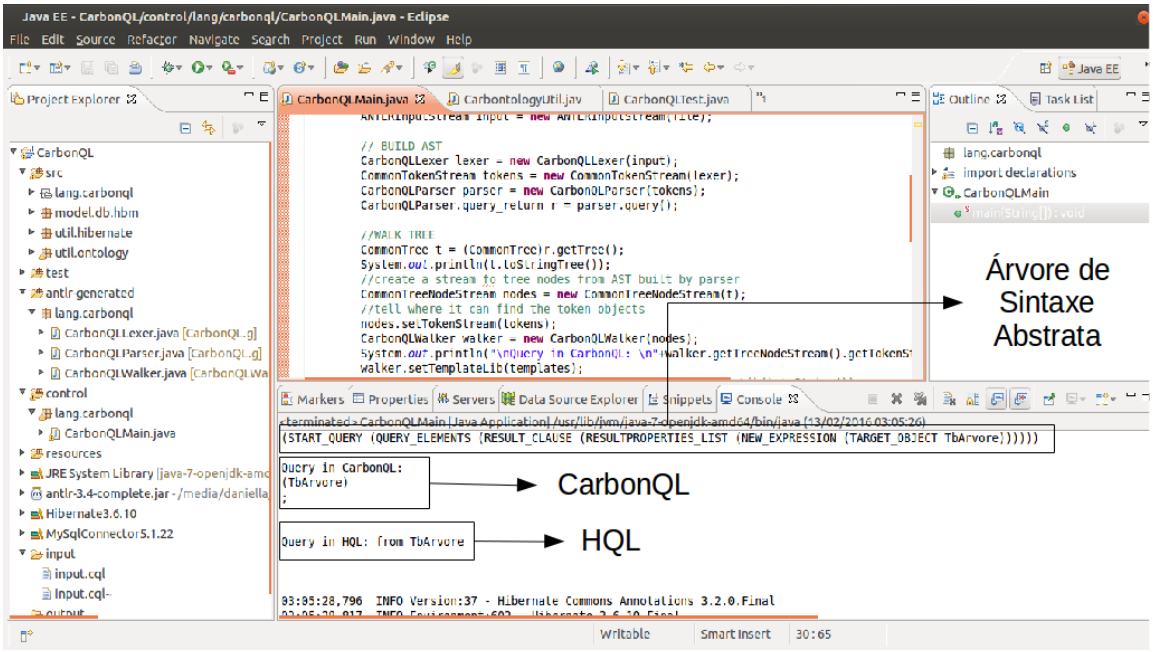


Figura 7.15: Logs de transformação entre as linguagens.

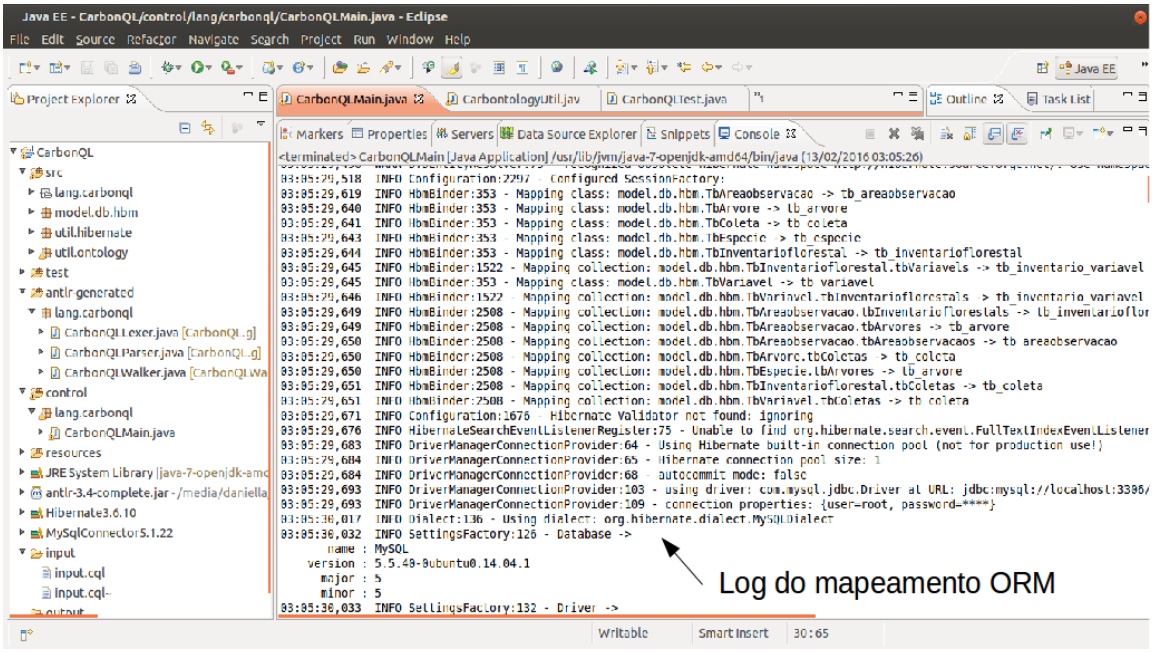


Figura 7.16: Logs do mapeamento ORM.

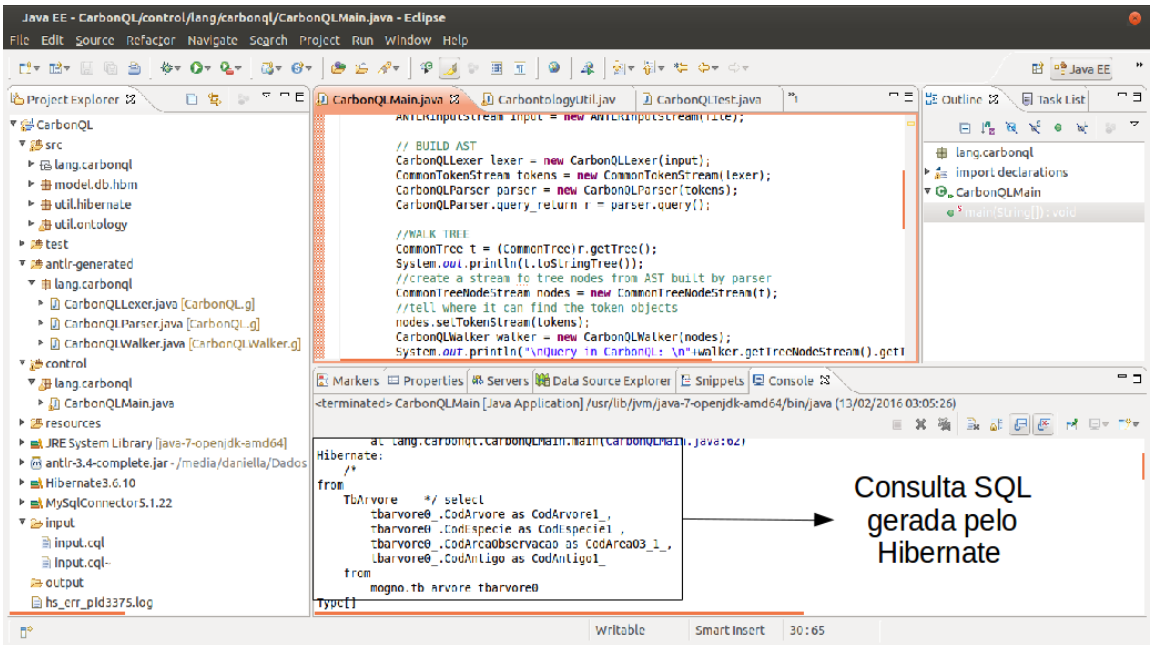


Figura 7.17: Logs da consulta gerada em SQL.

Um outro ponto de evolução é a própria sintaxe da linguagem que pode ser melhorada via estudo de usabilidade, com o objetivo de atender melhor a necessidade dos usuários.

7.2 Considerações Finais

A prova de conceito realizada serviu para sintetizar soluções de um domínio real, prático, desafiador, com vários requisitos interessantes de serem observados do ponto de vista arquitetural e evolutivo, incluindo a análise da similaridade e variabilidade. A prova serve como evidência do método proposto a outros pesquisadores que tenham interesse em aplicá-lo ou modificá-lo para obter vantagens adicionais, avaliar riscos de adoção, etc.

Além do mais, ajudou a documentar e comunicar decisões de projeto, permitiu a compreensão do que era para ser feito e o que era esperado, facilitando a visualização das dimensões do projeto e evitando retrabalho. Trouxe velocidade à implementação, pois foi possível prever os riscos, mitigá-los e conduziu a estruturação da família CarbonQL.

As ameaças que podem afetar o resultado da prova de conceito 1 seguem três categorias: confiabilidade, validade e sensibilidade. A coleta de dados sobre as visões de especificação da prova, as características, as particularidades e conexões entre as mesmas ajuda a mitigar riscos relacionados a confiabilidade das evidências. A definição dos critérios de sucesso ajuda a mitigar riscos relacionados a validade, eliminando a possibilidade de incluir uma prova que não atenda os critérios. O modelo teórico e a definição das variáveis latentes e observáveis ajuda a mitigar riscos relacionados à sensibilidade do estudo.

Capítulo 8

Prova de Conceito 2: Família PPL

O objetivo desse capítulo é apresentar a segunda prova de conceito para o método proposto, que é o protótipo estrutural da linguagem de domínio específico PPL, relacionada ao domínio florestal. Para tanto, foi adotada a seguinte organização: a Seção 8.1 aborda a modelagem arquitetural que comporta as visões: conceitual, de camadas e uso. A estratégia de implementação contempla a visão de código. A conexão entre essas visões é discutida na Seção 8.1.3. Para finalizar, algumas considerações finais são apresentadas na Seção 8.2.

8.1 PPL: uma DSL para rotas otimizadas de inventário florestal

O inventário de florestas nativas é uma atividade que exige intenso planejamento logístico, pois envolve a administração de recursos materiais e financeiros bem como o deslocamento de especialistas em áreas de observação da floresta.

O inventário florestal divide-se basicamente em duas etapas: o planejamento e a execução. No planejamento, os engenheiros florestais se concentram em especificar, por exemplo:

- Descrição da área onde se pretende realizar o inventário - localização, tamanho, tipo de terreno, acessibilidade, facilidade no transporte, tamanhos e formas das unidades amostrais, delineamento estatístico, precisão requerida para o inventário, bem como o tempo e custo para todas as fases do trabalho.
- Procedimentos para trabalhos de campo - apoio logístico e transporte, instruções sobre alocação de unidades de amostra e medições a serem realizadas

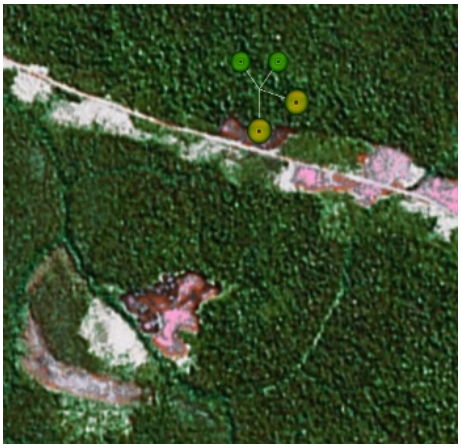
em campo, instrumentos, registros de dados e controle de qualidade.

Após o planejamento no qual são definidos os objetivos, os parâmetros e o tipo de amostragem, parte-se para a execução que compreende os trabalhos de campo.

Nos trabalhos de campo, as equipes devem ser convenientemente preparadas para a realização de tarefas como a localização das parcelas permanentes e a obtenção das variáveis de interesse como diâmetro, altura, peso, área basal, volume, classe de copa, crescimento e mortalidade das espécies florestais, entre outras. As espécies são numerosas e a identificação das mesmas requer um olhar atento do especialista para que não ocorram erros nesse processo. Por esses e outros fatores, métodos que minimizem o *custo temporal* do inventário florestal são bem vindos.

No contexto da Amazônia Brasileira o trabalho de campo pode tomar uma grande proporção logística, pois o inventário não ocorre só em parcelas permanentes contíguas. Às vezes, as parcelas podem estar isoladamente distribuídas em regiões distintas como mostrado na Figura 8.1(b).

(a) Área florestal delimitada por parcelas permanentes contíguas formando uma matriz 5x5.



(b) Parcela permanente não contígua ou isolada.

Figura 8.1: Parcelas Permanentes.

Isso implica em deslocar equipe, equipamentos e suprimentos. Esse deslocamento tem um elevado *custo financeiro* já que a Amazônia Brasileira é uma região carente de infraestrutura de transporte com rodovias e ferrovias escassas, malha aérea restrita e hidrovias abundantes. Para minimizar essa dificuldade, frequentemente se faz uso do *transporte multimodal*[1], combinando os meios de transporte.

[1]O Transporte Multimodal envolve mais de um meio de transporte para conduzir pessoas e

Em matéria de modais, os transportes podem ser divididos em rodoviários, ferroviários, aquaviários e aéreos [123]. Sendo o transporte fluvial naturalmente adequado à região amazônica, fatores de sazonalidade dos rios também deve ser observados, pois os mesmos regularmente inundam as florestas onde se pretende realizar o inventário ou tem o seu volume de água reduzido a ponto de se tornarem inavegáveis, logo a redução do *custo financeiro* é desejável.

No inventário florestal, as atividades logísticas envolvem tarefas operacionais de administração dos recursos pessoais, materiais e financeiros. A *logística integrada* concentra todas essas tarefas operacionais, incorporando novas funções particularmente adaptadas para região amazônica como o transporte multimodal para deslocamento da equipe que realizará o inventário, aquisição e armazenamento de suprimentos/equipamentos utilizados pela equipe para análise dos recursos florestais, o planejamento e execução dos inventários florestais, exploração, beneficiamento dos recursos madeireiros e finalmente o consumo final.

Quando a gestão de um processo produtivo envolve logística integrada, estima-se que um dos principais benefícios esteja presente no fluxo de informação gerado pelos produtos em movimento, permitindo a otimização do processo [123].

A Figura 8.2 ilustra um exemplo bem simples de transporte multimodal na região. Nesse contexto, o objetivo é sair de Manaus (ponto A) e chegar em Parintins (ponto B) para realizar um inventário florestal. O primeiro passo é descobrir quais são as opções modais para levar uma equipe até o destino do inventário. Nesse percurso é possível contar com a rodovia AM-10 (trecho Manaus-Itacoatiara), a hidrovia pelo Rio Amazonas (Manaus-Itacoatiara-Parintins) e por via aérea (trecho Manaus-Parintins).

A complexidade aumenta ao considerar que ao final do inventário a equipe precisa retornar à Manaus e para facilitar o entendimento dos vários trajetos, a Figura 8.3 ilustra o exemplo por meio de um grafo direcionado representando a rede de transporte, onde as arestas agrupam os modais e os nodos representam os municípios.

Nesse grafo é possível observar que existe uma opção **multimodal** (sentido ida Manaus-Parintins) de transporte que é Manaus-Itacoatiara via rodovia AM-10 e em seguida, Itacoatiara-Parintins por hidrovia, navegando em um trecho do rio Amazonas. As outras opções **unimodais** são: (1)Manaus-Parintins por via aérea; (2)Manaus-Parintins por hidrovia com possível escala em Itacoatiara. O retorno (sentido Parintins-Manaus) também pode ser multimodal e outras variáveis podem

produtos até o seu destino final. Podem ser utilizados veículos automotivos, barcos, aviões ou outro tipo de condução.

Figura 8.2: Hidrovias, rodovia e trecho aéreo entre Manaus-Parintins.

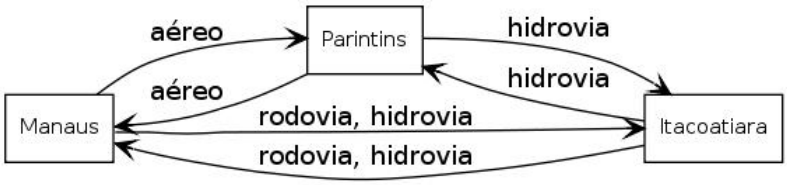


Figura 8.3: Possíveis modais de transporte entre Manaus e Parintins.

ser levadas em consideração como:

- O sentido do rio no caso do transporte fluvial - a favor da correnteza a viagem é mais rápida se comparada ao tempo gasto pelo mesmo trajeto contra a correnteza.
- A sazonalidade dos rios - em período de cheia os rios são facilmente navegáveis, mas em período de vazante, dependendo do trecho, a navegação pode se tornar inóspita ou inviável.
- Índice pluviométrico - chuvas intensas podem dificultar muito o inventário florestal ou até inviabilizá-lo.

Diante de todos esses fatores associados ao planejamento e execução do inventário, os engenheiros florestais se deparam com o problema de como *maximizar a quantidade de parcelas permanentes visitadas por inventário, minimizando o custo e o tempo de realizar tal atividade.*

O cálculo de um simples ciclo, levando em consideração a melhor rota de inventário florestal do ponto de vista prático, passando por todos os municípios que

concentram as parcelas permanentes sem repeti-los, semelhante ao problema do caixeiro viajante, classificado como NP-Difícil [124], não é algo trivial e exige métodos aprimorados de especificação, validação e processamento de cenários que representem esse domínio.

Este problema identificado no processo de inventário florestal ao mesmo tempo que se assemelha com o problema do caixeiro viajante na versão de otimização, também tem variabilidades no espaço do domínio que o torna uma instância diferenciada dentro dessa classe de problemas.

É uma atividade estratégica identificar rotas otimizadas de inventário florestal na Amazônia Brasileira para extrair informações da floresta quanto a vulnerabilidade, impactos sofridos e capacidade de regeneração. Devido a isso, modelos computacionais que ofereçam suporte a esse tipo de análise são bem-vindos, configurando assim um cenário interessante para se investigar como esses modelos podem ser representados por metamodelos, adaptados e reusados.

Com base nessa demanda real, surgiu a PPL (*Permanent Parcel Language*), uma linguagem de especificação de parcelas permanentes e do planejamento para a realização do inventário florestal com foco na transformação, configuração, componentização e reúso. Baseada em conceitos relevantes e *features* do domínio, a PPL segue o método proposto no Capítulo 5 e é apresentada como prova de conceito do tipo *protótipo estrutural*.

8.1.1 Projeto

Essa prova de conceito encontra-se no subdomínio de *Gestão do Inventário Florestal* (ver Figura 6.1) e visa a adequação do modelo proposto ao subdomínio e observar o impacto do mesmo sobre a variabilidade da modelagem arquitetural conforme requisitos apresentados na Tabela 8.1.

Uma especificação em PPL envolve as atividade conforme Figura 8.4.

Visão Conceitual

Essa visão tem como objetivo apresentar de forma integrada três dimensões: (i) a instância do problema no contexto do método proposto, (ii) como se dá a integração entre ED/EA e MDE no contexto do problema e (iii) onde efetivamente ocorre a variabilidade e a similaridade (passível de reúso por meio de uma família de software).

Tabela 8.1: Requisitos da PPL.

Prova de Conceito: PPL	
Arquitetura de Referência: Framework do Capítulo 5	
Requisitos funcionais:	Requisitos não-funcionais:
Especificar cenários de inventário florestal, considerando as variáveis do domínio	Variabilidade no espaço do problema
A linguagem de especificação deve ser formal e textual	Reusabilidade no espaço da solução por meio de configuração
Espera-se que essas especificações possam ser convertidas em instâncias de um problema de otimização compatível e previamente implementado	Consistência
Espera-se que as instâncias do problema possam ser processadas por métodos aproximados (heurísticos) de forma a se obter soluções válidas, mas não necessariamente ótimas	Usabilidade

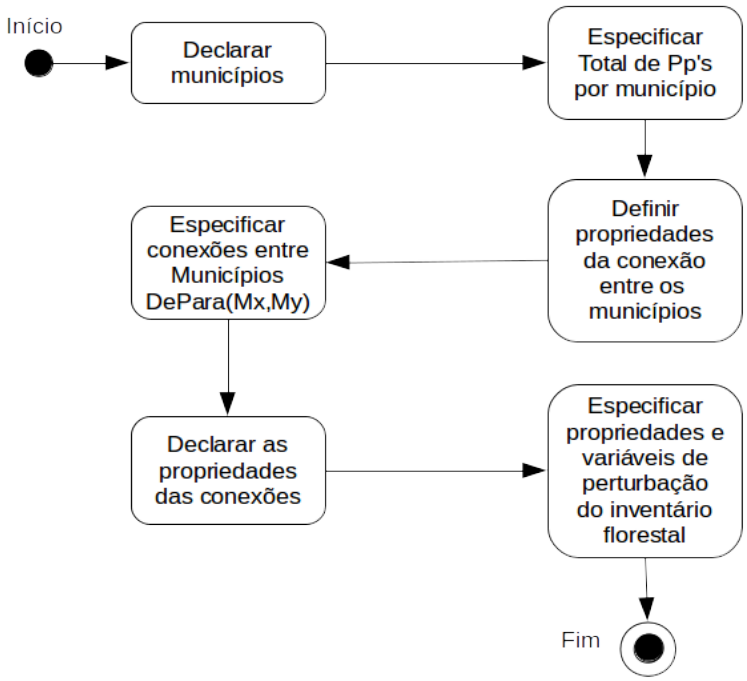


Figura 8.4: Uma especificação em PPL.

A Figura 8.5 apresenta a camada identificada como "zero", que representa o modelo específico de plataforma, mais próximo do nível de execução e restrito aos *frameworks* de otimização com o objetivo de reusá-los. Os *frameworks* concentram tanto a implementação de *problemas clássicos* na versão multi-objetivo como *métodos aproximados* de solução baseados em heurísticas e meta-heurísticas. O elemento denotacional entre o problema de otimização e a abordagem de solução é justamente o modelo de configuração, onde serão parametrizadas as variáveis de

influência sobre o problema com o objetivo de se extrair as melhores soluções, não necessariamente ótimas.

Em seguida, a camada "um" é a intermediária, responsável pela transformação de uma especificação em PPL como instância para uma implementação do TSP multi-objetivo e da integração com o modelo de configuração.

A camada de mais alto nível, identificada como "dois", representa o modelo independente de plataforma onde estão efetivamente representados os conceitos relativos a rota de inventário florestal, viabilizando aos especialistas, criar especificações que refletem o planejamento de expedições para coleta de dados em campo.

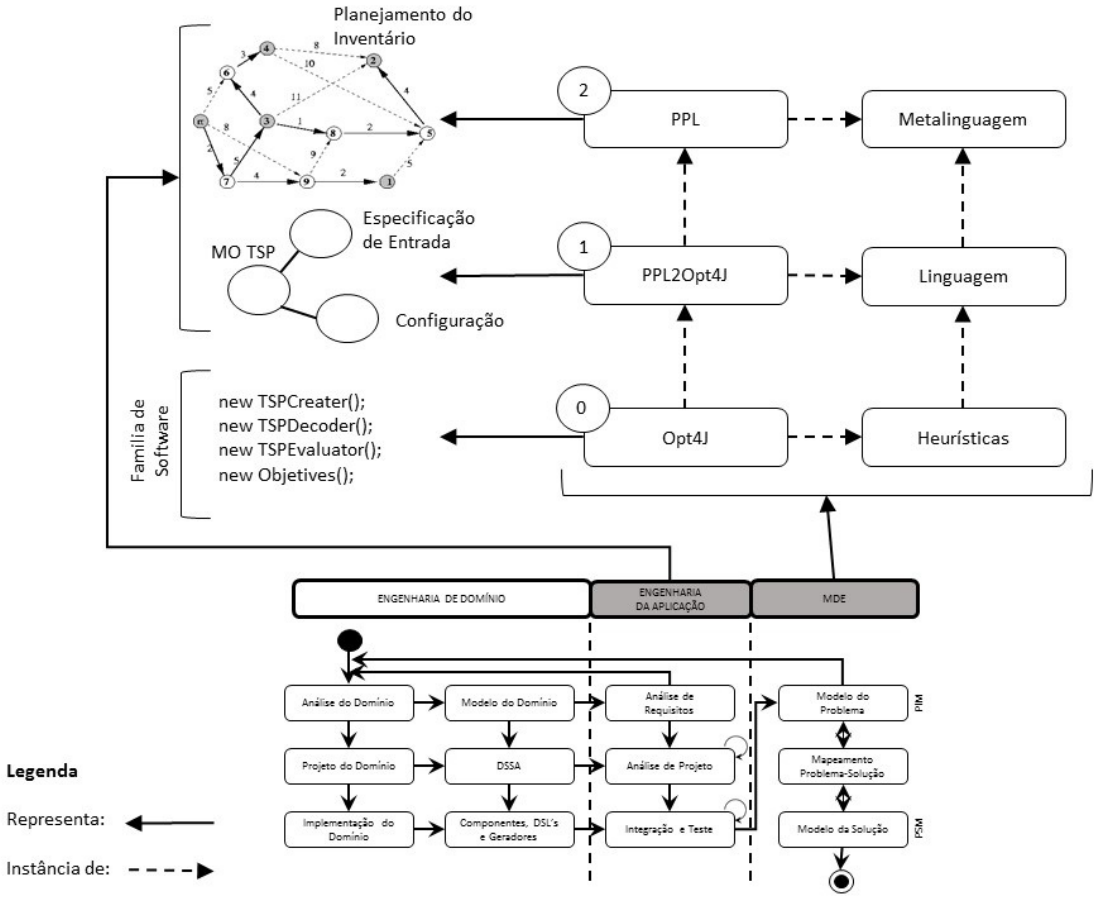


Figura 8.5: Visão Conceitual - Prova de Conceito PPL.

A Tabela 8.2 sintetiza o propósito da visão conceitual. Todas as etapas da engenharia de domínio e aplicação foram realizadas. Por ter uma representação direta a nível de domínio específico, a engenharia de aplicação entra em destaque, para indicar a equivalência entre os artefatos previstos e os gerados tanto no espaço do problema como no espaço da solução. O mesmo acontece com a MDE.

Tabela 8.2: Síntese da Visão Conceitual - PPL

Nome da Visão: Conceitual
1. Elementos, relacionamentos e propriedades
Tipos de elementos: modelos Tipos de relação: instância de, representa Tipos de propriedades: coesão, associação
2. Quem são os interessados que podem fazer uso da visão?
Testadores, mantenedores, arquitetos, analistas e engenheiros de linguagens
3. Questões respondidas por meio das informações contidas nessa visão?
- A correlação entre os modelos derivados a partir da arquitetura de referência podem ser facilmente visualizados? - A correlação entre o modelo de domínio específico gerado e o espaço do problema foi explicitamente representada? - A correlação entre o modelo de transição entre a solução e o problema foi explicitamente representado? - É possível visualizar a família de software?

Visão de Camadas

Essa visão tem como objetivo apresentar as camadas da arquitetura para o domínio específico. A Figura 8.6 ilustra as várias camadas e a relação entre elas.

A partir da observação da área física e do planejamento do inventário florestal, uma camada de georreferenciamento oferece suporte ao mapeamento dos pontos físicos. Sendo assim, a camada de georreferenciamento *usa* o ambiente físico para mapear dados de localização, que por sua vez, são persistidos em base de dados. A relação entre a camada de georreferenciamento e dados é bidirecional, pois ambas estão *autorizadas-a-usar* uma a outra em função da troca de mensagens de sucesso ou não tanto para persistência como para a coleta de dados.

A camada de negócio contempla os serviços de gerenciamento e faz a interface entre a camada do usuário e a de transmissão de dados. As três linearmente, estão *autorizadas-a-usar* a camada vizinha em função das trocas de mensagens de sucesso ou não para as requisições solicitadas.

A Tabela 8.3 sintetiza o propósito da visão de camadas. Alta coesão e baixo acoplamento são bem vindos uma vez que as camadas devem assumir responsabilidades bem definidas e a dependência entre as mesmas deve ser minimizada para evitar que a arquitetura seja inflexível a mudanças.

Visão de Uso

O propósito dessa visão é a extração rápida de subconjuntos e habilitar a construção incremental da PPL. Para isso, a Figura 8.7 apresenta um diagrama de

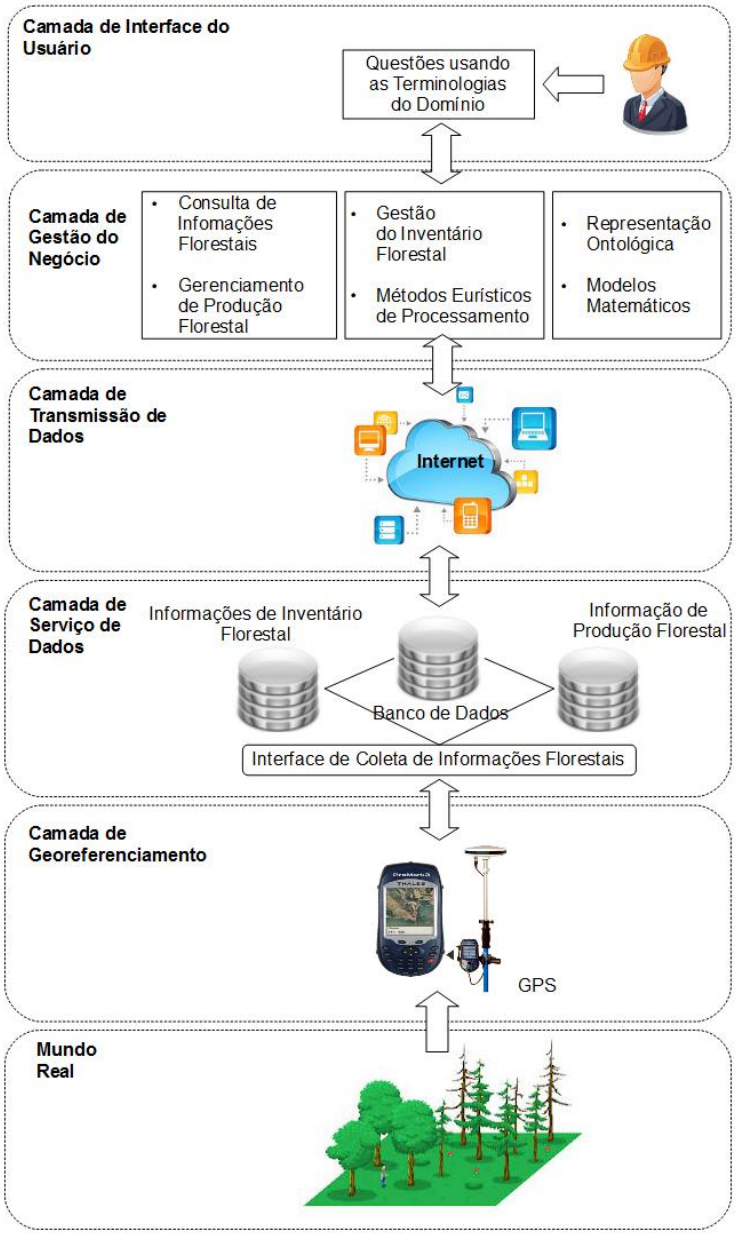


Figura 8.6: Visão de Camadas - Prova de Conceito PPL

contexto que ilustra a composição da CarbonQL, o que é esperado como modelo de entrada e saída e como ambos são estruturados.

Basicamente no contexto da PPL existem dois espaços de destaque: o da transformação e o da configuração. O *espaço da transformação* concentra também o espaço do problema onde o objetivo é receber uma especificação do planejamento do inventário florestal, transformar em uma instância válida para o TSP multiobjetivo

Tabela 8.3: Síntese da Visão de Camadas - PPL

Nome da Visão: Camadas
1. Elementos, relacionamentos e propriedades
Tipos de elementos: camadas Tipos de relação: autorizado-a-usar (unidirecional ou bidirecional) Tipos de propriedades: coesão, acoplamento
2. Quem são os interessados que podem fazer uso da visão?
Testadores, mantenedores, arquitetos, analistas, engenheiros de linguagens e usuários
3. Questões respondidas por meio das informações contidas nessa visão?
- Está claro o tipo de associação entre as camadas? - As camadas são coesas? - Qual o nível de acoplamento entre as camadas? Se alguma for removida, as demais funcionam? - É possível visualizar a correlação entre as camadas e as que foram estabelecidas no modelo de *feature* (serviço, funcional e ação)?

e agregar a configuração ao espaço da solução para que a entrada seja efetivamente processada de forma a gerar como saída rotas de inventário florestal otimizadas.

O núcleo do espaço da transformação é composto pelos metamodelos de origem e destino que são componentes.

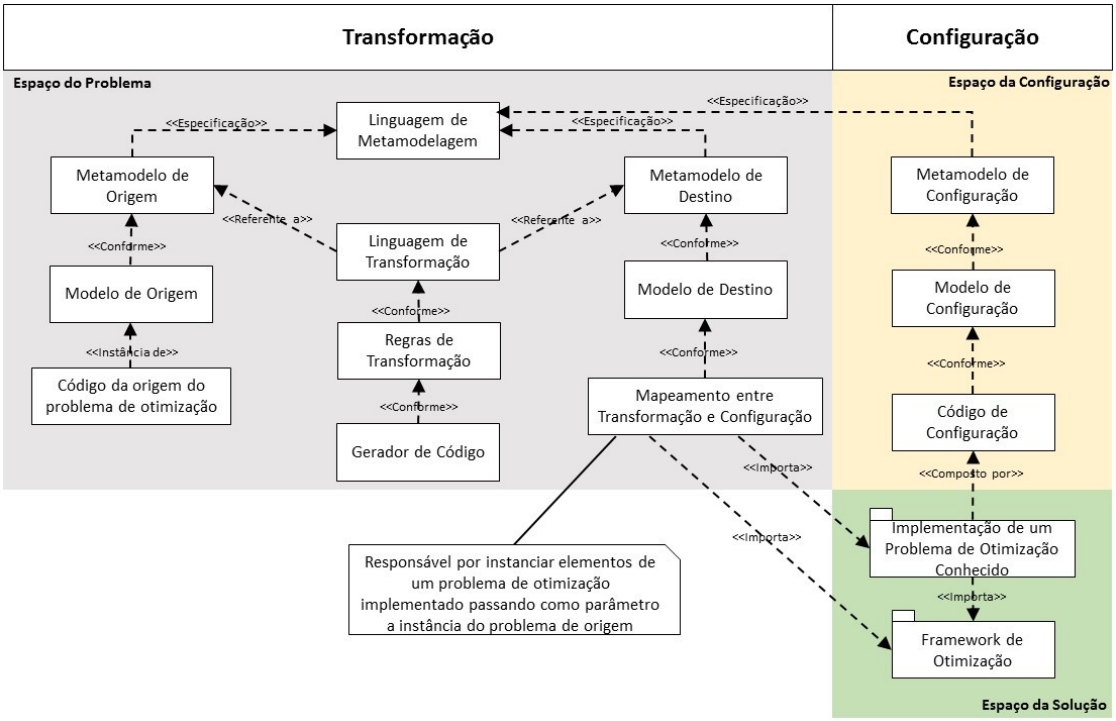


Figura 8.7: Visão de Uso - Prova de Conceito PPL

O *espaço da configuração* se conecta ao da solução e é responsável em mapear a relação entre os elementos de implementação e as *features*.

O *espaço da solução* contempla o reúso de heurísticas por meio de *frameworks* de otimização. Um benefício imediato dessa abordagem é que toda a informação se encontra na especificação do metamodelo, resultando em uma combinação de *features* para o contexto de um problema, oferecendo uma padronização e consistência de modelos, resultando em um link entre configuração e transformação entre problemas de otimização.

A Tabela 8.4 sintetiza o propósito da visão de uso que nos permite discutir possibilidades. O que acontece se a especificação do inventário florestal mudar e exigir um novo formalismo? O que acontece se o *framework* de otimização mudar? Com as devidas modificações, a arquitetura é capaz de suportar a modificação de requisitos ao longo do tempo? Ao menos para essas hipóteses, a arquitetura está preparada para mudanças.

Tabela 8.4: Síntese da Visão de Uso - PPL

Nome da Visão: Uso
1. Elementos, relacionamentos e propriedades
Tipos de elementos: modelos Tipos de relação: especificado-em, conforme, instância-de e entrada-para Tipos de propriedades: coesão, acoplamento
2. Quem são os interessados que podem fazer uso da visão?
Testadores, mantenedores, arquitetos, analistas, engenheiros de linguagens
3. Questões respondidas por meio das informações contidas nessa visão?
- Está claro o tipo de associação entre os modelos? - Os modelos são coesos? - O nível de acoplamento entre os modelos é satisfatório? - A visão permite identificar os elementos evolutivos?

8.1.2 Estratégia de Implementação

A Tabela 8.5 apresenta um exemplo de configuração do TSP. Alinhado a definição formal do TSP, dois requisitos funcionais são apresentados, bem como a regra de configuração e as *features*. Observe que é possível ter vários conjuntos de *features* para uma configuração. Da mesma forma, é possível ter muitas configurações um conjunto de requisitos. A Tabela 8.7, apresenta um conjunto de regras de transformação com base nos metamodelos de forma a estabelecer uma correspondência entre os mesmos.

Essa configuração se aplica a problemas de transformação. Como exemplo, a Tabela 8.6 apresenta a transformação do problema da mochila para o TSP, passando pelo modelo de origem, regra de transformação e modelo de destino.

Tabela 8.5: Configuração do TSP.

Especificação de Requisitos	Regras de Configuração	*Features* de Software
Criar/definir um conjunto de cidades e distâncias conhecidas entre cada par de cidades	Grafo completo G formado por um conjunto de vértices $V = \{city1, city2, ..., m\}$, um conjunto de arestas E conectando os vértices em V por uma função de custo c_{ij} para a aresta (i, j), para todo i, j em V representando a distância.	Funcionalidade: permitir criar/definir $G(V, E)$ por uma matriz de distância $d(c_i, c_j)$; Performance: tempo polinomial
Encontrar uma rota para visitar cada cidade uma única vez e minimizar a distância total de viagem	Uma permutação $\pi(1), ..., \pi(n)$ onde a soma $\sum_{1 \leq i \leq n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)})$ é a menor possível. Essa soma é o tamanho do percurso, iniciando na cidade $c_{\pi(1)}$, visitando todas as cidades e retornando para $c_{\pi(1)}$	Funcionalidade: encontrar o ciclo Hamiltoniano de custo mínimo; Performance: o método exato é executado em tempo exponencial.

Tabela 8.6: Transformação do problema da mochila para o TSP

Modelo de Origem	Regras de Transformação	Modelo de Destino.
Dada uma mochila com capacidade para $S = 15kg$ e um conjunto de itens I com o seu respectivo valor *commercial value* e/ou peso *weight*. $KP : N =$ conjunto de itens, $\Phi = \{S \subseteq N :$ *total wight of* $S \leq$ *knapsack capacity*$\}$	Grafo completo $G(V, E)$, onde $V = \{candidateItem1, candidateItem2, ..., i\}$, um conjunto de arestas E conectando os vértices em V por uma função de peso w_{ij} para a aresta (i, j), para too i, j em V. TSP.setInstance(KS.getInstance())	$TSP : N = E(G)$, $\Phi = \{S \subseteq N :$ *S induces a hamiltonian cycle of* $G\}$

A Figura 8.8 sintetiza a estratégia de implementação para a linguagem PPL onde cada elemento destacado por linhas tracejadas demandaram implementação e os demais foram meramente reusados.

Para que uma dada especificação em PPL seja efetivamente processada e executada, é necessário definir um gerador de código que traduza a especificação em PPL para uma instância do TSP multiobjetivo. Internamente, o TSP multiobjetivo implementa algumas interfaces disponibilizadas pelo opt4J.core como as classes *Creator, Decoder* e *Evaluator* para ter acesso aos métodos heurísticos.

Sendo assim, o gerador de código **PPL2Opt4J** necessita de regras de transformação que são definidas com base na linguagem **PPLWalker**. O resultado desse processo é um código java gerado automaticamente, e representa o elemento denotacional entre a implementação do TSP e o Opt4J.

Visão de Código

A Figura 8.9 apresenta o fluxo de transformação da PPL. Uma especificação (test.ppl) deve conter basicamente n localidades a serem visitadas para a realização do inventário florestal, bem como o total de parcelas por localidade, as conexões entre essas localidades, representadas pelos modais de transporte e o respectivo

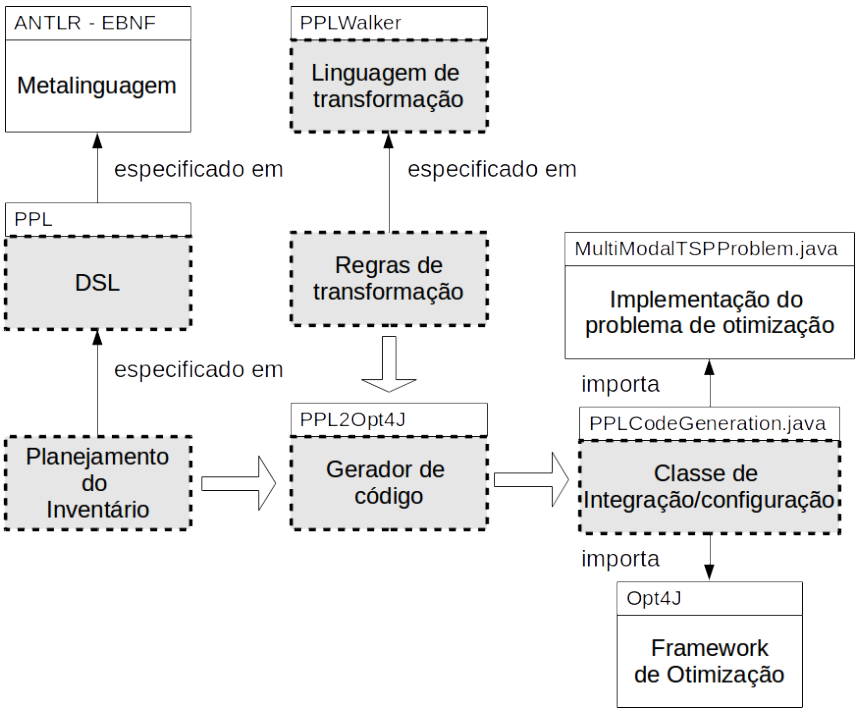


Figura 8.8: Escolhas tecnológicas para a implementação da PPL.

custo / tempo. A partir dessa especificação, ao caminhar na AST, o dado sobre o cenário de inventário florestal é coletado para uma estrutura de dados intermediária. A geração de código apenas recebe e passa objetos para a implementação do TSP, com o respectivo modelo de configuração que será processado pelo *framework* de otimização.

O processo de transformação segue os seguintes passos: (i) o código em PPL é transformado via sintaxe direta em uma árvore de sintaxe abstrata (AST-PPL); (ii) a AST-PPL é transformada em uma AST-Java; (iii) um código java é criado a partir da AST-Java.

A partir da gramática **PPL.g** especificada em EBNF, o ANTLR gera automaticamente as regras léxicas - **PPLLexer**, os símbolos terminais - **PPL.tokens** e os blocos de regras que compõem a análise sintática - **PPLParser**. A segunda gramática implementada foi a **PPLWalker** como a linguagem e transformação, cujas regras são sintetizadas pela Tabela 8.7.

Finalmente, a geração automática de código em Java segue o template definido no **PPL2Opt4J.stg**. A Figura 8.10 apresenta um fragmento do código.

A Tabela 8.8 sintetiza a visão de código. Os elementos apresentam estrutura coesa e estão de acordo com as especificações da visão conceitual, de uso e de camadas.

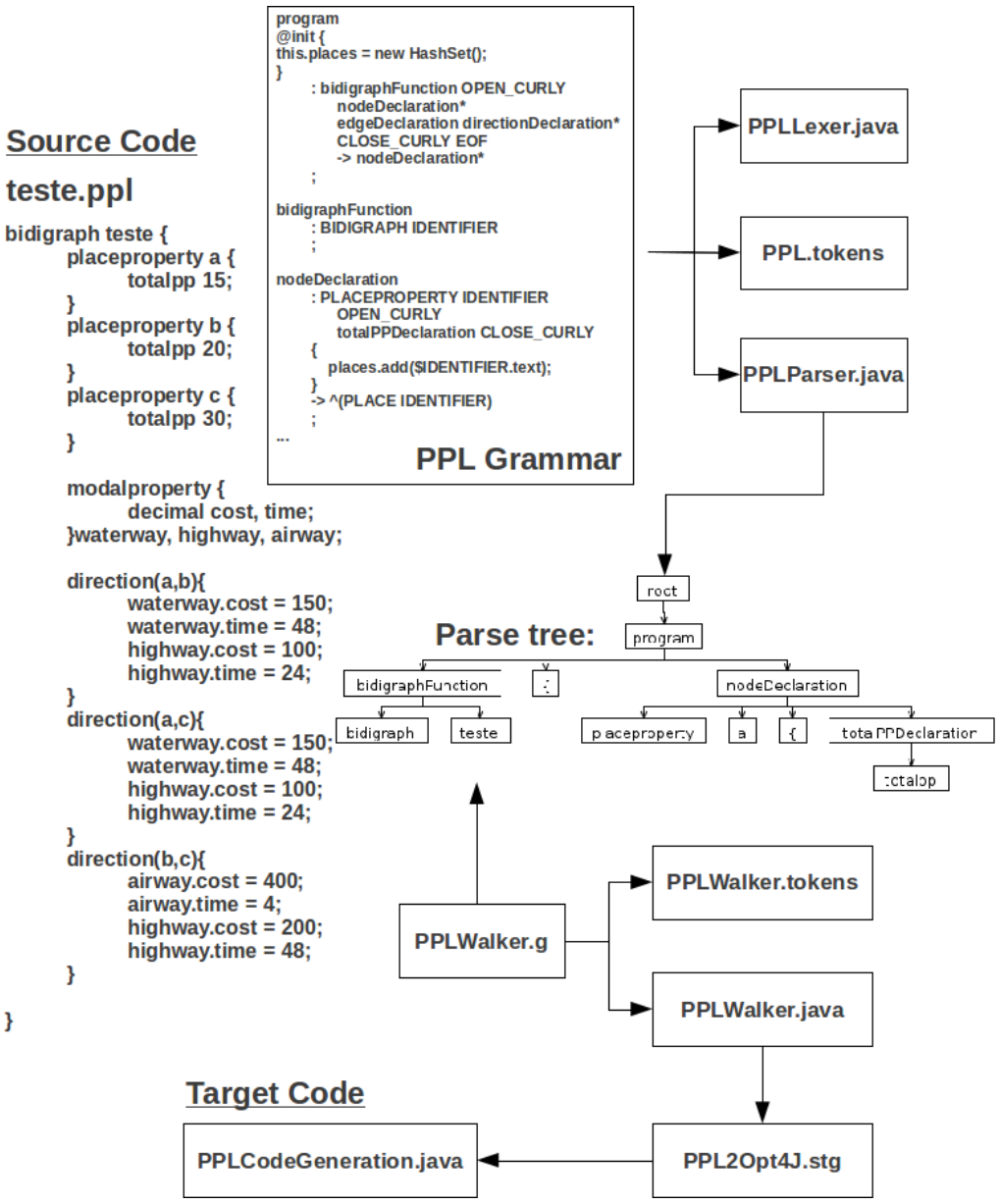


Figura 8.9: Projeto do Compilador PPL.

```
1 MultiModalTSPProblem tspProblem = new MultiModalTSPProblem();
2 tspProblem.setInstance(ppl.getInstance());
3 MultiModalTSPCreator tspCreator = new MultiModalTSPCreator(tspProblem);
4 MultiModalTSPDecoder tspDecoder = new MultiModalTSPDecoder();
5 MultiModalTSPEvaluator tspEvaluator = new MultiModalTSPEvaluator();
6 obj = new Objectives();
7 obj = tspEvaluator.evaluate((tspDecoder.decode(tspCreator.create())));
```

Figura 8.10: Núcleo do código gerado pelo *PPLCodeGeneration*.

Tabela 8.7: Regras de transformação da Linguagem PPL2Opt4J.

Regras PPL	Regras de Transformação	Efeito
nodedeclaration <place >	cities.add(<place >)	$\{nodeproperty \mid nodeproperty : IDENTIFIER\}$. Representa a declaração de nodos. No contexto da PPL, a semântica da declaração de um nodo é uma localidade, traduzida como cidade no contexto do TSP.
edgedeclaration <attribute, variable >	multimodal.add(<attribute >, <variable >)	$\{edgeproperty \mid edgeproperty : (\ attribute\ )register\}$. Representa a declaração de arestas. No contexto da PPL, trata-se da declaração de uma estrutura de dados composta por atributos e variáveis, traduzidas com uma instância multimodal, por exemplo, waterway.time, waterway.cost, onde *multimodal* é uma HashSet.
directiondeclaration <city, city, attribute, variable, value>	weight.add(<city>, <city>, <type>, <variable >, <value>)	$\{directionproperty \mid directionproperty : direction(place, place)attribute.register = VALUE;\}$. Representa a declaração de direção. Esse bloco de código representa semanticamente o peso da aresta.
IDENTIFIER	Traduzido como um *token* terminal	$\{IDENTIFIER : (a..z\|A..Z\|\_)(a..z\|A..Z\|\_\|0..9)*\}$.
VALUE	Traduzido como um *token* numérico	$\{VALUE : (INTEGER\|DECIMAL)\}$.

Tabela 8.8: Síntese da Visão de Código - PPL

Nome da Visão: Código
1. Elementos, relacionamentos e propriedades
Tipos de elementos: Classes e gramáticas Tipos de relação: associação Tipos de propriedades: coesão
2. Quem são os interessados que podem fazer uso da visão?
Testadores, mantenedores, arquitetos, analistas, engenheiros de linguagens
3. Questões respondidas por meio das informações contidas nessa visão?
- Qual a relação entre as classes? - Todos os elementos (classes e gramáticas) estão de acordo com a arquitetura? - Os elementos apresentam estrutura coesa? - Os elementos tecnológicos de suporte a implementação foram definidos?

8.1.3 Relação entre as Visões

A Figura 8.11 apresenta a relação entre as visões da PPL. Tudo começa pela *visão conceitual* que oferece um panorama de como o problema pode ser solucionado pelo método proposto. Essa visão contribui com diagramas que instanciam os vários domínios, elicita a variabilidade e similaridade e se há uma família de software para o contexto.

A partir dessa modelagem entra a *visão de camadas* que se apoia na especificação definida pela conceitual e complementa a percepção do domínio do ponto de vista funcional, elicitando serviços complementares que podem se conectar via relação de *uso* direcional ou bidirecional. A *visão de uso*, se apoia na visão conceitual para direcionar o desenvolvimento.

A *visão de código* foi a última a ser definida e integra todas as outras de forma di-

reta. A partir dessas visões é possível compreender como a arquitetura efetivamente funciona, implementar ou evoluir a implementação da linguagem.

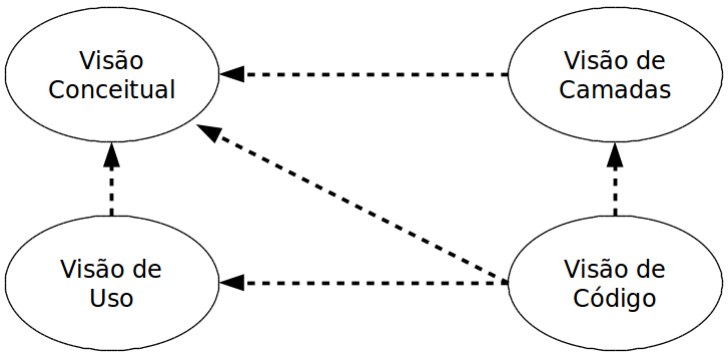


Figura 8.11: Relação entre as visões adotadas na prova de conceito PPL.

8.1.4 Aplicação da PPL

A natureza do problema discutido na Seção 8.1, estimulou a implementação inicial de um exemplo simples, considerando apenas algumas variáveis mapeadas a partir do domínio para avaliar a transformação, configuração e os métodos aproximados existentes no Opt4J. Algumas simplificações iniciais foram feitas como:

- Cada município, gerado randomicamente, se conecta com todos os outros, formando um grafo completo.
- Apenas um modal de transporte foi considerado.
- Cada aresta do grafo tem apenas um custo e tempo associado ao respectivo modal. Ambos são gerados randomicamente.

O critério de decisão pela melhor rota foi definido com base em duas funções-objetivo: uma para o custo e outra para o tempo; ambas de minimização.

A Figura 8.12 mostra os parâmetros considerados para uma simulação, onde foram definidos 50 municípios para compor a área de cobertura de um inventário florestal fictício, custo e tempo randômicos, algoritmo evolucionário como otimizador, as iterações como critério de diversidade para se gerar melhores resultados, o fator alpha que representa o tamanho da população, o fator *mu* que representa o número de pais por iteração, o fator *lambda* que define o total de descendentes por geração, bem como a taxa de cruzamento.

Camada	Elemento Geral	Valor		
Transformação	Linguagem de Origem	PPL		
	Linguagem de Transformação	PPLWalker		
	Gerador de Código	PPL2OptProblem		
	Cödigo de Destino	PPLCodeGeneration		
Configuração	Framework de Otimização	Opt4J		
	Problema de Otimização Conhecido	MM/MO TSP		
	Total de Lugares/Cidades	50		
	Custo/Tempo do Modal	Aleatório		
	Otimizador	Algoritmo Evolucionário		
	Elementos do Experimento	**Experimento 1**	**Experimento 2**	**Experimento 3**
	Código de Configuração	Experimnet1.conf.xml	Experiment2.conf.xml	Experiment3.conf.xml
	Total de Interações	100	100	100
	Fator Alfa	50	50	50
	Fator Um	25	30	35
	Fator Lambda	25	30	35
	Taxa Crossover	0,95	0,8	0,7
	Resultados			
	Total de Melhores Rotas	5	4	5

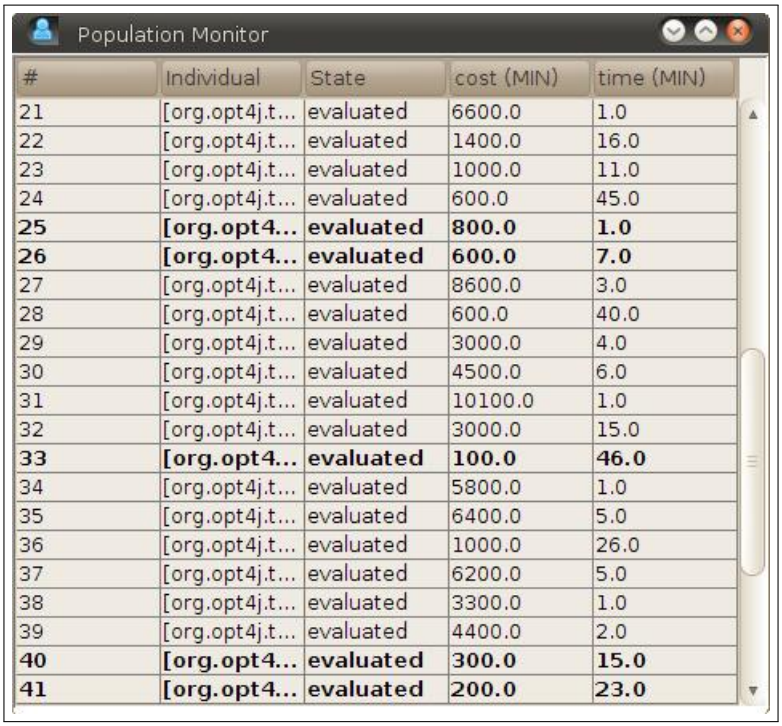
Figura 8.12: Parâmetros e experimentos realizados

Os Algoritmos Evolutivos mantêm uma população de possíveis soluções que competem entre si seguindo a metáfora da Teoria Evolutiva. A Figura 8.13 mostra o monitor do Opt4J para a população gerada na última iteração já que esses valores mudam em tempo de execução. As linhas que estão em negrito recebem esse destaque porque foram classificadas como melhores dentre a população de rotas geradas na respectiva iteração.

As melhores rotas encontradas são separadamente destacadas pelo gerenciador de melhores indivíduos. A cada iteração o gerenciador inclui ou exclui rotas com base nas funções objetivo de forma que ao final sobrem apenas as melhores como mostrado na Figura 8.14.

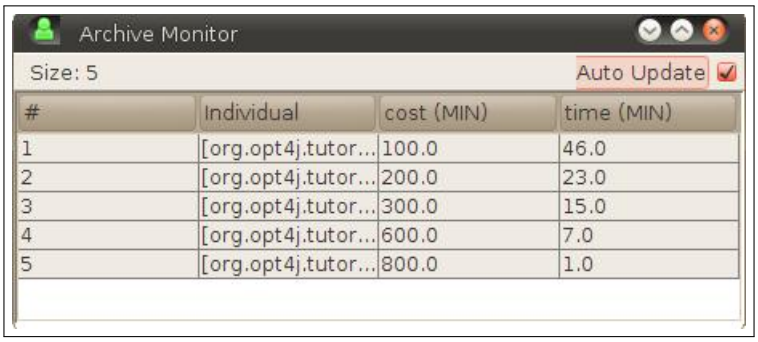
Observe que cinco possíveis rotas foram encontradas. Suponha que além das funções-objetivo o usuário tenha a opção de ponderar o resultado em 100% para o custo e 0% para o tempo, ou seja, a grande questão é orçamentária e não importa o tempo necessário para realizar o inventário. Nesse caso, a melhor opção seria o primeiro item da Figura 8.14, com custo de R$100,00 e tempo de 46hrs.

Um problema multiobjetivo, como definido por Arroyo e Armentano [125], consiste em determinar um vetor de variáveis de decisão que otimize um vetor de funções objetivo e satisfaça o conjunto de restrições envolvidas. Uma solução x é chamada eficiente, não-dominada ou Pareto ótima se o valor de alguma função objetivo $f_i(x)$



#	Individual	State	cost (MIN)	time (MIN)
21	[org.opt4j.t...	evaluated	6600.0	1.0
22	[org.opt4j.t...	evaluated	1400.0	16.0
23	[org.opt4j.t...	evaluated	1000.0	11.0
24	[org.opt4j.t...	evaluated	600.0	45.0
25	**[org.opt4...**	**evaluated**	**800.0**	**1.0**
26	**[org.opt4...**	**evaluated**	**600.0**	**7.0**
27	[org.opt4j.t...	evaluated	8600.0	3.0
28	[org.opt4j.t...	evaluated	600.0	40.0
29	[org.opt4j.t...	evaluated	3000.0	4.0
30	[org.opt4j.t...	evaluated	4500.0	6.0
31	[org.opt4j.t...	evaluated	10100.0	1.0
32	[org.opt4j.t...	evaluated	3000.0	15.0
33	**[org.opt4...**	**evaluated**	**100.0**	**46.0**
34	[org.opt4j.t...	evaluated	5800.0	1.0
35	[org.opt4j.t...	evaluated	6400.0	5.0
36	[org.opt4j.t...	evaluated	1000.0	26.0
37	[org.opt4j.t...	evaluated	6200.0	5.0
38	[org.opt4j.t...	evaluated	3300.0	1.0
39	[org.opt4j.t...	evaluated	4400.0	2.0
40	**[org.opt4...**	**evaluated**	**300.0**	**15.0**
41	**[org.opt4...**	**evaluated**	**200.0**	**23.0**

Figura 8.13: Opt4J - População para a rota de inventário florestal



#	Individual	cost (MIN)	time (MIN)
1	[org.opt4j.tutor...	100.0	46.0
2	[org.opt4j.tutor...	200.0	23.0
3	[org.opt4j.tutor...	300.0	15.0
4	[org.opt4j.tutor...	600.0	7.0
5	[org.opt4j.tutor...	800.0	1.0

Figura 8.14: Opt4J - Melhores rotas encontradas

não pode ser melhorada sem piorar ao menos uma das outras funções objetivo. Por exemplo, o problema biobjetivo desse exemplo com objetivos $f_{custo}(x)$ e $f_{tempo}(x)$ pode ser definido da seguinte forma:

$$Min\ Z\ = (f_{custo}(x),\ f_{tempo}(x)) \tag{8.1}$$

$$x\ \in\ X^* \tag{8.2}$$

O vetor solução é representado por x, Z corresponde à imagem de x ou espaço objetivo e X^* corresponde ao conjunto de soluções viáveis do problema. Se $x_1,\ x_2 \mid X^*$

são dois vetores solução do problema acima, considerando o modo como estão selecionados, há três possibilidades:

i) $f_{tempo}(x_1) < f_{tempo}(x_2)$ *e* $f_{custo}(x_1) < f_{custo}(x_2)$
(x_1 *domina* x_2)
ii) $f_{tempo}(x_1) > f_{tempo}(x_2)$ *e* $f_{custo}(x_1) > f_{custo}(x_2)$
(x_2 *domina* x_1)
iii) nenhuma das alternativas acima

Na condição *(iii)* x_1 e x_2 são indiferentes entre si. Por exemplo, na Figura 8.15, para os pontos $x2$ e $x3$, a relação é: $f_{tempo}(x_2) < f_{tempo}(x_3)$ e $f_{custo}(x_2) < f_{custo}(x_3)$, assim a solução $x2$ domina a solução $x3$, conforme condição *(i)*. Considerando os pontos $x1$ e $x2$, tem-se $f_{tempo}(x_1) > f_{tempo}(x_2)$ e $f_{custo}(x_1) < f_{custo}(x_2)$, então, as soluções x1 e x2 são indiferentes entre si (condição *(iii)*). A solução x^* é chamada eficiente ou Pareto ótima se não há outra solução $x \mid X_*$ que domine x^*. O conjunto de soluções eficientes determina a curva de Pareto. Para o experimento realizado, essa curva pode ser observada na Figura 8.15.

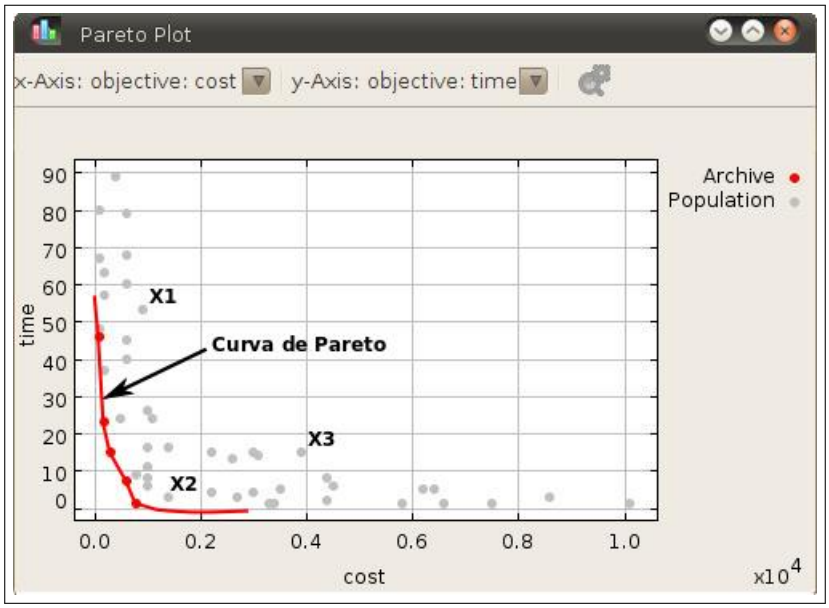


Figura 8.15: Diagrama de Pareto com destaque para a população e as melhores rotas encontradas

8.2 Considerações Finais

A prova de conceito realizada serviu para sintetizar soluções para um domínio real, prático, desafiador, com vários requisitos interessantes de serem observados do ponto de vista arquitetural e evolutivo, incluindo a análise da similaridade e variabilidade. Destaca-se como uma evidência do método proposto a outros pesquisadores que tenham interesse em aplicá-lo ou modificá-lo para obter vantagens adicionais, avaliar riscos de adoção, etc.

No contexto dessa prova de conceito, foi possível observar que o método proposto contribuiu para a modelagem arquitetural, permitiu a elaboração de modelos capazes de facilitar a compreensão das decisões arquiteturais e evitou retrabalho. Trouxe velocidade à implementação da família PPL, pois foi possível prever os riscos e mitigá-los.

As ameaças que podem afetar o resultado da provas seguem três categorias: confiabilidade, validade e sensibilidade. A coleta de dados sobre as visões, as características, as particularidades e conexões entre as mesmas ajuda a mitigar riscos relacionados a confiabilidade das evidências. A definição dos critérios de sucesso para a prova ajuda a mitigar riscos relacionados a validade, eliminando a possibilidade de incluir uma prova que não atenda os critérios. O modelo teórico e a definição das variáveis latentes e observáveis ajuda a mitigar riscos relacionados à sensibilidade do estudo.

Capítulo 9

Avaliação

O objetivo desse capítulo é apresentar a avaliação das provas de conceito, abordada na Seção 9.1. Essa avaliação inclui a análise dos critérios de sucesso, sobre os resultados obtidos nas provas de conceito. Para finalizar, algumas considerações finais são apresentadas na Seção 9.2.

9.1 Avaliação do Método por meio das Provas de Conceito

As provas de conceito foram analisadas com base na Tabela 6.2 que descreve os critérios de sucesso que podem ser atendidos em sua totalidade, parcialmente ou podem não ser atendidos.

A partir dessa classificação, a Tabela 9.1 sintetiza o resultado da avaliação para cada prova de conceito, onde S = sim, critério de sucesso atendidos totalmente; P = parcialmente e N = o critério de sucesso não foi atendido.

O CS_01 examina se a integração entre as engenharias (SPL+MDE) atende os requisitos funcionais e não-funcionais. No caso da CarbonQL, todos os seus requisitos funcionais foram atendidos. Já os não-funcionais, a usabilidade foi o único que não foi validado. No entanto, apesar de ser desejável, esse requisito não tem tanto impacto do ponto de vista da validação do método proposto e da arquitetura. Mesmo assim, a elicitação desse requisito nos permite conhecê-lo e planejar uma investigação mais aprofundada sobre a usabilidade em trabalhos futuro. O mesmo pode ser observado no caso da PPL. A usabilidade não foi validada e da mesma forma, não tem tanto impacto do ponto de vista da validação do método proposto e da arquitetura. Isso faz com que o julgamento para esse critério seja de parcialidade

Tabela 9.1: Avaliação dos Critérios de Sucesso

		CarbonQL			PPL		
Dimensão 1 - Integração entre as Engenharias							
Índice	Categoria	S	P	N	S	P	N
CS_01	Capacidade		x			x	
CS_02	Consistência	x			x		
Dimensão 2 - Framework Arquitetural							
CS_03	Efetividade	x			x		
CS_04	Produtividade		x			x	
Dimensão 3 - Variabilidade da Modelagem Arquitetural							
CS_05	Eficiência	x			x		
CS_06	Eficácia	x			x		
Dimensão 4 - Qualidade do Projeto Final							
CS_07	Clareza	x			x		
CS_08	Flexibilidade	x			x		

para as duas provas de conceito. Mas, suponha que a usabilidade estivesse dentro do escopo do projeto, como a integração entre as engenharias poderia atender a esse requisito? A partir da análise do domínio (ED), podemos extrair critérios de usabilidade e via refluxo de *customização* sobre integração e teste (EA), a usabilidade pode ser analisada e o resultado pode ser incorporado no modelo do problema - PIM (acompanhar pela Figura 5.3).

O CS_02 examina se todas as etapas previstas pela integração entre as engenharias (SPL+MDE) foram executadas. Tanto na CarbonQL como na PPL, todas as etapas previstas foram executadas. Isso faz com que o julgamento para esse critério seja de atendimento total, capacitando a *dimensão-1* como factível, no contexto das provas de conceito realizadas.

O CS_03 examina se a documentação sugerida pelo *framework* atende a necessidade de identificação da variabilidade arquitetural. Em ambas as provas, esse critério foi atendido, principalmente pelo suporte das visões, com destaque para a de camadas que foca na alta coesão e baixo acoplamento, de forma que se os requisitos mudarem ao longo do tempo ou se uma camada deixar ou passar a existir, as demais não deixem de funcionar em função da variabilidade arquitetural.

O CS_04 examina se os modelos gerados são usados de forma automática na geração de outros modelos. Tanto na CarbonQL como na PPL esse critério foi atendido parcialmente. Falta suporte para que os modelos além de documentação, seja efetivamente usados como artefato de implementação e geração automática de código em sentido bidirecional, ou seja, a partir do código, também obter modelos. Com base nos resultados para o CS_03 e CS_04, a *dimensão-2* é classificada como factível.

O CS_05 examina se a identificação da variabilidade exógena da modelagem arquitetural, entre o *framework* e a arquitetura, orientou o reúso. Em ambas as provas, esse critério foi atendido. O *framework* sugere um conjunto de visões para detalhar as decisões arquiteturais de sistemas com demanda por transformação e/ou configuração. Seguir o *framework* já é um reúso a nível de projeto e documentação. A variabilidade exógena pode ser comprovada a partir da construção de duas arquiteturas para domínios diferentes (consultas e planejamento do inventário) via *framework* proposto.

O CS_06 examina se a identificação da variabilidade endógena da modelagem arquitetural contribuiu para a construção incremental e demonstração da portabilidade das DSLs. Em ambas as provas esse critério foi atendido. As várias visões e a conexão entre elas, viabilizaram a construção incremental e o exercício de abstração da portabilidade das DSLs considerado outros paradigmas, como foi o caso da CarbonQL, caso o mapeamento passe de ORM para orientado a objeto e no caso da PPL, caso o método heurístico mude e tenha outros parâmetros. Com base nos resultados para o CS_05 e CS_06, a *dimensão-3* é classificada como factível.

O CS_07 examina se houve melhoria na comunicação do projeto. Esse critério foi atendido em ambas as provas de conceito e foi validado por meio de reuniões com especialistas e discussão sobre as decisões de projeto.

O CS_08 examina se houve incremento na identificação/flexibilidade à mudanças arquiteturais. Esse critério foi atendido em ambas as provas de conceito. Apesar das visões apresentarem um resultado consolidado, ao longo do projeto, cada modelo foi evoluindo gradativamente com base no refluxo de customização na etapa de análise de projeto, integração e teste. Com base nos resultados para o CS_07 e CS_08, a *dimensão-4* é classificada como factível.

9.2 Considerações Finais

As provas de conceito realizadas serviram para sintetizar soluções para um domínio real, prático, desafiador, com vários requisitos interessantes de serem observados do ponto de vista arquitetural e evolutivo, incluindo a análise da similaridade e variabilidade. As provas servem como evidência do método proposto a outros pesquisadores que tenham interesse em aplicá-lo ou modificá-lo para obter vantagens adicionais, avaliar riscos de adoção, etc.

Além do mais, as provas permitiram a compreensão do que era para ser feito

e o que era esperado, facilitando a visualização das dimensões do planejamento do projeto e evitando retrabalho. Trouxe velocidade à implementação do projeto, pois foi possível prever os riscos e mitigá-los.

Para ambas as provas, todas as cinco etapas sugeridas pelo método foram seguidas. Na Etapa-1 foi feito o diagnóstico da situação-problema e mapeadas as Famílias CarbonQL e PPL. Durante essa etapa foi feito o levantamento dos domínios e extraída a variabilidade e similaridade.

Para cada família foi modelada na Etapa-2 a respectiva arquitetura de integração entra as engenharias. No caso da Família CarbonQL,

As ameaças que podem afetar o resultado das provas seguem três categorias: confiabilidade, validade e sensibilidade. A coleta de dados sobre as visões de especificação das provas, as características, as particularidades e conexões entre as mesmas ajuda a mitigar riscos relacionados a confiabilidade das evidências. A definição dos critérios de sucesso para as provas ajuda a mitigar riscos relacionados a validade, eliminando a possibilidade de incluir uma prova que não atenda os critérios. O modelo teórico e a definição das variáveis latentes e observáveis ajuda a mitigar riscos relacionados à sensibilidade do estudo.

Capítulo 10

Trabalhos Relacionados

A Seção 10.1 apresenta uma análise dos trabalhos relacionados e responde as seguintes perguntas: (iii) Quais métodos se propõe a dar suporte à modelagem arquitetural de forma genérica? (iv) MD-SLP precisa de um método específico para modelagem arquitetural? (v) Quais métodos são voltados para MD-SPL? Por fim, na Seção 10.2 são discutidas as considerações finais.

10.1 Estudo Comparativo

A modelagem arquitetural de SPL é uma área que vem produzindo cada vez mais pesquisas centradas em três principais sub-áreas: (i) a integração, variação ou derivação endógena de modelos (dentro de uma mesma arquitetura), (ii) a integração, variação ou derivação exógena de modelos (entre arquiteturas) e (iii) a gestão de decisões de modelagem arquitetural passando pela rastreabilidade e recuperação das escolhas arquiteturais. Para as sub-áreas (i) e (ii), apesar de terem trabalhos em *architecture description language* (ADL) ou técnicas orientadas a aspecto, todos os trabalhos relacionados discutidos a seguir usam derivações a partir da transformação de modelos, ou seja, MDE.

10.1.1 Métodos de Modelagem Arquitetural

Cabello *et al.* [126] apresentam um método centrado em MD-SPL para gerar o esqueleto de uma arquitetura. Basicamente o método usa uma arquitetura de referência (na forma de metamodelo) e o modelo de *feature* como entrada, processa via o que os autores chamam de plano de produção, que na verdade é uma transformação de modelos. A saída desse processo de transformação é o esqueleto da arquitetura de

família de software na forma da visão componente-conector. O método foi aplicado na prática em arquitetura de sistemas especialistas.

Duran-Limon *et al.* [127] comentam que em SPL a maioria das pesquisas focam em instanciar as variantes selecionadas no produto final ao invés de trabalhar na derivação em nível arquitetural. Foi pensando nesse cenário que os autores criaram um *framework* de derivação de arquitetura de produto baseado em ontologia (OntoAD). A solução usa linguagem independente de modelo para especificar a arquitetura de linha de produto e MDE para atividades de derivação da arquitetura. A ontologia foi utilizada para geração automática de regras de transformação de modelo-para-modelo baseada na seleção de *features*. O método foi aplicado na prática em arquitetura de sistemas de voz sobre IP.

Insfran *et al.* [128] apresentam um método que integra atributos de qualidade em estágios iniciais da modelagem arquitetural, posteriormente chamado de QuaDAI [129]. O método é baseado em multimodelo[1] que explicitamente representa a linha de produto de software a partir de múltiplas visões (variabilidade, funcional e qualidade) e a relação entre elas, bem como o processo de derivação que faz uso do multimodelo para derivar a arquitetura com os mesmos requisitos de qualidade para outras arquiteturas. A viabilidade do método é demonstrada na arquitetura de sistemas automotivos.

Guana e Correal [130] desenvolveram um método para identificação pontos críticos na arquitetura. Essa estratégia seleciona automaticamente componentes e combina as funcionalidades do produto com requisitos de qualidade. O método foi aplicado na prática em software de cuidados com a saúde para plataforma de dispositivos móvel.

Deng *et al.* [45] apresenta uma abordagem arquitetural centrada em MD-SPL com o intuito de minimizar o problema de evolução do domínio. Para ilustrar a abordagem é apresentado um estudo de caso de um sistema embarcado com o intuito de discutir os principais desafios de evolução do domínio e como envolver arquitetura de linha de produtos de forma sistemática e minimizar a intervenção humana. Uma linguagem de modelagem para sistemas embarcados foi desenvolvida e aplicada uma técnica de transformação de modelo-para-modelo, com o intuito de definir um metamodelo, de forma a simplificar a complexidade de evolução da arquitetura da linha de produto. Uma discussão profunda e muito relevante foi feita quanto à evolução da linha de produto e a viabilidade de se conduzir as dificuldades

[1]Multimodelo é um conjunto de modelos inter-relacionados que representam diferentes visões de um sistema em particular.

relacionadas a esse processo via MDE.

10.1.2 Métodos de Suporte a Decisão Arquitetural

O processo de decisão arquitetural é um mecanismo que serve para explicar as escolhas feitas pelos arquitetos. O tema é muito discutido na comunidade de arquitetura de software principalmente do ponto de vista evolutivo, pois deixar as decisões em evidência e viabiliza a comunicação entre a equipe em qualquer etapa o desenvolvimento.

Capilla *et al.* [131] apresenta um metamodelo de decisão arquitetural para criar e manter relações de dependência entre entidades durante a identificação e confirmação de decisões tomadas, manter a rastreabilidade da evolução das decisões e dar suporte às decisões em tempo de execução. O metamodelo contempla o nível da decisão, registro dos envolvidos com a decisão, a causa que motivou uma possível mudança (como a fundamentação, a recomendação, etc.), As alternativas mapeadas para essa mudança (elementos contra e a favor) e os resultados obtidos com essa mudança (justificativa, opções candidatas como alternativa, consequências, etc.). O metamodelo foi aplicado em um estudo de caso para demonstrar na prática a tomada de decisão sobre a elaboração de uma arquitetura orientada a serviço (SOA).

Eray Tuzun *et al.* [132] contribui com uma perspectiva interessante sobre a viabilidade de adotar as práticas de linha de produto de software por meio de um modelo de suporte a decisão. Trata-se de um estudo profundo, que ilustra a complexidade de variáveis associadas às escolhas arquiteturais. Em seu mapeamento, motivações comerciais, grau de controle sobre a especificação do produto, retorno sobre o investimento, potencial de mercado, tolerância a risco, variabilidade e similaridade do produto, estabilidade do domínio e estabilidade da tecnologia são alguns dos fatores levados em consideração na hora de se fazer escolhas arquiteturais.

10.2 Considerações Finais

Os trabalhos relacionados foram mapeados a partir da revisão da literatura sobre estudos primários e são classificados basicamente por trabalhos com objetivos e ferramental tecnológico em comum.

Existem vários paradigmas que sugerem um ponto de partida à obtenção de soluções de tal problema. Um deles são as engenharias trabalhando de forma integrada (SPL+MDE), com o objetivo de contribuir com seus exemplos de modu-

Tabela 10.1: Síntese dos trabalhos relacionados

Método	Elementos de Entrada	Elementos de Saída	Camada de Solução	Sub-área da Modelagem	Aplicação Prática
Cabello [126]	Arquitetura de referência, modelo de feature e plano de produção	Esqueleto da arquitetura na forma de visão componente-conector	Operação	(ii)	Sistema especialista
OntoAD [127]	Modelo de feature, modelo de arquitetura e a ontologia OSPL	Código ADL (Architecture Description Language)	Operação	(ii)	Voz sobre IP
QuaADI [128, 129]	Padrões de projeto, transformação dirigida por qualidade, multimodelos	Visão de transformação	Operação	(i) e (ii)	Sistema automotivo
Guana e Correal [130]	Modelos de contexto, relação com componentes previamente implementados	Visão de impacto e propriedades	Decisão e Operação	(i), (ii) e (iii)	Sistema de cuidados com a saúde
Deng {et al.} [45]	Padrões de projeto	Framework de componentes arquiteturais	Operação	(ii)	Sistema embarcado
Capilla {et al.} [131]	Metamodelos de decisão, requisitos funcionais e não-funcionais, artefatos de projeto	Visão de decisão arquitetural	Decisão	(iii)	Decisão sobre uma arquitetura orientada a serviço

laridade principalmente nas fases de análise e projeto, porém ainda existem outros fatores que dificultam a adoção das práticas de reúso nesse contexto. Entre esses, destacam-se problemas como: (i) o fato de recair sobre o projetista a responsabilidade de garantir a consistência, corretude dos artefatos gerados - verificação, validação e a rastreabilidade de artefatos; (ii) a evolução do espaço do problema com mínimo ou nenhum impacto sobre o espaço das soluções em andamento.

Capítulo 11

Discussão e Conclusões

Nesse capítulo são apresentadas as considerações finais sobre o trabalho realizado. Uma compilação dos resultados obtidos é discutida na Seção 11.1. As ameaças à validade são relatadas na Seção 11.2. As limitações são descritas na Seção 11.3. O trabalho futuro é discutido na Seção 11.4. Para finalizar, a Seção 11.5 apresenta uma análise da contribuição inovadora dessa pesquisa.

11.1 Retornando às Questões de Pesquisa

QP-1: A variabilidade da modelagem arquitetural influencia na eficiência e eficácia das provas de conceito? Sim. A influência foi observada na prática tanto do ponto de vista endógeno como exógeno. No caso da variabilidade exógena, o fato de ter uma arquitetura de referência faz toda diferença para o reúso. Contar com uma referência, permite que as melhores práticas sejam adotadas e soluções sejam idealizadas a partir de projetos passados. Já no caso da variabilidade endógena, a quantidade versus tipo de visões deve garantir um equilíbrio entre o esperado e praticado. A *Prova de Conceito 1 - CarbonQL* apresentou uma situação de extrapolação sobre a quantidade recomendada pelo framework arquitetural em função de uma necessidade de domínio, por demandar mais formalizações. Desde que devidamente justificadas, esse tipo de comportamento do fenômeno observado só fortalece a evidência da influência positiva sobre a eficiência e eficácia.

QP-2: Como o framework arquitetural proposto pode auxiliar o reúso de software? Pode servir como um guia prático tanto no *desenvolvimento de componentes reusáveis* como no desenvolvimento de *sistemas com reúso de componentes*. Essa situação pode ser observada nas provas de conceito. Como exemplo, tem

a Carbontology como elemento reusável e a própria PPL2Opt4J como linguagem de transformação com reúso de componentes (modelo de configuração).

QP-3: Como a integração entre as engenharias (SPL+MDE) pode influenciar no desenvolvimento de sistemas mais eficientes e incrementais? Essa integração pode influenciar positivamente no desenvolvimento de sistemas mais eficientes e incrementais desde que a análise custo benefício de adoção a revele como um caminho válido para o contexto do projeto. Não é correto afirmar que a integração entre engenharias é uma abordagem generalista aplicada a qualquer tipo de projeto, até por que são necessários mais estudos para mapear essa fronteira de aplicação. Essa tese identificou que projetos com uma clara demanda por transformação e/ou configuração de modelos, atrelada ao mapeamento de similaridades e variabilidades, pode se beneficiar da abordagem e evoluir incrementalmente por meio do arcabouço oferecido.

QP-4: Qual o impacto da variabilidade da modelagem arquitetural sobre o método proposto? O método proposto é composto por dois elementos: o esquema de integração entre as engenharias e o framework de suporte a tal integração. A variabilidade da modelagem arquitetural tem impacto direto sobre esses dois elementos do ponto de vista evolutivo. Ambos estão atrelados a uma abordagem promissora, porém emergente. Estudos sobre a integração entre as engenharias (SPL+MDE) e mais aplicações do método proposto devem ser realizadas para evitar o viés da pesquisa.

11.2 Ameaças à validade

As ameaças que podem afetar os resultados dessa pesquisa seguem três categorias: confiabilidade, validade e sensibilidade. A coleta de dados sobre as visões de especificação das provas, as características, as particularidades e conexões entre as mesmas ajuda a mitigar riscos relacionados à confiabilidade das evidências. A definição dos critérios de sucesso para as provas ajuda a mitigar riscos relacionados à validade, eliminando a possibilidade de incluir uma prova que não atenda aos critérios. O modelo teórico e a definição das variáveis latentes e observáveis ajuda a mitigar riscos relacionados à sensibilidade do estudo.

11.3 Limitações

De forma geral, quanto às limitações do trabalho é preciso melhorar:

- A gestão de dependências entre os pontos de variabilidade de um projeto conduzido com base no método proposto e de preferência, que haja ferramental e suporte automático, pois a forma como foi conduzida nas duas provas de conceito, tudo ficou centralizado em uma única pessoa para não perder o controle das decisões. Em equipes descentralizadas, esse controle se tornaria muito trabalhoso e fragilizaria o projeto como um todo;
- A evolução dos mapeamentos/modelos de variabilidade que envolvem tomada de decisão por quais variabilidades deixam de ser necessárias, envolvendo um trabalho manual e engenharia de regras para garantir a verificação de consistência e propagação de valores;
- A diversificação das observações da variabilidade da modelagem arquitetural sobre o método proposto;
- O critério de avaliação por meio da inserção de abordagens formais.

11.4 Trabalho Futuro

No decorrer desse trabalho, surgiram alguns pontos que não foram tratados devido à delimitação do escopo inicialmente proposto. Esses pontos de pesquisa, que envolvem os principais temas de continuação considerados para esse trabalho são os seguintes:

- Aplicar o método proposto em outros domínios e no contexto de outros sistemas;
- Realizar novos estudos do ponto de vista arquitetural relativos à integração entre as engenharias;
- Avaliar o método proposto com base em técnicas quantitativas;
- Evoluir incrementalmente a CarbonQL de protótipo estrutural para uma linguagem em ambiente produtivo;

- Evoluir incrementalmente a PPL de protótipo estrutural para uma linguagem em ambiente produtivo; e

- Investigar o equilíbrio da quantidade e quais visões usar para uma mesma arquitetura, já que esse é um problema combinatorial que envolve variáveis diversas, como a análise custo benefício de se gerar mais uma visão.

11.5 Mensurando as Contribuições

Essa tese contribuiu com uma percepção endógena e exógena da variabilidade da modelagem arquitetura, apresentou uma arquitetura para integração entre as engenharias e, com o intuito de dar suporte à implementação dessa integração, foi concebido o framework arquitetural. Secundariamente, contribuiu com a metodologia ARDev.

Essas contribuições foram guiadas pelo resultado da revisão da literatura que forneceu um panorama das pesquisas conduzidas e de outras teses de doutorado desenvolvidas dentro da mesma área de conhecimento. A nível Brasil, não foi encontrado nenhum trabalho de doutorado ou mestrado no banco de tese da CAPES que se concentre na integração das engenharias (SPL+MDE) e que explore questões arquiteturais essenciais para a evolução dessa integração. Essa ausência de trabalhos pode ser compreendida de duas formas: pesquisadores não investiram esforços por ser um tema irrelevante ou é simplesmente uma área pouco explorada.

A revisão da literatura indica que se trata da segunda opção, pois a área a nível internacional é pulsante e movimenta conferências, workshops, vários trabalhos acadêmicos já foram concluídos e outros estão em andamento. Sendo assim, esse trabalho reúne elementos que o caracterizam como pioneiro e as contribuições são inovadoras. As publicações desse trabalho são listadas a seguir a começar pela mais recente:

- Daniella Bezerra, Arilo Dias-Neto e Raimundo Barreto. *ARDev: A Methodology for Managing Research on Software Technologies.* Journal of Software: Evolution and Process (JSEM). Special Issue - ASPD. Em processo de *major revision.* QUALIS: B1.

- Daniella Bezerra, Arilo Dias-Neto e Raimundo Barreto. *ARDev: A Methodology Based on Scrum Principles to Support Research Management on Software*

Technologies. 24th Annual International Conference on Computer Science and Software Engineering (CASCON), Toronto, Canadá. 2014. QUALIS: B1.

- Daniella Bezerra, Raimundo Barreto. *Domain Engineering: A Practical Application in Analysis and Design of a Generative Query Language.* 8th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), Maceió, Alagoas. 2014. QUALIS: B3.

- Daniella Bezerra, Rosiane de Freitas Rodrigues e Raimundo Barreto. *Inventário Florestal na Amazônia: uma modelagem da Logística Integrada Multimodal.* Anais do Encontro Regional de Pesquisa Operacional do Norte (ERPO-NO), Manaus, Amazonas, 2011.

- Daniella Bezerra, Rosiane de Freitas Rodrigues, Ulisses Cunha e Raimundo Barreto. *Manejo Florestal Sustentável: Dificuldade Computacional e Otimização de Processos.* Anais da Escola Regional de Informática (ERIN), Manaus, Amazonas, 2010. *Premiado como melhor artigo científico.

Referências Bibliográficas

[1] D. Bezerra, A. Dias-Neto, and R. Barreto. Ardev: A methodology based on scrum principles to support research management on software technologies. In *24rd Annual International Conference on Computer Science and Software Engineering*, CASCON '14, pages 363–366, Toronto, CA, 2014. CAS Research.

[2] A. Daniluk. Visual modeling for scientific software architecture design. a practical approach. *Computer Physics Communications*, 183(2):213 – 230, 2012.

[3] T. Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Bookshelf, 2007.

[4] KyoC. Kang and Hyesun Lee. Variability modeling. In Rafael Capilla, Jan Bosch, and Kyo-Chul Kang, editors, *Systems and Software Variability Management*, pages 25–42. Springer Berlin Heidelberg, 2013.

[5] Faheem Ahmed and Luiz Fernando Capretz. An architecture process maturity model of software product line engineering. *Innovations in Systems and Software Engineering*, 7(3):191–207, 2011.

[6] T. M. Araújo, N. Higuchi, and J. A. Carvalho Jr. Comparison of formulae for biomass content determination in a tropical rainforest site in the state of pará, brazil. *Forest Ecology and Management Journal*, 117(5388):43–52, 1999.

[7] Steffen Thiel. On the definition of a framework for an architecting process supporting product family development. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, PFE '01, pages 125–142, London, UK, 2002. Springer-Verlag.

[8] Sridhar Chimalakonda and Dan Hyung Lee. On the evolution of software and systems product line standards. *SIGSOFT Softw. Eng. Notes*, 41(3):27–30, June 2016.

[9] Ebrahim Bagheri, David Benavides, Klaus Schmid, and Per Runeson. Foreword to the special issue on empirical evidence on software product line engineering. *Empirical Software Engineering*, 21(4):1579–1585, 2016.

[10] Jean-Marc Jézéquel. Model-driven engineering for software product lines. *ISRN Software Engineering*, 2012:1–24, 2012.

[11] Kathrin Berg, Judith Bishop, and Dirk Muthig. Tracing software product line variability: From problem to solution space. In *Proceedings of the 2005 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*, SAICSIT '05, pages 182–191, Republic of South Africa, 2005. South African Institute for Computer Scientists and Information Technologists.

[12] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wkasowski. A survey of variability modeling in industrial practice. In *Proceedings of the 7th International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '13, pages 1–8, New York, NY, USA, 2013. ACM.

[13] Lianping Chen and Muhammad Ali Babar. A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology*, 53(4):344 – 362, 2011. Special section: Software Engineering track of the 24th Annual Symposium on Applied ComputingSoftware Engineering track of the 24th Annual Symposium on Applied Computing.

[14] Iris Reinhartz-Berger. Towards automatization of domain modeling. *Data Knowl. Eng.*, 69(5):491–515, may 2010.

[15] Dines Bjorner. Domain theory: practice and theories a discussion of possible research topics. In *Proceedings of the 4th international conference on Theoretical aspects of computing*, ICTAC '07, pages 1–17, 2007.

[16] Robert Cartwright, Rebecca Parsons, and Moez A. AbdelGawad. Domain theory: An introduction. Technical report, Cornell University, 2016.

[17] Maarit Harsu. A survey on domain engineering. Technical report, 2002.

[18] Abdelrahman Osman Elfaki. A rule-based approach to detect and prevent inconsistency in the domain-engineering process. *Expert Systems*, 33(1):3–13, 2016.

[19] Requirements engineering for software product lines: A systematic literature review. *Information and Software Technology*, 52(8):806 – 820, 2010.

[20] Mahvish Khurum and Tony Gorschek. A systematic review of domain analysis solutions for product lines. *Journal of Systems and Software*, 82(12):1982 – 2003, 2009.

[21] Thainá Mariani, Thelma Elita Colanzi, and Silvia Regina Vergilio. Preserving architectural styles in the search based design of software product line architectures. *Journal of Systems and Software*, 115:157 – 173, 2016.

[22] Ruben Heradio, Hector Perez-Morago, David Fernandez-Amoros, Francisco Javier Cabrerizo, and Enrique Herrera-Viedma. A bibliometric analysis of 20 years of research on software product lines. *Information and Software Technology*, 72:1 – 15, 2016.

[23] E.D. De Souza Filho, R. De Oliveira Cavalcanti, D.F.S. Neiva, T.H.B. Oliveira, L.B. Lisboa, E.S. De Almeida, and S.R. De Lemos Meira. Evaluating domain design approaches using systematic review. *Lecture Notes in Computer Science*, 5292 LNCS:50–65, 2008.

[24] Uzma Afzal, Tariq Mahmood, and Zubair Shaikh. Intelligent software product line configurations: A literature review. *Computer Standards Interfaces*, 48:30 – 48, 2016.

[25] Mojtaba Shahin, Peng Liang, and Muhammad Ali Babar. A systematic review of software architecture visualization techniques. *Journal of Systems and Software*, 94:161 – 185, 2014.

[26] Benjamin Cool, Christoph Knieke, Andreas Rausch, Mirco Schindler, Arthur Strasser, Martin Vogel, Oliver Brox, and Stefanie Jauns-Seyfried. From product architectures to a managed automotive software product line architecture. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, pages 1350–1353, New York, NY, USA, 2016. ACM.

[27] Bovstjan Slivnik. Measuring the complexity of domain-specific languages developed using mdd. *Software Quality Journal*, 24(3):737–753, 2016.

[28] Miguel Goulão, Vasco Amaral, and Marjan Mernik. Quality in model-driven engineering: a tertiary study. *Software Quality Journal*, 24(3):601–633, 2016.

[29] Krzysztof Czarnecki, Thomas Bednasch, Peter Unger, and Ulrich Eisenecker. *Generative Programming for Embedded Software: An Industrial Experience Report*, pages 156–172. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[30] Daniel Calegari and Nora Szasz. Verification of model transformations: A survey of the state-of-the-art. *Electronic Notes in Theoretical Computer Science*, 292:5 – 25, 2013.

[31] K. Holldobler, B. Rumpe, and I. Weisemoller. Systematically deriving domain-specific transformation languages. In *ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems*, (MODELS '15), pages 136–145, Sept 2015.

[32] M. Fowler. *Domain-Specific Languages*. Addison-Wesley Professional, 2010.

[33] T. Parr. *Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages*. Addison-Wesley Professional, 2010.

[34] Tomaz Kosar, Sudev Bohra, and Marjan Mernik. Domain-specific languages: A systematic mapping study. *Information and Software Technology*, 71:77 – 91, 2016.

[35] Heiko Koziolek, Thomas Goldschmidt, Thijmen de Gooijer, Dominik Domis, Stephan Sehestedt, Thomas Gamer, and Markus Aleksy. Assessing software product line potential: an exploratory industrial case study. *Empirical Software Engineering*, 21(2):411–448, 2016.

[36] Michael Szvetits and Uwe Zdun. Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime. *Software Systems Modeling*, 15(1):31–69, 2016.

[37] Colin Atkinson, Ralph Gerbig, and Mathias Fritzsche. A multi-level approach to modeling language extension in the enterprise systems domain. *Information Systems*, 54:289 – 307, 2015.

[38] Czarnecki Krzysztof, Antkiewicz Michal, Kim Chang Hwan Peter, Lau Sean, and Pietroszek Krzysztof. Model-driven software product lines. In *Companion to the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '05, pages 126–127, New York, NY, USA, 2005. ACM.

[39] Julio Ariel Hurtado, María Cecilia Bastarrica, Sergio F. Ochoa, and Jocelyn Simmonds. Mde software process lines in small companies. *Journal of Systems and Software*, 86(5):1153 – 1171, 2013.

[40] Andreas Pleuss, Goetz Botterweck, Deepak Dhungana, Andreas Polzer, and Stefan Kowalewski. Model-driven support for product line evolution on feature level. *Journal of Systems and Software*, 85(10):2261 – 2274, 2012. Automated Software Evolution.

[41] Jihen Maazoun, Nadia Bouassida, and Hanene Ben-Abdallah. Change impact analysis for software product lines. *Journal of King Saud University - Computer and Information Sciences*, pages 1 – 17, 2016.

[42] Felix Schwagerl, Thomas Buchmann, Sabrina Uhrig, and Bernhard Westfechtel. *Realizing a Conceptual Framework to Integrate Model-Driven Engineering, Software Product Line Engineering, and Software Configuration Management*, pages 21–44. Springer International Publishing, 2015.

[43] Jean-Claude Royer and Hugo Arboleda. *Model-Driven and Software Product Line Engineering*. Wiley-ISTE, October 2012.

[44] David Blanes and Emilio Insfran. A comparative study on model-driven requirements engineering for software product line. *Revista de Sistemas e Computação*, 2(1):3 – 13, 2012.

[45] Gan Deng, Douglas Schimidt, Aniruddha Gokhale, Jeff Gray, Yuehua Lin, and Gunther Lenz. *Designing Software-Intensive Systems: Methods and Principles*, chapter Evolution in model-driven software product line architecture, pages 102–132. Information Science Reference, 2008.

[46] Daniel Lucredio, Renata P. M. Fortes, Eduardo S. Almeida, and Silvio R. L. Meira. Designing domain architectures for model-driven engineering. In *Fourth*

Brazilian Symposium on Software Components, Architectures and Reuse, SBCARS 2010, Salvador, Bahia, Brazil, September 27 - October 1, 2010, pages 100–109, 2010.

[47] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[48] M. Schlesinger and B. McMurray. The past, present, and future of computational models of cognitive development. *Cognitive Development*, 27(4):326 – 348, 2012.

[49] Graeme S. Halford, Glenda Andrews, William H. Wilson, and Steven Phillips. Computational models of relational processes in cognitive development. *Cognitive Development*, 27(4):481 – 499, 2012.

[50] Stuart Marcovitch and Philip David Zelazo. The potential contribution of computational modeling to the study of cognitive development: When, and for what topics? *Cognitive Development*, 27(4):323 – 325, 2012.

[51] P. C. R. Lane and F. Gobet. Developing reproducible and comprehensible computational models. *Artif. Intell.*, 144(1-2):251–263, March 2003.

[52] Dana Scott. Outline of a mathematical theory of computation. *4th Princeton Conference on Information Sciences and Systems*, pages 169 – 176, 1970.

[53] Samson Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51(1):1 – 77, 1991.

[54] Samson Abramsky and Achim Jung. Domain theory. In *Handbook of Logic in Computer Science*, pages 1–168. Clarendon Press, 1994.

[55] Amir Molzam Sharifloo, Andreas Metzger, Clement Quinton, Luciano Baresi, and Klaus Pohl. Learning and evolution in dynamic software product lines. In *11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '16, pages 158–164, New York, NY, USA, 2016. ACM.

[56] Bruno B. P. Cafeo, Claus Hunsen, Alessandro Garcia, Sven Apel, and Jaejoon Lee. Segregating feature interfaces to support software product line mainte-

nance. In *15th International Conference on Modularity*, MODULARITY 2016, pages 1–12, New York, NY, USA, 2016. ACM.

[57] Stefan Fischer. Reducing the test effort of variability-rich systems by using feature interaction knowledge and variability-aware source code analysis. In *38th International Conference on Software Engineering Companion*, ICSE '16, pages 855–858, New York, NY, USA, 2016. ACM.

[58] Thomas Devine, Katerina Goseva-Popstojanova, Sandeep Krishnan, and Robyn R. Lutz. Assessment and cross-product prediction of software product line quality: Accounting for reuse across products, over multiple releases. *Automated Software Eng.*, 23(2):253–302, June 2016.

[59] Stefan Fischer, Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. Enhancing clone-and-own with systematic reuse for developing software variants. In *IEEE International Conference on Software Maintenance and Evolution*, ICSME '14, pages 391–400, Washington, DC, USA, 2014. IEEE Computer Society.

[60] Wesley Klewerton and Guez Assuncao. Search-based migration of model variants to software product line architectures. In *37th International Conference on Software Engineering - Volume 2*, ICSE '15, pages 895–898, Piscataway, NJ, USA, 2015. IEEE Press.

[61] Harvey Siy and Audris Mockus. Measuring domain engineering effects on software change cost. In *Proceedings of the 6th International Symposium on Software Metrics*, METRICS '99, page 304, Washington, DC, USA, 1999.

[62] Ana Paula T. B. Blois, Claudia M. L. Werner, and Karin Becker. Towards a components grouping technique within a domain engineering process. In *Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, EUROMICRO '05, pages 18–27, Washington, DC, USA, 2005.

[63] A. van Deursen, P. Klint, and J. Visser. Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, 35(6):26–36, June 2000.

[64] A. Asmaa, R. Ounsa, S. Nissrine, and S. Camille. Selecting spl modeling languages: A practical guide. In *3rd World Conference on Complex Systems (WCCS)*, pages 1–6, Nov 2015.

[65] William B. Frakes and Kyo Kang. Software reuse research: Status and future. *IEEE Trans. Softw. Eng.*, 31(7):529–536, jul 2005.

[66] Tom Mens, Pieter Van Gorp, Dániel Varró, and Gabor Karsai. Applying a model transformation taxonomy to graph transformation technology. *Electronic Notes in Theoretical Computer Science*, 152:143 – 159, 2006. Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005)Graph and Model Transformation 2005.

[67] Laurence Tratt. Model transformations and tool integration. *Software & Systems Modeling*, 4(2):112–122, 2005.

[68] Karsten Ehrig, Claudia Ermel, and Stefan Hänsgen. Towards model transformation in generated eclipse editor plug-ins. *Electron. Notes Theor. Comput. Sci.*, 152:39–52, mar 2006.

[69] Eugene Syriani and Jeff Gray. Challenges for addressing quality factors in model transformation. In *Fifth IEEE International Conference on Software Testing, Verification and Validation, ICST 2012, Montreal, QC, Canada, April 17-21, 2012*, pages 929–937, 2012.

[70] Johan Haan. Site com artigos técnicos sobre MDE. Disponível em: http://www.theenterprisearchitect.eu. Acesso em: 11-Fevereiro-2016.

[71] J. Hutchinson, M. Rouncefield, and J. Whittle. Model-driven engineering practices in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 633–642, New York, NY, USA, 2011. ACM.

[72] Valeria De Castro, Esperanza Marcos, and Juan Manuel Vara. Applying cim-to-pim model transformations for the service-oriented development of information systems. *Information and Software Technology*, 53(1):87 – 105, 2011.

[73] Emna Mezghani, Riadh Ben Halima, Ismael Bouassida Rodriguez, and Khalil Drira. A model driven methodology for enabling autonomic reconfiguration of service oriented architecture. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1772–1773, New York, NY, USA, 2013. ACM.

[74] Verónica Andrea Bollati, Juan Manuel Vara, Álvaro Jiménez, and Esperanza Marcos. Applying mde to the (semi-)automatic development of model transformations. *Information and Software Technology*, 55(4):699 – 718, 2013.

[75] S. Kent. Model driven engineering. In *Proceedings of the 3rd International Conference on Integrated Formal Methods*, IFM '02, pages 286–298, London, UK, 2002. Springer-Verlag.

[76] Marco Torchiano, Federico Tomassetti, Filippo Ricca, Alessandro Tiso, and Gianna Reggio. Relevance, benefits, and problems of software modelling and model driven techniques - a survey in the italian industry. *Journal of Systems and Software*, 86(8):2110 – 2126, 2013.

[77] Jean-Marie Favre. Towards a basic theory to model model driven engineering. pages 262–271, 2004.

[78] Gan Deng, Gunther Lenz, and Douglas C. Schmidt. *Addressing Domain Evolution Challenges in Software Product Lines*, pages 247–261. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[79] Correa Chessman K. F., Oliveira Toacy C., and Werner Claudia M. L. An analysis of change operations to achieve consistency in model-driven software product lines. In *Proceedings of the 15th International Software Product Line Conference, Volume 2*, SPLC '11, pages 1–24, New York, NY, USA, 2011. ACM.

[80] Thomas Buchmann, Alexander Dotor, and Bernhard Westfechtel. Mod2-scm: A model-driven product line for software configuration management systems. *Information and Software Technology*, 55(3):630 – 650, 2013. Special Issue on Software Reuse and Product Lines.

[81] Xiaorui Zhang, Haugen O., and Moller-Pedersen B. Model comparison to synthesize a model-driven software product line. In *Software Product Line Conference (SPLC), 2011 15th International*, pages 90–99, Aug 2011.

[82] Vinay Kulkarni. Use of sple to deliver custom solutions at product cost: Challenges and a way forward. In *Proceedings of the 2nd International Workshop on Product Line Approaches in Software Engineering*, PLEASE '11, pages 1–5, New York, NY, USA, 2011. ACM.

[83] Parra Carlos, Joya Diego, Giral Leonardo, and Infante Alvaro. An soa approach for automating software product line adoption. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 1231–1238, New York, NY, USA, 2014. ACM.

[84] Cuong Cu and Yongjie Zheng. Architecture-centric derivation of products in a software product line. In *Proceedings of the 8th International Workshop on Modeling in Software Engineering*, MiSE '16, pages 27–33, New York, NY, USA, 2016. ACM.

[85] Hamad I. Alsawalqah, Sungwon Kang, and Jihyun Lee. A method to optimize the scope of a software product platform based on end-user features. *Journal of Systems and Software*, 98:79 – 106, 2014.

[86] Karma Sherif, Radha Appan, and Zhangxi Lin. Resources and incentives for the adoption of systematic software reuse. *International Journal of Information Management*, 26(1):70 – 80, 2006.

[87] Daniel Lucrédio, Kellyton dos Santos Brito, Alexandre Alvaro, Vinicius Cardoso Garcia, Eduardo Santana de Almeida, Renata Pontin de Mattos Fortes, and Silvio Lemos Meira. Software reuse: The brazilian industry scenario. *Journal of Systems and Software*, 81(6):996 – 1013, 2008. Agile Product Line Engineering.

[88] Karma Sherif and Ajay Vinze. Barriers to adoption of software reuse: A qualitative study. *Information and Management*, 41(2):159 – 175, 2003.

[89] Veronika Bauer and Antonio Vetro. Comparing reuse practices in two large software-producing companies. *Journal of Systems and Software*, 117:545 – 582, 2016.

[90] Li Bin, Zhou Yun-fei, and Tang Xiao-qi. A research on open cnc system based on architecture/component software reuse technology. *Computers in Industry*, 55(1):73 – 85, 2004.

[91] Myoungkyu Song and Eli Tilevich. Reusing metadata across components, applications, and languages. *Science of Computer Programming*, 98, Part 4:617 – 644, 2015.

[92] Douwe Postmus and Theo Dirk Meijler. Aligning the economic modeling of software reuse with reuse practices. *Information and Software Technology*, 50(7-8):753 – 762, 2008.

[93] Rafael Capilla, Anton Jansen, Antony Tang, Paris Avgeriou, and Muhammad Ali Babar. 10 years of software architecture knowledge management: Practice and future. *Journal of Systems and Software*, pages 1 – 15, 2015.

[94] David Ameller, Xavier Burgués, Oriol Collell, Dolors Costal, Xavier Franch, and Mike P. Papazoglou. Development of service-oriented architectures using model-driven development: A mapping study. *Information and Software Technology*, 62:42 – 66, 2015.

[95] Olaf Zimmermann, Christoph Miksovic, and Jochen M. Kuster. Reference architecture, metamodel, and modeling principles for architectural knowledge management in information technology services. *Journal of Systems and Software*, 85(9):2014 – 2033, 2012.

[96] J. Ven, A. Jansen, J. Nijhuis, and J. Bosch. *Design decisions: The bridge between rationale and architecture*, pages 329–348. Springer Berlin Heidelberg, 2006. Rationale Management in Software Engineering.

[97] Paul C. Clements. A survey of architecture description languages. In *Proceedings of the 8th International Workshop on Software Specification and Design*, IWSSD '96, pages 16–28, Washington, DC, USA, 1996. IEEE Computer Society.

[98] David Garlan and Joao Sousa. Documenting software architecture: recommendations for industrial practice. Technical report, Carnegie Mellon University, October 2000.

[99] Christine Hofmeister, Philippe Kruchten, Robert L. Nord, Henk Obbink, Alexander Ran, and Pierre America. A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software*, 80(1):106 – 126, 2007.

[100] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software Architecture: Foundations, Theory, and Practice.* Wiley Publishing, 2009.

[101] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[102] Zhenyi Jin. *A software architecture-based testing technique*. PhD thesis, George Mason University, 2000.

[103] Antonia Bertolino and Paola Inverardi. Architecture-based software testing. In *Joint Proceedings of the 2nd International Software Architecture Workshop (ISAW-2) and International Workshop on Multiple Perspectives in Software Development (Viewpoints '96) on SIGSOFT '96 Workshops*, ISAW '96, pages 62–64, New York, NY, USA, 1996. ACM.

[104] Antonia Bertolino, Paola Inverardi, and Henry Muccini. *Formal Methods in Testing Software Architectures*, pages 122–147. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[105] Hongyu Pei Breivold, Ivica Crnkovic, and Magnus Larsson. A systematic review of software architecture evolution research. *Information and Software Technology*, 54(1):16 – 40, 2012.

[106] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[107] Julio Sincero, Reinhard Tartler, Daniel Lohmann, and Wolfgang Schroder-Preikschat. Efficient extraction and analysis of preprocessor-based variability. In *Proceedings of the 9th International Conference on Generative Programming and Component Engineering*, GPCE '10, pages 33–42, New York, NY, USA, 2010. ACM.

[108] CharlesW. Krueger. *Easing the Transition to Software Mass Customization*, pages 282–293. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[109] Vander Alves, Pedro Matos, Leonardo Cole, Paulo Borba, and Geber Ramalho. *Extracting and Evolving Mobile Games Product Lines*, pages 70–81. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[110] Kentaro Kumaki, Ryosuke Tsuchiya, Hironori Washizaki, and Yoshiaki Fukazawa. Supporting commonality and variability analysis of requirements and

structural models. In *16th International Software Product Line Conference - Volume 2*, SPLC '12, pages 115–118, New York, NY, USA, 2012. ACM.

[111] Marko Rosenmuller, Norbert Siegmund, Sven Apel, and Gunter Saake. Flexible feature binding in software product lines. *Automated Software Engineering*, 18(2):163–197, jun 2011.

[112] Venkat Chakravarthy, John Regehr, and Eric Eide. Edicts: Implementing features with flexible binding times. In *Proceedings of the 7th International Conference on Aspect-oriented Software Development*, AOSD '08, pages 108–119, New York, NY, USA, 2008. ACM.

[113] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering*, EASE'08, pages 68–77, Swinton, UK, UK, 2008. British Computer Society.

[114] D. Li and C. K. Chang. Initiating and institutionalizing software product line engineering: From bottom-up approach to top-down practice. In *2009 33rd Annual IEEE International Computer Software and Applications Conference*, volume 1, pages 53–60, July 2009.

[115] Vincent Lussenburg, Tijs van der Storm, Jurgen Vinju, and Jos Warmer. Mod4j: A qualitative case study of model-driven software development. In *Model Driven Engineering Languages and Systems*, pages 346–360, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[116] Perovich D., Rossel P.O., and Bastarrica M.C. Feature model to product architectures: Applying mde to software product lines. In *Software Architecture, 2009 European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*, pages 201–210, 2009.

[117] IPCC. Ipcc volume1. Technical report, 2006. General Guidance and Reporting.

[118] IPCC. Guidelines for national greenhouse gas inventories. Technical report, 2006. Agriculture, Forestry and Other Land Use.

[119] R. Tairas, M. Mernik, and J. Gray. Models in software engineering. chapter Using Ontologies in the Domain Analysis of Domain-Specific Languages, pages 332–342. Springer-Verlag, Berlin, Heidelberg, 2009.

[120] Site com informações sobre o framework de mapeamento objeto-relacional Hibernate. Disponível em: http://www.hibernate.org/. Acesso em: 11-Fevereiro-2016.

[121] Documentação on line da linguagem Hibernate Query Language HQL. Disponível em: http://docs.jboss.org /hibernate/orm/3.3/ reference/en/html/-queryhql.html. Acesso em: 11-Fevereiro-2016.

[122] Site com informações sobre o gerador de parser Antlr. Disponível em: http://www.antlr.org/. Acesso em: 11-Fevereiro-2016.

[123] C.F. Almeida. *Elaboração de Rede de Transporte Multimodal de Carga para a Região Amazônica sob o Enfoque de Desenvolvimento Econômico.* PhD thesis, Universidade de Brasília, Departamento de Engenharia Civil e Ambiental, Brasília, DF, Junho 2008.

[124] Ernesto Bonomi and Jean-Luc Lutton. The n-city travelling salesman problem: Statistical mechanics and the metropolis algorithm. *SIAM Review*, 26(4):551–568, 1984.

[125] José Elias Claudio Arroyo and Vinícius Amaral Armentano. Genetic local search for multi-objective flowshop scheduling problems. *European Journal of Operational Research*, 167(3):717–738, 2005.

[126] Maria Eugenia Cabello, Isidro Ramos, Oscar Alberto Santana, and Saúl Iván Beristain. A generic process for the design and generation of software product line skeleton architectures. *International Journal of Software Engineering and Knowledge Engineering*, 24(09):1301–1335, 2014.

[127] H. A. Duran-Limon, C. A. Garcia-Rios, F. E. Castillo-Barrera, and R. Capilla. An ontology-based product architecture derivation approach. *IEEE Transactions on Software Engineering*, 41(12):1153–1168, Dec 2015.

[128] Emilio Insfran, Silvia Abrahão, Javier González-Huerta, John D. McGregor, and Isidro Ramos. *A Multimodeling Approach for Quality-Driven Architecture Derivation*, pages 205–218. Springer US, Boston, MA, 2013.

[129] Javier Gonzalez-Huerta, Emilio Insfran, Silvia Abrahao, and Giuseppe Scanniello. Validating a model-driven software architecture evaluation and improve-

ment method: A family of experiments. *Information and Software Technology*, 57:405 – 429, 2015.

[130] Victor Guana and Dario Correal. Improving software product line configuration: A quality attribute-driven approach. *Information and Software Technology*, 55(3):541 – 562, 2013. Special Issue on Software Reuse and Product LinesSpecial Issue on Software Reuse and Product Lines.

[131] Rafael Capilla, Olaf Zimmermann, Uwe Zdun, Paris Avgeriou, and Jochen M. Küster. An enhanced architectural knowledge metamodel linking architectural design decisions to other artifacts in the software engineering lifecycle. In *5th European Conference on Software Architecture*, ECSA'11, pages 303–318, Berlin, Heidelberg, 2011. Springer-Verlag.

[132] Eray Tuzun, Bedir Tekinerdogan, Mert Emin Kalender, and Semih Bilgen. Empirical evaluation of a decision support model for adopting software product line engineering. *Information and Software Technology*, 60:77 – 101, 2015.

Apêndice A

Revisão da Literatura

Este Apêndice apresenta a revisão *ad-hoc* da literatura que foi realizada sobre a integração entre Linha de Produto de Software e Engenharia Guiada por Modelo, com o objetivo de identificar abordagens e necessidades. O objetivo do estudo é obter uma visão geral das pesquisas que se concentram na interseção dessas áreas, com base nas seguintes atividades: (i) busca por publicações relevantes, (ii) seleção das publicações e (iii) mapeamento das publicações.

A seção A.1 apresenta o critério de busca utilizado. Já o critério de seleção das publicações é apresentado na Seção A.2. Os estudos primários identificados que se concentram na integração das áreas SPL e MDE são apresentados na Seção A.2.1. O objetivo é obter uma visão geral das pesquisas que se concentram em ambas as categorias de estudo com base nas seguintes atividades: (i) busca por publicações relevantes, (ii) seleção das publicações e (iii) agrupamento. Para finalizar, a Seção A.3 apresenta algumas considerações sobre revisão da literatura.

A.1 Estratégia de Busca

A busca por publicações relevantes iniciou com uma revisão exploratória da literatura, onde foi observado que enquanto a comunidade de SPL se concentra em desenvolver um arcabouço de reúso fundamentado em produzir uma família de software em um domínio específico, em paralelo, temos a comunidade MDE que se concentra em avançar o estado da arte na manipulação de modelos e suas várias formas de transformação para o desenvolvimento do software.

Apesar de terem surgido em paralelo e com linhas isoladas de desenvolvimento, desde o ano de 2005, com a publicação de Czarnecki *et al.* [38], é possível observar na comunidade científica uma crescente discussão entre os pontos de interseção das

áreas dando origem a uma fusão chamada *Model-Driven Software Product Line* MD-SPL.

As seguintes bibliotecas digitais foram consultadas:

- ACM Digital Library;
- Science Direct;
- CiteSeerX;
- IEEE.

A escolha dessas bibliotecas se deu primeiramente por serem tradicionais e indexarem publicações relevantes e de impacto científico e, em segundo lugar, por estarem acessíveis a partir da rede institucional da Universidade. Foram pesquisados periódicos, conferências, relatórios técnicos, dissertações e teses, sem restrição de tempo.

A.2 Seleção das Publicações

A seleção foi baseada na pertinência do artigo em função das questões de pesquisa definidas. O primeiro filtro de seleção foi realizado com base na leitura do título, resumo e palavras-chave. Para cada publicação analisada, foram considerados os seguintes critérios:

- Inclusão - (i) a publicação tem que estar em inglês ou português; (ii) tem que estar disponível na íntegra e; (iii) deve indicar aprofundamento no tema.
- Exclusão - (i) a publicação que não está nos idiomas selecionados; (ii) que esteja fragmentadas, com partes omitidas ou parcialmente indisponíveis e; (iii) publicação que não esteja relacionada com o tema.

A leitura na íntegra das publicações que passaram pelo primeiro filtro compõe o processo do segundo filtro. Nesta etapa o objetivo era a coleta de dados com base nas seguintes questões de pesquisa:

- *Que desafios são relatados na integração entre MDE e ED/EA?* Este esclarecimento permite validar a relevância de pesquisas passadas e futuras;

- *Onde essas pesquisas são publicadas?* Existem conferências e *workshops* especializados no tema, mas o objetivo é identificar se há inserção do tema em veículos de publicação mais generalistas;
- *Que tópicos são investigados?* A resposta para essa pergunta permite traçar direcionamentos para a construção do conhecimento incremental; e
- *Que tipos de pesquisas são realizadas?* As abordagens bem fundamentadas agregam credibilidade às evidências.

A Tabela A.1 mostra o número de artigos retornados em cada biblioteca digital e o número de artigos selecionados após o segundo filtro que é a leitura do artigo na íntegra.

Tabela A.1: Artigos retornados versus selecionados por biblioteca digital.

	Publicações	
Biblioteca Digital	Retornadas	Selecionadas
ACM	9	6
Science Direct	11	6
CiteSeerX	76	16
IEEE	1	1

A principal conclusão foi que, embora a combinação entre as engenharias pareça ser uma boa ideia, faltam descrição de como implementar essa ideia. 29 trabalhos discutem essa combinação, com margem para explorar em profundidade questões básicas, tais como:

- A representação da variabilidade e similaridades; e
- Os paradigmas arquiteturais e padrões de projeto que podem ser usados na implementação.

Considerando a falta de descrições detalhadas sobre como integrar as engenharias, o Capítulo 5 propõe um método para tal e no Capítulo **??** são apresentadas algumas provas de conceito que servem de evidência e detalham mais a parte de implementação da integração entre as engenharias. Ainda assim, essas evidências podem melhorar ao diversificar as provas de conceito, realizar estudo de caso, simulação, etc. que avancem sobre outros tipos de domínios e sistemas além de DSLs.

A.2.1 Agrupamento das Publicações

A Tabela A.2 apresenta os principais veículos de publicação para MD-SPL. O tema se mantém realmente ativo a nível de *workshops* e conferências especializadas. Isso indica que a comunidade científica está concentrando esforços para debater, evoluir e amadurecer o tema.

Tabela A.2: Principais veículos de publicação

Conferências e *Workshops*	
Sigla	Descrição
SPLC	International Software Product Line Conference
SAM	International Conference on System Analysis and Modeling: theory and practice
ICSE	International Conference on Software engineering
NFPinDSML	International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages
CLOUDCOM	International Conference on Cloud Computing Technology and Science
PLEASE	International Workshop on Product LinE Approaches in Software Engineering
SOFSEM	Conference on Current Trends in Theory and Practice of Computer Science
ICSEA	International Conference on Software Engineering Advances
OOPSLA	ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications
ECMFA	European Conference on Modelling Foundations and Applications
EA	Workshop on Early Aspects
MDPLE	International Workshop on Model-Driven Product Line Engineering
CAiSE	International Conference on Advanced Information Systems
GPCE	Generative Programming and Component Engineering
Journals	
IST	Information and Software Technology
SoSyM	Software and Systems Modeling
TAOSD	Transactions on Aspect-Oriented Software Development
JSS	Journal of Systems and Software

O resultado da busca realizada na ACM é sintetizado na Tabela A.3. O estudo da variabilidade está em alta e se destaca em meio aos demais, possivelmente por ser um ponto que desperta muito interesse, ou seja, um elo de conexão com outras áreas da engenharia de software de forma a investigar e derivar conhecimento sobre o existente.

O resultado da busca realizada na Science Direct é sintetizado na Tabela A.4. O conjunto resultante é de publicações mais recentes e com evidências mais consolidadas. Após uma década de investigação, os resultados estão prontos para serem discutidos com mais rigor.

Tabela A.3: Artigos Selecionados - ACM

Autor	Publicação	Tipo	Ano
Chessman Corrêa *et al.*	An analysis of change operations to achieve consistency in model-driven software product lines	Artigo / SPLC	2011
Victor Guana e Dario Correal	Variability quality evaluation on component-based software product lines	Artigo / SPLC	2011
Chessman Corrêa	Towards automatic consistency preservation for model-driven software product lines	Artigo / SPLC	2011
Javier González-Huerta *et al.*	Non-functional requirements in model-driven software product line engineering	Artigo / NF-PinDSML	2012
João Filho *et al.*	Leveraging variability modeling for multi-dimensional model-driven software product lines	Artigo / PLEASE	2012
Andres Yie *et al.*	Multi-step concern refinement	Artigo / WEA	2008

Tabela A.4: Artigos Selecionados - Science Direct.

Autor	Publicação	Tipo	Ano
Thomas Buchmann *et al.*	MOD2-SCM: A model-driven product line for software configuratio nmanagement systems	Artigo / IST	2013
Andreas Pleuss *et al.*	Model-driven support for product line evolution on feature level	Artigo / JSS	2012
Steven She *et al.*	Efficient synthesis of feature models	Artigo / IST	2014
Deepak Dhungana *et al.*	Structuring the modeling space and supporting evolution in software product line engineering	Artigo / JSS	2010
Wolfgang Heider *et al.*	Simulating evolution in model-based product line engineering	Artigo / IST	2010
Javier Gonzalez-Huerta *et al.*	Validating a model-driven software architecture evaluation and improvement method: A family of experiments	Artigo / IST	2015

O resultado da busca realizada na CiteSeerX é sintetizado na Tabela A.5. O conjunto resultante é mais volumoso e concentra resultados mais emergentes. A investigação da variabilidade e arquitetura mais uma vez se destaca, o que leva a crer que são relevantes para o domínio.

A busca na IEEE retornou apenas uma publicação e a mesma foi incluída no agrupamento, a citar:

- Autor: Xiaorui Zhang *et al.*; Título: Model Comparison to Synthesize a Model-Driven Software Product Line; Tipo: Artigo / SPLC; Ano: 2011.

Tabela A.5: Artigos Selecionados - CiteSeerX.

Autor	Publicação	Tipo	Ano
Marten Sijtema	Managing Variability in Model Transformation for Model-Driven Product Line	Dissertação	2010
Iris Groher e Markus Voelter	Aspect-Oriented Model-Driven SoftwareProduct Line Engineering	Artigo / TAOSD	2009
Gang Deng *et al.*	Evolution in Model-Driven Software Product Line Architecture	Cap. de Livro	2008
Czarnecki *et al.*	Model Driven Software Product Line	Artigo / OOPSLA	2005
Christoph Elsner	Subdomain-oriented Implementation of Model-driven Software Product Lines	Resumo Palestra	2009
Lidia Fuentes *et al.*	Feature-Oriented Model-Driven Software Product Lines: The TENTE approach	Artigo / CAiSE	2009
Hugo Arboleda	Dealing with Constraints during a Feature Configuration Process in a Model-Driven Software Product Line	Artigo / OOPSLA	2007
Stephane Bonnet *et al.*	A Model-Driven Approach for Smart Card Configuration	Artigo / GPCE	2004
Kelly Garces *et al.*	Variability Management in Model-Driven Software Product Line	Artigo / Revista	2007
Sean Quan Lau	Domain Analysis of E-Commerce Systems Using Feature-Based Model Templates	Dissertação	2006
Christoph Elsner	Towards Separation of Concerns in Model Transformation Workflows	Artigo / SPLC	2008
Gan Deng	Addressing Domain Evolution Challenges in Model-Driven Software Product Line Architectures	Artigo / Workshop do MODELS* ed. única	2005
Iris Groher *et al.*	Integrating Model-Driven Development and Software Product Line Engineering	Artigo / Eclipse Modeling Symposium	2007
Nele Andersen *et al.*	Efficient Synthesis of Feature Models	Artigo / SPLC	2012
Nicolas Anquetil *et al.*	Traceability for Model Driven, Software Product Line Engineering	Artigo / ECMDA Traceability Workshop	2008
Hassan Gomaa e Michael Shin	Variability Modeling in Model-Driven Software Product Line Engineering	Artigo / MDPLE	2010

A Tabela A.6 sintetiza as teses e dissertações sobre MD-SPL. Todas foram desenvolvidas em Universidade estrangeiras. O Banco de Teses e Dissertações da CAPES foi consultado, mas não há registro de nenhuma tese ou dissertação sobre MD-SPL que tenha sido defendida no Brasil.

Tabela A.6: Dissertações e Teses na área de MD-SPL.

Autor	Titulo	Tipo	Ano de Defesa
Marten Sijtema	Managing Variability in Model Transformation for Model-Driven Product Line	Dissertação	2010
Sean Quan Lau	Domain Analysis of E-Commerce Systems Using Feature-Based Model Templates	Dissertação	2006
Salvador Trujillo Gonzalez	Feature Oriented Model Driven Product Line	Tese	2007
Abel Gómez Llana	Model Driven Software Product Line Engineering: System Variability View and Process Implications	Tese	2012
Maider Azanza Sesé	Model Driven Product Line Engineering: Core Asset and Process Implications	Tese	2011
Rasha Tawhid	Integrating Performance Analysis in Model Driven Software Product Line Engineering	Tese	2012

A.3 Considerações Finais

A revisão ajudou a identificar critérios importantes para o estudo como relevância, oportunidades de pesquisa e principalmente o mapeamento dos trabalhos que se aproximam dessa tese. A integração entre SPL e MDE é uma área de investigação promissora. Várias tecnologias emergentes necessitam quebrar a barreira do mapeamento denotacional e da arquitetura para avançar. Isso faz com que essas engenharias sejam estudadas e essencialmente aplicadas nos mais diversos domínio com o intuito de minimizar ou resolver problemas dessa natureza.

Apêndice B

A Carbontology

```
<?xml version="1.0"?>


<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<Ontology xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="http://www.semanticweb.org/ontologies/2012/1/
  Ontology1329152069873.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     ontologyIRI="http://www.semanticweb.org/ontologies/2012/1/
  Ontology1329152069873.owl">
    <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
    <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="bibtex" IRI="http://bibtexml.sf.net/"/>
    <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
    <Prefix name="mathml" IRI="http://www.w3.org/1998/Math/MathML"/>
    <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
    <Import>http://www.openmath.org/ontology#</Import>
    <Annotation>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <Literal xml:lang="pt" datatypeIRI="&rdf;PlainLiteral">Uma ontologia
  para modelagem matemÃ¡tica do conhecimento sobre metodologias de
  previsÃ£o do carbono em florestas.

</Literal>
    </Annotation>
    <Annotation>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">An ontology for
         mathematical modeling of the knowledge about forest carbon estimation
         methodologies.</Literal>
    </Annotation>
    <Annotation>
        <AnnotationProperty abbreviatedIRI="owl:versionInfo"/>
        <Literal datatypeIRI="&xsd;string">version 1.0</Literal>
    </Annotation>
    <Declaration>
        <Class IRI="#Aboveground"/>
```

```
48      </Declaration>
49      <Declaration>
50          <Class IRI="#Afforestation"/>
51      </Declaration>
52      <Declaration>
53          <Class IRI="#Alive"/>
54      </Declaration>
55      <Declaration>
56          <Class IRI="#Belowgraund"/>
57      </Declaration>
58      <Declaration>
59          <Class IRI="#Biomass"/>
60      </Declaration>
61      <Declaration>
62          <Class IRI="#Biome"/>
63      </Declaration>
64      <Declaration>
65          <Class IRI="#Boreal"/>
66      </Declaration>
67      <Declaration>
68          <Class IRI="#BottonUp"/>
69      </Declaration>
70      <Declaration>
71          <Class IRI="#CarbonBalance"/>
72      </Declaration>
73      <Declaration>
74          <Class IRI="#CarbonBudget"/>
75      </Declaration>
76      <Declaration>
77          <Class IRI="#CarbonCycle"/>
78      </Declaration>
79      <Declaration>
80          <Class IRI="#CarbonDioxide"/>
81      </Declaration>
82      <Declaration>
83          <Class IRI="#CarbonEstimationMethod"/>
84      </Declaration>
85      <Declaration>
86          <Class IRI="#CarbonFluxe"/>
87      </Declaration>
88      <Declaration>
89          <Class IRI="#CarbonMensuration"/>
90      </Declaration>
91      <Declaration>
92          <Class IRI="#CarbonPool"/>
93      </Declaration>
94      <Declaration>
95          <Class IRI="#CarbonStock"/>
96      </Declaration>
97      <Declaration>
98          <Class IRI="#Coefficient"/>
99      </Declaration>
100     <Declaration>
101         <Class IRI="#Constant"/>
102     </Declaration>
103     <Declaration>
104         <Class IRI="#Dead"/>
105     </Declaration>
106     <Declaration>
107         <Class IRI="#Deforestation"/>
108     </Declaration>
109     <Declaration>
110         <Class IRI="#DirectActivity"/>
111     </Declaration>
112     <Declaration>
113         <Class IRI="#DirectEstimation"/>
114     </Declaration>
```

```
115     <Declaration>
116         <Class IRI="#Disturbance"/>
117     </Declaration>
118     <Declaration>
119         <Class IRI="#Emittion"/>
120     </Declaration>
121     <Declaration>
122         <Class IRI="#Equation"/>
123     </Declaration>
124     <Declaration>
125         <Class IRI="#Forest"/>
126     </Declaration>
127     <Declaration>
128         <Class IRI="#ForestInventory"/>
129     </Declaration>
130     <Declaration>
131         <Class IRI="#GreenhouseGas"/>
132     </Declaration>
133     <Declaration>
134         <Class IRI="#HumanInducedActivity"/>
135     </Declaration>
136     <Declaration>
137         <Class IRI="#IndirectActivity"/>
138     </Declaration>
139     <Declaration>
140         <Class IRI="#IndirectEstimation"/>
141     </Declaration>
142     <Declaration>
143         <Class IRI="#MathematicalConcept"/>
144     </Declaration>
145     <Declaration>
146         <Class IRI="#ModellingApproach"/>
147     </Declaration>
148     <Declaration>
149         <Class IRI="#ObservationArea"/>
150     </Declaration>
151     <Declaration>
152         <Class IRI="#PermanentParcel"/>
153     </Declaration>
154     <Declaration>
155         <Class IRI="#Region"/>
156     </Declaration>
157     <Declaration>
158         <Class IRI="#Removing"/>
159     </Declaration>
160     <Declaration>
161         <Class IRI="#Root"/>
162     </Declaration>
163     <Declaration>
164         <Class IRI="#Sink"/>
165     </Declaration>
166     <Declaration>
167         <Class IRI="#Temperate"/>
168     </Declaration>
169     <Declaration>
170         <Class IRI="#TopDown"/>
171     </Declaration>
172     <Declaration>
173         <Class IRI="#TotalBiomass"/>
174     </Declaration>
175     <Declaration>
176         <Class IRI="#TreeMensuration"/>
177     </Declaration>
178     <Declaration>
179         <Class IRI="#Tropical"/>
180     </Declaration>
181     <Declaration>
```

```
182         <Class IRI="#Variable"/>
183     </Declaration>
184     <Declaration>
185         <ObjectProperty IRI="#calculatedBy"/>
186     </Declaration>
187     <Declaration>
188         <ObjectProperty IRI="#canBeEstimatedBy"/>
189     </Declaration>
190     <Declaration>
191         <ObjectProperty IRI="#hasEquation"/>
192     </Declaration>
193     <Declaration>
194         <ObjectProperty IRI="#isApplicableIn"/>
195     </Declaration>
196     <Declaration>
197         <ObjectProperty IRI="#isEstimatedPer"/>
198     </Declaration>
199     <Declaration>
200         <ObjectProperty IRI="#locatedIn"/>
201     </Declaration>
202     <Declaration>
203         <DataProperty IRI="#hasDescription"/>
204     </Declaration>
205     <Declaration>
206         <DataProperty IRI="#hasDimension"/>
207     </Declaration>
208     <Declaration>
209         <DataProperty IRI="#hasName"/>
210     </Declaration>
211     <Declaration>
212         <DataProperty IRI="#hasResult"/>
213     </Declaration>
214     <Declaration>
215         <DataProperty IRI="#hasValue"/>
216     </Declaration>
217     <Declaration>
218         <DataProperty IRI="#mathExpression"/>
219     </Declaration>
220     <Declaration>
221         <NamedIndividual IRI="#BiomassExpansionFactor"/>
222     </Declaration>
223     <Declaration>
224         <NamedIndividual IRI="#CrownDiameter"/>
225     </Declaration>
226     <Declaration>
227         <NamedIndividual IRI="#DBH"/>
228     </Declaration>
229     <Declaration>
230         <NamedIndividual IRI="#EmissionFactor"/>
231     </Declaration>
232     <Declaration>
233         <NamedIndividual IRI="#FreshBiomass"/>
234     </Declaration>
235     <Declaration>
236         <NamedIndividual IRI="#INPAMethod"/>
237     </Declaration>
238     <Declaration>
239         <NamedIndividual IRI="#Manaus"/>
240     </Declaration>
241     <Declaration>
242         <NamedIndividual IRI="#Negative"/>
243     </Declaration>
244     <Declaration>
245         <NamedIndividual IRI="#Neutral"/>
246     </Declaration>
247     <Declaration>
248         <NamedIndividual IRI="#Positive"/>
```

```
249     </Declaration>
250     <Declaration>
251         <NamedIndividual IRI="#TrunkHeight"/>
252     </Declaration>
253     <Declaration>
254         <NamedIndividual IRI="#brazilianAmazonForest"/>
255     </Declaration>
256     <Declaration>
257         <NamedIndividual IRI="#carbonAboveground"/>
258     </Declaration>
259     <Declaration>
260         <NamedIndividual IRI="#croplands"/>
261     </Declaration>
262     <Declaration>
263         <NamedIndividual IRI="#desert"/>
264     </Declaration>
265     <Declaration>
266         <NamedIndividual IRI="#dryAbovegroundBiomass"/>
267     </Declaration>
268     <Declaration>
269         <NamedIndividual IRI="#dryTotalBiomass"/>
270     </Declaration>
271     <Declaration>
272         <NamedIndividual IRI="#freshAbovegroundBiomass_1"/>
273     </Declaration>
274     <Declaration>
275         <NamedIndividual IRI="#freshAbovegroundBiomass_2"/>
276     </Declaration>
277     <Declaration>
278         <NamedIndividual IRI="#selectTrees"/>
279     </Declaration>
280     <Declaration>
281         <NamedIndividual IRI="#semidesert"/>
282     </Declaration>
283     <Declaration>
284         <NamedIndividual IRI="#temperateGrasslands"/>
285     </Declaration>
286     <Declaration>
287         <NamedIndividual IRI="#totalCarbonEquation"/>
288     </Declaration>
289     <Declaration>
290         <NamedIndividual IRI="#totalFreshBiomass"/>
291     </Declaration>
292     <Declaration>
293         <NamedIndividual IRI="#tropicalSavannas"/>
294     </Declaration>
295     <Declaration>
296         <NamedIndividual IRI="#tundra"/>
297     </Declaration>
298     <Declaration>
299         <NamedIndividual IRI="#wetlands"/>
300     </Declaration>
301     <Declaration>
302         <AnnotationProperty IRI="#source"/>
303     </Declaration>
304     <EquivalentClasses>
305         <Class IRI="#Biomass"/>
306         <ObjectSomeValuesFrom>
307             <ObjectProperty IRI="#isEstimatedPer"/>
308             <Class IRI="#ObservationArea"/>
309         </ObjectSomeValuesFrom>
310     </EquivalentClasses>
311     <EquivalentClasses>
312         <Class IRI="#CarbonEstimationMethod"/>
313         <ObjectUnionOf>
314             <Class IRI="#DirectEstimation"/>
315             <Class IRI="#IndirectEstimation"/>
```

```
316             </ObjectUnionOf>
317         </EquivalentClasses>
318         <SubClassOf>
319             <Class IRI="#Aboveground"/>
320             <Class IRI="#Biomass"/>
321         </SubClassOf>
322         <SubClassOf>
323             <Class IRI="#Afforestation"/>
324             <Class IRI="#DirectActivity"/>
325         </SubClassOf>
326         <SubClassOf>
327             <Class IRI="#Alive"/>
328             <Class IRI="#Aboveground"/>
329         </SubClassOf>
330         <SubClassOf>
331             <Class IRI="#Belowgraund"/>
332             <Class IRI="#Biomass"/>
333         </SubClassOf>
334         <SubClassOf>
335             <Class IRI="#Biomass"/>
336             <Class IRI="#TreeMensuration"/>
337         </SubClassOf>
338         <SubClassOf>
339             <Class IRI="#Boreal"/>
340             <Class IRI="#Forest"/>
341         </SubClassOf>
342         <SubClassOf>
343             <Class IRI="#BottonUp"/>
344             <Class IRI="#ModellingApproach"/>
345         </SubClassOf>
346         <SubClassOf>
347             <Class IRI="#CarbonBalance"/>
348             <Class IRI="#CarbonMensuration"/>
349         </SubClassOf>
350         <SubClassOf>
351             <Class IRI="#CarbonBudget"/>
352             <Class IRI="#CarbonMensuration"/>
353         </SubClassOf>
354         <SubClassOf>
355             <Class IRI="#CarbonCycle"/>
356             <Class IRI="#CarbonMensuration"/>
357         </SubClassOf>
358         <SubClassOf>
359             <Class IRI="#CarbonDioxide"/>
360             <Class IRI="#GreenhouseGas"/>
361         </SubClassOf>
362         <SubClassOf>
363             <Class IRI="#CarbonEstimationMethod"/>
364             <DataSomeValuesFrom>
365                 <DataProperty IRI="#hasName"/>
366                 <Datatype abbreviatedIRI="xsd:string"/>
367             </DataSomeValuesFrom>
368         </SubClassOf>
369         <SubClassOf>
370             <Class IRI="#CarbonFluxe"/>
371             <Class IRI="#CarbonMensuration"/>
372         </SubClassOf>
373         <SubClassOf>
374             <Class IRI="#CarbonPool"/>
375             <Class IRI="#CarbonMensuration"/>
376         </SubClassOf>
377         <SubClassOf>
378             <Class IRI="#CarbonStock"/>
379             <Class IRI="#CarbonMensuration"/>
380         </SubClassOf>
381         <SubClassOf>
382             <Class IRI="#Coefficient"/>
```

```
383         <Class IRI="#MathematicalConcept"/>
384     </SubClassOf>
385     <SubClassOf>
386         <Class IRI="#Coefficient"/>
387         <DataSomeValuesFrom>
388             <DataProperty IRI="#hasName"/>
389             <Datatype abbreviatedIRI="xsd:string"/>
390         </DataSomeValuesFrom>
391     </SubClassOf>
392     <SubClassOf>
393         <Class IRI="#Coefficient"/>
394         <DataSomeValuesFrom>
395             <DataProperty IRI="#hasValue"/>
396             <Datatype abbreviatedIRI="xsd:string"/>
397         </DataSomeValuesFrom>
398     </SubClassOf>
399     <SubClassOf>
400         <Class IRI="#Constant"/>
401         <Class IRI="#MathematicalConcept"/>
402     </SubClassOf>
403     <SubClassOf>
404         <Class IRI="#Dead"/>
405         <Class IRI="#Aboveground"/>
406     </SubClassOf>
407     <SubClassOf>
408         <Class IRI="#Deforestation"/>
409         <Class IRI="#DirectActivity"/>
410     </SubClassOf>
411     <SubClassOf>
412         <Class IRI="#DirectActivity"/>
413         <Class IRI="#HumanInducedActivity"/>
414     </SubClassOf>
415     <SubClassOf>
416         <Class IRI="#DirectEstimation"/>
417         <Class IRI="#CarbonEstimationMethod"/>
418     </SubClassOf>
419     <SubClassOf>
420         <Class IRI="#Disturbance"/>
421         <Class IRI="#CarbonMensuration"/>
422     </SubClassOf>
423     <SubClassOf>
424         <Class IRI="#Emittion"/>
425         <Class IRI="#CarbonMensuration"/>
426     </SubClassOf>
427     <SubClassOf>
428         <Class IRI="#Equation"/>
429         <Class IRI="#MathematicalConcept"/>
430     </SubClassOf>
431     <SubClassOf>
432         <Class IRI="#Forest"/>
433         <Class IRI="#Biome"/>
434     </SubClassOf>
435     <SubClassOf>
436         <Class IRI="#Forest"/>
437         <ObjectSomeValuesFrom>
438             <ObjectProperty IRI="#locatedIn"/>
439             <Class IRI="#Region"/>
440         </ObjectSomeValuesFrom>
441     </SubClassOf>
442     <SubClassOf>
443         <Class IRI="#IndirectActivity"/>
444         <Class IRI="#HumanInducedActivity"/>
445     </SubClassOf>
446     <SubClassOf>
447         <Class IRI="#IndirectEstimation"/>
448         <Class IRI="#CarbonEstimationMethod"/>
449     </SubClassOf>
```

```
450     <SubClassOf>
451         <Class IRI="#IndirectEstimation"/>
452         <ObjectSomeValuesFrom>
453             <ObjectProperty IRI="#hasEquation"/>
454             <Class IRI="#Equation"/>
455         </ObjectSomeValuesFrom>
456     </SubClassOf>
457     <SubClassOf>
458         <Class IRI="#PermanentParcel"/>
459         <Class IRI="#ObservationArea"/>
460     </SubClassOf>
461     <SubClassOf>
462         <Class IRI="#PermanentParcel"/>
463         <ObjectSomeValuesFrom>
464             <ObjectProperty IRI="#locatedIn"/>
465             <Class IRI="#Forest"/>
466         </ObjectSomeValuesFrom>
467     </SubClassOf>
468     <SubClassOf>
469         <Class IRI="#Removing"/>
470         <Class IRI="#CarbonMensuration"/>
471     </SubClassOf>
472     <SubClassOf>
473         <Class IRI="#Root"/>
474         <Class IRI="#Belowgraund"/>
475     </SubClassOf>
476     <SubClassOf>
477         <Class IRI="#Sink"/>
478         <Class IRI="#CarbonMensuration"/>
479     </SubClassOf>
480     <SubClassOf>
481         <Class IRI="#Temperate"/>
482         <Class IRI="#Forest"/>
483     </SubClassOf>
484     <SubClassOf>
485         <Class IRI="#TopDown"/>
486         <Class IRI="#ModellingApproach"/>
487     </SubClassOf>
488     <SubClassOf>
489         <Class IRI="#TotalBiomass"/>
490         <Class IRI="#TreeMensuration"/>
491     </SubClassOf>
492     <SubClassOf>
493         <Class IRI="#TotalBiomass"/>
494         <ObjectSomeValuesFrom>
495             <ObjectProperty IRI="#canBeEstimatedBy"/>
496             <Class IRI="#CarbonEstimationMethod"/>
497         </ObjectSomeValuesFrom>
498     </SubClassOf>
499     <SubClassOf>
500         <Class IRI="#Tropical"/>
501         <Class IRI="#Forest"/>
502     </SubClassOf>
503     <SubClassOf>
504         <Class IRI="#Variable"/>
505         <Class IRI="#MathematicalConcept"/>
506     </SubClassOf>
507     <SubClassOf>
508         <Class IRI="#Variable"/>
509         <DataSomeValuesFrom>
510             <DataProperty IRI="#hasDimension"/>
511             <Datatype abbreviatedIRI="xsd:string"/>
512         </DataSomeValuesFrom>
513     </SubClassOf>
514     <SubClassOf>
515         <Class IRI="#Variable"/>
516         <DataSomeValuesFrom>
```

```
517             <DataProperty IRI="#hasName"/>
518             <Datatype abbreviatedIRI="xsd:string"/>
519         </DataSomeValuesFrom>
520     </SubClassOf>
521     <DisjointClasses>
522         <Class IRI="#Aboveground"/>
523         <Class IRI="#Belowgraund"/>
524     </DisjointClasses>
525     <DisjointClasses>
526         <Class IRI="#DirectEstimation"/>
527         <Class IRI="#IndirectEstimation"/>
528     </DisjointClasses>
529     <ClassAssertion>
530         <Class IRI="#Coefficient"/>
531         <NamedIndividual IRI="#BiomassExpansionFactor"/>
532     </ClassAssertion>
533     <ClassAssertion>
534         <Class IRI="#Variable"/>
535         <NamedIndividual IRI="#CrownDiameter"/>
536     </ClassAssertion>
537     <ClassAssertion>
538         <Class IRI="#Variable"/>
539         <NamedIndividual IRI="#DBH"/>
540     </ClassAssertion>
541     <ClassAssertion>
542         <Class IRI="#Coefficient"/>
543         <NamedIndividual IRI="#EmissionFactor"/>
544     </ClassAssertion>
545     <ClassAssertion>
546         <Class IRI="#Alive"/>
547         <NamedIndividual IRI="#FreshBiomass"/>
548     </ClassAssertion>
549     <ClassAssertion>
550         <Class IRI="#IndirectEstimation"/>
551         <NamedIndividual IRI="#INPAMethod"/>
552     </ClassAssertion>
553     <ClassAssertion>
554         <Class IRI="#Region"/>
555         <NamedIndividual IRI="#Manaus"/>
556     </ClassAssertion>
557     <ClassAssertion>
558         <Class IRI="#CarbonBalance"/>
559         <NamedIndividual IRI="#Negative"/>
560     </ClassAssertion>
561     <ClassAssertion>
562         <Class IRI="#CarbonBalance"/>
563         <NamedIndividual IRI="#Neutral"/>
564     </ClassAssertion>
565     <ClassAssertion>
566         <Class IRI="#CarbonBalance"/>
567         <NamedIndividual IRI="#Positive"/>
568     </ClassAssertion>
569     <ClassAssertion>
570         <Class IRI="#Variable"/>
571         <NamedIndividual IRI="#TrunkHeight"/>
572     </ClassAssertion>
573     <ClassAssertion>
574         <Class IRI="#Tropical"/>
575         <NamedIndividual IRI="#brazilianAmazonForest"/>
576     </ClassAssertion>
577     <ClassAssertion>
578         <Class IRI="#Equation"/>
579         <NamedIndividual IRI="#carbonAboveground"/>
580     </ClassAssertion>
581     <ClassAssertion>
582         <Class IRI="#Biome"/>
583         <NamedIndividual IRI="#croplands"/>
```

```
584     </ClassAssertion>
585     <ClassAssertion>
586         <Class IRI="#Biome"/>
587         <NamedIndividual IRI="#desert"/>
588     </ClassAssertion>
589     <ClassAssertion>
590         <Class IRI="#Equation"/>
591         <NamedIndividual IRI="#dryAbovegroundBiomass"/>
592     </ClassAssertion>
593     <ClassAssertion>
594         <Class IRI="#Equation"/>
595         <NamedIndividual IRI="#dryTotalBiomass"/>
596     </ClassAssertion>
597     <ClassAssertion>
598         <Class IRI="#Equation"/>
599         <NamedIndividual IRI="#freshAbovegroundBiomass_1"/>
600     </ClassAssertion>
601     <ClassAssertion>
602         <Class IRI="#Equation"/>
603         <NamedIndividual IRI="#freshAbovegroundBiomass_2"/>
604     </ClassAssertion>
605     <ClassAssertion>
606         <Class IRI="#Equation"/>
607         <NamedIndividual IRI="#selectTrees"/>
608     </ClassAssertion>
609     <ClassAssertion>
610         <DataMinCardinality cardinality="10">
611             <DataProperty IRI="#hasValue"/>
612             <Datatype abbreviatedIRI="xsd:float"/>
613         </DataMinCardinality>
614         <NamedIndividual IRI="#selectTrees"/>
615     </ClassAssertion>
616     <ClassAssertion>
617         <Class IRI="#Biome"/>
618         <NamedIndividual IRI="#semidesert"/>
619     </ClassAssertion>
620     <ClassAssertion>
621         <Class IRI="#Biome"/>
622         <NamedIndividual IRI="#temperateGrasslands"/>
623     </ClassAssertion>
624     <ClassAssertion>
625         <Class IRI="#Equation"/>
626         <NamedIndividual IRI="#totalCarbonEquation"/>
627     </ClassAssertion>
628     <ClassAssertion>
629         <Class IRI="#Equation"/>
630         <NamedIndividual IRI="#totalFreshBiomass"/>
631     </ClassAssertion>
632     <ClassAssertion>
633         <Class IRI="#Biome"/>
634         <NamedIndividual IRI="#tropicalSavannas"/>
635     </ClassAssertion>
636     <ClassAssertion>
637         <Class IRI="#Biome"/>
638         <NamedIndividual IRI="#tundra"/>
639     </ClassAssertion>
640     <ClassAssertion>
641         <Class IRI="#Biome"/>
642         <NamedIndividual IRI="#wetlands"/>
643     </ClassAssertion>
644     <ObjectPropertyAssertion>
645         <ObjectProperty IRI="#hasEquation"/>
646         <NamedIndividual IRI="#INPAMethod"/>
647         <NamedIndividual IRI="#dryAbovegroundBiomass"/>
648     </ObjectPropertyAssertion>
649     <ObjectPropertyAssertion>
650         <ObjectProperty IRI="#hasEquation"/>
```

```
651         <NamedIndividual IRI="#INPAMethod"/>
652         <NamedIndividual IRI="#selectTrees"/>
653     </ObjectPropertyAssertion>
654     <ObjectPropertyAssertion>
655         <ObjectProperty IRI="#hasEquation"/>
656         <NamedIndividual IRI="#INPAMethod"/>
657         <NamedIndividual IRI="#dryTotalBiomass"/>
658     </ObjectPropertyAssertion>
659     <ObjectPropertyAssertion>
660         <ObjectProperty IRI="#hasEquation"/>
661         <NamedIndividual IRI="#INPAMethod"/>
662         <NamedIndividual IRI="#totalFreshBiomass"/>
663     </ObjectPropertyAssertion>
664     <ObjectPropertyAssertion>
665         <ObjectProperty IRI="#hasEquation"/>
666         <NamedIndividual IRI="#INPAMethod"/>
667         <NamedIndividual IRI="#freshAbovegroundBiomass_1"/>
668     </ObjectPropertyAssertion>
669     <ObjectPropertyAssertion>
670         <ObjectProperty IRI="#hasEquation"/>
671         <NamedIndividual IRI="#INPAMethod"/>
672         <NamedIndividual IRI="#freshAbovegroundBiomass_2"/>
673     </ObjectPropertyAssertion>
674     <ObjectPropertyAssertion>
675         <ObjectProperty IRI="#hasEquation"/>
676         <NamedIndividual IRI="#INPAMethod"/>
677         <NamedIndividual IRI="#totalCarbonEquation"/>
678     </ObjectPropertyAssertion>
679     <ObjectPropertyAssertion>
680         <ObjectProperty IRI="#isApplicableIn"/>
681         <NamedIndividual IRI="#carbonAboveground"/>
682         <NamedIndividual IRI="#brazilianAmazonForest"/>
683     </ObjectPropertyAssertion>
684     <ObjectPropertyAssertion>
685         <ObjectProperty IRI="#isApplicableIn"/>
686         <NamedIndividual IRI="#dryAbovegroundBiomass"/>
687         <NamedIndividual IRI="#brazilianAmazonForest"/>
688     </ObjectPropertyAssertion>
689     <ObjectPropertyAssertion>
690         <ObjectProperty IRI="#isApplicableIn"/>
691         <NamedIndividual IRI="#dryTotalBiomass"/>
692         <NamedIndividual IRI="#brazilianAmazonForest"/>
693     </ObjectPropertyAssertion>
694     <ObjectPropertyAssertion>
695         <ObjectProperty IRI="#isApplicableIn"/>
696         <NamedIndividual IRI="#freshAbovegroundBiomass_1"/>
697         <NamedIndividual IRI="#Manaus"/>
698     </ObjectPropertyAssertion>
699     <ObjectPropertyAssertion>
700         <ObjectProperty IRI="#isApplicableIn"/>
701         <NamedIndividual IRI="#freshAbovegroundBiomass_2"/>
702         <NamedIndividual IRI="#brazilianAmazonForest"/>
703     </ObjectPropertyAssertion>
704     <ObjectPropertyAssertion>
705         <ObjectProperty IRI="#isApplicableIn"/>
706         <NamedIndividual IRI="#totalCarbonEquation"/>
707         <NamedIndividual IRI="#brazilianAmazonForest"/>
708     </ObjectPropertyAssertion>
709     <ObjectPropertyAssertion>
710         <ObjectProperty IRI="#isApplicableIn"/>
711         <NamedIndividual IRI="#totalFreshBiomass"/>
712         <NamedIndividual IRI="#brazilianAmazonForest"/>
713     </ObjectPropertyAssertion>
714     <DataPropertyAssertion>
715         <DataProperty IRI="#hasName"/>
716         <NamedIndividual IRI="#CrownDiameter"/>
717         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">
```

```
718             Crown Diameter</Literal>
719         </DataPropertyAssertion>
720         <DataPropertyAssertion>
721             <DataProperty IRI="#hasDimension"/>
722             <NamedIndividual IRI="#FreshBiomass"/>
723             <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">kg</Literal>
724         </DataPropertyAssertion>
725         <DataPropertyAssertion>
726             <DataProperty IRI="#hasName"/>
727             <NamedIndividual IRI="#INPAMethod"/>
728             <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">
729             INPA Method</Literal>
730         </DataPropertyAssertion>
731         <DataPropertyAssertion>
732             <DataProperty IRI="#hasName"/>
733             <NamedIndividual IRI="#TrunkHeight"/>
734             <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">
735             Trunk Height</Literal>
736         </DataPropertyAssertion>
737         <DataPropertyAssertion>
738             <DataProperty IRI="#mathExpression"/>
739             <NamedIndividual IRI="#carbonAboveground"/>
740             <Literal datatypeIRI="&rdf;XMLLiteral">&lt;math xmlns=&quot;
741             http://www.w3.org/1998/Math/MathML&quot;&gt;
742 &lt;mrow&gt;
743   &lt;msub&gt;
744     &lt;mi&gt;C&lt;/mi&gt;
745     &lt;mrow&gt;
746       &lt;mi&gt;a&lt;/mi&gt;
747       &lt;mi&gt;b&lt;/mi&gt;
748       &lt;mi&gt;g&lt;/mi&gt;
749     &lt;/mrow&gt;
750   &lt;/msub&gt;
751   &lt;mo&gt;=&lt;/mo&gt;
752   &lt;mrow&gt;
753     &lt;mo maxsize=&quot;1.00em&quot;&gt;(&lt;/mo&gt;
754     &lt;mi&gt;D&lt;/mi&gt;
755     &lt;msub&gt;
756       &lt;mi&gt;B&lt;/mi&gt;
757       &lt;mrow&gt;
758         &lt;mi&gt;a&lt;/mi&gt;
759         &lt;mi&gt;b&lt;/mi&gt;
760         &lt;mi&gt;g&lt;/mi&gt;
761       &lt;/mrow&gt;
762     &lt;/msub&gt;
763     &lt;mo maxsize=&quot;1.00em&quot;&gt;)&lt;/mo&gt;
764   &lt;/mrow&gt;
765   &lt;mo&gt;*&lt;/mo&gt;
766   &lt;mn&gt;0,485&lt;/mn&gt;
767 &lt;/mrow&gt;
768 &lt;/math&gt;</Literal>
769         </DataPropertyAssertion>
770         <DataPropertyAssertion>
771             <DataProperty IRI="#mathExpression"/>
772             <NamedIndividual IRI="#dryAbovegroundBiomass"/>
773             <Literal datatypeIRI="&rdf;XMLLiteral">&lt;math xmlns=&quot;
774             http://www.w3.org/1998/Math/MathML&quot;&gt;
775 &lt;mrow&gt;
776   &lt;mi&gt;D&lt;/mi&gt;
777   &lt;msub&gt;
778     &lt;mi&gt;B&lt;/mi&gt;
779     &lt;mrow&gt;
780       &lt;mi&gt;a&lt;/mi&gt;
781       &lt;mi&gt;b&lt;/mi&gt;
782       &lt;mi&gt;g&lt;/mi&gt;
783     &lt;/mrow&gt;
784   &lt;/msub&gt;
```

```
785    &lt;mo&gt;=&lt;/mo&gt;
786    &lt;mrow&gt;
787      &lt;mo maxsize=&quot;1.00em&quot;&gt;(&lt;/mo&gt;
788      &lt;mi&gt;F&lt;/mi&gt;
789      &lt;msub&gt;
790        &lt;mi&gt;W&lt;/mi&gt;
791        &lt;mrow&gt;
792          &lt;mi&gt;a&lt;/mi&gt;
793          &lt;mi&gt;b&lt;/mi&gt;
794          &lt;mi&gt;g&lt;/mi&gt;
795        &lt;/mrow&gt;
796      &lt;/msub&gt;
797      &lt;mo maxsize=&quot;1.00em&quot;&gt;)&lt;/mo&gt;
798    &lt;/mrow&gt;
799    &lt;mo&gt;*&lt;/mo&gt;
800    &lt;mn&gt;0,592&lt;/mn&gt;
801  &lt;/mrow&gt;
802  &lt;/math&gt;</Literal>
803      </DataPropertyAssertion>
804      <DataPropertyAssertion>
805          <DataProperty IRI="#mathExpression"/>
806          <NamedIndividual IRI="#dryTotalBiomass"/>
807          <Literal datatypeIRI="&rdf;XMLLiteral">&lt;math xmlns=&quot;
808          http://www.w3.org/1998/Math/MathML&quot;&gt;
809  &lt;mrow&gt;
810    &lt;mi&gt;D&lt;/mi&gt;
811    &lt;msub&gt;
812      &lt;mi&gt;B&lt;/mi&gt;
813      &lt;mrow&gt;
814        &lt;mi&gt;t&lt;/mi&gt;
815        &lt;mi&gt;o&lt;/mi&gt;
816        &lt;mi&gt;t&lt;/mi&gt;
817      &lt;/mrow&gt;
818    &lt;/msub&gt;
819    &lt;mo&gt;=&lt;/mo&gt;
820    &lt;mrow&gt;
821      &lt;mo maxsize=&quot;1.00em&quot;&gt;(&lt;/mo&gt;
822      &lt;mi&gt;F&lt;/mi&gt;
823      &lt;msub&gt;
824        &lt;mi&gt;W&lt;/mi&gt;
825        &lt;mrow&gt;
826          &lt;mi&gt;t&lt;/mi&gt;
827          &lt;mi&gt;o&lt;/mi&gt;
828          &lt;mi&gt;t&lt;/mi&gt;
829        &lt;/mrow&gt;
830      &lt;/msub&gt;
831      &lt;mo maxsize=&quot;1.00em&quot;&gt;)&lt;/mo&gt;
832    &lt;/mrow&gt;
833    &lt;mo&gt;*&lt;/mo&gt;
834    &lt;mn&gt;0,584&lt;/mn&gt;
835  &lt;/mrow&gt;
836  &lt;/math&gt;</Literal>
837      </DataPropertyAssertion>
838      <DataPropertyAssertion>
839          <DataProperty IRI="#mathExpression"/>
840          <NamedIndividual IRI="#freshAbovegroundBiomass_1"/>
841          <Literal datatypeIRI="&rdf;XMLLiteral">&lt;math xmlns=&quot;
842          http://www.w3.org/1998/Math/MathML&quot;&gt;
843  &lt;mrow&gt;
844    &lt;mi&gt;F&lt;/mi&gt;
845    &lt;msub&gt;
846      &lt;mi&gt;W&lt;/mi&gt;
847      &lt;mrow&gt;
848        &lt;mi&gt;a&lt;/mi&gt;
849        &lt;mi&gt;b&lt;/mi&gt;
850        &lt;mi&gt;g&lt;/mi&gt;
851      &lt;/mrow&gt;
```

```
852    &lt;/msub&gt;
853    &lt;mo&gt;=&lt;/mo&gt;
854    &lt;mn&gt;2,2737&lt;/mn&gt;
855    &lt;mo&gt;*&lt;/mo&gt;
856    &lt;msup&gt;
857       &lt;mi&gt;D&lt;/mi&gt;
858       &lt;mn&gt;1,9156&lt;/mn&gt;
859    &lt;/msup&gt;
860 &lt;/mrow&gt;
861 &lt;/math&gt;</Literal>
862       </DataPropertyAssertion>
863       <DataPropertyAssertion>
864            <DataProperty IRI="#mathExpression"/>
865            <NamedIndividual IRI="#freshAbovegroundBiomass_2"/>
866            <Literal datatypeIRI="&rdf;XMLLiteral">&lt;math xmlns=&quot;
867             http://www.w3.org/1998/Math/MathML&quot;&gt;
868 &lt;mrow&gt;
869    &lt;mi&gt;F&lt;/mi&gt;
870    &lt;msub&gt;
871       &lt;mi&gt;W&lt;/mi&gt;
872       &lt;mrow&gt;
873          &lt;mi&gt;a&lt;/mi&gt;
874          &lt;mi&gt;b&lt;/mi&gt;
875          &lt;mi&gt;g&lt;/mi&gt;
876       &lt;/mrow&gt;
877    &lt;/msub&gt;
878    &lt;mo&gt;=&lt;/mo&gt;
879    &lt;mn&gt;0,0039&lt;/mn&gt;
880    &lt;mo&gt;*&lt;/mo&gt;
881    &lt;msup&gt;
882       &lt;mi&gt;D&lt;/mi&gt;
883       &lt;mn&gt;1,5268&lt;/mn&gt;
884    &lt;/msup&gt;
885    &lt;mo&gt;*&lt;/mo&gt;
886    &lt;msup&gt;
887       &lt;mi&gt;H&lt;/mi&gt;
888       &lt;mn&gt;2,2973&lt;/mn&gt;
889    &lt;/msup&gt;
890 &lt;/mrow&gt;
891 &lt;/math&gt;</Literal>
892       </DataPropertyAssertion>
893       <DataPropertyAssertion>
894            <DataProperty IRI="#mathExpression"/>
895            <NamedIndividual IRI="#totalCarbonEquation"/>
896            <Literal datatypeIRI="&rdf;XMLLiteral">&lt;math xmlns=&quot;
897             http://www.w3.org/1998/Math/MathML&quot;&gt;
898 &lt;mrow&gt;
899    &lt;msub&gt;
900       &lt;mi&gt;C&lt;/mi&gt;
901       &lt;mrow&gt;
902          &lt;mi&gt;t&lt;/mi&gt;
903          &lt;mi&gt;o&lt;/mi&gt;
904          &lt;mi&gt;t&lt;/mi&gt;
905       &lt;/mrow&gt;
906    &lt;/msub&gt;
907    &lt;mo&gt;=&lt;/mo&gt;
908    &lt;mrow&gt;
909       &lt;mo maxsize=&quot;1.00em&quot;&gt;(&lt;/mo&gt;
910       &lt;mi&gt;D&lt;/mi&gt;
911       &lt;msub&gt;
912          &lt;mi&gt;B&lt;/mi&gt;
913          &lt;mrow&gt;
914             &lt;mi&gt;t&lt;/mi&gt;
915             &lt;mi&gt;o&lt;/mi&gt;
916             &lt;mi&gt;t&lt;/mi&gt;
917          &lt;/mrow&gt;
918       &lt;/msub&gt;
```

```
919      &lt;mo maxsize=&quot;1.00em&quot;&gt;)&lt;/mo&gt;
920    &lt;/mrow&gt;
921    &lt;mo&gt;*&lt;/mo&gt;
922    &lt;mn&gt;0,485&lt;/mn&gt;
923 &lt;/mrow&gt;
924 &lt;/math&gt;</Literal>
925      </DataPropertyAssertion>
926      <DataPropertyAssertion>
927          <DataProperty IRI="#mathExpression"/>
928          <NamedIndividual IRI="#totalFreshBiomass"/>
929          <Literal datatypeIRI="&rdf;XMLLiteral">&lt;math xmlns=&quot;
930          http://www.w3.org/1998/Math/MathML&quot;&gt;
931 &lt;mrow&gt;
932    &lt;msub&gt;
933      &lt;mi&gt;C&lt;/mi&gt;
934      &lt;mrow&gt;
935        &lt;mi&gt;t&lt;/mi&gt;
936        &lt;mi&gt;o&lt;/mi&gt;
937        &lt;mi&gt;t&lt;/mi&gt;
938      &lt;/mrow&gt;
939    &lt;/msub&gt;
940    &lt;mo&gt;=&lt;/mo&gt;
941    &lt;mrow&gt;
942      &lt;mo maxsize=&quot;1.00em&quot;&gt;(&lt;/mo&gt;
943      &lt;mi&gt;D&lt;/mi&gt;
944      &lt;msub&gt;
945        &lt;mi&gt;B&lt;/mi&gt;
946        &lt;mrow&gt;
947          &lt;mi&gt;t&lt;/mi&gt;
948          &lt;mi&gt;o&lt;/mi&gt;
949          &lt;mi&gt;t&lt;/mi&gt;
950        &lt;/mrow&gt;
951      &lt;/msub&gt;
952      &lt;mo maxsize=&quot;1.00em&quot;&gt;)&lt;/mo&gt;
953    &lt;/mrow&gt;
954    &lt;mo&gt;*&lt;/mo&gt;
955    &lt;mn&gt;0,485&lt;/mn&gt;
956 &lt;/mrow&gt;
957 &lt;/math&gt;</Literal>
958      </DataPropertyAssertion>
959      <ObjectPropertyDomain>
960          <ObjectProperty IRI="#isEstimatedPer"/>
961          <Class IRI="#Biomass"/>
962      </ObjectPropertyDomain>
963      <ObjectPropertyRange>
964          <ObjectProperty IRI="#isEstimatedPer"/>
965          <Class IRI="#ObservationArea"/>
966      </ObjectPropertyRange>
967      <SubDataPropertyOf>
968          <DataProperty IRI="#hasDimension"/>
969          <DataProperty abbreviatedIRI="owl:topDataProperty"/>
970      </SubDataPropertyOf>
971      <DataPropertyDomain>
972          <DataProperty IRI="#hasName"/>
973          <DataSomeValuesFrom>
974              <DataProperty IRI="#hasName"/>
975              <Datatype abbreviatedIRI="xsd:string"/>
976          </DataSomeValuesFrom>
977      </DataPropertyDomain>
978      <AnnotationAssertion>
979          <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
980          <IRI>#Afforestation</IRI>
981          <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">The direct
982          human-induced conversion of land that has not been forested for a
983          period of at least 50 years to forested land through planting,
984          seeding and/or the human-induced promotion of natural seed
985          sources.</Literal>
```

```
986     </AnnotationAssertion>
987     <AnnotationAssertion>
988         <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
989         <IRI>#Alive</IRI>
990         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Standing alive
991         aboveground biomass or vertical aboveground biomass is composed of
992         trees and shrubs (not considering the roots).
993 </Literal>
994     </AnnotationAssertion>
995     <AnnotationAssertion>
996         <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
997         <IRI>#Belowgraund</IRI>
998         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Belowground
999         biomass is composed of roots.
1000 </Literal>
1001     </AnnotationAssertion>
1002     <AnnotationAssertion>
1003         <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
1004         <IRI>#Biomass</IRI>
1005         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Biomass is
1006         defined as the quantity, expressed in mass units, of the vegetal
1007         material content per unit area in a forest. In general, the estimated
1008         biomass components are: vertical aboveground biomass or standing
1009         alive aboveground biomass, composed of trees and shrubs (not
1010         considering the roots), dead aboveground biomass, composed of litter
1011         and fallen trunks, and the belowground biomass, composed of roots.
1012         The sum of all considered components provides the total biomass.
1013
1014 </Literal>
1015     </AnnotationAssertion>
1016     <AnnotationAssertion>
1017         <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1018         <IRI>#BiomassExpansionFactor</IRI>
1019         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">A multiplication
1020         factor that expands growing stock, or commercial round-wood harvest volume,
1021         or growing stock volume increment data, to account for non-merchantable
1022         biomass components such as branches, foliage, and non-commercial trees.
1023         </Literal>
1024     </AnnotationAssertion>
1025     <AnnotationAssertion>
1026         <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1027         <IRI>#BiomassExpansionFactor</IRI>
1028         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Biomass Expansion
1029         Factor</Literal>
1030     </AnnotationAssertion>
1031     <AnnotationAssertion>
1032         <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
1033         <IRI>#Biome</IRI>
1034         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Source: IPCC
1035         anexe I</Literal>
1036     </AnnotationAssertion>
1037     <AnnotationAssertion>
1038         <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1039         <IRI>#Biome</IRI>
1040         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">A biome is a
1041         major and distinct regional element of the biosphere, typically
1042         consisting of several ecosystems (e.g. forests, rivers, pounds,
1043         swamps within a region). Biomes are characterised by typical
1044         communities of plants and animals.</Literal>
1045     </AnnotationAssertion>
1046     <AnnotationAssertion>
1047         <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1048         <IRI>#Biome</IRI>
1049         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Biome</Literal>
1050     </AnnotationAssertion>
1051     <AnnotationAssertion>
1052         <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
```

```
1053        <IRI>#BottonUp</IRI>
1054        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">A modelling
1055        approach which starts from processes at a detailed scale (i.e., plot/
1056        stand/ecosystems scale) and provides results at a larger, aggregated
1057        scale (regional/national/continental/global).</Literal>
1058    </AnnotationAssertion>
1059    <AnnotationAssertion>
1060        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1061        <IRI>#BottonUp</IRI>
1062        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Botton-up
1063        </Literal>
1064    </AnnotationAssertion>
1065    <AnnotationAssertion>
1066        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1067        <IRI>#CarbonBalance</IRI>
1068        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Carbon Balance
1069        </Literal>
1070    </AnnotationAssertion>
1071    <AnnotationAssertion>
1072        <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1073        <IRI>#CarbonBudget</IRI>
1074        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">The balance of the
1075        exchanges of carbon between carbon pools or between one specific loops
1076        (e.g., atmosphere - biosphere) of the carbon cycle. The examination of
1077        the budget of a pool or reservoir will provide information whether it
1078        is acting as a source or a sink.</Literal>
1079    </AnnotationAssertion>
1080    <AnnotationAssertion>
1081        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1082        <IRI>#CarbonBudget</IRI>
1083        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Carbon Budget
1084        </Literal>
1085    </AnnotationAssertion>
1086    <AnnotationAssertion>
1087        <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1088        <IRI>#CarbonCycle</IRI>
1089        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">All parts
1090        (pools) and fluxes of carbon; usually thought of as a series of
1091        the four main pools of carbon interconnected by pathways of exchange.
1092        The four pools are atmosphere, biosphere, oceans and sediments.
1093        Carbon exchanges from pool to pool by chemical, physical and biological
1094        processes.</Literal>
1095    </AnnotationAssertion>
1096    <AnnotationAssertion>
1097        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1098        <IRI>#CarbonCycle</IRI>
1099        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Carbon Cycle
1100        </Literal>
1101    </AnnotationAssertion>
1102    <AnnotationAssertion>
1103        <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1104        <IRI>#CarbonDioxide</IRI>
1105        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">CO2</Literal>
1106    </AnnotationAssertion>
1107    <AnnotationAssertion>
1108        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1109        <IRI>#CarbonDioxide</IRI>
1110        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Carbon Dioxide
1111        </Literal>
1112    </AnnotationAssertion>
1113    <AnnotationAssertion>
1114        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1115        <IRI>#CarbonEstimationMethod</IRI>
1116        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Carbon
1117        Estimation Method</Literal>
1118    </AnnotationAssertion>
1119    <AnnotationAssertion>
```

```
1120        <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1121        <IRI>#CarbonFluxe</IRI>
1122        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Transfer of
1123        carbon from one pool to another in units of measurement of mass per
1124        unit of area and time (e.g.,
1125        tonnes C ha-1 yr-1).</Literal>
1126    </AnnotationAssertion>
1127    <AnnotationAssertion>
1128        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1129        <IRI>#CarbonFluxe</IRI>
1130        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Carbon Fluxe
1131        </Literal>
1132    </AnnotationAssertion>
1133    <AnnotationAssertion>
1134        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1135        <IRI>#CarbonMensuration</IRI>
1136        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Carbon
1137        Mensuration</Literal>
1138    </AnnotationAssertion>
1139    <AnnotationAssertion>
1140        <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1141        <IRI>#CarbonPool</IRI>
1142        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">A reservoir.
1143        A system which has the capacity to accumulate or release carbon.
1144        Examples of carbon pools are forest biomass, wood products, soils
1145        and the atmosphere. The units are mass.</Literal>
1146    </AnnotationAssertion>
1147    <AnnotationAssertion>
1148        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1149        <IRI>#CarbonPool</IRI>
1150        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Carbon Pool
1151        </Literal>
1152    </AnnotationAssertion>
1153    <AnnotationAssertion>
1154        <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1155        <IRI>#CarbonStock</IRI>
1156        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">The quantity of
1157        carbon in a pool.</Literal>
1158    </AnnotationAssertion>
1159    <AnnotationAssertion>
1160        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1161        <IRI>#CarbonStock</IRI>
1162        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Carbon Stock
1163        </Literal>
1164    </AnnotationAssertion>
1165    <AnnotationAssertion>
1166        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1167        <IRI>#CrownDiameter</IRI>
1168        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Crown Diameter
1169        </Literal>
1170    </AnnotationAssertion>
1171    <AnnotationAssertion>
1172        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1173        <IRI>#DBH</IRI>
1174        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Diameter at Breast
1175        Height</Literal>
1176    </AnnotationAssertion>
1177    <AnnotationAssertion>
1178        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
1179        <IRI>#Dead</IRI>
1180        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Dead aboveground
1181        biomass is composed of litter and fallen trunks.
1182</Literal>
1183    </AnnotationAssertion>
1184    <AnnotationAssertion>
1185        <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1186        <IRI>#Deforestation</IRI>
```

```
1187        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">The direct
1188        human-induced conversion of forested land to non-forested land.
1189        </Literal>
1190    </AnnotationAssertion>
1191    <AnnotationAssertion>
1192        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1193        <IRI>#DirectActivity</IRI>
1194        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Direct Activity
1195        </Literal>
1196    </AnnotationAssertion>
1197    <AnnotationAssertion>
1198        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
1199        <IRI>#DirectEstimation</IRI>
1200        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">The direct
1201        method is expensive and destructive, consisting of the cutting
1202        and weighing of the aboveground material in an established area.
1203</Literal>
1204    </AnnotationAssertion>
1205    <AnnotationAssertion>
1206        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1207        <IRI>#DirectEstimation</IRI>
1208        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Direct Estimation
1209        </Literal>
1210    </AnnotationAssertion>
1211    <AnnotationAssertion>
1212        <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1213        <IRI>#Disturbance</IRI>
1214        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Processes
1215        that reduce or redistribute carbon pools in terrestrial ecosystems
1216        </Literal>
1217    </AnnotationAssertion>
1218    <AnnotationAssertion>
1219        <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1220        <IRI>#EmissionFactor</IRI>
1221        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">A coefficient
1222        that relates the activity data to the amount of chemical compound
1223        which is the source of later emissions. Emission factors are often
1224        based on a sample of measurement data, averaged to develop a
1225        representative rate of emission for a given activity level under a given
1226        set of operating conditions.</Literal>
1227    </AnnotationAssertion>
1228    <AnnotationAssertion>
1229        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1230        <IRI>#EmissionFactor</IRI>
1231        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Emission Factor
1232        </Literal>
1233    </AnnotationAssertion>
1234    <AnnotationAssertion>
1235        <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1236        <IRI>#Emittion</IRI>
1237        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">The release of
1238        greenhouse gases and/or their precursors into the atmosphere over a
1239        specified area and period of time.</Literal>
1240    </AnnotationAssertion>
1241    <AnnotationAssertion>
1242        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
1243        <IRI>#Forest</IRI>
1244        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Source: IPPC anexe I
1245        </Literal>
1246    </AnnotationAssertion>
1247    <AnnotationAssertion>
1248        <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1249        <IRI>#Forest</IRI>
1250        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Forest is a
1251        minimum area of land of 0.05 - 1.0 hectares with tree crown cover
1252        (or equivalent stocking level) of more than 10 - 30 per cent with
1253        trees with the potential to reach a minimum height of 2 - 5
```

```
1254          metres at maturity insitu. A forest may consist either of closed
1255          forest formations where trees of various storeys and undergrowth
1256          cover a high portion of the ground or open forest. Young natural
1257          stands and all plantations which have yet to reach a crown density of
1258          10 - 30 per cent or tree height of 2 - 5 metres are included under
1259          forest, as are areas normally forming part of the forest area
1260          which are temporarily unstocked as a result of human intervention
1261          such as harvesting or natural causes but which are expected to
1262          revert to forest.
1263 </Literal>
1264     </AnnotationAssertion>
1265     <AnnotationAssertion>
1266         <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1267         <IRI>#Forest</IRI>
1268         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Forest</Literal>
1269     </AnnotationAssertion>
1270     <AnnotationAssertion>
1271         <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1272         <IRI>#GreenhouseGas</IRI>
1273         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">The gases
1274         that have global warming potentials (GWPs).</Literal>
1275     </AnnotationAssertion>
1276     <AnnotationAssertion>
1277         <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1278         <IRI>#HumanInducedActivity</IRI>
1279         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Human-induced
1280         Activity</Literal>
1281     </AnnotationAssertion>
1282     <AnnotationAssertion>
1283         <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1284         <IRI>#IndirectActivity</IRI>
1285         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Indirect
1286         Activity</Literal>
1287     </AnnotationAssertion>
1288     <AnnotationAssertion>
1289         <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
1290         <IRI>#IndirectEstimation</IRI>
1291         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">The indirect
1292         method utilizes mathematical models. It relates the tree variable
1293         parameters obtained from forest inventories, such as diameter at
1294         breast height (DBH, the trunk diameter at approximately 1.3 m from
1295         the soil level), trunk height, crown diameter, total tree height,
1296         tree species, etc.
1297 </Literal>
1298     </AnnotationAssertion>
1299     <AnnotationAssertion>
1300         <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1301         <IRI>#IndirectEstimation</IRI>
1302         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Indirect
1303         Estimation</Literal>
1304     </AnnotationAssertion>
1305     <AnnotationAssertion>
1306         <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1307         <IRI>#ModellingApproach</IRI>
1308         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Modelling
1309         Approach</Literal>
1310     </AnnotationAssertion>
1311     <AnnotationAssertion>
1312         <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
1313         <IRI>#Region</IRI>
1314         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Source:
1315         IPCC anexe I</Literal>
1316     </AnnotationAssertion>
1317     <AnnotationAssertion>
1318         <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1319         <IRI>#Region</IRI>
1320         <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">A region is
```

```
1321        a territory characterised by specific geographical and climatological
1322        features. The climate of a region is affected by regional and local
1323        scale foreings like topography, land use characteristics, lakes, etc.,
1324        as well as remore influences from other regions.</Literal>
1325    </AnnotationAssertion>
1326    <AnnotationAssertion>
1327        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1328        <IRI>#Region</IRI>
1329        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Region</Literal>
1330    </AnnotationAssertion>
1331    <AnnotationAssertion>
1332        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
1333        <IRI>#Root</IRI>
1334        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Roots of the tree are
1335        a belowgraund biomass</Literal>
1336    </AnnotationAssertion>
1337    <AnnotationAssertion>
1338        <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1339        <IRI>#Sink</IRI>
1340        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Any process,
1341        activity or mechanism which removes a greenhouse gas, an aerosol,
1342        or a precursor of a greenhouse gas from the atmosphere. Notation
1343        in the final stages of reporting is the negative (-)
1344        sign.</Literal>
1345    </AnnotationAssertion>
1346    <AnnotationAssertion>
1347        <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
1348        <IRI>#TopDown</IRI>
1349        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">A modelling
1350        approach which aims to infer processes and parameters at a smaller
1351        scale from measurements taken at an aggregated scale
1352        (regional/national/continental/global).</Literal>
1353    </AnnotationAssertion>
1354    <AnnotationAssertion>
1355        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1356        <IRI>#TopDown</IRI>
1357        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Top-down
1358        </Literal>
1359    </AnnotationAssertion>
1360    <AnnotationAssertion>
1361        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1362        <IRI>#TrunkHeight</IRI>
1363        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Trunk
1364        Height</Literal>
1365    </AnnotationAssertion>
1366    <AnnotationAssertion>
1367        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1368        <IRI>#brazilianAmazonForest</IRI>
1369        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Brazilian
1370        Amazon Forest</Literal>
1371    </AnnotationAssertion>
1372    <AnnotationAssertion>
1373        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1374        <IRI>#carbonAboveground</IRI>
1375        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Carbon
1376        Aboveground Equation</Literal>
1377    </AnnotationAssertion>
1378    <AnnotationAssertion>
1379        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1380        <IRI>#croplands</IRI>
1381        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Croplands</Literal>
1382    </AnnotationAssertion>
1383    <AnnotationAssertion>
1384        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1385        <IRI>#desert</IRI>
1386        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Desert</Literal>
1387    </AnnotationAssertion>
```

```
1388    <AnnotationAssertion>
1389        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1390        <IRI>#dryAbovegroundBiomass</IRI>
1391        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Dry Above Ground
1392        Biomass Equation</Literal>
1393    </AnnotationAssertion>
1394    <AnnotationAssertion>
1395        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1396        <IRI>#dryTotalBiomass</IRI>
1397        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Dry Total Biomass
1398        Equation</Literal>
1399    </AnnotationAssertion>
1400    <AnnotationAssertion>
1401        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1402        <IRI>#freshAbovegroundBiomass_1</IRI>
1403        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Fresh Aboveground
1404        Biomass Equation</Literal>
1405    </AnnotationAssertion>
1406    <AnnotationAssertion>
1407        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1408        <IRI>#freshAbovegroundBiomass_2</IRI>
1409        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Fresh Aboveground
1410        Biomass Generic</Literal>
1411    </AnnotationAssertion>
1412    <AnnotationAssertion>
1413        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1414        <IRI>#selectTrees</IRI>
1415        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">Select Trees
1416        </Literal>
1417    </AnnotationAssertion>
1418    <AnnotationAssertion>
1419        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1420        <IRI>#semidesert</IRI>
1421        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">
1422        Semidesert</Literal>
1423    </AnnotationAssertion>
1424    <AnnotationAssertion>
1425        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1426        <IRI>#temperateGrasslands</IRI>
1427        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">
1428        Temperate Grasslands</Literal>
1429    </AnnotationAssertion>
1430    <AnnotationAssertion>
1431        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1432        <IRI>#totalCarbonEquation</IRI>
1433        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">
1434        Total Carbon Equation</Literal>
1435    </AnnotationAssertion>
1436    <AnnotationAssertion>
1437        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1438        <IRI>#totalFreshBiomass</IRI>
1439        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">
1440        Total Fresh Biomass Equation</Literal>
1441    </AnnotationAssertion>
1442    <AnnotationAssertion>
1443        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1444        <IRI>#tropicalSavannas</IRI>
1445        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">
1446        Tropical Savannas</Literal>
1447    </AnnotationAssertion>
1448    <AnnotationAssertion>
1449        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1450        <IRI>#tundra</IRI>
1451        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">
1452        Tundra</Literal>
1453    </AnnotationAssertion>
1454    <AnnotationAssertion>
```

```
1455            <AnnotationProperty abbreviatedIRI="rdfs:label"/>
1456            <IRI>#wetlands</IRI>
1457            <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">
1458            Wetlands</Literal>
1459        </AnnotationAssertion>
1460        <SubAnnotationPropertyOf>
1461            <AnnotationProperty IRI="#source"/>
1462            <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
1463        </SubAnnotationPropertyOf>
1464    </Ontology>
1465    <!-- Generated by the OWL API (version 3.2.5.1912) http://owlapi.sourceforge.net -->
```