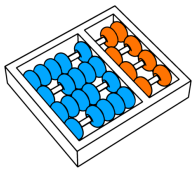
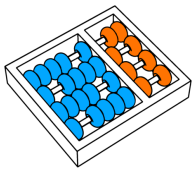


Fernanda Yara dos Santos Foschiani

"Uma abordagem de Linhas de Produtos de Software para apoiar e-Science"

CAMPINAS
2013



Universidade Estadual de Campinas
Instituto de Computação

Fernanda Yara dos Santos Foschiani

"Uma abordagem de Linhas de Produtos de Software para apoiar e-Science"

Orientador(a): **Profa. Dra. Cecília Mary Fischer Rubira**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Mestre em Ciência da Computação.

Este exemplar corresponde à versão final da Dissertação defendida por Fernanda Yara dos Santos Foschiani, sob orientação de Profa. Dra. Cecília Mary Fischer Rubira.

Assinatura do Orientador(a)

CAMPINAS
2013

FICHA CATALOGRÁFICA ELABORADA POR
ANA REGINA MACHADO - CRB8/5467
BIBLIOTECA DO INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E
COMPUTAÇÃO CIENTÍFICA - UNICAMP

Foschiani, Fernanda Yara dos Santos, 1983-
F783a Uma abordagem de linhas de produtos de software para apoiar e-Science / Fernanda Yara dos Santos Foschiani. – Campinas, SP : [s.n.], 2012.

Orientador: Cecília Mary Fischer Rubira.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Engenharia de linha de produto de software. 2. Software - Arquitetura. 3. Componente de software. 4. Reengenharia de software. 5. Engenharia de software - Desenvolvimento. I. Rubira, Cecília Mary Fischer,1964-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em inglês: A software product lines approach to support e-Science
Palavras-chave em inglês:
Software product line engineering
Software architecture
Component software
Software reengineering
Software engineering - Development
Área de concentração: Ciência da Computação
Titulação: Mestra em Ciência da Computação
Banca examinadora:
Cecília Mary Fischer Rubira [Orientador]
Rosana Teresinha Vaccare Braga
Eliane Martins
Data de defesa: 14-12-2012
Programa de Pós-Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 14 de Dezembro de 2012, pela Banca examinadora composta pelos Professores Doutores:

Rosana Braga

Profª. Drª. Rosana Teresinha Vaccare Braga
ICMC / USP

E. Martins

Profª. Drª. Eliane Martins
IC / UNICAMP

Cecília Mary Fischer Rubira

Prof. Dr. Cecília Mary Fischer Rubira
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Uma abordagem de Linhas de Produtos de Software para apoiar e-Science

Fernanda Yara dos Santos Foschiani

14 de Dezembro de 2012

Banca Examinadora:

- Profa. Dra. Cecília Mary Fischer Rubira (Orientadora)
- Profa. Dra. Rosana Teresinha Vaccare Braga
  Instituto de Ciências Matemáticas e de Computação, USP
- Profa. Dra. Eliane Martins
  Instituto de Computação - UNICAMP

Abstract

With the increasing demand for software in order to reduce development costs and effort, and to reduce the time-to-market, several techniques are being used, including the Software Product Line (SPL). Computational resources are commonly used in the research field, in order to facilitate data and computational services sharing. The context in which computing becomes a fundamental for the success of scientific research is called e-Science. The systems diversity, simulators and computational data involved in experiments leads to the necessity of environments that provide facilities for technology use and matching, for example the scientific workflow environment.

The proposed solution in this thesis is a scientific workflow environment that allows the researchers to create their own personalized workflows, using components provided by the development team as well as components developed by themselves, regardless of the language being used. The basis for this workflow environment is a component based software product line, developed from legacy systems.

The proposed method for the software product line development is supported by the Feature-oriented Reengineering framework, which is divided into four steps. The first one, SPL Reverse Engineering, extracts information about the legacy system in order to understand the application domain and collect the features that need to exist in the product line. The second step, SPL Analysis, refines the feature model obtained in the previous step and, using PLUS modeling techniques, develops software assets based on use cases. The SPL Project step, which is the third approach step, applies the FArM method to obtain a mapping of the feature model to architectural components, and specifies the components' interface, creating the final architecture. The last step is the product line development. For the components development we used the COSMOS* model and legacy code.

We performed three case studies: two of them to evaluate if the product line is capable of replacing the legacy system and enhance components reuse, and the third one to evaluate the workflow customization capability, by the addition of a component developed in MatLab.

All the case studies had a positive result, showing that the proposed solution of this

thesis facilitates the product line architectures evolution and allows the researchers to customize their workflows, aiding the research process.

keywords: software product line engineering, software architecture, component software, software reengineering, software engineering - development

Resumo

Com o aumento da demanda por software no mercado, a fim de reduzir custos e esforço em desenvolvimento e reduzir o tempo de entrega de software, diversas técnicas vêm sendo utilizadas e entre elas estão as Linhas de Produtos de Software (LPS). Por outro lado, os recursos computacionais vêm sendo muito utilizados não só na indústria, mas também em ambientes de pesquisa, facilitando o compartilhamento de dados e serviços computacionais. Este contexto, em que a computação torna-se parte integrante e fundamental para o sucesso na realização de pesquisas científicas, é chamado de *e-Science*. A diversidade de programas, simuladores e dados computacionais envolvidos em experimentos levam à necessidade de ambientes que forneçam facilidades para o uso e combinação das tecnologias, como por exemplo, ambientes de *workflows* científicos.

A solução proposta nesta dissertação é uma infraestrutura para a execução de *workflows* científicos, que permite a pesquisadores criarem seus fluxos de trabalho de maneira personalizada, podendo utilizar componentes disponibilizados pela equipe de desenvolvimento, assim como componentes criados por eles mesmos, independentemente da linguagem de programação utilizada. A base para esta infraestrutura é uma linha de produtos baseada em componentes, desenvolvida a partir de sistemas legados.

O método proposto para o desenvolvimento da linha de produtos é apoiado pelo arcabouço da Reengenharia Orientada a Características, separado em quatro fases. A primeira fase, Engenharia Reversa do Sistema Legado, extrai informações do código legado a fim de entender o domínio da aplicação e coletar as características que deverão existir na linha de produtos. A segunda fase do método, Análise da LPS, refina o modelo de características obtido na fase anterior e, utilizando técnicas de modelagem do método PLUS, desenvolve artefatos de software baseados em casos de uso. A fase de Projeto da LPS, terceira fase, aplica o método FArM para obter um mapeamento do modelo de características para modelos de arquitetura de linha de produtos baseada em componentes e especifica as interfaces dos componentes, gerando assim a arquitetura final. A quarta e última fase trata do desenvolvimento da linha de produtos. Para o desenvolvimento dos componentes foi utilizado o padrão COSMOS* de componentização, e código legado.

Foram realizados três estudos de caso: os dois primeiros para avaliar se a solução pro-

posta é capaz de substituir o sistema legado e também avaliar o reúso de componentes, e o terceiro estudo para avaliar a capacidade de customização da linha de produtos, a partir da adição de um novo componente desenvolvido em MatLab. Os três estudos apresentaram resultados positivos, indicando que a solução proposta nesta dissertação facilita a modificação da linha de produtos, além de permitir aos pesquisadores a customização de fluxos de trabalho, auxiliando assim o processo de pesquisa científica.

palavras-chave: engenharia de linha de produto de software, software - arquitetura, componente de software, reengenharia de software, engenharia de software - desenvolvimento.

Agradecimentos

Dedico esta dissertação aos meus pais, que me deram a vida e me ensinaram a viver com dignidade.

Durante a construção deste trabalho muitas pessoas foram de suma importância, me ajudando a trilhar este longo caminho.

Primeiro, agradeço a Deus pela vida que me deu, e por ter colocado nela pessoas tão especiais, sem as quais eu nada seria.

Agradeço à minha orientadora Cecília Rubira por estar sempre presente com sua sabedoria e dedicação, mesmo passando por momentos tão difíceis durante nossa jornada.

Gostaria de agradecer aos meus pais. Sem eles nada disso seria possível. Não bastaria um muito obrigada para agradecer àqueles que iluminaram meus caminhos com afeto e dedicação, e que renunciaram seus sonhos para que eu pudesse realizar os meus.

À minha irmã Danielle, minha melhor amiga e cúmplice, que sempre teve participação em toda minha vida, e desta vez não seria diferente. Mesmo estando tão longe, sempre esteve perto.

Ao meu noivo, amor, amigo e companheiro André Bertolini, que me sustentou todas as vezes nos momentos em que o medo, a frustração e as dúvidas me derrubaram, me ajudando ao seu modo, a me levantar e sempre seguir em frente.

Um agradecimento muito especial ao meu amigo Leonardo Tizzei. Não tenho palavras para agradecer sua dedicação tamanha, me guiando e ensinando desde o começo. Sem sua ajuda tudo teria sido muito mais difícil!

Agradeço ao Denis Schiozer, coordenador do Grupo UNISIM, e Alberto Santos, gerente de projetos do Grupo UNISIM, pela oportunidade que me foi dada para a realização deste mestrado. Obrigada pela compreensão e apoio durante estes anos.

Agradeço ainda a todos meus amigos, familiares e colegas do UNISIM e IC que, direta ou indiretamente, contribuíram para a realização deste trabalho.

"Um dia você aprende ... Aprende que as circunstâncias e os ambientes têm influência sobre nós, mas nós somos responsáveis por nós mesmos. Começa a aprender que não

se deve comparar com os outros, mas com o melhor que pode ser. Descobre que se leva muito tempo para se tornar a pessoa que quer ser, e que o tempo é curto. Aprende que não importa onde já chegou, mas onde está indo, mas se você não sabe para onde está indo, qualquer lugar serve. ... Portanto, plante seu jardim e decore sua alma ao invés de esperar que alguém lhe traga flores. E você aprende que realmente pode suportar, que realmente é forte, e que pode ir muito mais longe depois de pensar que não se pode mais. E que realmente a vida tem valor e que você tem valor diante da vida! Nossas dúvidas são traidoras e nos fazem perder o bem que poderíamos conquistar se não fosse o medo de tentar."

Autor Desconhecido

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

A demanda por software no mercado vem aumentando a cada dia, juntamente com a complexidade e tamanho dos produtos exigidos. Com isso, a competitividade vem crescendo e, por parte dos clientes, existe cada vez mais a exigência por produtos de qualidade e agilidade na entrega. Com o intuito de auxiliar o desenvolvimento de software, reduzir custos e diminuir o tempo de mercado (do inglês *time-to-market*), várias técnicas vêm sendo criadas e utilizadas, e entre elas estão as Linhas de Produtos de Software (LPS).

A ideia fundamental da abordagem de LPS é realizar o desenvolvimento de um conjunto de produtos como uma única atividade coerente de desenvolvimento. Os produtos são construídos a partir de um conjunto de ativos centrais (do inglês *core asset*), uma coleção de ativos que foram especificamente desenvolvidos para utilização na LPS. Esta abordagem tem se mostrado eficiente para capacitar melhorias em qualidade, tempo de mercado, custo e produtividade, se comparada a um processo de desenvolvimento de software tradicional [21].

Em LPS, um conceito muito importante é o gerenciamento de variabilidade. Pode-se entender variabilidade como sendo as diferenças entre os produtos gerados pela LPS, e pontos de variação são os pontos onde estas variabilidades diferem em cada produto.

A arquitetura utilizada na LPS é um fator fundamental para sua evolução uma vez que ela abstrai detalhes de implementação. A variabilidade arquitetural, através de um conjunto de pontos de variação localizados na configuração dos seus componentes e conectores, reflete no projeto a diferença entre os diversos produtos gerados a partir de uma LPS. É a variabilidade arquitetural que difere a arquitetura de uma LPS de arquiteturas de sistemas únicos. Geralmente, a variabilidade em LPS é identificada usando um modelo de características e implementada pela arquitetura da LPS [55].

Assim como a abordagem de LPS, existem vários outros métodos de desenvolvimento de software baseados em reutilização através do gerenciamento de variabilidades. Porém, de um modo geral, os métodos existentes são muito difíceis de serem implementados

em um ambiente já em produção, em parte pela falta de um mapeamento explícito de funcionalidades comuns e variáveis dos produtos desenvolvidos e também pela dificuldade da troca de metodologias adotadas.

Recursos computacionais vêm sendo muito utilizados não só nas indústrias, como também no desenvolvimento da pesquisa, beneficiando o trabalho das comunidades científicas, facilitando o compartilhamento de dados e serviços computacionais, além de contribuir para a construção de uma infraestrutura de dados e de uma comunidade científica distribuída. Este contexto, em que a computação torna-se parte integrante e fundamental para o sucesso na realização de pesquisas científicas das mais variadas áreas, é o contexto de *e-Science* [44].

O uso de estudos experimentais é motivado pelo desejo de controlar e manipular o comportamento do ambiente de forma precisa, direta e sistemática. Tempos atrás, os pesquisadores realizavam seus experimentos em bancadas de laboratório ou em ambientes reais, porém cada vez torna-se mais comum o uso de recursos computacionais na simulação destes ambientes [8]. A diversidade de programas, simuladores e dados computacionais envolvidos em experimentos levam à necessidade de ambientes que forneçam facilidades para o uso e combinação das tecnologias, como por exemplo, o ambiente de execução de *workflows* científicos.

Segundo a *Workflow Management Coalition* (WfMC) [8], *workflow* é a automação do processo de negócio, na sua totalidade ou em partes, onde documentos, informações ou tarefas são passadas de um participante para o outro para execução de uma ação, de acordo com um conjunto de regras e procedimentos. *Workflow* científico é a aplicação do conceito de *workflow* em ambientes científicos (*e-Science*).

Um experimento é uma das formas utilizadas pelos cientistas para apoiar a formulação de novas teorias. O *workflow* científico representa a definição da orquestração de sequências de processos que manipulam dados de modo a construir uma simulação. Ele é o principal recurso do experimento científico. Um experimento se caracteriza pela composição e execução de diversas variações de um *workflow*. Essas variações incluem a mudança de dados de entrada, parâmetros, programas ou ainda a combinação delas [42] [25].

O *workflow* científico é definido como o *template* do estudo, onde a sequência das atividades essenciais estão descritas [26]. O *workflow* científico serve como uma abstração para representar o encadeamento de atividades para experimentação [42].

A tarefa de concepção do *workflow* é o momento no qual o pesquisador define quais são as atividades, a sua ordem de execução e o seu escopo, isto é, o fluxo de execução do seu experimento [39].

Este trabalho tem como objetivo propor um método baseado em LPS para projetar uma infraestrutura de apoio a um sistema de gerenciamento de *workflow* científico (do inglês *Workflow Management Systems*, WfMS) para ser implantado em um grupo de

desenvolvimento de sistemas de software utilizados como apoio à pesquisa na área de estratégia de busca e exploração de reservatórios de petróleo.

1.1 Motivação

As atividades de *e-Science* vêm crescendo cada vez mais e, junto com este crescimento, são criadas novas metodologias, novas ferramentas de estudo, e obtidos novos dados e resultados. A proliferação de programas e dados é o principal motivador para estudos de como organizar e armazenar estas informações de forma a compartilhar e reutilizar experimentos.

O UNISIM [7] é um grupo de pesquisa criado em 1996 que atua na área de Simulação Numérica e Gerenciamento de Reservatórios de Petróleo. O Grupo, pertencente ao Departamento de Engenharia de Petróleo da Faculdade de Engenharia Mecânica da Unicamp, conta com o apoio do Centro de Estudos de Petróleo (CEPETRO).

Os sistemas de software desenvolvidos pelo UNISIM visam fornecer aos pesquisadores (engenheiros de reservatórios de petróleo, economistas, geólogos, etc.) ferramentas que os auxiliem na tomada de decisões e que sejam úteis na automatização de tarefas que utilizam a simulação numérica de reservatórios.

A simulação de reservatório, segundo Aziz [11], é uma mistura de engenharia, física, química, matemática, análise numérica, programação de computadores, experiência e prática do engenheiro. No passado, antes do advento de técnicas numéricas para solução de equações não-lineares e de programas eficientes, modelos físicos e análogos foram extensivamente utilizados para representar o reservatório. No gerenciamento de reservatórios de petróleo, a simulação é uma ferramenta extremamente útil que permite estimar o comportamento de pressões, saturações e produções de uma jazida de hidrocarbonetos submetida a configurações alternativas de poços ou condições de produção [52].

O simulador matemático de reservatórios consiste em um conjunto de equações diferenciais parciais que expressam a conservação de massa e energia. Além disso, acrescenta algumas leis que descrevem os processos de fluxos atuantes no reservatório. Exemplos destas leis são Darcy (fluxo de fluidos), Fourier (condução de calor) e Fick (transporte de soluto por difusão ou dispersão). As equações diferenciais do modelo são geralmente não-lineares e requerem uma solução numérica, surgindo então os simuladores numéricos de reservatório. Estes são programas de computador que resolvem equações de diferenças finitas, obtidas a partir das equações diferenciais originais, sujeitas a condições iniciais e de contorno adequadas [52].

O processo de simulação é composto das seguintes etapas: (1) Caracterização de reservatórios, (2) Definição e construção do modelo de simulação, (3) Ajuste do histórico de produção, (4) Previsão de comportamento (extrapolação) e (5) Documentação. Todas es-

tas etapas são realizadas através de um trabalho multidisciplinar que envolve engenharia de reservatórios, geologia, geofísica, engenharia de produção e laboratórios.

Segundo Mello [41], a expectativa de aumento significativo da produção de petróleo no Brasil deve-se, entre outros fatores, às descobertas recentes de imensos reservatórios de petróleo no pré-sal (reservatórios localizados abaixo da camada de sal).

Não apenas já foram confirmados volumes expressivos nos campos Carioca, Franco, Iara, Tiro, Parque das Baleias, Lula e Júpiter, como ainda há grandes campos em etapa de avaliação, como Bem-te-vi, Caratinga, Caramba, Guará, Iracema, Libra, Azulão e Parati [29]. O desenvolvimento da produção a partir desses reservatórios promete grandes recompensas, mas enfrenta desafios do ponto de vista científico, técnico e econômico.

Os reservatórios do pré-sal brasileiros possuem, em geral, óleo leve, com alta densidade e baixo teor de enxofre [16], características atraentes economicamente pelo rendimento em derivados de maior valor agregado, como gasolina e lubrificantes. Muitos desses volumes situam-se em localizações marítimas muito distantes, a mais de 200 quilômetros da costa, a grandes profundidades de lâmina d'água, acima de 2000 metros, possuem grandes volumes de gás associado, requerem a perfuração da camada de sal mecanicamente instável e de reservatórios carbonáticos fraturados, exigem elevados investimentos e a sua produção enfrenta consideráveis obstáculos técnicos e de custos de tecnologia.

O sucesso na seleção de estratégias para a produção a partir desses reservatórios é essencial para o desenvolvimento do setor petrolífero do Brasil e da economia nacional em médio e longo prazo, e depende da pesquisa e simulação de reservatórios confiável [41].

1.2 Contexto do Problema

Além do desenvolvimento de pesquisa em temas relacionados às necessidades da indústria de Exploração e Produção (E&P), os principais objetivos do UNISIM são a formação de recursos humanos na área de engenharia e simulação de reservatórios de petróleo, o aumento da confiabilidade nos estudos e metodologias que empregam a simulação numérica de reservatórios, melhorias no processo de tomada de decisão na exploração e produção de petróleo, a integração entre a simulação de reservatórios com outras áreas de E&P, tais como Geociências e Economia e o desenvolvimento de técnicas para aumentar a eficiência de estudos através do gerenciamento de múltiplas simulações.

Os principais temas de pesquisa desenvolvidos pelo UNISIM são o uso do simulador para caracterização de reservatórios, a automatização de metodologias que utilizam simuladores de reservatórios, tais como, análise de risco, ajuste de histórico assistido e otimização de explotação, o desenvolvimento e gerenciamento de reservatórios de petróleo, o estudo de técnicas de construção de simuladores e a utilização de técnicas de computação paralela e distribuída aplicadas a esses processos.

Para atingir seus objetivos, o UNISIM busca desenvolver sistemas de software para pesquisa na área petrolífera que forneçam as principais facilidades fornecidas pelos sistemas de software comerciais. Além de diversas ferramentas que fazem interfaces com sistemas comerciais, o UNISIM conta hoje com dois sistemas desenvolvidos em parceria com a Petrobras: o sistema UNIPAR e o sistema MERO.

O sistema MERO ainda está em início de desenvolvimento e, portanto, não será utilizado para o estudo desta dissertação.

O objetivo do sistema UNIPAR é fornecer ao engenheiro de reservatórios de petróleo ferramentas que o auxiliem na tomada de decisões e que sejam úteis na automatização de algumas tarefas que utilizam simulação numérica de reservatórios. A utilização de recursos de computação distribuída para realizar a distribuição das simulações, permite atingir este objetivo com maior eficiência.

O sistema UNIPAR apoia três tópicos de pesquisa - Análise de Incertezas e Risco, Ajuste de Histórico e Análise Econômica - através de cinco módulos principais e um conjunto de ferramentas para apoio à execução dos módulos. O UNIPAR será detalhado no Capítulo 3, Seção 3.3.

A maioria dos arquivos de entrada e saída do sistema UNIPAR utiliza o padrão *eXtensible Markup Language* (XML) de representação de dados e, por isto, a utilização da sua interface gráfica para a criação e alteração dos arquivos é recomendada. Além disso, caso o usuário conheça as regras do XML, ele pode gerar seu arquivo com a ajuda de um editor de texto.

O sistema UNIPAR é um sistema legado que vem sendo atualizado constantemente, porém apresenta uma série de problemas que dificultam sua manutenção, como por exemplo, código pouco modularizado; estrutura pouco flexível; muito código duplicado; código sem documentação; e custo de manutenção alto.

Estes problemas dificultam a atualização do sistema, a alteração do código existente, a inclusão de novas metodologias de pesquisa, além da sua integração com outros programas.

O método proposto nesta dissertação será aplicado ao sistema UNIPAR, com o objetivo de extrair uma LPS a partir desse sistema legado.

1.3 Solução Proposta

Para apoiar a solução da organização e armazenamento das ferramentas e dados utilizados em experimentos científicos de *e-Science*, propõe-se um ambiente de apoio à execução de *workflows* científicos, que é uma ferramenta flexível para executar fluxos de trabalho geralmente complexos. Cada fluxo é formado por um ou mais programas e, após executado, produz os resultados desejados contendo saídas gráficas ou textuais.

O *workflow* científico fornece um ambiente facilitador para a criação e execução de

experimentos científicos com base em dados e programas disponíveis. Com o uso dos *workflows* científicos, um usuário pode representar e reproduzir experimentos já realizados ou criar novos experimentos facilmente reproduzíveis. Em outras palavras, o *workflow* científico apoia a modelagem e registro do conhecimento do domínio [39].

A solução proposta é uma infraestrutura de apoio para executar *workflows* científicos onde os usuários possam utilizar componentes disponíveis e também integrar componentes desenvolvidos por eles ou terceiros (outros pesquisadores) aos componentes existentes, gerando experimentos científicos personalizados. A base desta infraestrutura é uma LPS baseada em componentes, desenvolvida a partir do sistema UNIPAR, chamada de UNIPAR-LPS. A Figura 1.1 mostra a arquitetura em camadas da solução proposta.

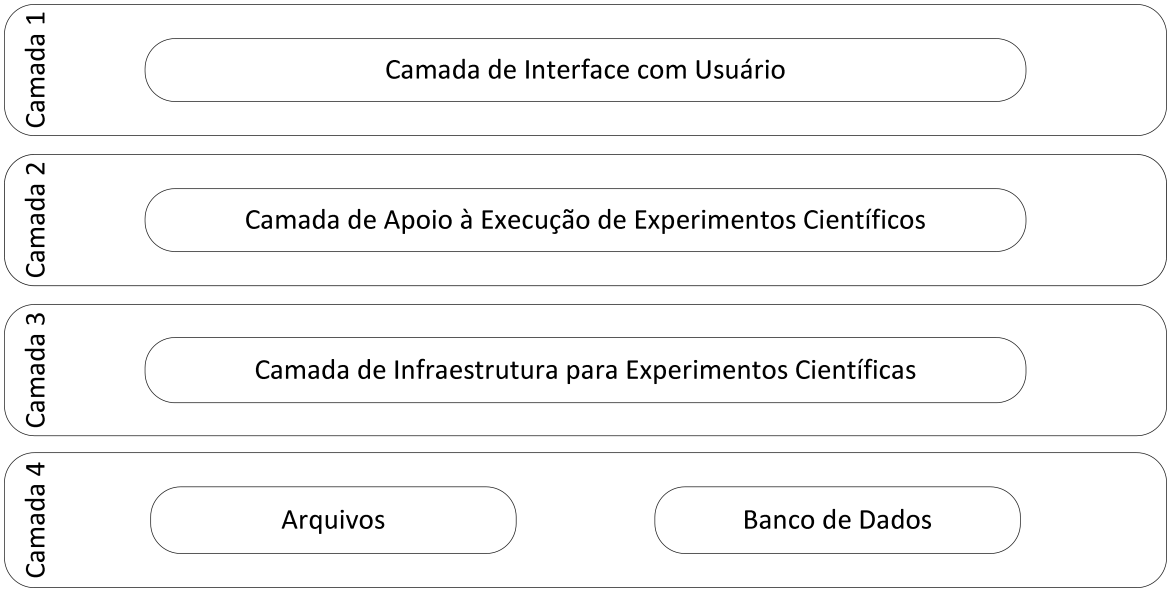


Figura 1.1: Visão geral da solução proposta

A Camada 1, Interface com Usuário, é a responsável pela interface do sistema com o usuário. A interface inicial é via linhas de comando. A Camada 2, Apoio à execução de Experimentos Científicos, é a camada que provê serviços necessários à execução dos experimentos científicos. A Camada 3, Infraestrutura para Experimentos Científicos, é o foco deste trabalho. Ela oferece serviços para a concepção dos experimentos. É nesta camada que foi implementada a UNIPAR-LPS. A Camada, Arquivos e Dados, é a camada relativa ao armazenamento dos dados dos experimentos, sejam em bancos de dados ou em arquivos.

A UNIPAR-LPS foi desenvolvida baseada na arquitetura *pipe-and-filter* (Seção 2.4.2), utilizando componentes de software gerados a partir da implementação dos módulos do sistema UNIPAR. O módulo de gerenciamento e execução de *workflow* científico foi de-

senvolvido baseado na arquitetura *call-and-return* (Seção 2.4.3).

O trabalho deve dar continuidade e se integrar com outros trabalhos do Grupo de Engenharia de Software e Tolerância a Falhas do Instituto de Computação da Unicamp, nas áreas de arquitetura de software, ferramentas e evolução de sistemas baseados em componentes.

1.4 Organização do Documento

O restante desta dissertação é apresentado da seguinte forma: o Capítulo 2 apresenta os fundamentos teóricos sobre reutilização de software e reengenharia, necessários para o entendimento deste trabalho. A solução proposta é detalhada no Capítulo 3, onde também são apresentados alguns trabalhos relacionados. São apresentados três estudos de caso no Capítulo 4, bem como uma análise dos resultados obtidos. Finalmente, no Capítulo 5 são discutidas as conclusões desta dissertação, destacando suas principais contribuições e indicando possíveis trabalhos futuros.

Capítulo 2

Reutilização e Reengenharia de Software

Neste capítulo são apresentados conceitos fundamentais sobre reúso de software, linhas de produtos de software, variabilidade, modelos de características, arquitetura de software, desenvolvimento baseado em componentes e outros conceitos relevantes para a pesquisa.

Na Seção 2.1 são apresentados os conceitos de linhas de produtos de software, seus objetivos, abordagens de desenvolvimento, vantagens, riscos e dificuldades de implementação, além de apresentar o método PLUS de adoção de engenharia de linhas de produtos. A Seção 2.2 define os conceitos características e modelo de características, e descreve detalhadamente o método FArM, utilizado para geração de arquiteturas baseadas em componentes. A Seção 2.3 discorre sobre reengenharia de software e apresenta um arcabouço que propõe uma reengenharia em sistemas legados para adoção de linhas de produtos de software. A Seção 2.4 conta com conceitos sobre arquitetura de software e apresenta três estilos arquiteturais utilizados neste projeto. Por fim, a Seção 2.5 aborda sobre desenvolvimento baseado em componentes, o processo UML *Components* e o modelo de componente COSMOS*, utilizado neste projeto.

2.1 Linhas de Produtos de Software

Linhas de Produtos de Software (LPS, do inglês *Software Product Lines*) é um *framework* de processos que visam o desenvolvimento de uma família de produtos. Ou seja, uma família de produtos representa o conjunto de sistemas de software gerados pela LPS [21].

O desenvolvimento de uma LPS objetiva a construção de componentes que podem ser reutilizados em sistemas de mesmo domínio, e a identificação de variabilidades que podem ser implementadas e combinadas para a formação de diferentes produtos. Para tal,

é necessário que ambos os aspectos sejam planejados para todo o ciclo de desenvolvimento afim de que possam ser centralizados em uma estrutura comum, facilitando a manutenção e reusabilidade do sistema [21].

Na definição para LPS de Clements e Northrop, [21], destacam-se os termos: **segmento particular de mercado ou missão**, **conjunto comum de ativos** e **forma preestabelecida.** Estes termos são muito importantes em uma LPS e sustentam suas principais vantagens propostas. Definimos os termos:

- **Segmento particular ou missão:** Também chamado de domínio, refere-se ao corpo de conhecimento ou à área de especialização em que a LPS atua. Está diretamente relacionado com o conjunto de funcionalidades dos produtos da linha;
- **Conjunto comum de ativos:** Correspondem a um conjunto de elementos customizáveis utilizados na construção dos sistemas de software produzidos (produtos), por exemplo, componentes de software, documentos, *design-patterns*, a arquitetura da linha, etc.;
- **Forma preestabelecida:** O processo de produção de sistemas de software através de uma LPS é realizado através de planos de produção. Esses planos são definidos para cada produto que será produzido pela linha. Ao definir um plano, deve-se relacionar quais ativos farão parte deste produto e assim estabelecer um vínculo aos processos anexos de cada ativo utilizado, processos estes que definem o que o ativo faz, qual a sua flexibilidade, e qual a técnica de configuração do ativo (se ele for flexível).

De acordo com Durscki [28], o processo sugere dividir o produto em pequenas partes, mais compreensíveis e gerenciáveis (ativos); possuir especificações claras e sem ambiguidades para produção/montagem (forma preestabelecida); e atender a um domínio específico (segmento particular ou missão) para que, como em uma linha de montagem da indústria tradicional, cada novo produto seja uma reutilização de partes de produtos construídos anteriormente, pertencentes à mesma família e, de preferência, com um mínimo de esforço adicional de construção.

Para desenvolver uma LPS é necessário identificar as semelhanças e diferenças de um conjunto de software de determinado domínio. Tendo as semelhanças e diferenças identificadas, podem ser implementados os artefatos comuns aos produtos de tal maneira que cada produto da linha possa ser gerado a partir dos artefatos comuns integrados (ou não) a artefatos que representam suas diferenças.

Produzir um conjunto de produtos de software como uma LPS tem permitido às organizações alcançar uma maior qualidade nos produtos, melhoria da produtividade e reduções significativas de custo e tempo de mercado. O uso das técnicas das LPS vem

crescendo, porém a adoção desta abordagem envolve muitos desafios. Seu uso também pode permitir a entrada rápida no mercado e resposta flexível, e fornecem uma capacidade de customização em massa [21].

A iniciativa PLP (*Product Line Pratice*), criada pelo SEI (*Software Engineering Institute*) [6], estabelece três atividades essenciais de uma abordagem de LPS: **Desenvolvimento do Núcleo de Artefatos** (*Core Asset Development*), também conhecida como Engenharia de Domínio; **Desenvolvimento do Produto** (*Product Development*), também conhecida como Engenharia de Aplicação; e **Gerenciamento** (*Management*) [21].

A Figura 2.1, adaptada do arcabouço de LPS do SEI, ilustra a interligação entre as três atividades. Elas estão fortemente ligadas e são atividades dinâmicas. Qualquer alteração em alguma das atividades acarreta em impacto nas demais.

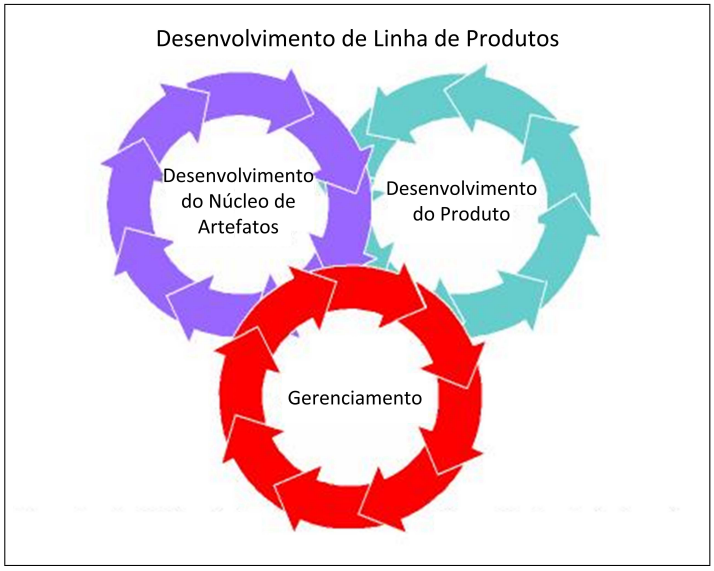


Figura 2.1: As três atividades essenciais de uma LPS

As atividades não seguem uma ordem preestabelecida, mas para que produtos sejam gerados, é necessário que existam artefatos que possam ser integrados. Na medida em que os produtos são desenvolvidos, a fase de Desenvolvimento do Núcleo de Artefatos pode passar por alterações, sejam elas revisões ou inclusões de novos artefatos, assim como novos produtos podem ser gerados enquanto o núcleo de artefatos passa por alterações. A atividade de Gerenciamento influencia diretamente estas duas atividades. As seções a seguir descrevem cada uma das atividades, de acordo com [21].

2.1.1 Desenvolvimento do Núcleo de Artefatos

O objetivo do Desenvolvimento do Núcleo de Artefatos é estabelecer uma capacidade de produção para os produtos. Quatro dos mais importantes fatores contextuais são:

- **Restrições do produto:** São as semelhanças e as variações entre os produtos que constituirão a LPS, as normas que devem ser seguidas para definir as características dos produtos e os limites de desempenho;
- **Restrições de produção:** São os componentes COTS (*Commercial-Off-The-Shelf*) que serão utilizados, os componentes legados que serão reutilizados e as normas que serão seguidas;
- **Estratégia de produção:** Determina a origem da arquitetura e seus componentes associados, e o caminho para seu crescimento;
- **Artefatos preexistentes:** São os sistemas legados e produtos existentes que incorporam os conhecimentos de domínio de uma organização e/ou definem a sua presença no mercado.

As três saídas que o Desenvolvimento do Núcleo de Artefatos gera são:

- **Escopo da linha de produto:** Descrição dos produtos que constituirão a LPS ou que esta é capaz de produzir;
- **Núcleo de artefatos:** Base com todos os artefatos disponíveis para a produção de produtos a partir da LPS;
- **Plano de produção:** Descreve como os produtos são gerados a partir do núcleo de artefatos.

Estas três saídas alimentam a atividade de Desenvolvimento de Produtos, o que implica em produtos que atendem a um determinado cliente ou nicho de mercado.

2.1.2 Desenvolvimento de Produtos

A atividade de Desenvolvimento de Produtos depende das três saídas da atividade anterior, além da especificação de cada produto individualmente, conforme segue:

- **Escopo da linha de produto:** Indica se é ou não viável incluir algum produto na LPS;
- **Núcleo de artefatos:** Artefatos que integrados compõe o produto;

- **Plano de produção:** Detalha como os artefatos serão utilizados;
- **Descrição de um produto específico:** Define as semelhanças e variações de determinado produto contido no escopo da LPS.

2.1.3 Gerenciamento da Linha de Produtos de Software

A atividade de Gerenciamento da LPS deve garantir que todas as atividades técnicas sejam coordenadas e supervisionadas. Pode ser classificado de duas maneiras:

- **Gerenciamento organizacional:** Garante que as unidades organizacionais recebem os recursos corretos (ex. treinamento) em quantidades suficientes;
- **Gerenciamento técnico:** Coordena tecnicamente a equipe de desenvolvimento nas atividades de desenvolvimento do núcleo de artefatos e do produto.

Uma das ações mais importantes desta atividade é a criação de um plano de adoção que descreva o estado desejado da organização e uma estratégia para alcançar tal estado. O gerenciamento técnico e o gerenciamento organizacional também contribuem com o núcleo de artefatos, disponibilizando artefatos de gerenciamento para reúso, especialmente cronogramas e orçamentos usados no desenvolvimento de produtos da LPS.

Apesar do gerenciamento da LPS ser muito importante, ela não entra no escopo deste trabalho.

2.1.4 Vantagens do Uso de Linhas de Produtos de Software

São vários os benefícios que o uso de LPS pode trazer e vários estudos de caso mostram isso. Nesta seção são apresentados estes benefícios de acordo com Cohen [22].

Cohen diz que no início do planejamento de uma LPS, uma organização deve definir os objetivos de negócio e definir os benefícios que espera alcançar. Ele classificou os benefícios do uso de LPS em duas categorias: tangíveis e intangíveis.

- **Tangíveis:** Benefícios que podem ser medidos diretamente, como redução do tempo de mercado ou redução de defeitos;
- **Intangíveis:** Benefícios que os desenvolvedores relatam, mas que não podem ser medidos em termos de métricas. Podem incluir satisfação do desenvolvedor ou aceitação do consumidor.

As Tabelas 2.1 e 2.2 mostram os benefícios mapeados por Cohen, em cada uma das categorias (tabelas adaptadas de [22]).

Tabela 2.1: Benefícios Tangíveis de uma LPS

Fator	Benefício para a Empresa adotando o uso de LPS
Rentabilidade	O núcleo de ativos permite à organização produzir produtos voltados para um segmento específico de mercado. É observado um aumento da participação de mercado e da rentabilidade da empresa.
Qualidade	Uma redução no número de defeitos relatados é comum em sistemas desenvolvidos utilizando LPS. A qualidade também pode ser medida em termos de redução do tempo de correções e da redução do efeito *ripple*, ou seja, geração de novos defeitos a partir de correções executadas.
Desempenho	A utilização de ativos aumenta o desempenho em relação ao desenvolvimento tradicional. Com o aumento da LPS, os ativos fornecem pontos de variação que podem ser substituídos por ativos que possuam algoritmos mais rápidos.
Tempo de Integração	Construções incrementais são completadas mais rapidamente.
Volume de Código	O número de objetos construídos para subsistemas utilizando LPS é menor que o estimado para um sistema baseado no desenvolvimento tradicional.
Produtividade	A abordagem requer uma equipe de desenvolvimento menor acarretando em um corte de custo de desenvolvimento e maior velocidade de liberação do produto.

Tabela 2.2: Benefícios Intangíveis de uma LPS

Fator	Benefício para a Empresa adotando o uso de LPS
Desgaste de Profissionais	Ocorre uma baixa rotatividade e desgaste de pessoal com o uso de linha de produto em comparação com outros projetos que não utilizam esta abordagem.
Aceitabilidade dos Desenvolvedores	Após um treinamento inicial, os desenvolvedores relatam satisfação em trabalhar com a abordagem baseada em ativos e arquitetura comuns.
Satisfação Profissional	Os desenvolvedores entendem que o trabalho braçal já foi realizado (desenvolvimento dos ativos de software) e assim eles podem se concentrar em atividades mais interessantes, como o aperfeiçoamento ou a melhoria de elementos específicos.
Satisfação do Cliente	Os ativos reduzem os riscos, aumentando a previsibilidade da entrega e diminuindo a taxa de defeitos. Os clientes retornam preferindo produtos baseados em LPS.

2.1.5 Dificuldades e Riscos da Implantação

Um projeto de implantação de uma LPS pode ser considerado um projeto de inovação tecnológica, ou seja, um projeto de adaptação a uma nova tecnologia ou a uma nova maneira de fazer negócio. Especificamente no caso de LPS, ambas as definições se aplicam, tornando o projeto ainda mais delicado. Assim como qualquer mudança tecnológica, antes da implantação de uma LPS, deve haver uma avaliação inicial do status atual da empresa, a definição do estado desejado e a elaboração de um plano para atingir este estado [28].

As LPS interferem diretamente na maneira de trabalhar da empresa e por isso outros fatores além dos tecnológicos devem ser considerados, como a adaptação dos recursos, treinamentos necessários e até mesmo a preparação do cliente para a nova abordagem.

As dificuldades de implantação de uma linha são diversas e podem vir de várias fontes. Pelo fato do processo envolver tanto pessoas como tecnologias, ele pode sofrer muita resistência dentro de uma empresa. Neste contexto, Cohen [22] cita como os principais problemas: falta de um líder comprometido, estratégia de implantação inadequada, falta de compromisso da gerência e da equipe, adaptação insuficiente, evolução da abordagem e falta de disseminação.

Provavelmente nenhuma empresa irá enfrentar todos os problemas citados na implantação de uma LPS. A pesquisa realizada por Cohen revelou que pelo menos três ou quatro destes elementos estiveram ou estão presentes nas organizações entrevistadas.

2.1.6 Variabilidade em Linhas de Produtos de Software

De acordo com Bachmann e Clements [14], dentre as principais características de uma LPS está a **variabilidade**. Variabilidade é a habilidade de um sistema, um ativo ou um ambiente de desenvolvimento apoiar a produção de um conjunto de artefatos que diferem um dos outros de uma maneira pré-planejada. Variabilidade significa a habilidade de ativos se adaptarem ao uso em diferentes produtos de uma LPS.

Além da definição de variabilidade apresentada acima, existem várias outras definições na literatura. A mais utilizada atualmente é a definição dada por Van Gurp *et al.* [58] que diz que "variabilidade de software é a capacidade de um sistema de software ou artefato ser modificado, customizado ou configurado, para ser utilizado em um contexto específico".

Ainda de acordo com [58], um **ponto de variação** é o local do artefato de software em que uma decisão de projeto pode ser tomada, e variantes são alternativas de projeto associadas a este ponto.

Quanto mais alta a capacidade de um artefato ser modificado, mais ele é reutilizável, podendo adaptar-se a várias necessidades diferentes. Assim, uma alta variabilidade permite que detalhes de implementação ou funcionalidades de produtos sejam facilmente modificados ou incluídos em sistemas de software baseados em LPS.

Existem na literatura diversos mecanismos de implementação de variabilidade. Segundo Anastasopoulos *et al.* [9], a implementação das variabilidades depende muito da escolha de uma linguagem de programação e de uma documentação muito bem definida dos artefatos da LPS.

Existem também diferentes tempos de ligação (do inglês *binding time*) para resolver as variabilidades, como em tempo de compilação ou execução [54]. Algumas das técnicas de implementação de variabilidade propostas são:

- **Agregação/Delegação:** A variabilidade pode ser tratada colocando-se a funcionalidade obrigatória no objeto que delega e a variante no objeto delegado;
- **Herança:** Adiciona funções básicas às superclasses e funções especializadas às subclasses;
- **Parametrização:** O comportamento do componente é determinado pelos valores dos parâmetros definidos;
- **Compilação Condicional:** Realizada em tempo de compilação, este mecanismo possibilita o controle sobre os segmentos de código a serem incluídos ou excluídos da compilação de um programa;
- **Reflexão:** Nesta técnica o programa manipula, na forma de dados, algo que representa o estado de um programa durante sua própria execução.

2.1.7 Abordagens de Adoção de Linhas de Produtos de Software

De acordo com Krueger [36] o projeto de implantação de uma LPS pode ser abordado de três maneiras:

- **Proativa:** Nesta abordagem a organização analisa, projeta e implementa uma LPS para suportar todo o escopo de produtos necessários. A partir da análise e do projeto são implementados os artefatos de software (componentes, arquitetura da linha, definição de processo, definição do escopo e requisitos, entre outros) para a LPS.
- **Reativa:** Nesta abordagem a organização cresce sua linha de produtos incrementalmente conforme a demanda de novos produtos ou novos requisitos em produtos existentes. Conforme a necessidade, os artefatos são construídos, utilizados nos produtos já existentes e também alimentando o núcleo de artefatos da LPS. Esta abordagem oferece uma transição para LPS mais rápida e menos dispendiosa do que a proativa.

- **Extrativa:** Nesta abordagem a organização tira proveito de sistemas de software existentes, extraindo o código comum e variável para a LPS. Esta abordagem permite à organização adotar a LPS muito rapidamente.

A abordagem extrativa é apropriada nos casos em que se têm artefatos de sistemas existentes que podem ser reutilizados. Ela é ainda mais apropriada quando os sistemas têm uma quantidade significativa de partes em comum e também diferenças consistentes entre eles [36]. Além disso, não é necessário realizar extração de todos os sistemas preexistentes de uma só vez. As extrações podem ser feitas incrementalmente. Após a linha de produção ter sido populada, as instâncias dos produtos são criadas. Por estes motivos, a abordagem extrativa foi a escolhida para a realização deste projeto.

2.1.8 Método PLUS

Existem na literatura diversas metodologias de desenvolvimento de LPS, fundamentadas em diferentes abordagens, como abordagens baseadas em pontos de vista, em metas, em metas e cenários, em casos de uso, e as orientadas a características. O método PLUS é baseado em casos de uso, e será utilizado neste trabalho.

O método PLUS (*Product Line UML-Based Software Engineering*) [32] é baseado em UML (*Unified Modeling Language*) e composto pelo processo ESPLEP (*Evolutionary Software Product Line Engineering Process*) que apresenta uma perspectiva para o desenvolvimento de LPS. Nesta metodologia o objetivo é explicitar as variabilidades e os pontos em comuns de uma LPS.

A variabilidade é descrita através de casos de uso e modelos de características. O PLUS é composto das etapas: **Modelagem de Requisitos**, **Análise**, **Projeto** e **Engenharia da Aplicação de Software**. As etapas são brevemente descritas a seguir.

Modelagem de Requisitos

Esta etapa consiste na concepção dos artefatos: modelo de casos de uso, modelo de características e a relação entre os modelos de casos de uso e o modelo de características. Os casos de uso são classificados em três tipos, conforme segue:

- **Casos de uso obrigatórios:** Necessários para todos os membros da LPS;
- **Casos de uso opcionais:** Necessários para apenas alguns membros da LPS;
- **Casos de uso alternativos:** Sugere a existência de alternativas e que pelo menos um caso de uso alternativo deva ser selecionado.

A variabilidade pode estar presente nos atores, que também podem ser obrigatórios, opcionais ou alternativos. O modelo de características, conceito chave em LPS, especifica

características comuns, opcionais e alternativas, assim como uma forma de representação das características em UML. Com o modelo de características e o modelo de casos de uso prontos, então é realizado o relacionamento entre eles.

Etapa de Análise para LPS

Esta etapa é composta de quatro modelos:

- **Modelo estático:** Define as entidades estáticas e seus relacionamentos na LPS;
- **Modelo dinâmico:** Cada caso de uso terá sua representatividade em um modelo dinâmico interativo, criado a partir de diagramas UML, para representar a interatividade e as mensagens do sistema;
- **Modelo dinâmico de máquina de estados:** Consiste na criação de máquinas de estado, que definem os estados de cada entidade, o ciclo de vida de cada uma delas e proveem um meio de interpretação para elas;
- **Modelo de dependência de características:** Determina as dependências de cada classe com as características dos sistemas.

Etapa de Projeto para LPS

Esta etapa é composta de duas fases:

- **Arquitetura de software:** Define a estrutura arquitetural e os padrões de projeto para o desenvolvimento da LPS (diagrama de pacotes).;
- **Projeto de software baseado em componentes:** A LPS é dividida em componentes e são definidas as interfaces de comunicação entre eles.

Engenharia da Aplicação de Software

É nesta etapa que são desenvolvidos dos membros da LPS, utilizando os modelos desenvolvidos nas etapas anteriores.

2.2 Modelo de Características e o Método FArM

Uma **característica** (do inglês *feature*) é um aspecto importante visível ao usuário, qualidade ou característica de um software ou sistema. As características definem os aspectos comuns do domínio, bem como diferenças entre os sistemas relacionados no domínio. Gurp *et al.* [58] definem característica como sendo uma unidade lógica de comportamento que corresponde a um conjunto de requisitos funcionais e de qualidade.

Originalmente proposto na fase de Engenharia de Domínio, como parte do método FODA (*Feature-Oriented Domain Analysis*), um **modelo de características** (do inglês *feature model*) representa as características de uma família de produtos em um domínio, as relações entre elas e suas semelhanças e diferenças (variabilidades). Ele é usado para definir e gerenciar os pontos comuns e as variabilidades em LPS. A construção do modelo de características é o primeiro passo para interpretar e ordenar os requisitos [58].

De acordo com Kang *et al.* [35] as características podem ser classificadas como **mandatórias**, **opcionais** e **alternativas**. A Figura 2.2 apresenta um exemplo de modelo de características de um carro.

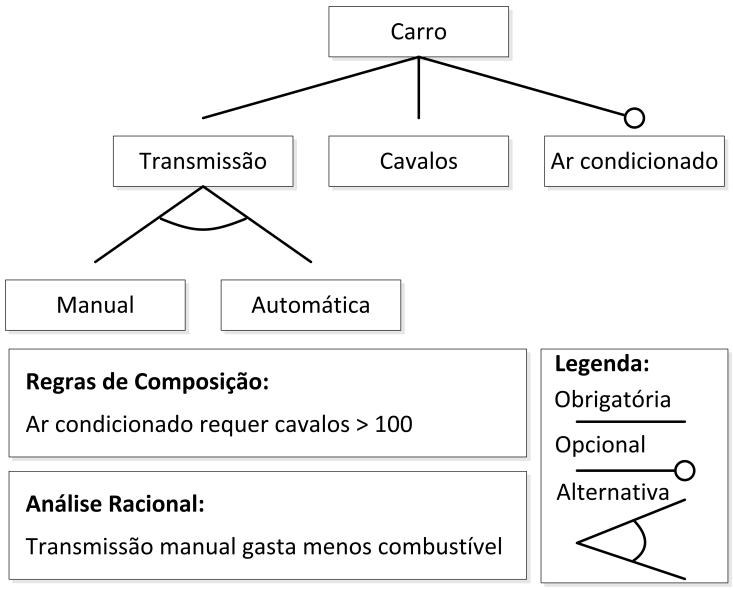


Figura 2.2: Modelo de características de um carro

Na figura temos representadas três características: **Transmissão**, **Cavalos** e **Ar condicionado**, sendo que as duas primeiras são obrigatórias e a última opcional. As características alternativas **Manual** e **Automático** são especialidades da característica **Transmissão**. A **regra de composição** define a semântica existente entre as características. Neste exemplo, a regra de composição impõe que para que o carro tenha ar condicionado, sua potência deve ser maior que 100 (Ar condicionado requer cavalos maior que 100). Já a **análise racional** é apenas uma recomendação sobre uma ou mais características. No exemplo, é dito que a transmissão manual é mais econômica (Transmissão manual gasta menos combustível).

Existem alguns métodos encontrados na literatura que propõem diretrizes para o mapeamento de características para elementos arquiteturais. Em alguns desses métodos, como por exemplo no trabalho de Zhang *et al.* [61], cada característica é mapeada para

um único elemento arquitetural. Em outros, o modelo de características é refinado antes de realizar o mapeamento, como no método *Feature-Architecture Mapping* (FArM) [53], utilizado neste trabalho.

O método FArM propõe quatro transformações aplicadas no modelo de características, visando criar uma arquitetura inicial (baseada em componentes) de LPS. A Figura 2.3 mostra as quatro transformações propostas pelo método FArM, seus artefatos de entrada e a arquitetura inicial da LPS como saída.

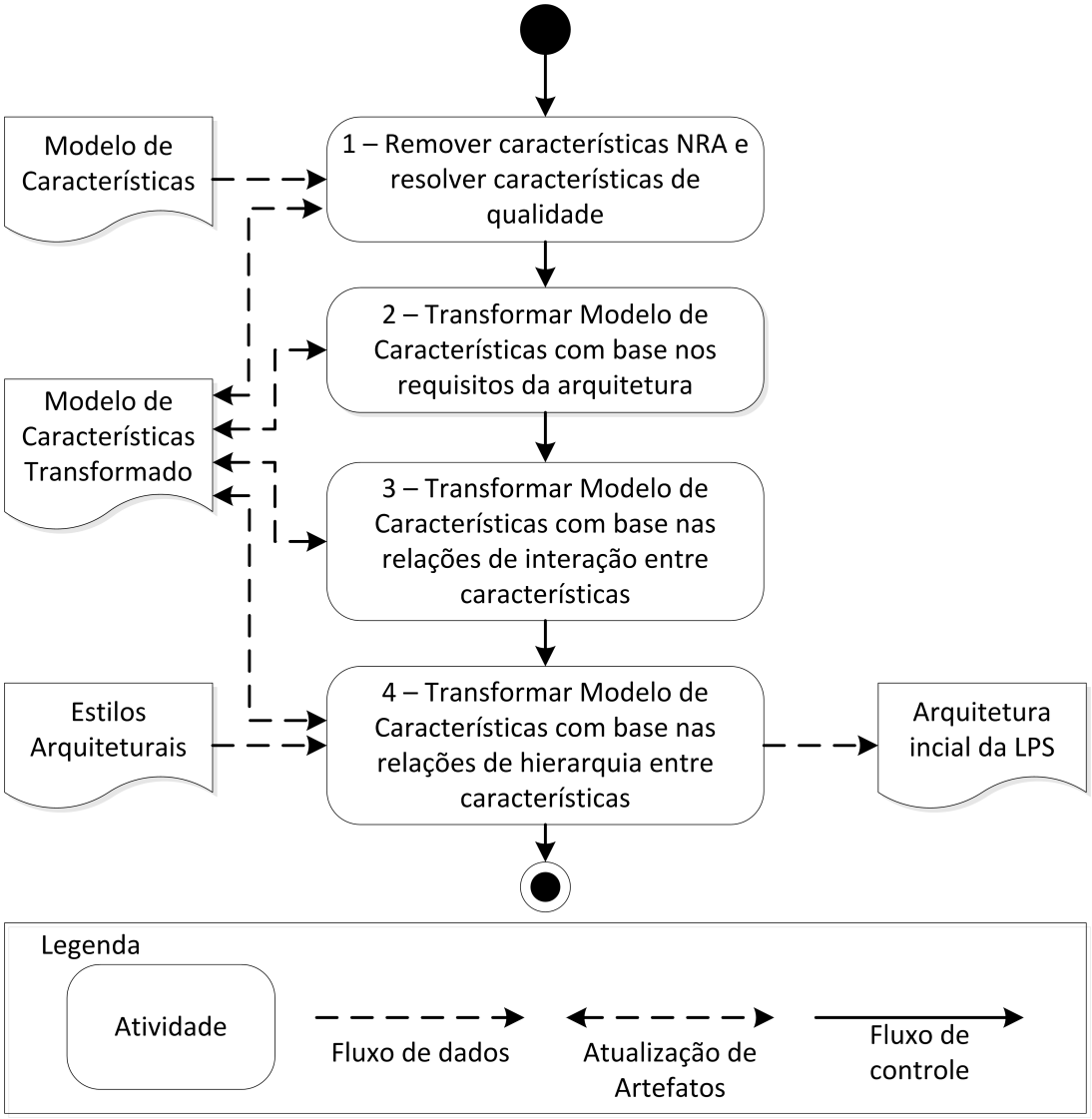


Figura 2.3: Transformações do método FArM e seus artefatos de entrada e saída

A primeira entrada do FArM é o modelo de características. Na primeira transformação proposta, este modelo inicial dará origem a um modelo de características transformado. As

duas próximas transformações atualizam este modelo e finalmente, na quarta transformação, combinado a estilos arquiteturais, ele dá origem ao modelo de arquitetura inicial da LPS. Cada uma das transformações é descrita a seguir. Utiliza-se como exemplo de cada transformação trechos de um modelo de características desenvolvido para um telefone celular.

2.2.1 Remoção de Características NRA e Resolução de Características de Qualidade

características não relacionadas à arquitetura (NRA) são aquelas que não são consideradas requisitos significantes para a arquitetura. O método FArM classifica as NRA em duas categorias: (1) características físicas e (2) características de negócio. Na primeira transformação do FArM essas características são removidas do modelo de características inicial. Após a remoção das NRA, é necessário identificar quais características representam requisitos de qualidade e integrá-las com aquelas que representam requisitos funcionais. Caso não existam características funcionais passíveis de serem integradas com as de qualidade, pode-se incluir uma nova característica funcional para a integração. O objetivo desta integração é manter a representação das características de qualidade na arquitetura.

A parte esquerda da Figura 2.4 mostra um trecho do modelo de características inicial, e a parte direita o modelo após a primeira transformação.

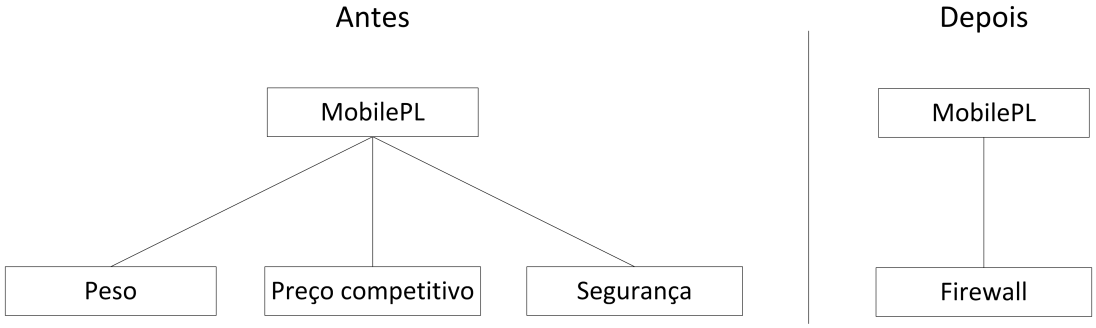


Figura 2.4: Modelo de características antes e depois da primeira transformação do FArM

As características **Peso** e **Preço competitivo** são NRA física e de negócios, respectivamente. Elas foram removidas do modelo. **Segurança** representa um requisito de qualidade. Para resolver esta característica foi adicionada a nova característica funcional **Firewall**.

2.2.2 Transformação Baseada nos Requisitos Arquiteturais

A segunda transformação é baseada em requisitos arquiteturais e tem como objetivo contrabalancear a influência que usuários finais possuem na especificação do modelo de características. O método sugere que requisitos arquiteturais que não tenham sido representados no modelo de características, mas que exercem influência sobre a arquitetura, sejam representados explicitamente como características e sejam integrados com características existentes ou com novas características criadas para representá-los, assim como ocorre com as características de qualidade.

A Figura 2.5 mostra um exemplo desta transformação.

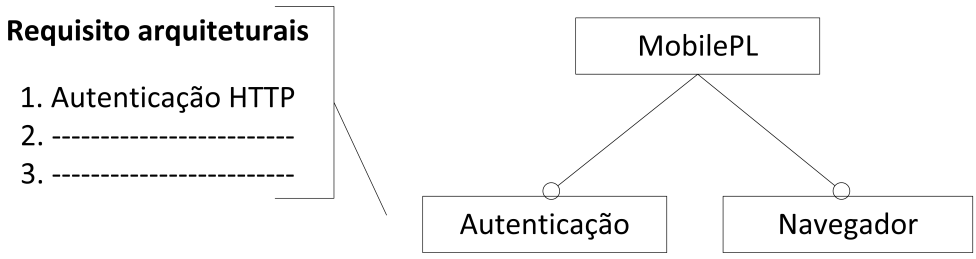


Figura 2.5: Modelo de características depois da segunda transformação do FArM

Supondo que o telefone celular possua acesso à internet, **Autenticação HTTP** é um requisito arquitetural que não aparece no modelo de características. Refere-se à autenticação necessária para acessar algumas páginas na internet. Para reduzir o tráfego de dados pela rede, foi adicionada a característica **Autenticação** que armazena senhas e usuários, e que são automaticamente passados para as páginas quando requeridos. Caso a autenticação falhe, é necessário exibir uma mensagem no navegador. Para tanto, esta parte do requisito foi integrada à característica existente **Navegador**.

2.2.3 Transformação Baseada nas Relações de Interação entre Características

A terceira transformação é com base nas relações de interação entre características. São especificadas as comunicações entre as características introduzindo relações de interação entre elas. O FArM classifica essas relações como três tipos:

- **Tipo 1 - usa:** Relaciona duas características onde uma usa a funcionalidade de outra;
- **Tipo 2 - modifica:** Relaciona duas características onde o funcionamento correto de uma altera o comportamento de outra;

- **Tipo 3 - contradiz:** Relaciona duas características onde o funcionamento correto de uma contradiz o comportamento correto de outra.

Esses três tipos de relações devem ser adicionados ao modelo de características transformado. A Figura 2.6 mostra um exemplo de cada tipo de relação de interação.

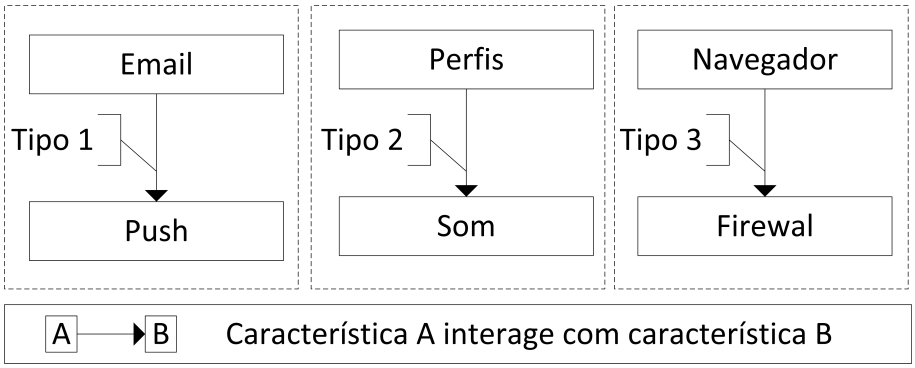


Figura 2.6: Tipos de relacionamento de interações entre características

Neste exemplo, a característica **Email** usa a funcionalidade *Push* para receber emails automaticamente (relação tipo "usa"). A presença da característica **Perfis** indica que uma alteração de configuração, por exemplo o modo silencioso, faz com que a característica **Som** seja alterada (relação tipo "modifica"). A presença da característica *Firewal* previne o **Navegador** de realizar downloads de arquivos executáveis, por exemplo (relação tipo "contradiz").

Após esta etapa, é feita uma nova transformação com base em dois critérios:

- **Critério 1:** O tipo de relação entre as características;

- **Critério 2** O número de relações de interação entre características.

Baseado nesses critérios, características devem ser integradas ou novas características podem ser adicionadas. Um exemplo de relação que segue o primeiro critério é por exemplo a relação entre **Navegador - Firewall**, onde uma característica pode contradizer a outra. Assim, a integração entre essas duas características pode resolver a contradição. A Figura 2.7 mostra um exemplo de transformação baseada no segundo critério. Neste exemplo a característica **Interface com usuário** tem um grande número de interações e depende de outras características, como por exemplo **Email**. Desta maneira, ela é integrada a cada característica que a utiliza.

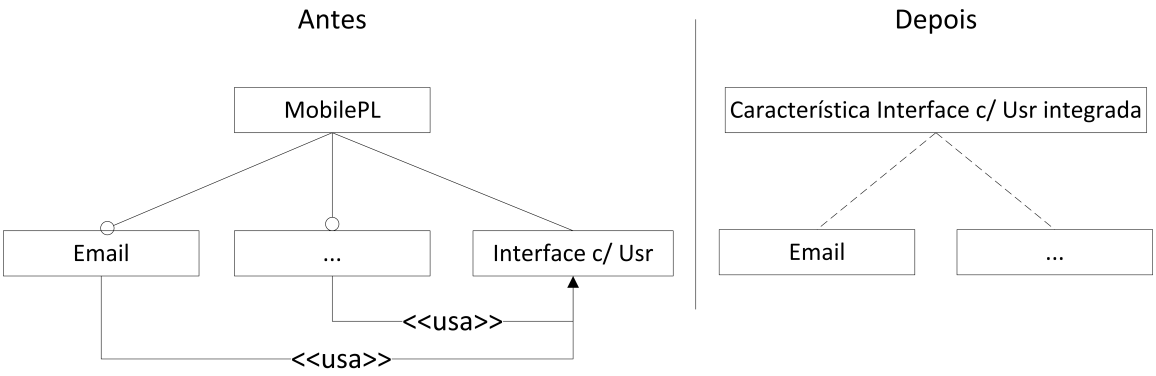


Figura 2.7: Modelo de características depois da terceira transformação do FArM

2.2.4 Transformação Baseada nas Relações de Hierarquia entre Características

A última transformação proposta pelo método FArM trata as características com base em suas relações de hierarquia. São dois os tipos de características: (1) supercaracterística e (2) subcaracterística. O FArM define três tipos de relações de hierarquia:

- **Tipo especialização:** Uma subcaracterística especializa uma supercaracterística;
- **Tipo agregação:** Uma subcaracterística é parte de uma supercaracterística;
- **Tipo alternativa:** As subcaracterísticas representam alternativas para a supercaracterística;

A Figura 2.8 mostra exemplos desses tipos de relação de hierarquia. As subcaracterísticas **Email** e **SMS** são especializações da supercaracterística **Mensagem**. As relações entre as subcaracterísticas são do tipo alternativa.

O método propõe que todas as relações de hierarquia do modelo de características transformado sejam de um dos três tipos definidos. Relações inválidas devem ser removidas e novas relações podem ser criadas. Feito isso, as características são mapeadas para componentes da seguinte maneira: as supercaracterísticas são mapeadas para componentes que proveem acesso às funcionalidades dos componentes que implementam as subcaracterísticas. Neste exemplo, a supercaracterística **Mensagem** foi mapeada para o componente **Mensagem** que oferece acesso às funcionalidades dos componentes que implementam as características **Email** e **SMS**. A variabilidade das características e seus relacionamentos são mantidos nos componentes.

A fim de criar uma arquitetura inicial da LPS, são utilizados estilos arquiteturais de referências (detalhados na Seção 2.4) para especificar as relações entre os componentes

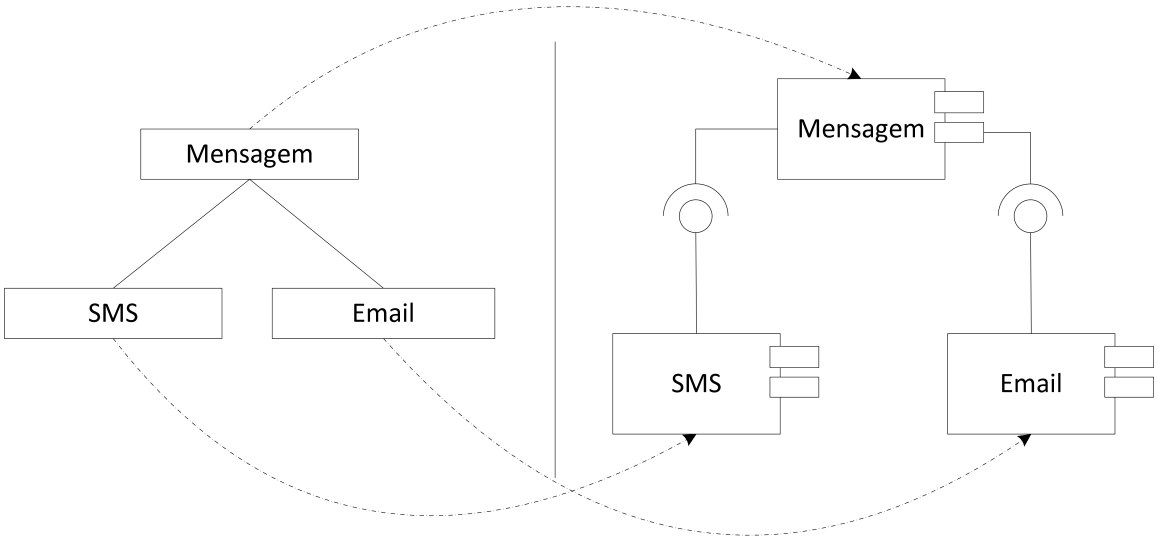


Figura 2.8: Mapeamento de características para componentes

identificados. Esta arquitetura não é a final da LPS pois o método não prevê a identificação das interfaces, nem a especificação da assinatura de seus métodos.

2.3 Reengenharia de Software

A dificuldade em atualizar sistemas de software para a utilização de novas tecnologias tem motivado pesquisadores a buscar soluções que garantam um tempo de vida maior para o sistema, diminuam custos de desenvolvimento e facilitem sua manutenção [27]. Uma das maneiras utilizadas é melhorar ou transformar um sistema existente de forma que ele possa ser entendido, controlado e usado novamente. Este é o princípio da **reengenharia de software**. Ela é realizada quando um sistema é usado regularmente, mas possui erros constantes e um alto custo de manutenção [47]. A reengenharia de software é importante para a recuperação e reutilização de artefatos existentes, controlando os custos de manutenção, e para estabelecer uma base para a evolução do sistema [51].

Basicamente, a reengenharia é refazer um sistema legado existente que tornou-se caro para manter, ou cuja arquitetura ou implementação são obsoletos. A dificuldade está na compreensão do sistema existente. Normalmente os sistemas legados não possuem documentação de requisitos, de projeto e de código, ou são muito desatualizadas.

O primeiro passo da reengenharia é a **engenharia reversa**, que identifica os componentes do software legado e os representa em um nível mais alto de abstração, para fins de re-documentação ou de recuperação do projeto [20]. Na engenharia reversa, os

requisitos, estrutura, projeto, conteúdo e regras de negócio do sistema legado devem ser recapturados. Os principais objetivos da engenharia reversa são: gerar visões alternativas, recuperar informações perdidas, detectar efeitos colaterais, sintetizar abstrações elevadas e facilitar a reutilização de artefatos. A engenharia reversa não envolve alterações no sistema legado ou a criação de um novo sistema.

Após a engenharia reversa, as informações extraídas do sistema legado são utilizadas para o projeto, documentação e desenvolvimento do novo sistema. Esta etapa é chamada de **engenharia avante**. A principal desvantagem da abordagem de reengenharia é o risco de fracasso. O novo sistema não deve conter erros que afetem funcionalidades do usuário [43].

A abordagem extrativa de adoção de LPS, vista na Seção 2.1.8, utiliza sistemas de software já existentes. Os sistemas passam pelo processo de engenharia reversa para serem entendidos e para que sejam extraídos seus requisitos, e depois pelo processo de engenharia avante, durante a concepção da LPS. O arcabouço proposto por Kang *et al.* propõe a abordagem de reengenharia para a adoção extrativa de LPS.

2.3.1 Reengenharia Orientada a Características

Kang *et al.* propôs o arcabouço Reengenharia Orientada a características (ROC) [34] para o desenvolvimento extrativo de LPS orientado a características, apresentando um estudo de caso utilizando aplicações da indústria. O arcabouço, exibido na Figura 2.9, é composto por quatro fases, explicadas a seguir.

As duas primeiras fases corresponde à engenharia reversa dos sistemas legados. Na fase 1, **Recuperação da Arquitetura**, foi realizada a extração de componentes e informações arquiteturais do sistema legado. Na fase 2, **Modelagem de características**, baseado nas informações recuperadas e no conhecimento do domínio, foi gerado um modelo de características da aplicação legada.

As três próximas fases correspondem à engenharia da LPS. Considerando possíveis oportunidades de negócio e alterações de tecnologia, na fase 3, chamada **Refinamento do modelo de características**, os modelos de características dos sistemas legados foram refeitos, adicionando novas características e informações de variabilidade, dando origem ao modelo de características refinado. Na fase de **Projeto da LPS**, fase 4, com base no modelo de características refinado e princípios de engenharia, foram desenvolvidos os artefatos e projetada a nova arquitetura e componentes para a LPS. Na fase 5, **Implementação da LPS**, a LPS é desenvolvida.

De acordo com os autores, a abordagem utilizada pode diminuir custos de desenvolvimento e aumentar a flexibilidade do software.

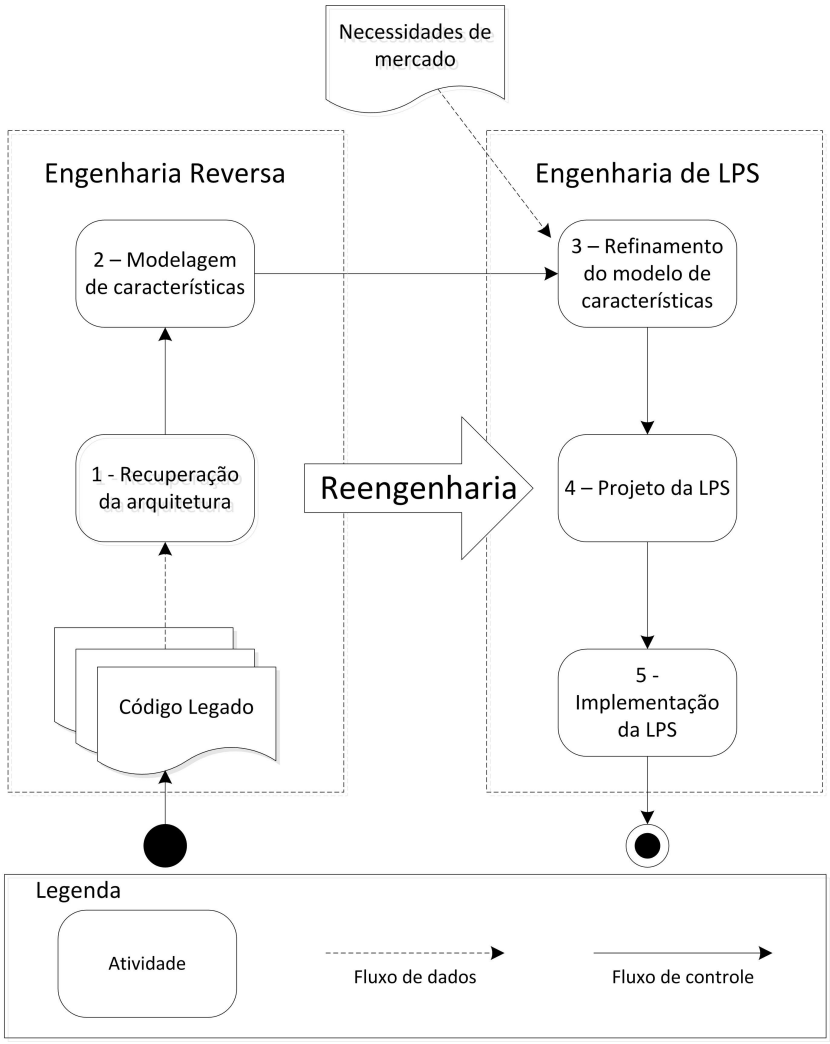


Figura 2.9: Arcabouço da reengenharia orientada a características

2.4 Arquitetura de Software

Arquitetura de software engloba estruturas de sistemas de grande porte. A visão de arquitetura de um sistema é abstrata, não se preocupa com detalhes do algoritmo, implementação, e representação de dados, concentrando-se no comportamento e interação entre elementos vistos como "caixas-pretas".

Ela é desenvolvida como o primeiro passo para se projetar um sistema que tem uma coleção de propriedades desejadas. Pode-se pensar na arquitetura como uma estrutura ou um conjunto de estruturas do sistema, que incluem elementos de software, as propriedades visíveis externamente desses elementos e as relações entre eles [18].

A arquitetura de software define uma estrutura de alto nível do sistema em termos de elementos arquiteturais, abstraindo detalhes de sua implementação [15], [55]. Um **componente arquitetural** é responsável pelo processamento e armazenagem de dados relacionados às funcionalidades do sistema [45]. Um **conector arquitetural** é responsável por intermediar a comunicação entre componentes arquiteturais e um conjunto de componentes arquiteturais ligados por conectores arquiteturais define uma configuração arquitetural [37].

Aqui serão brevemente descritos três modelos de arquitetura muito utilizados em sistemas de informação: arquitetura em camadas, arquitetura *pipe-and-filter* e arquitetura *call-and-return*.

2.4.1 Arquitetura em Camadas

De acordo com Bachmann *et al.* [13] a **arquitetura em camadas** é um dos tipos de arquitetura mais utilizados. Assim como qualquer outra arquitetura, ela reflete uma divisão do software em unidades. Neste caso, as unidades são as camadas.

Cada camada representa uma máquina virtual, que é uma coleção de software que, juntos, fornecem um conjunto coeso de serviços que outro software pode utilizar sem saber como esses serviços são implementados. Além disso, cada camada é capaz de interagir apenas com sua camada imediatamente anterior.

O objetivo da divisão em camadas é definir máquinas virtuais que são pequenas o suficiente para serem entendidas, mas abrangentes o suficiente para que prováveis mudanças afetem apenas uma camada.

2.4.2 Arquitetura *Pipe-and-Filter*

Em um sistema com arquitetura *pipe-and-filter* cada componente possui um conjunto de entradas e um conjunto de saídas. Cada componente lê fluxos de dados em suas entradas e produz fluxos de dados em suas saídas. Os componentes são denominados **filtros** (do

inglês *filter*). As saídas de um filtro são transmitidas para as entradas de outros filtros através dos **conectores** (do inglês *pipe*). O exemplo mais conhecido de arquitetura *pipe-and-filters* são os programas escritos no *shell* Unix [18].

Os sistemas com arquitetura *pipe-and-filter* possuem diversas vantagens, como por exemplo, a facilidade de compreensão do sistema dado que este pode ser visto como uma simples composição de filtros individuais; a facilidade de reuso, uma vez que os filtros necessitam apenas especificar suas entradas e saídas; e a facilidade de manutenção e evolução do sistema, através de trocas de filtros.

Porém, as regras de comunicação existentes nos conectores (necessárias para padronizar a comunicação e facilitar a modificação dos filtros) podem ser uma desvantagem, pois podem exigir que os filtros modifiquem seus fluxos de entrada e saída para respeitarem estas regras, afetando o desempenho e a construção dos filtros. Além disso, existem dependências entre fluxos de dados, sendo necessário que os filtros tratem as correspondências entre os fluxos.

2.4.3 Arquitetura *Call-and-Return*

O estilo arquitetural *call-and-return* vem sendo o estilo dominante de sistemas de grande porte nos últimos 30 anos [2]. Um sistema de arquitetura *call-and-return* é decomposto em pequenos "pedaços"a fim de melhor lidar com a complexidade e alcançar modificabilidade.

Sistemas que seguem este estilo arquitetural geralmente possuem um único componente de controle. Cada um dos componentes do sistema é executado apenas quando recebe o controle de outro componente, e assim que sua execução termina, o controle volta ao componente anterior. Durante a execução, um componente pode passar o controle a outro componente.

Cada componente de um sistema *call-and-return* é chamado de sub-rotina. O componente que tem o controle total do sistema é chamado de programa principal.

2.5 Desenvolvimento Baseado em Componentes

A ideia de reutilizar partes de sistemas já existentes surgiu para atacar problemas como diminuir custos e manejar a complexidade no desenvolvimento de software, e melhorar a qualidade dos sistemas produzidos [39].

A divisão de um sistema em módulos foi a base para o surgimento de uma metodologia de desenvolvimento fundamentada na composição de componentes de software chamada **desenvolvimento baseado em componentes** (DBC) [46]. O DBC se baseia na construção rápida de sistemas a partir de componentes pré-fabricados [12]. Um dos seus principais objetivos é a redução de custo e tempo de desenvolvimento [1]. O DBC está

cada vez mais ganhando adeptos e é uma das propostas promissoras para a melhoria da produção de software nos próximos anos [60].

Na literatura existem várias definições para um **componente de software**, e a definição atualmente mais encontrada em trabalhos acadêmicos é a definição dada por Clemens Szyperski [1]:

"Um componente é uma unidade de composição com interfaces especificadas contratualmente e com dependências de contexto explícitas; um componente de software pode ser implantado de forma independente ou se combinar com outros".

Uma **interface** de um componente identifica um ponto de interação entre um componente e o seu ambiente [30]. Um componente possui **interfaces providas**, por meio das quais ele declara os serviços oferecidos ao ambiente, e **interfaces requeridas**, pelas quais ele declara os serviços do ambiente dos quais depende para funcionar [30]. A Figura 2.10 mostra um exemplo da representação de um componente e suas interfaces.

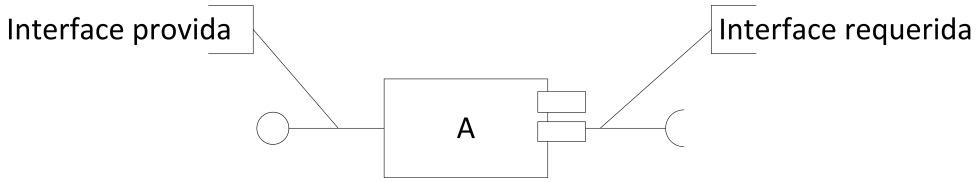


Figura 2.10: Interfaces provida e requerida do componente A

O objetivo do DBC é a definição de componentes interoperáveis, com interfaces bem definidas, evidenciando os tipos de relacionamentos permitidos por este componente. Desta forma, a complexidade no desenvolvimento é reduzida, assim como os custos, através da reutilização de componentes já testados [60].

O DBC permite que uma aplicação seja construída pela composição de componentes de software que já foram previamente especificados, construídos e testados, o que resulta em ganho de produtividade e qualidade no software produzido. O aumento da produtividade advém da reutilização de componentes existentes em novos sistemas, enquanto o aumento da qualidade advém do fato do componente já ter sido utilizado, modificado e testado em outros sistemas, eliminando boa parte dos erros que poderiam ocorrer. Com isso, os produtos finais não são mais produtos de software monolíticos e fechados, mas baseados em componentes que podem ser integrados com outros produtos [46].

2.5.1 O Processo UML *Components*

O UML *Components* [19] é um processo de desenvolvimento de sistemas de software baseado em componentes, visando o reuso de componentes e a facilidade de manutenção e modificabilidade do sistema, obtendo assim um ganho de produtividade e qualidade. Em sistemas baseados em componentes, o principal objetivo é a facilidade de troca de um componente por outro. Daí vem a importância da arquitetura do sistema e a gerência de seus componentes.

A fim de simplificar o processo de desenvolvimento, o UML *Components* adota uma arquitetura pré-definida em cinco camadas, enfatizando duas camadas: (i) camada de sistema, que contém os componentes que implementam os cenários específicos do sistema; e (ii) camada de negócio, que contém os componentes identificados a partir do domínio do negócio e, por isso, são os principais candidatos a serem reutilizados em diferentes aplicações.

No processo UML *Components*, o desenvolvimento é dividido em fases, brevemente descritas a seguir.

A primeira fase é a de especificação de requisitos, onde o desenvolvedor especifica os requisitos funcionais do sistema utilizando um conjunto de casos de uso. Nesta fase também é especificado o modelo conceitual de negócio, que representa as entidades básicas do domínio com o qual o sistema está relacionado.

A fase seguinte, a mais detalhada pelo processo, é a especificação dos componentes. Esta fase é responsável pelo projeto da arquitetura do sistema, incluindo a identificação de interfaces, componentes, conectores e configurações arquiteturais. Essa fase é dividida em três subfases: (i) identificação de componentes, responsável por identificar os componentes e as interfaces entre eles; (ii) interação dos componentes, responsável por especificar as configurações arquiteturais e novas operações do sistema; e (iii) especificação final dos componentes, responsável por refatorar interfaces e componentes, antes de finalizar o projeto da arquitetura.

Ao final da especificação da arquitetura de componentes, é executada a fase de provisão de componentes, que é responsável pela aquisição dos componentes, seja por meio da implementação, reutilização interna da própria empresa, ou aquisição de terceiros. Em seguida, na fase de montagem, os conectores especificados são implementados e os componentes conectados para materializar a configuração arquitetural.

O processo UML *Components* propõe diagramas utilizados para especificar os componentes e suas relações, entre eles o diagrama de comunicação, utilizado na fase 3.1.4: Projeto da LPS do método proposto. O diagrama de comunicação era conhecido como diagrama de colaboração até a versão 1.5 da UML, tendo seu nome modificado para diagrama de comunicação a partir da versão 2.0. As informações mostradas nesse diagrama são, com frequência, as mesmas apresentadas no diagrama de sequência, porém com um

enfoque diferente, uma vez que este diagrama não se preocupa com a temporalidade do processo, e concentra-se em como os objetos estão vinculados e quais mensagens trocam entre si durante o processo.

2.5.2 Modelo COSMOS*

COSMOS* (*Component Structuring Model for Object-oriented Systems*) [31] é um modelo de estruturação de componentes para mapeamento de arquiteturas de componentes em linguagens de programação. Ele define elementos para a implementação de componentes e conectores, participantes da composição de software. O modelo age, principalmente, na organização do sistema em termos de seus componentes, conectores e interações entre eles.

O modelo de estruturação de componentes proposto no COSMOS* preocupa-se principalmente em garantir reusabilidade (promovida pelo baixo nível de acoplamento do sistema, uma vez que os componentes interagem somente através de conectores, e um componente não tem conhecimento dos demais componentes), adaptabilidade (possibilidade de modificar ou substituir um componente, que participa de uma composição de software, isolando o maior número possível de modificações na implementação dos conectores) e modificabilidade (apenas a interface do componente é conhecida do usuário, possibilitando assim a modificação da implementação do componente sem afetar a sua utilização). Ele é independentemente de plataforma tecnológica para projeto e implementação de arquiteturas de software baseadas em componentes. Assim, é possível adaptar o modelo COSMOS* para o uso com C++, linguagem de desenvolvimento utilizada neste projeto.

Internamente, o componente é dividido em duas partes: a especificação e a implementação. A especificação é a visão externa do componente, que também é subdividida em duas partes: a que especifica os serviços oferecidos e a que explicita as dependências de outros serviços. A implementação possui uma infraestrutura para instanciar o componente e implementar os serviços oferecidos pelas interfaces providas.

O modelo COSMOS* permite a definição recursiva de componentes internos a outros componentes e estabelece diretrizes para especificar e instanciar esses componentes sem quebrar o encapsulamento dos externos. Por fim, ele também oferece diretrizes para especificar um componente de sistema que possa ser implantado e executado, assim como as formas como esse componente deve ser empacotado e instanciado.

2.6 Resumo do Capítulo

Este capítulo apresentou os principais conceitos sobre LPS, reengenharia e reutilização de software, desenvolvimento baseado em componentes, arquitetura de software, além de

detalhar os métodos de adoção de LPS e geração de arquitetura que foram utilizados no projeto.

Com relação aos conceitos de LPS, o capítulo detalhou as três atividades essenciais para o desenvolvimento de LPS, definidos pelo SEI através da iniciativa PLP (Desenvolvimento do Núcleo de Artefatos, Desenvolvimento do Produto e Gerenciamento), discorreu sobre as vantagens e dificuldades e riscos da implantação de LPS, apresentou os conceitos de variabilidade de software, pontos de variação e como as variabilidades estão diretamente ligadas à LPS. Foi apresentado o método PLUS, método proativo de adoção de LPS.

O capítulo também apresentou o conceito de características em LPS e descreveu detalhadamente o método FArM, utilizado para a geração de arquiteturas baseadas em componentes. Com relação ao desenvolvimento baseado em componentes, seus principais conceitos foram apresentados e também o modelo COSMOS* de geração de componentes.

A reengenharia de software também foi abordada neste capítulo, e foi descrito o arcabouço ROC para adoção extrativa de LPS, que utiliza conceitos de reengenharia.

Ainda neste capítulo foram descritos três estilos arquiteturais: arquitetura em camadas, arquitetura *pipe-and-filter* e a arquitetura *call-and-return*. Os três estilos foram utilizados neste projeto. Foi detalhado também o método FArM, utilizado para a geração de uma arquitetura de LPS baseada em componentes.

Por fim, foram apresentados conceitos de desenvolvimento baseado em componentes, e o modelo COSMOS* utilizado no projeto.

Os conceitos apresentados serão utilizados no próximo capítulo, onde será apresentado o método proposto para a adoção extrativa de LPS e o uso da infraestrutura gerada. O método proposto tem como base o arcabouço ROC, utiliza técnicas de modelagem do método PLUS e aplica o método FArM para a geração de uma arquitetura inicial de LPS baseada em componentes.

Capítulo 3

Solução Proposta: UNIPAR-LPS

O sistema UNIPAR, conforme mencionado no Capítulo 1, apresenta grandes dificuldades de manutenção e atualização, o que acaba por dificultar o processo de pesquisa científica do grupo UNISIM. A fim de solucionar alguns dos problemas mencionados (código pouco modularizado, estrutura pouco flexível, muito código duplicado, levando a um custo de manutenção alto) e facilitar a organização e compartilhamento de dados utilizados em experimentos científicos, a solução proposta neste trabalho é uma infraestrutura de apoio à execução destes experimentos, que tem como base uma linha de produtos de software baseada em componentes, desenvolvida a partir do sistema legado UNIPAR, chamada de UNIPAR-LPS.

Com a LPS baseada em componentes, espera-se resolver os problemas de modularização e estrutura de código, e também eliminar a duplicação de código, uma vez que os componentes gerados poderão ser utilizados em diferentes produtos. A infraestrutura gerada dará apoio à geração dos *workflows* científicos (são eles que definem a ordem de execução das tarefas dos experimentos) e permitirá o compartilhamento de dados e resultados obtidos.

Na Seção 3.1 deste capítulo será apresentado o método proposto para o desenvolvimento de uma LPS a partir de um sistema legado. A Seção 3.3 discorre sobre a aplicação do método proposto no sistema UNIPAR e detalha a UNIPAR-LPS. Finalmente, a Seção 3.4 explica como resolver a variabilidade da UNIPAR-LPS para a geração dos produtos finais (os *workflows* científicos).

3.1 Método Extrativo Proposto para Adoção de LPS

Como já visto na Seção 2.1.8 do Capítulo 2, uma das maneiras de se adotar a engenharia de LPS é utilizar uma abordagem extrativa. Esta abordagem é apropriada quando existem sistemas customizados que podem ser reusados. Ela é ainda mais apropriada quando esses

sistemas tem quantidade significante de partes em comum e diferenças consistentes entre eles [36]. Neste trabalho propõe-se um método extrativo para o desenvolvimento de uma LPS baseada em componentes, implementada a partir de um sistema legado.

3.1.1 Visão Geral do Método Proposto

O método proposto neste trabalho é baseado na Reengenharia Orientada a características (ROC), vista na Seção 2.3.1, e também utiliza técnicas de modelagem do método PLUS (Seção 2.1.8) e do método FArM (Seção 2.2) para a criação da arquitetura inicial da LPS baseada em componentes. Para o desenvolvimento dos componentes é proposta a utilização do modelo COSMOS* (Seção 2.5.2).

Para facilitar a compreensão deste método, pode-se representá-lo através de quatro fases principais: (1) Engenharia Reversa do Sistema Legado, (2) Análise da LPS, (3) Projeto da LPS e (4) Implantação da LPS. A Figura 3.1 representa as fases do método.

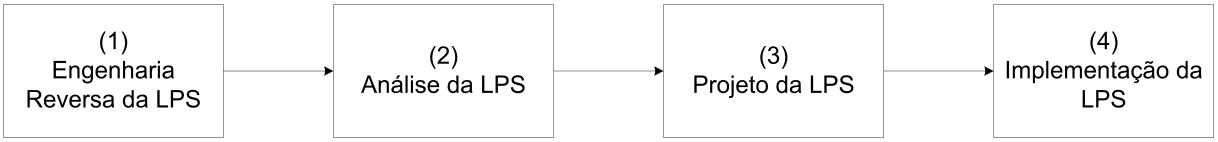


Figura 3.1: Visão geral do método proposto para adoção extrativa de LPS

Todos os artefatos de software gerados durante a execução do método proposto compõe o núcleo de artefatos. As próximas seções descrevem detalhadamente cada uma das fases do método.

3.1.2 Fase 1: Engenharia Reversa do Sistema Legado

De acordo com Chikofsky *et al.* [20], a Engenharia Reversa pode ser definida como um processo de análise de um sistema, a fim de identificar os componentes desse sistema e seus relacionamentos, e criar representações do sistema em uma outra forma ou em um nível mais alto de abstração.

A Engenharia Reversa objetiva a extração de documentação de projeto a partir do código, representando uma atividade ou um conjunto de técnicas que visam facilitar a compreensão de um sistema existente.

Neste trabalho, propõe-se a Fase 1: Engenharia Reversa, composta por duas atividades: (1.1) Recuperação da Arquitetura e (1.2) Modelagem de características dos sistemas legados. As duas são etapas propostas pelo arcabouço da Reengenharia Orientada a características, conforme descrito na Seção 2.3.1. A Figura 3.2 ilustra as duas etapas desta fase.

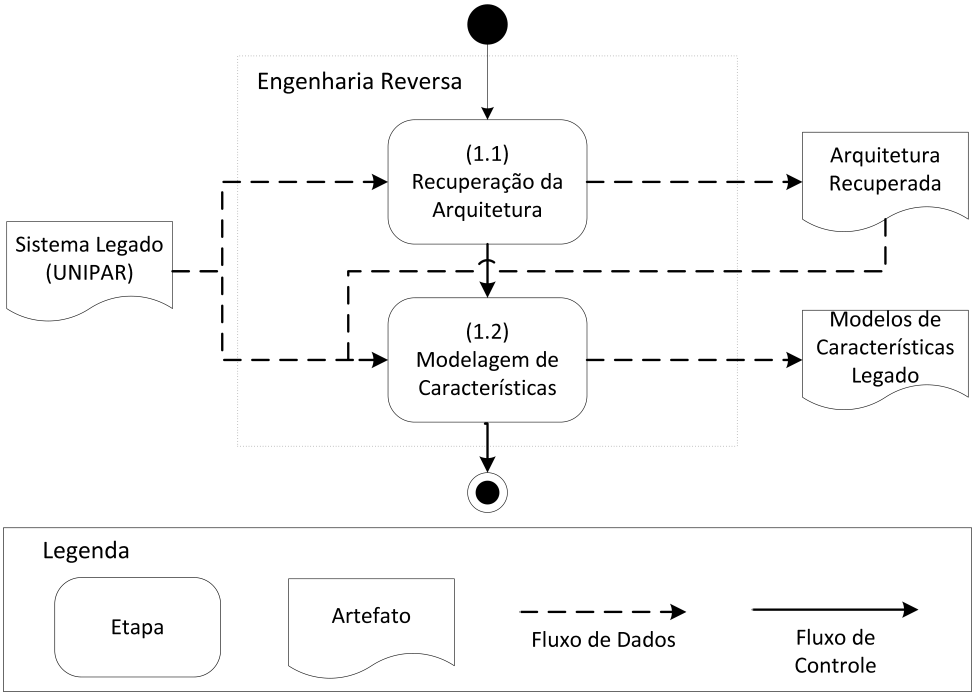


Figura 3.2: Atividades da fase de Engenharia Reversa

Em relação aos sistemas legados, a Engenharia Reversa é de grande utilidade pois esses sistemas costumam se encontrar em operação há anos nas organizações, sendo modificados ao longo do tempo sem que a sua documentação seja atualizada em paralelo. Assim, o código fonte acaba se tornando a única fonte de informação atualizada [50].

Fase 1.1: Recuperação da Arquitetura

A recuperação da arquitetura é uma atividade de Engenharia Reversa que visa a obtenção de uma arquitetura documentada para um sistema existente [57]. A descrição arquitetural de um sistema comunica decisões de projeto essenciais, as quais em relação aos sistemas legados, foram tomadas no passado. Recuperar a arquitetura requer, portanto, a recuperação de decisões passadas de projeto, levando o desenvolvedor a inferir informações arquiteturais que não estão imediatamente evidentes [50].

O método propõe a recuperação da arquitetura nesta fase para que o sistema legado seja melhor compreendido. A compreensão do sistema auxilia na sua manutenção, reutilização, e neste caso, para a migração para uma abordagem de engenharia de linhas de produtos. As arquiteturas legadas são analisadas e se possível utilizadas na implementação da LPS, mais precisamente para o desenvolvimento do núcleo de artefatos.

Fase 1.2: Modelagem de Características

A próxima atividade proposta é a geração de um modelo de características para cada um dos sistemas legados. Para a construção desses modelos, deve-se levar em conta informações obtidas na atividade anterior, Fase 1.1: Recuperação da Arquitetura, e pode ser necessário conhecimento acerca dos sistemas legados. Esses modelos serão utilizados na Fase 2: Análise da LPS, para a definição de quais características dos sistemas legados deverão fazer parte da LPS.

3.1.3 Fase 2: Análise da LPS

Esta fase foi adaptada da fase de Modelagem de Requisitos do método PLUS, descrito na Seção 2.1.8. A Figura 3.3 mostra as atividades da fase.

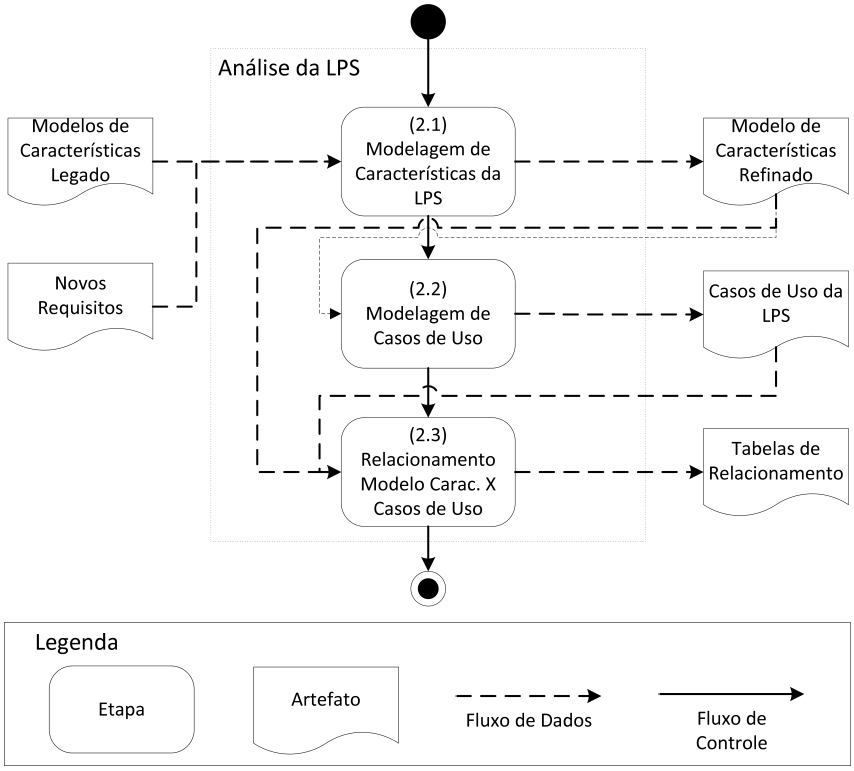


Figura 3.3: Fase de Análise da LPS

Durante esta fase, é desenvolvido um modelo de requisitos onde os requisitos funcionais do sistema são definidos em termos de atores e casos de uso. Ela é composta por três atividades: (2.1) Modelagem de características, (2.2) Modelagem de Casos de Uso e (2.3) Relacionamento entre características e Casos de Uso. O artefato de entrada desta fase é o modelo de características do sistema legado, gerado na Fase 1: Recuperação da Arquitetura. Como saídas da fase, são gerados os seguintes artefatos: um modelo de

características refinado, modelos de casos de uso, e é criado um relacionamento entre os casos de uso e as características.

Fase 2.1: Modelagem de Características

Nesta fase, o objetivo é gerar o modelo de características da LPS. Para isso, propõe-se que os modelos de características dos sistemas legados gerados na Fase 1 sejam analisados e, a partir deles, seja construído o novo modelo de características. O modelo de características da LPS deve conter todas as características dos sistemas legados, além de poder conter novas características (necessidades de mercado).

Fase 2.2: Modelagem de Casos de Uso

De acordo com Gomma [32], a abordagem de caso de uso tem sido muito bem sucedida para modelar os requisitos funcionais de sistemas únicos. Todos os requisitos de software que foram especificados para um sistema único devem ser fornecidos pelo sistema. Já em uma LPS, como já visto no Capítulo 2, apenas a alguns requisitos são comuns a todos os membros da família. Para especificar os requisitos funcionais de uma LPS, é importante capturar os requisitos comuns da LPS, ou seja, os requisitos que são comuns a todos os membros da família, os requisitos obrigatórios, bem como os requisitos opcionais e alternativos.

Ao estender a abordagem de modelagem de casos de uso para especificar os requisitos funcionais de LPS, definem-se diferentes tipos de casos de uso: casos de uso do tipo *kernel*, que são necessários a todos os membros da linha de produtos; casos de uso do tipo opcionais, que são necessários por apenas alguns membros da linha de produtos, e os casos de uso do tipo alternativos, onde diferentes casos de uso são necessários por diferentes membros da linha de produtos. Para distinguir os casos de uso obrigatórios, opcionais e alternativos, são utilizados os estereótipos «kernel», «optional» e «alternative», respectivamente.

Fase 2.3: Relacionamento entre Casos de Uso e Modelo de características

Tanto os casos de uso quanto o modelo de características representam requisitos do sistema. O método PLUS sugere que sejam criadas tabelas para representar os relacionamentos entre os casos de uso e as características, que não são necessariamente de um-para-um. Eventualmente um caso de uso pode se associar a mais de um elemento no modelo de características e vice-versa.

A rastreabilidade da variabilidade entre os dois modelos facilita a manutenção e evolução da variabilidade [17], uma vez que ao alterar um caso de uso sabe-se exatamente qual ou quais características são afetadas com esta alteração. Ou seja, o relacionamento feito entre os casos de uso e características são fundamentais para a evolução da LPS.

3.1.4 Fase 3: Projeto da LPS

Assim como a fase de Projeto da LPS do método PLUS, visto na Seção 2.1.8, esta fase do método proposto trata da geração da arquitetura da LPS e do seu projeto baseado em componentes. A Figura 3.4 mostra as atividades e os artefatos de entrada e saída de cada uma delas.

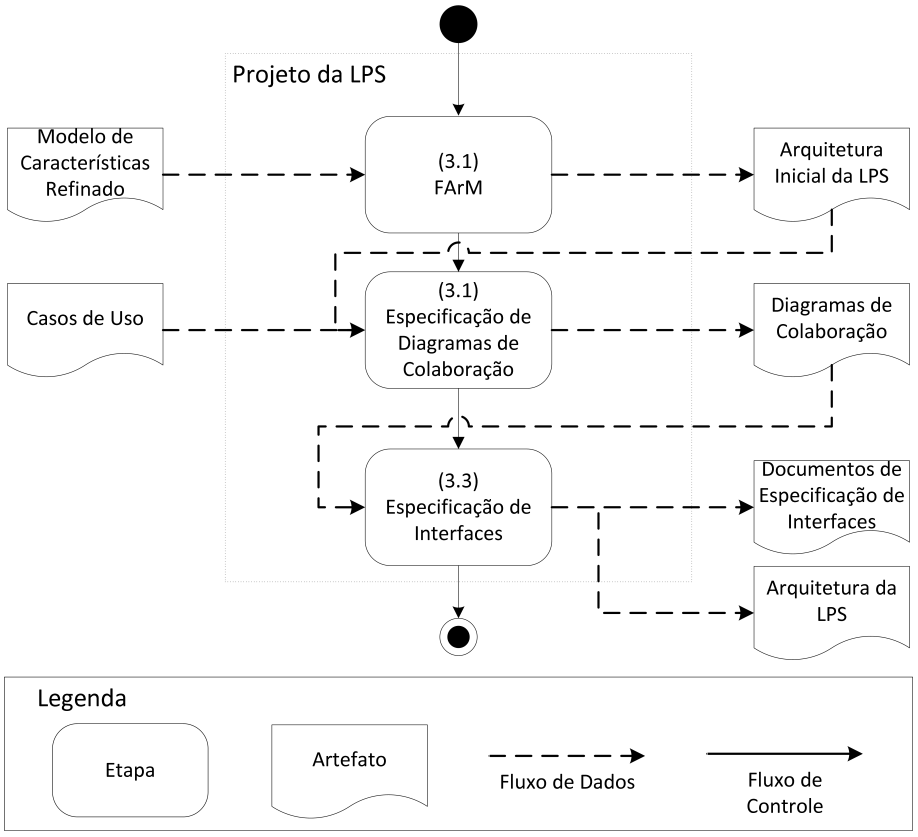


Figura 3.4: Fase de Projeto da LPS

Esta fase conta com três atividades: (3.1) o método FArM, utilizado para a geração de uma arquitetura inicial, (3.2) Especificação de Diagramas de Comunicação e (3.3) Especificação de Interfaces. A entrada desta fase é o modelo de características refinado e os modelos de casos de uso, gerados na fase anterior, Fase 2: Análise da LPS, e como saída temos a arquitetura final da LPS, diagramas de comunicação e documentos de especificação das interfaces dos componentes. As seções a seguir detalham as atividades desta fase.

Fase 3.1: Método FArM

Para a geração da arquitetura inicial foi utilizado o método FArM, apresentado detalhadamente na Seção 2.2, no Capítulo 2. O método FArM foi a escolha para este projeto

pelo fato de ser baseado em características e componentes, além de ser bastante conhecido na literatura.

Como visto, este método é composto de quatro transformações feitas em cima do modelo de características refinado, sendo que a última transformação resulta na arquitetura sugerida, baseada em componentes.

Após as quatro transformações realizadas, e com base em estilos arquiteturais e arquiteturas de referência, devem ser especificadas as relações entre os componentes resultantes da quarta transformação. Ainda não se pode chamar esta arquitetura de final pois as interfaces entre os componentes não estão especificadas. Para a especificação das interfaces, sugerimos a geração de diagramas de comunicação para identificar quais componentes se comunicam, e a partir destes diagramas, especificar as interfaces entre os componentes.

Fase 3.2: Especificação de Diagramas de Comunicação

Com base nos componentes identificados na arquitetura inicial obtida após a execução do método FArM na atividade anterior, Fase 3.2: Método FArM, e com base nos casos de uso gerados na Fase 2: Análise da LPS, é sugerida a geração de um diagrama de comunicação completo, ou seja, com todas as características obrigatórias, opcionais e alternativas. Este diagrama será utilizado na próxima atividade, Fase 3.3: Especificação de Interfaces, para a especificação das interfaces providas e requeridas de cada componente.

Fase 3.3: Especificação de Interfaces

A última atividade da fase de Projeto da LPS consiste na especificação das interfaces entre os módulos. No caso deste projeto, escolhemos a arquitetura *pipe-and-filter* para a arquitetura da LPS, onde os *filters* são os componentes e os *pipes* são arquivos. Ou seja, os componentes comunicam-se através de arquivos.

A implementação dos componentes da LPS será detalhada na próxima fase.

3.1.5 Fase 4: Implementação da LPS

Esta é a última fase do método proposto para a especificação e implementação de uma LPS a partir de um sistema legado. É nesta fase que os componentes são implementados.

Com base nos modelos de saída da fase anterior, Fase 3: Projeto da LPS, os componentes que compõem a LPS são desenvolvidos de forma incremental. Os componentes gerados também fazem parte do núcleo de artefatos, assim como os artefatos gerados nas fases anteriores. Propõe-se o modelo COSMOS*, descrito da Seção 2.5.2 do Capítulo 2, para a padronização dos componentes. O modelo COSMOS* é independente de plataforma tec-

nológica, e por isso pode ser facilmente adaptado a diversas linguagens de programação.

3.2 O Sistema Legado UNIPAR

Como já dito no Capítulo 1, o sistema UNIPAR é desenvolvido pelo UNISIM [7], e busca fornecer aos pesquisadores da área de reservatórios de petróleo (engenheiro de reservatórios de petróleo, economistas, geólogos, etc.) ferramentas que o auxiliem na tomada de decisões e que sejam úteis na automatização de algumas tarefas que utilizam simulação numérica de reservatórios.

O sistema UNIPAR hoje trabalha com três tópicos de pesquisa e conta com cinco módulos, além de um conjunto de ferramentas auxiliares. Os módulos do sistema UNIPAR são:

- Módulo de Análise de Incertezas e Risco (MAI), que trabalha com o tópico de pesquisa de Análise de Risco;
- Módulo de Redução de Incertezas (MRI), que trabalha com uma combinação entre os tópicos de Análise de Risco e Ajuste de Histórico;
- Módulo de Cálculos Econômicos (MEC), que trabalha com o tópico de pesquisa de Análise Econômica;
- Módulo de Distribuição de simulações (MPS), responsável por distribuir simulações em *clusters*;
- Módulo Distribui simulações e Gera UNIPRO (DSGU), módulo auxiliar, utilizado pelos módulos MAI e MEC.

O sistema UNIPAR possui 223 KLOC, e destas, 136 KLOC foram consideradas neste trabalho (a interface gráfica foi desconsiderada).

Ele pode ser executado tanto em ambiente Linux quanto em ambiente Windows, mas as simulações distribuídas são feitas apenas em Linux (*clusters*). Para tanto, ele conta com uma ferramenta que permite a comunicação entre máquinas Windows e os *clusters*.

Existem ainda ferramentas complementares a estes módulos, como por exemplo conversores de arquivos de saída (formato XML) para formato texto ou planilhas Excel, e validadores de dados de entrada e saída, que também não foram consideradas no projeto.

As próximas seções descrevem brevemente os módulos MAI, MEC, DSGU e MPS, que são os módulos utilizados neste projeto. O DSGU e MPS entram no escopo do projeto por serem utilizados pelo MAI e MEC.

3.2.1 MAI - Módulo de Análise de Incertezas e Risco

O MAI aborda a metodologia para a quantificação do impacto de incertezas na avaliação econômica de reservatórios de petróleo. Tal metodologia baseia-se na simulação de diversos modelos de fluxo que representam os possíveis cenários do reservatório, utilizando a combinação dos atributos incertos que o caracterizam. O número total de modelos a serem simulados é definido pelo número de atributos e pelos níveis de incerteza de cada atributo.

Para cada atributo selecionado, são atribuídos valores que representam a sua incerteza e a probabilidade de ocorrência de cada um destes níveis. Para atributos que são funções contínuas, como porosidade e permeabilidade absoluta, usualmente discretiza-se em três níveis representando o valor alto (otimista), provável e baixo (pessimista).

Os modelos de simulação do MAI são construídos utilizando as técnicas de árvore de derivação (AD) ou hipercubo latino discreto (HCLD). No caso da AD, cada ramo final dela corresponde a um modelo completo para simulação, com a combinação dos atributos críticos. A probabilidade resultante de cada modelo é equivalente ao produto das probabilidades dos atributos que o compõem.

O procedimento de combinação das variáveis críticas reduz o número de modelos de fluxos quando comparado à combinação de todos os fatores que são fontes de incerteza na caracterização do reservatório. Como a simulação de escoamento é uma etapa bastante dispendiosa, esta redução permite uma economia de tempo na etapa de previsão de produção do reservatório. A diferença do HCLD para a AD é que no caso do HCLD, apenas parte dos modelos é considerada no momento da análise.

3.2.2 MEC - Módulo de Cálculos Econômicos

O valor do dinheiro no tempo é a fundamentação da análise econômica e é a base para as medidas de lucro para selecionar investimentos. Geralmente, a avaliação econômica de projetos petrolíferos é feita por meio do fluxo de caixa descontado, do qual são obtidos os principais indicadores econômicos, entre os quais destaca-se o valor presente líquido (VPL) que é definido como o somatório dos valores das entradas e saídas do fluxo de caixa, descontados a uma taxa mínima de atratividade e a uma determinada data.

O VPL representa o patrimônio ganho ou perdido pela empresa devido ao projeto. Na avaliação de projetos de E&P é importante que os indicadores escolhidos sejam utilizados já na fase de decisão técnica ligada ao processo decisório de como deve ser a explotação dos campos de petróleo, por isso, é importante que os conceitos sejam claros e integrados com o processo de simulação e gerenciamento dos campos.

O MEC é o módulo responsável por realizar cálculos de índices econômicos para campos, poços e grupos de poços de petróleo, possibilitando o estudo da viabilidade de um

projeto de exploração de petróleo. Para efetuar tais cálculos, o MEC utiliza as informações resultantes de uma simulação do comportamento da produção no campo.

Definidos os dados de produção ao longo do tempo, o MEC aplica parâmetros definidos pelo usuário para calcular os índices econômicos. O MEC utiliza os seguintes dados de saída do simulador: produções acumuladas de óleo, produções acumuladas de gás, produções acumuladas de água, e os tempos nos quais ocorrem as produções citadas. O MEC calcula também o Valor Presente Líquido (VPL), utilizado para calcular a atratividade de investimentos. Os resultados do MEC permitem avaliar a viabilidade de um projeto de exploração de petróleo.

3.2.3 MPS - Módulo de Distribuição de simulações

O MPS foi desenvolvido com o objetivo de tornar mais fácil e rápido o processo de simulações simultâneas de reservatórios de petróleo. O MPS executa e gerencia diversas instâncias de simuladores de reservatórios com o auxílio do pacote de paralelização *Parallel Virtual Machine* (PVM), com o software de gerenciamento de filas *Load Sharing Facility* (LSF), ou ainda com o software *Torque Resource Manager*.

Além da distribuição das simulações em diversos nós e máquinas de *clusters*, o MPS também paraleliza as simulações em dois ou mais processadores, utilizando recursos fornecidos pelos simuladores.

Pode-se considerar o MPS como uma caixa preta que, a partir dos arquivos de dados do simulador, gera os arquivos de saída.

3.2.4 DSGU - Módulo Distribui simulações e Gera UNIPRO

O DSGU é um módulo auxiliar, utilizado pelo MAI e MEC, que divide-se em duas partes: a que distribui as simulações (utilizando o MPS) e aquela que gera os arquivos UNIPRO. Estes últimos são arquivos internos do sistema UNIPAR, no formato XML. Eles armazenam dados de saída das simulações que são relevantes ao sistema UNIPAR.

3.3 Criação da UNIPAR-LPS

Esta seção descreve como foi aplicado o método extrativo de adoção de LPS, proposto na Seção 3.1 deste capítulo, no sistema legado UNIPAR. A LPS gerada foi chamada de UNIPAR-LPS. Ao longo desta seção serão mostradas cada fase do método proposto, seus artefatos de entrada e saída. Os artefatos gerados durante a execução do método compõe o núcleo de artefatos da UNIPAR-LPS.

3.3.1 Fase 1: Engenharia Reversa do Sistema UNIPAR

Esta fase tem como entrada o código legado do sistema UNIPAR e como saídas a arquitetura recuperada e o modelo de características do sistema.

Fase 1.1: Recuperação da Arquitetura

O objetivo da recuperação da arquitetura neste projeto é identificar as relações entre os módulos do sistema UNIPAR e entender seu funcionamento para, a partir dos artefatos obtidos, construir o modelo de características do sistema legado. Sugere-se aqui a recuperação de uma visão estática e uma visão dinâmica. Neste projeto não foi utilizada nenhuma ferramenta para a recuperação da arquitetura, apenas o conhecimento do sistema legado e do domínio.

Pelo fato do UNIPAR ser um sistema grande e complexo, com código estruturado, optou-se por recuperar uma visão estática de alto nível, apenas para documentar o relacionamento de seus módulos. Foi gerado um diagrama com uma visão macro da comunicação entre os módulos e as bibliotecas, Figura 3.5. O diagrama ajudou o entendimento do sistema como um todo, além de mostrar as dependências entre os módulos.

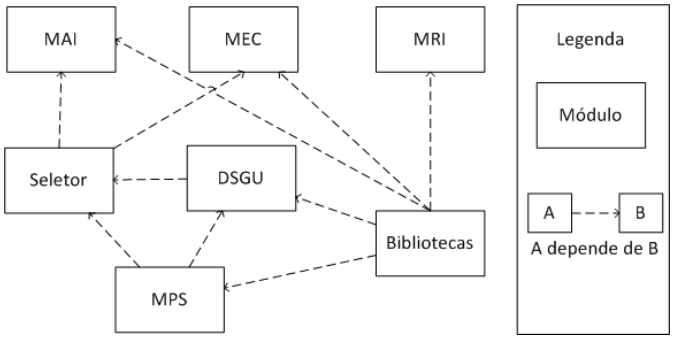


Figura 3.5: Visão geral da arquitetura do UNIPAR

Para compreender o funcionamento dos módulos do sistema UNIPAR foram gerados fluxogramas para cada módulo. Cada fluxograma é uma representação esquemática do processo de cada um dos módulos. Além dos fluxogramas de alto nível foram gerados fluxogramas detalhados para cada módulo, explicitando os parâmetros disponíveis e suas influências na execução do programa. A Figura 3.6 apresenta os fluxogramas de alto nível dos módulos MEC e MAI.

A semelhança entre o MAI e MEC foi o principal motivador da escolha desses dois módulos para o desenvolvimento deste estudo. A partir dos fluxogramas é possível identificar as tarefas que o módulo deve realizar e também como a escolha de parâmetros de execução influenciam em seu funcionamento. A recuperação da arquitetura foi realizada sem a utilização de técnicas ou ferramentas automatizadas, mas apenas com o conheci-

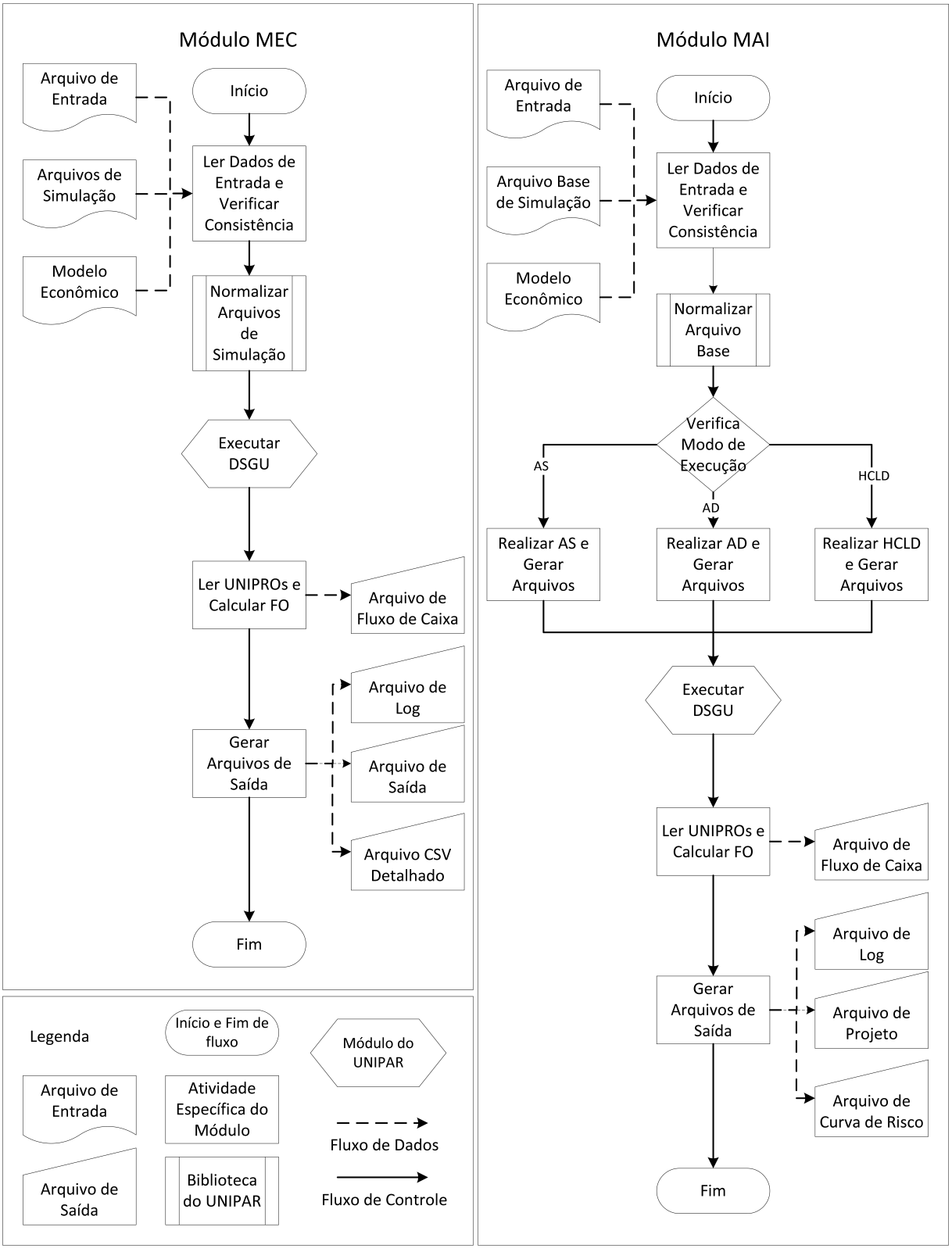


Figura 3.6: Fluxogramas dos módulos MEC e MAI

mento do domínio.

Fase 1.2: Modelagem de Características

Com base nos fluxogramas gerados na Fase 1.1, construiu-se um modelo de características para cada um dos módulos do sistema UNIPAR. A notação para a modelagem de características utilizada neste projeto é a notação proposta por Kang *et al.* (Seção 2.2).

As Figuras 3.7 e 3.8 mostram os modelos de características de dois dos módulos do sistema UNIPAR, o MEC e o MAI, respectivamente.

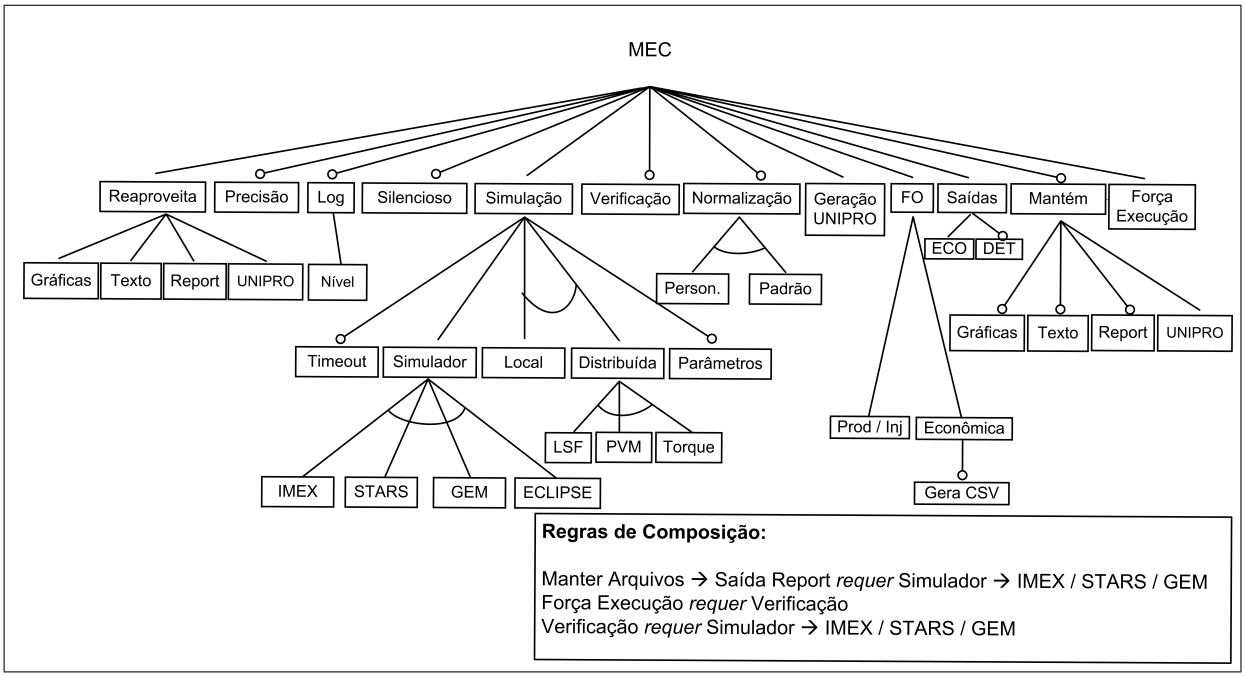


Figura 3.7: Modelo de características do MEC

Nestes modelos foram representadas todas as características de cada um dos módulos, indicando quais são obrigatórias, opcionais e alternativas, bem como explicitadas as regras de composição.

O mapeamento do fluxograma para o modelo de características não foi um-para-um, mas todas as atividades do fluxograma foram mapeadas em uma ou mais características.

3.3.2 Fase 2: Análise da UNIPAR-LPS

Esta fase tem como entrada os modelos de características dos módulos do sistema UNIPAR, gerados na Fase 1.1. Como saída, é gerado um modelo de características refinado, modelos de casos de uso, e é criado um relacionamento entre os casos de uso e as características. As próximas seções detalham cada uma das fases e exibem alguns dos artefatos

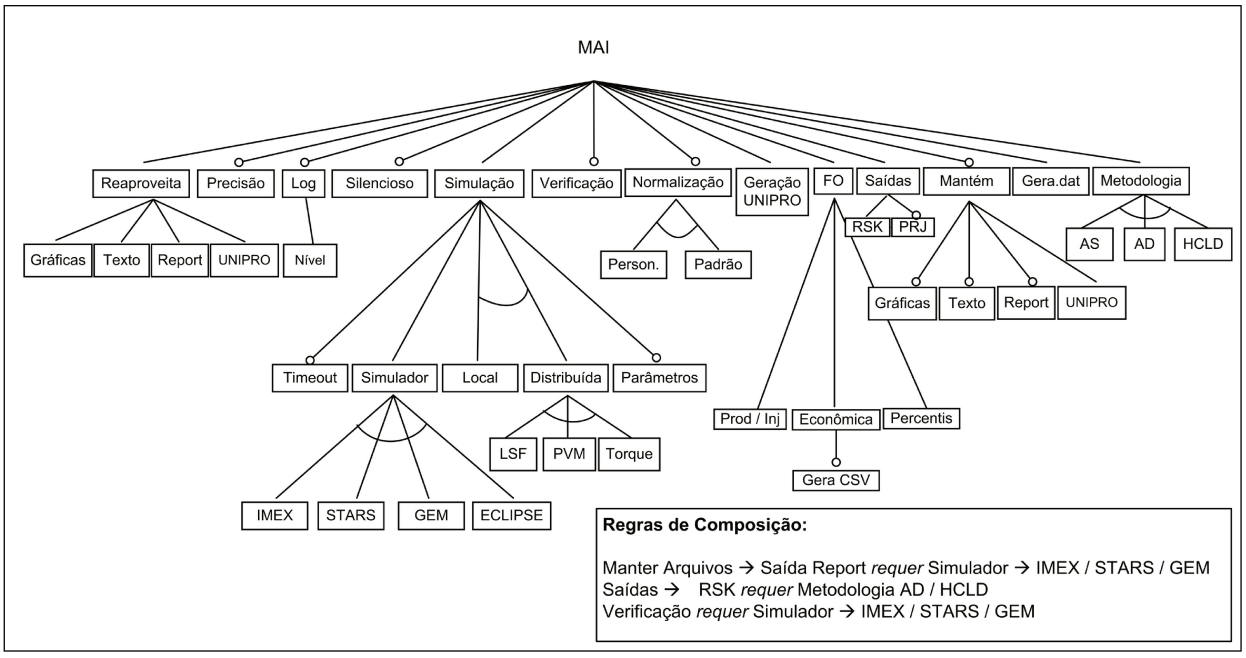


Figura 3.8: Modelo de características do MAI

gerados.

Fase 2.1: Modelagem de Características

Os modelos de características dos módulos do sistema UNIPAR foram analisados, unificados, refinados, e alguns novos requisitos foram adicionados, gerando um único modelo de características que representa a UNIPAR-LPS. Todas as características do sistema UNIPAR estão presentes no modelo de características da UNIPAR-LPS.

Fase 2.2: Modelagem de Casos de Uso

Assim como as características, os casos de uso podem ser dos tipos obrigatórios, opcionais e alternativos. Para distinguir os tipos de casos de uso são utilizados os estereótipos «kernel» (para casos de uso do tipo obrigatório), «optional» (para casos de uso do tipo opcional) e «alternative» (para casos de uso do tipo alternativo). Assim sendo, foram gerados casos de uso para todos os cenários de execução dos módulos do sistema UNIPAR, identificando-os com os estereótipos descritos.

A Figura 3.9 representa um caso de uso de um cenário de execução do MAI do sistema UNIPAR, utilizando a metodologia HCLD. Mais exemplos dos casos de uso gerados podem ser encontrados no Apêndice A.

Fase 2.3: Relacionamento entre Casos de Uso e Modelo de características

Para realizar o relacionamento entre os casos de uso e o modelo de características

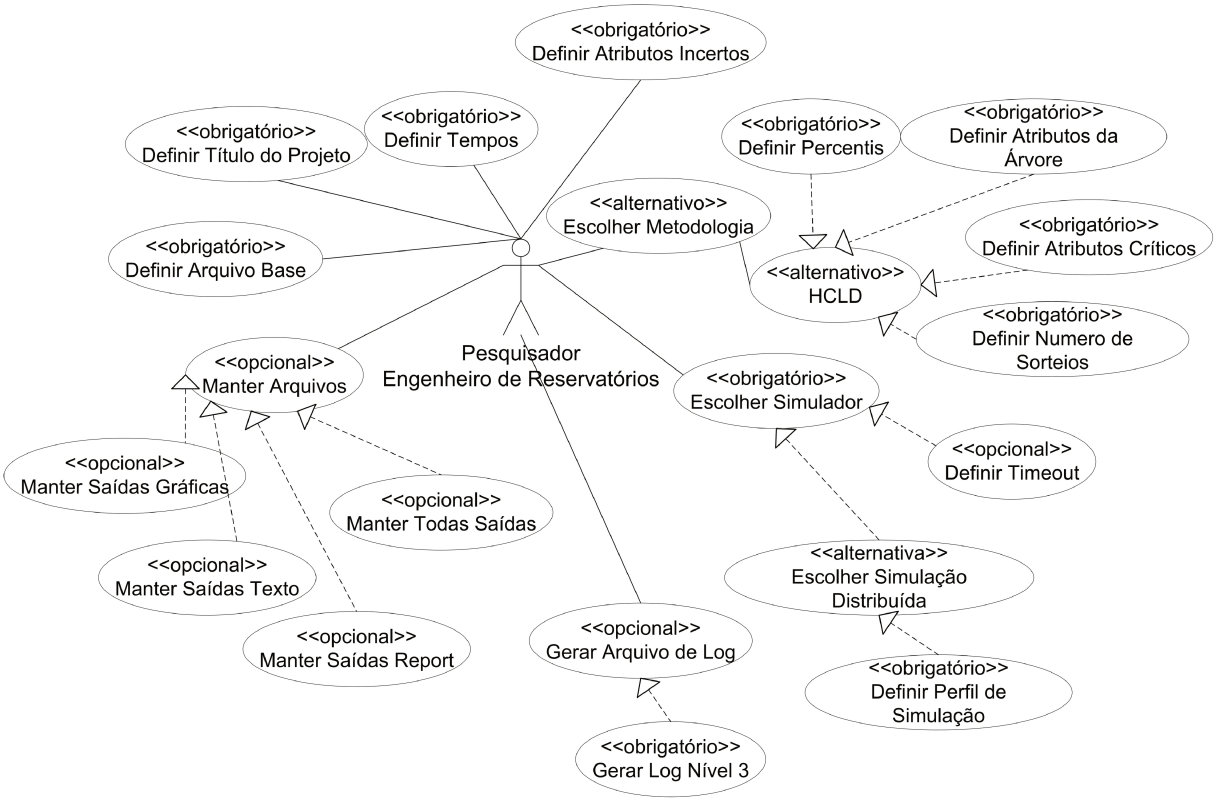


Figura 3.9: Caso de uso: pesquisador (engenheiro de reservatórios) utilizando a metodologia HCLD

foram geradas três tabelas.

A primeira tabela, chamada UC-LPS, foi alimentada com todos os casos de uso identificados e numerados pelo prefixo "UC_". A segunda tabela, chamada FM-LPS, foi gerada contendo todas as características do modelo de características da LPS, identificadas e numeradas pelo prefixo "FM_". Por fim, a última tabela, UC-FM-LPS, é a responsável por fazer o relacionamento entre os casos de uso da primeira tabela com as características presentes na segunda. Partes das tabelas podem ser vistas na figura 3.10.

Como já mencionado na Seção 3.1.3, a tabela UC-FM-LPS gerada permite a rastreabilidade da variabilidade entre os modelos, facilitando a manutenção e evolução da variabilidade da UNIPAR-LPS. Parte das tabelas geradas pode ser vista na figura 3.10.

3.3.3 Fase 3: Projeto da UNIPAR-LPS

Nesta fase foi aplicado o método FArM para obter uma arquitetura inicial para a LPS, e geramos diagramas de comunicação para nos auxiliar a definir as interfaces entre os componentes.

UC-LPS	
UC_1	Definir Tempos
UC_2	Definir Att Incertos
UC_3	Verificar Arquivos
UC_4	Definir Arquivos
UC_5	Escolher Metodologia
UC_6	Análise de Sensibilidade
UC_7	Árvore de Derivação
UC_8	HCLD
UC_9	Definir Percentis
UC_10	Definir Att da Árvore
UC_11	Definir Número Sorteios
UC_12	Escolher Simulador
UC_13	Passar Parâmetros para Simulador
UC_14	Definir Timeout
UC_15	Escolher Simulação Local
UC_16	Escolher Simulação Distribuída
UC_17	Definir Perfil de Simulação
UC_18	Gerar Arquivos de Log

FM-LPS	
FM_1	Normalização
FM_2	Personalizada
FM_3	Padrão
FM_4	Precisão
FM_5	Log
FM_6	Nível
FM_7	Refaz C. E.
FM_8	Simulação
FM_9	Timeout
FM_10	Simulador
FM_11	Local
FM_12	Distribuída
FM_13	Configuração
FM_14	Parâmetros
FM_15	Verificação
FM_16	Modo Silencioso
FM_17	Mantém Arquivos
FM_18	Mantém Saídas Gráficas

UC-FM-LPS

	U_1	U_2	U_3	U_4	U_5	U_6	U_7	U_8	U_9	U_10	U_11	U_12	U_13	U_14	U_15	U_16	U_1
F_1												X					
F_2												X					
F_3												X					
F_4																	
F_5																	
F_6																	
F_7																	
F_8			X									X					
F_9														X			
F_10																	
F_11															X		
F_12																X	
F_13												X					X
F_14												X					
F_15																	
F_16																	

Figura 3.10: Fluxogramas dos módulos MEC e MAI

Fase 3.1: Método FArM

Como visto na Seção 2.2, o método FArM é composto de quatro transformações feitas em cima do modelo de características refinado, sendo que a última transformação resulta na arquitetura sugerida, baseada em componentes.

A primeira transformação proposta pelo método sugere a remoção de características não relacionadas à arquitetura e a resolução de características de qualidade. Nosso modelo de características refinado (saída da primeira atividade da fase de Análise da LPS, Seção 3.1.3) não contém características relacionadas à arquitetura e as características de qualidade já estão resolvidas. Portanto, a primeira transformação não fez nenhuma alteração no modelo de características.

Um exemplo de característica de qualidade é a "Distribuição". A distribuição de simulações pode ser feita com três escalonadores de tarefas: LSF, PVM e Torque, conforme visto na Figura 3.11. A Figura 3.11 representa parte do modelo de características da LPS, e será utilizada nos exemplos das transformações do método FArM.

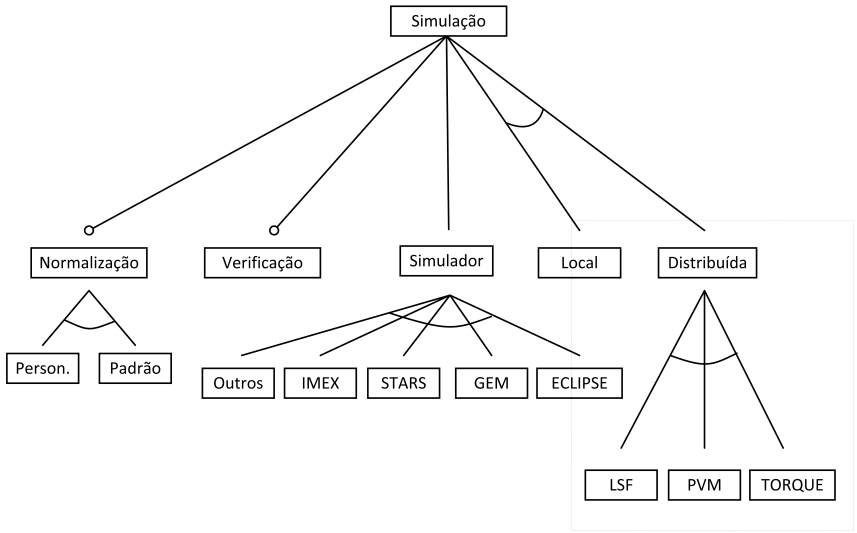


Figura 3.11: Parte do modelo de características da UNIPAR-LPS

A segunda transformação do FArM é baseada em requisitos arquiteturais. O método sugere que esses requisitos sejam representados explicitamente no modelo de características e sejam integrados com características existentes ou que sejam criadas novas características para representá-los. Assim, a segunda transformação também não fez nenhuma alteração no modelo de características.

Na terceira transformação é proposto especificar a comunicação entre características, baseada em três tipos de relação (tipos "usa", "modifica"e "contradiz"), e depois realizar uma nova transformação com base nos critérios tipo de relação e número de

relações de interação. No modelo de características da LPS deste projeto temos relações do tipo "usa", que seguem o critério de número de relações. Como exemplo, a característica "Simulador"foi integrada às características "Normalização", "Verificação", "Local"e "Distribuída". A Figura 3.12 mostra parte do modelo de características da UNIPAR-LPS após as duas atividades desta transformação.

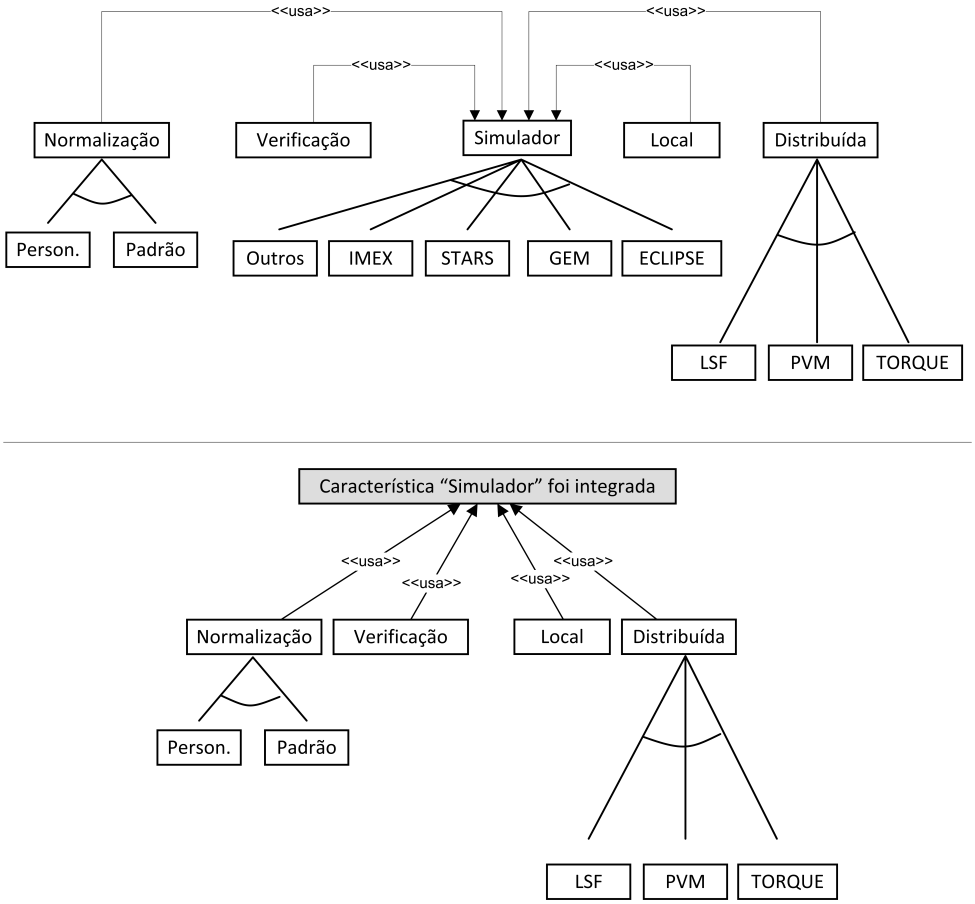


Figura 3.12: Modelo de características da UNIPAR-LPS após a terceira transformação do FArM

Por fim, a última transformação verifica se todas as relações de hierarquia entre características são de um dos três tipos definidos pelo FArM. Aquelas que não são de nenhum dos três tipos, são removidas e outras relações podem ser criadas.

A Figura 3.13 mostra o modelo de características final, após as transformações do método FArM.

Após estas transformações, as características são mapeadas para componentes. As supercaracterísticas são mapeadas para componentes que proveem acesso às funcionalidades dos componentes que implementam as subcaracterísticas. As relações entre as caracterís-

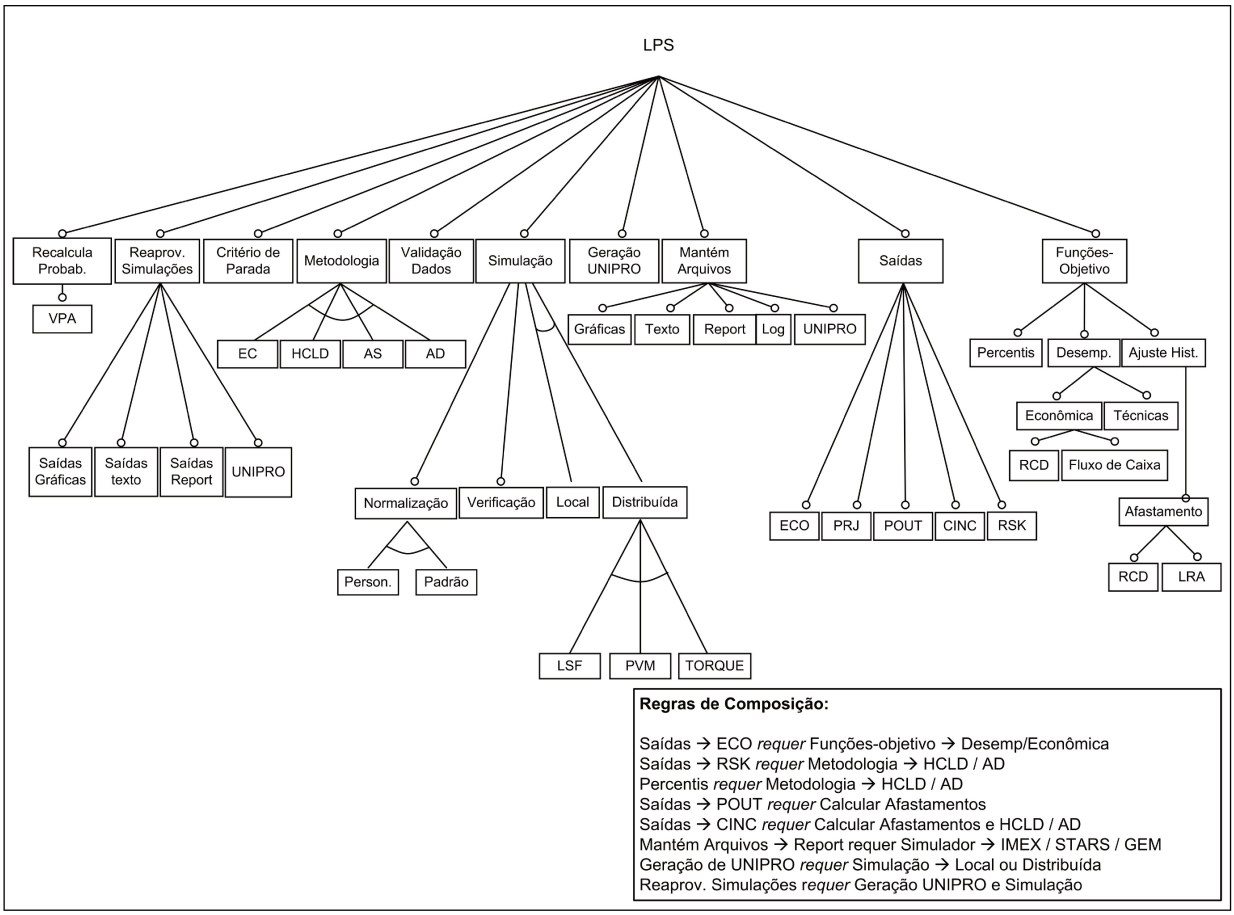


Figura 3.13: Modelo de características final da UNIPAR-LPS

ticas são mantidas nos componentes. A Figura 3.14 mostra o resultado deste mapeamento para o trecho do modelo de características utilizado nos exemplos.

Com base em estilos arquiteturais e arquiteturas de referência, foram especificadas as relações entre os componentes resultantes da quarta transformação. Ainda não pode-se chamar esta arquitetura de final pois as interfaces entre os componentes não estão especificadas. Esta tarefa é realizada nas próximas atividades do método proposto.

Fase 3.2: Especificação de Diagramas de Comunicação

Com base nos componentes identificados na arquitetura inicial, gerada após o método FArM, e com base nos casos de uso gerados na fase de Análise da LPS, geramos um diagrama de comunicação completo, ou seja, com todas as características obrigatórias, opcionais e alternativas. A Figura 3.15 apresenta o diagrama de comunicação gerado.

Na figura exibida, os subcomponentes da UNIPAR-LPS foram omitidos. Apenas seus supercomponentes foram considerados. Por questões de simplificação, as interfaces foram

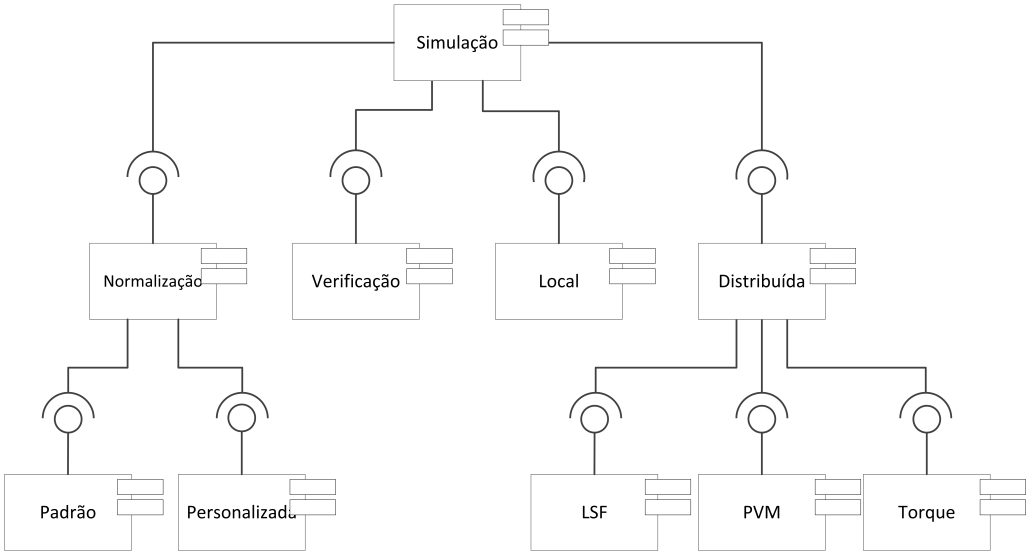


Figura 3.14: Parte do modelo de características após a quarta transformação do FArM mapeado para componentes

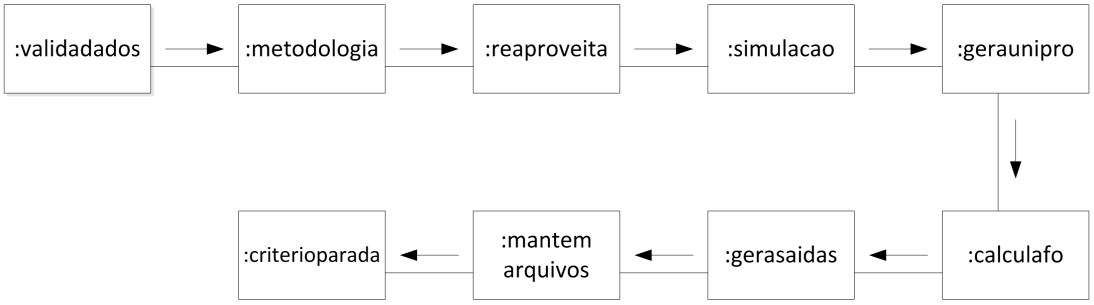


Figura 3.15: Diagrama de comunicação

omitidas no diagrama.

Fase 3.3: Especificação de Interfaces

A última atividade da Fase 3: Projeto da LPS, consiste na especificação das interfaces entre os módulos. No caso deste projeto, escolhemos a arquitetura *pipe-and-filter* para a arquitetura da LPS, onde os **filtros** são os componentes e os **conectores** são arquivos. Ou seja, os componentes comunicam-se através de arquivos. Todos os componentes da LPS trabalham com uma lista de arquivos de simulação como base.

Por exemplo, o componente **Metodologia** gera uma lista de arquivos de simulação que serão simulados pelo componente **Simulação**. Ou seja, a saída do componente **Metodologia** é uma lista de arquivos de simulação, assim como a entrada do componente

Simulação é uma lista de arquivos de simulação. O arquivo de saída do componente **Metodologia** é a entrada do componente **Simulação**.

O formato dos arquivos XML foi mantido para facilitar a migração para a LPS. Foram gerados arquivos do tipo XSD para todos os formatos de arquivos aceitos pelos componentes da LPS, e utilizamos documentos no formato HTML e PDF para especificar os arquivos XSD. Um exemplo completo da documentação gerada pode ser encontrado no Apêndice B.

3.3.4 Fase 4: Implementação da UNIPAR-LPS

Esta é a última fase do método proposto para a especificação e implementação de uma LPS a partir de um sistema legado. Usualmente é nesta fase que os componentes são implementados e os produtos são gerados. No caso deste projeto, os produtos serão gerados *on the fly*, ou seja, o pesquisador resolve a variabilidade e escolhe as características que seu produto terá no momento da execução. Estas escolhas são informadas em um arquivo de especificação que representa um *workflow* científico.

Com base nos modelos gerados nas atividades anteriores, os componentes que compõem a LPS são desenvolvidos de forma incremental, populando o núcleo de artefatos. Neste projeto, os componentes foram desenvolvidos a partir do código do sistema UNIPAR e foi utilizado o modelo COSMOS* (Seção 2.5.2) para a padronização. A maioria do código do sistema UNIPAR é escrito nas linguagens C/C++ e por este motivo esta linguagem foi escolhida para o desenvolvimento dos componentes.

Para a codificação dos componentes foi utilizada a ferramenta Eclipse. Utilizou-se como repositório de componentes o *Concurrent Version System* (Sistema de Versões Concorrentes), o CVS. Ele é um sistema de controle de versão que permite trabalhar com diversas versões de arquivos organizados em um diretório local ou remotamente, mantendo suas versões antigas e os *logs* de por quem e quando os arquivos foram manipulados. Optamos por esta ferramenta por dois motivos: primeiro, pela facilidade de uso e existência de *plug-ins* de integração com o Eclipse. Segundo, por ser o repositório em uso pelo UNISIM e ser integrado com algumas ferramentas internas de desenvolvimento de software.

A variabilidade do sistema foi implementada em dois níveis: em nível arquitetural, através de componentes obrigatórios, opcionais e alternativos, e através da parametrização, uma das técnicas de implementação de variabilidade vista na Seção 2.1.6, Capítulo 2. Ou seja, existem componentes obrigatórios, opcionais e alternativos, assim como existem parâmetros para especificar o comportamento requerido de cada componente.

Em alguns casos, os parâmetros servem para indicar qual subcomponente deve ser executado. Pode-se citar como exemplo o componente **Metodologia**, mostrado na Figura

3.16. Ele conta com quatro subcomponentes que especificam o tipo de metodologia a ser utilizada: AS, AD, HCLD e EC. Para que seja executada a metodologia HCLD, por exemplo, deve-se escolher o componente **Metodologia** passando o parâmetro '-c'.

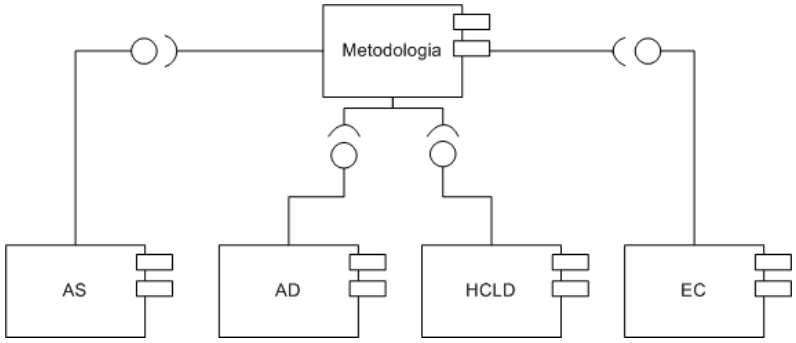


Figura 3.16: Componente Metodologia e seus quatro subcomponentes

Foi criado um novo componente, chamado ***workflow***, para o apoio à execução dos experimentos científicos. Este componente não foi adicionado ao modelo de características por não se tratar de uma característica visível ao usuário. É este componente que implementa a função principal da arquitetura *call-and-return* da solução proposta. Ele é responsável pela validação do *workflow* científico a ser executado, bem como responsável pela chamada dos componentes que o compõem.

A estrutura de diretórios da UNIPAR-LPS é exibida na Figura 3.17, e cada um dos diretórios são detalhados a seguir.

- **Diretório "etc"**

  Algumas informações de configuração necessárias são armazenadas em um diretório comum a toda LPS, o "etc". Neste diretório encontram-se arquivos com informações sobre os simuladores, *clusters* e configurações do ambiente.

- **Diretório "includes"**

  O diretório "includes"possui um ou mais subdiretórios que indica qual a arquitetura da máquina na qual foi compilado o código da LPS, como por exemplo "LINUX64", ou "WIN32". Nesses subdiretórios são armazenados todos os arquivos de includes (".h"e ".hh") de bibliotecas da LPS, que podem ser utilizados por outros componentes, bem como includes de bibliotecas externas, como "tclap", uma biblioteca externa biblioteca utilizada para o tratamento das linhas de comando.

- **Diretório "lib"**

Assim como o diretório "includes". o diretório "lib"também possui os subdiretórios referentes às arquiteturas de compilação. Nesses subdiretórios são armazenados os arquivos ".a"das bibliotecas da LPS.

- **Diretório "bin"**

  O diretório "bin"segue o padrão dos dois diretórios anteriores e em seus subdiretórios são armazenados os executáveis dos componentes. Caso um pesquisador crie seu próprio componente, basta armazenar seu executável neste diretório e ele estará disponível para o uso.

- **Diretório "*scripts*"**

  Este diretório armazena *scripts* para reproduzir exatamente os módulos do sistema UNIPAR, e também um *script* que, caso a execução do aplicativo esteja sendo feita em ambiente LINUX, configura o ambiente conforme necessário.

- **Diretório "componentes"**

  Neste diretório estão armazenados os códigos fonte e arquivos objetos e executáveis relacionados aos componentes, respeitando uma estrutura interna:

  - **Diretório "bin":** Onde são armazenados os executáveis.

  - **Diretório "obj":** Onde são armazenados os arquivos objeto.

  - **Diretório "src":** Onde são armazenados os códigos fonte.

  Os dois primeiros diretórios possuem os subdiretórios indicando a arquitetura da máquina na qual o código foi compilado.

- **Diretório "*workflow*"**

  Este diretório armazena o componente central da UNIPAR-LPS, responsável por validar o *workflow* especificado pelo usuário, e executar os componentes na ordem em que foram solicitados.

A definição do fluxo de trabalho em tempo de execução é a premissa do *workflow* científico, que será explicado na próxima seção.

3.4 Execução de Experimentos Científicos

Segundo Cruz *et al.* [56], apesar dos *workflows* terem surgido no ambiente de negócios, cada vez mais Ciências como Biologia e Ciências ligadas à Matemática e à Natureza (Química, Física, Geografia, Engenharia, entre outras) utilizam *workflows*. Na literatura

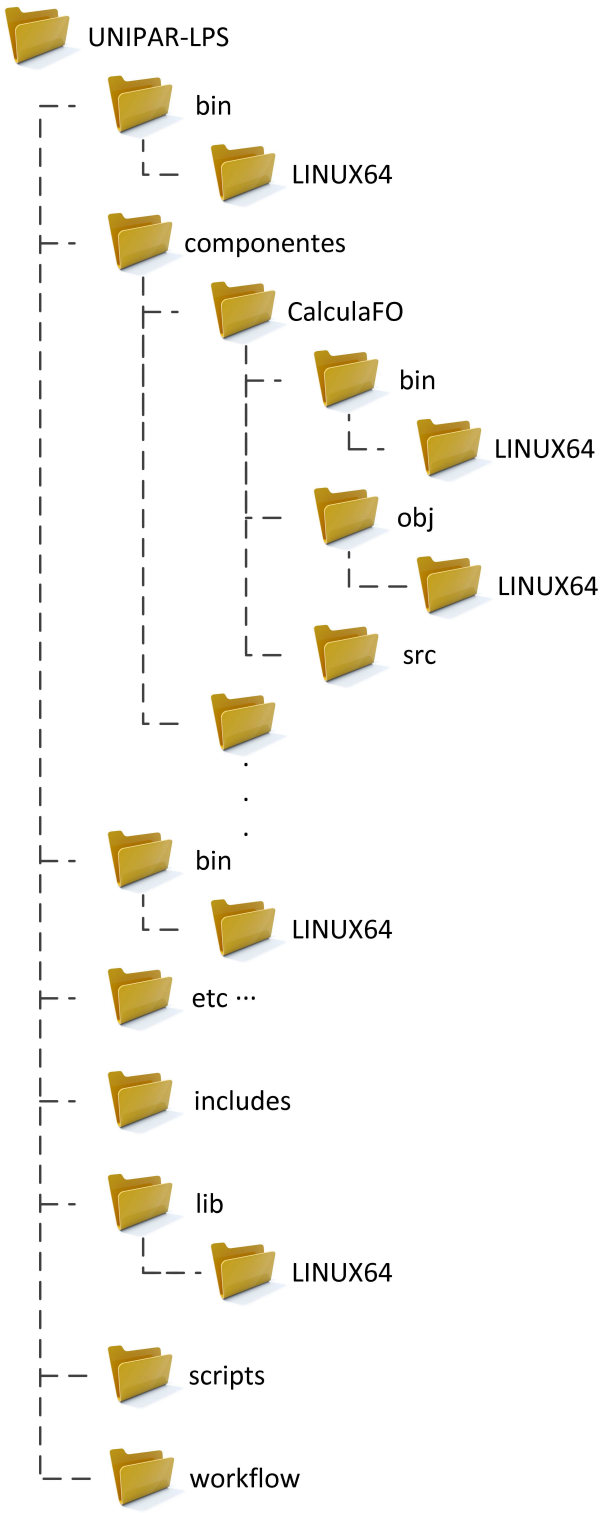


Figura 3.17: Estrutura de diretórios da UNIPAR-LPS

encontramos diversos tipos de *workflows*, muitos produzidos através do uso de linguagens de *scripts*. Esses *scripts* definem em um só momento a sequência de execução e os parâmetros de execução do fluxo de trabalho.

O uso de *scripts* pode não ser trivial para pesquisadores. A complexidade de um *script* pode dificultar sua manutenção e reduzir a capacidade de reutilização. Alguns pontos com relação aos *workflows* devem ter atenção especial, tais como dificuldades em registrar as execuções dos programas, a origem dos dados utilizados, as transformações aplicadas aos dados, os resultados obtidos, entre outras.

Um experimento científico é a montagem de uma estratégia concreta a partir da qual se organizam diversas ações observáveis direta ou indiretamente, de forma a provar a plausibilidade ou falsidade de uma dada hipótese, ou de forma a estabelecer relações de causa/efeito entre fenômenos. A experiência científica é uma das abordagens fundamentais necessárias, empirista à ampliação do conhecimento humano. O *workflow* científico é utilizado para especificar a ordem das tarefas que compõem o experimento.

Independentemente de qual seja o domínio de aplicação, os *workflows* devem obedecer a um conjunto mínimo de requisitos [40]:

1. Os experimentos devem ser reprodutíveis e disseminados. Para tanto, eles precisam ser bem documentados, permitindo que outros cientistas repliquem esses experimentos ou experimentos semelhantes, e que confirmem ou não os resultados. Os resultados também devem ser documentados e podem ser usados com base para futuras pesquisas.

2. Os experimentos devem ser projetados a partir de um ponto inicial ou, o que é mais comum, podem reutilizar ou combinar experimentos realizados anteriormente.

3. Geralmente os experimentos consistem em um número de passos com restrições complexas, passos estes que podem ser executados paralelamente em sistemas distribuídos. Pode ocorrer de algum passo precisar ser feito novamente. Para tanto, as condições experimentais deverão ser documentadas e controladas ao longo das atividades. Assim, caso seja necessário refazer uma etapa, garante-se os mesmos resultados.

Uma das maiores dificuldades do sistema UNIPAR, por se tratar de um sistema legado, sem documentação, é o fato de ter um custo de manutenção muito alto. O código do sistema UNIPAR possui um forte acoplamento, demandando muito esforço para alterações que teoricamente deveriam ser simples. Outro ponto importante a ser ressaltado é que qualquer alteração ou nova funcionalidade que um pesquisador necessite, precisa ser passada à equipe de desenvolvimento e esperar até que uma nova versão do software seja liberada.

Para o UNISIM, o ambiente de trabalho ideal é aquele no qual o pesquisador possa criar fluxos de trabalho utilizando componentes existentes combinando-os ou não aos componentes desenvolvidos pelo próprio pesquisador, gerando assim produtos personalizados. A infraestrutura proposta utilizada em conjunto com os *workflows* científicos facilitará a inclusão à UNIPAR-LPS de novas metodologias desenvolvidas pelos pesquisadores, sem a necessidade da equipe de desenvolvimento.

Para garantir o compartilhamento de dados, propõe-se a criação de um novo componente, responsável por armazenar na LPS as informações dos experimentos executados e também responsável por copiar um experimento para um local compartilhado. É necessário tomar cuidado com a privacidade dos dados. Propõe-se que o pesquisador indique ao componente de compartilhamento de dados, através de um parâmetro, se ele deseja que seus dados sejam públicos ou privados. Caso sejam públicos, pode-se copiar o experimento para o local compartilhado com permissão do diretório para todo o grupo de pesquisa. Caso seja privado, deve-se copiar com permissão apenas para o pesquisador responsável pelo experimento.

3.4.1 Uso da UNIPAR-LPS

A infraestrutura fornecida será utilizada tanto pelos pesquisadores do UNISIM, para a especificação de *workflows* científicos que definem os experimentos científicos, quanto pela equipe de desenvolvimento do UNISIM, que dará continuidade ao desenvolvimento e manutenção da UNIPAR-LPS.

Uma característica na UNIPAR-LPS é sempre mapeada para um componente ou parâmetro de componente. Assim, uma nova característica na LPS significa um novo componente ou um novo parâmetro para um componente. No caso de ser adicionado um novo parâmetro a um componente, esta adição deve ser feita pela equipe de desenvolvimento. Os pesquisadores não têm acesso ao código fonte da LPS.

No caso da variabilidade representada por um componente, ele deve ser desenvolvido e adicionado ao núcleo de artefatos da LPS, e pode ser desenvolvido por um pesquisador, independente de linguagem de programação. É necessário que o arquivo de **Especificação de Componentes** (armazenado no diretório "etc"da UNIPAR-LPS) seja atualizado. Neste arquivo deve-se indicar quais componentes são passíveis de serem invocados após sua execução, bem como seus parâmetros aceitos. Caso esta configuração não seja realizada, não será possível validar o fluxo de trabalho e este não poderá ser executado. Este arquivo pode ser visto no Apêndice C.

A infraestrutura proposta permite a criação de produtos personalizados para os pesquisadores, gerados *on the fly*, utilizando os componentes disponibilizados pela UNIPAR-LPS, ou componentes desenvolvidos por eles mesmos ou terceiros. A Figura 3.18 ex-

emplifica o tipo de combinações que a infraestrutura permite. Tomando como exemplo componentes quaisquer, A, B, C e D, o pesquisador pode utilizar o fluxo completo, Trocar um componente por outro implementado por ele (desde que siga as especificações de interface), neste caso B', ou ainda não utilizar um componente, alterando o fluxo de trabalho.

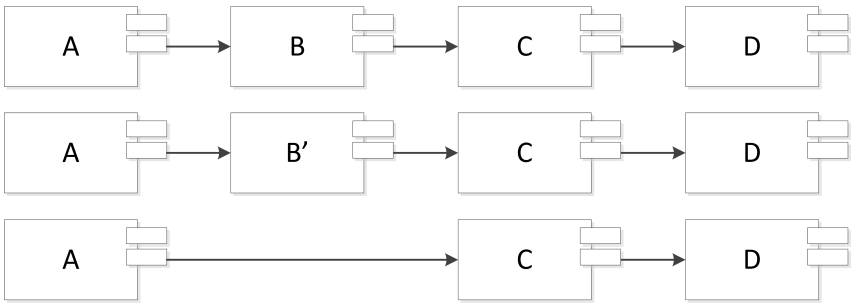


Figura 3.18: Exemplos de alteração de fluxos de trabalho

Para a adição de uma nova característica na UNIPAR-LPS, tanto pelo pesquisador quanto pela equipe de desenvolvimento, é necessário adicioná-la ao modelo de características. Não é permitida a criação de novos pontos de variação, mas é permitida a criação de variantes em pontos de variação já existentes. A Figura 3.19 mostra o caso de uso onde o usuário pode utilizar a infraestrutura proposta, e o caso de uso de como é feita a adição de uma variabilidade arquitetural.

Para a execução de um experimento científico, é necessário que o usuário especifique dados iniciais para a execução, tais como o simulador que será utilizado, nome do arquivo base de simulação, atributos para o projeto, etc. Esses dados iniciais devem ser informados em um arquivo de entrada comum a todos os componentes da LPS, em formato XML. Logo no início da execução, o componente central da UNIPAR-LPS, responsável pela execução do *workflow*, gera uma cópia deste arquivo com o nome "default.xml"para que todos os componentes da UNIPAR-LPS consigam encontrar as informações necessárias.

Também através de um arquivo XML, chamado de "especificação do *workflow*", o usuário compõe o *workflow* utilizando os componentes existentes e/ou adicionando seus próprios componentes, desde que os novos componentes respeitem a entrada e a saída exigida pelo fluxo de trabalho, ou seja, obedeça as interfaces do componente anterior e posterior à sua execução. O componente central da UNIPAR-LPS é responsável por validar o *workflow* gerado e continuar ou não a execução. Também é neste arquivo que são informados os parâmetros e seus respectivos valores para cada componente que será executado.

Pelo fato da comunicação entre os componentes ser realizada através de arquivos, a infraestrutura aqui proposta é totalmente independente de linguagem de programação.

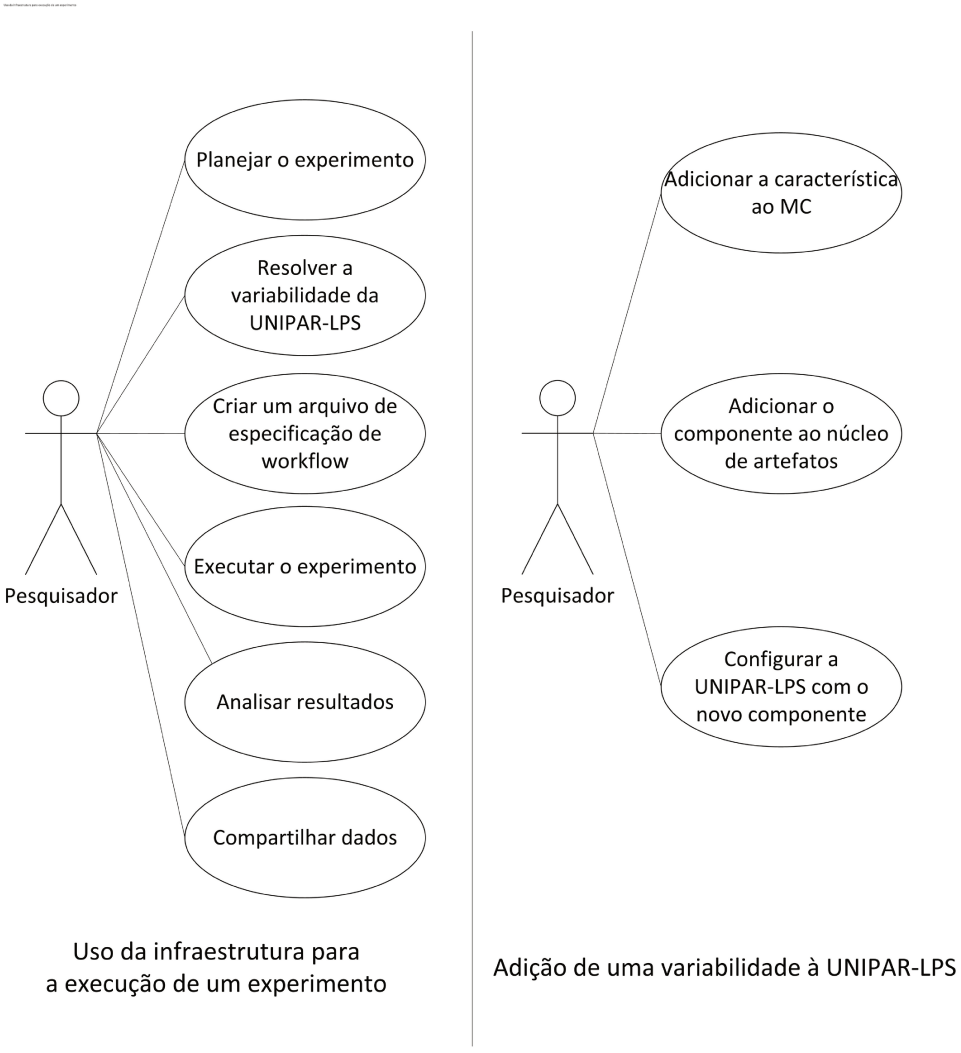


Figura 3.19: Casos de uso: uso da infraestrutura e adição de uma nova variante à UNIPAR-LPS

Ou seja, independente da linguagem a ser utilizada na implementação de um componente, ele pode ser facilmente adicionado à LPS, desde que siga as especificações de entrada e saída de dados.

A Figura 3.20 representa o conjunto de atividades para a execução de um experimento utilizando o sistema UNIPAR e utilizando a infraestrutura UNIPAR-LPS.

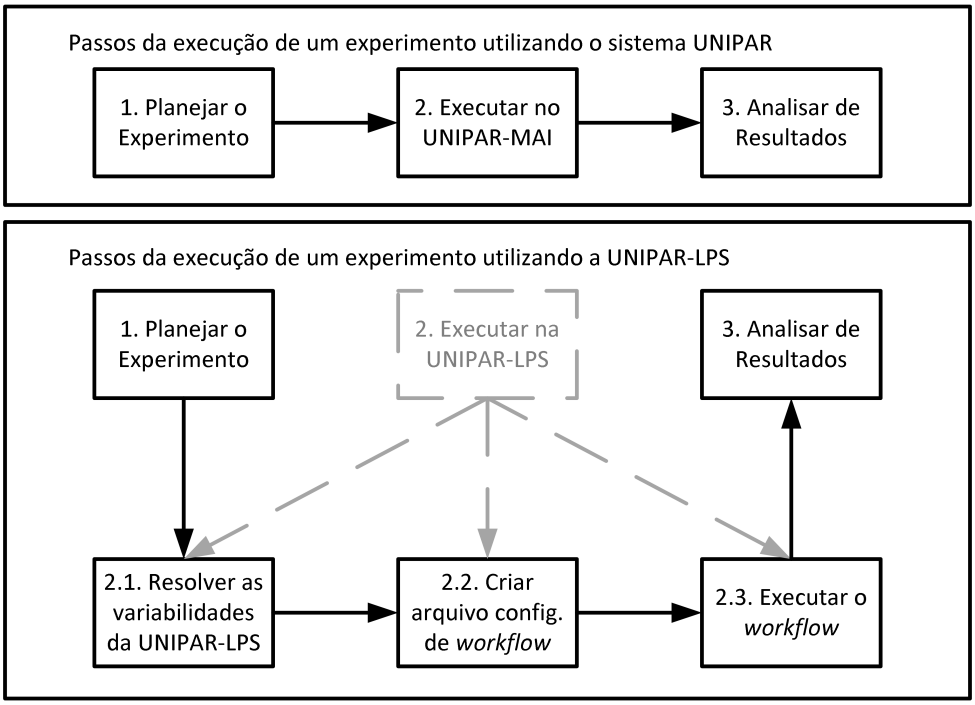


Figura 3.20: Conjunto de atividades para a execução do sistema UNIPAR e da UNIPAR-LPS

Conforme mostra a Figura 3.20, para a execução de um experimento científico utilizando o sistema UNIPAR, são necessárias três atividades: (1) Planejamento do Experimento, (2) execução do UNIPAR e (3) Análise dos Resultados. Utilizando a UNIPAR-LPS, a execução é parecida, porém a atividade (2) é dividida em três passos: (2.1) é o passo onde o usuário resolve as variabilidades da UNIPAR-LPS a fim de selecionar as características que farão parte do produto que será gerado, (2.2) onde o usuário gera o arquivo de especificação do *workflow*, indicando quais componentes e parâmetros deverão ser executados e em que ordem, e (2.3) é a execução do *workflow* através da linha de comando. São estes passos que permitem a geração de produtos personalizados.

3.5 Trabalhos Relacionados

FeatuRSEB [10] e KobrA [33] são dois métodos baseados em casos de uso para derivar uma arquitetura baseada em componentes. A fim de prover um mapeamento entre as características e os elementos arquiteturais, FeatuRSEB usa *traces* enquanto KobrA propõe o modelo de decisão e não utiliza o conceito de características. A derivação de componentes de LPS através de casos de uso promovem o espalhamento e entrelaçamento de características. Uma característica é descrita em partes de vários casos de uso. Derivar componentes a partir de casos de uso eventualmente leva ao desenvolvimento de características mapeadas a muitos elementos arquiteturais. No entanto, casos de uso derivados de componentes de arquitetura mostram um alto encapsulamento e um baixo acoplamento, facilitando a manutenção. O uso de *traces* e modelos de decisão para o mapeamento das características para a arquitetura é uma solução para LPS de tamanhos médios. Sistemas grandes levam a um número muito elevado de *traces* e decisões para o modelo de decisão, tornando a criação e manutenção da LPS uma tarefa difícil.

Couto *et al.* [24] descreveram a extração de oito características complexas de um sistema de software real (ArgoUML, uma ferramenta *open source* utilizada para projetos de sistemas em UML) a fim de gerar uma LPS. O código responsável pela implementação de oito características foi alterado utilizando diretivas de compilação condicional, resultando na linha de produtos chamada ArgoUML-SPL. As características foram escolhidas para o estudo por representarem funcionalidades importantes para o software. Na LPS extraída, as diretivas de pré-processamento foram utilizadas para delimitar o código opcional associado a cada característica. Por um lado, o uso de diretivas de pré-processamento poluem o código, tornando o programa menos legível e difícil de ser entendido, mantido e evoluído. Por outro lado, estas diretivas têm vantagens como expressividade, já que suportam a anotação de qualquer trecho de código. A intenção nesse trabalho foi prover uma LPS que pudesse ser usada para evoluir outras modularizações e separação de conceitos de tecnologia. A extração da ArgoUML-SPL foi motivada pelo fato de que a pesquisa na área de linhas de produtos é geralmente baseada em demonstrações de sistemas construídos em laboratório, e não em sistemas reais.

Fazendo-se uma analogia com o conceito de LPS, o trabalho de Ogasawara *et al.* [42] propõe o conceito de linhas de experimentos. A proposta é uma abordagem para apoiar as etapas de concepção e utilização de *workflows* científicos baseada em técnicas de reutilização de software e gerência de configuração. Uma linha de experimentos pode ser entendida como sendo o estabelecimento de um *workflow* padrão que contém um conjunto de modelos de atividades encadeadas entre si. Para cada modelo de atividade deve-se indicar especificamente quais são as atividades que podem ser utilizadas na experimentação. Estas atividades podem ser algoritmos e componentes pré-existentes no sistema gerenciador

de *workflow* científico e também *subworkflow*. O conceito de *workflow* científico promove certo grau de reutilização de experimentos, possibilitando que eles sejam re-executados. Entretanto, além de faltarem mecanismos que impeçam a desorganização de diferentes experimentos em inúmeros *workflows* de experimentação (que, se desordenados, podem perder a semântica), também falta formalismo para a separação entre uma configuração específica e um processo padrão de experimentação, dentre outras limitações.

Com relação a *workflows* científicos, existe o projeto myGrid [3], financiado pelo *Engineering and Physical Sciences Research Council* (EPSRC). O projeto envolve cinco universidades do Reino Unido, o grupo *European Bioinformatics* e alguns colaboradores da indústria. Seu principal objetivo é explorar a tecnologia de computação em grid e prover serviços úteis às necessidades da bioinformática. O myGrid conta com serviços para integração de dados e recursos de aplicações, como descoberta de recursos, execução de *workflows* e processamento distribuído de consultas. Além disso, também possibilita a gerência de dados de proveniência, metadados e personalização, o que permite integração e um modelo semântico para a área de bioinformática.

Na área petrolífera, pode-se citar o software Petrel [4], um software comercial muito utilizado na pesquisa científica, desenvolvido pela empresa Schlumberger. A partir do Petrel, geofísicos, geologistas e engenheiros de reservatórios podem desenvolver *workflows* colaborativos e integrar conhecimento de diferentes disciplinas. É possível desenvolver *plug-ins* personalizados para o Petrel. Para tanto, é disponibilizado pela Schlumberger o *framework* Ocean, que disponibiliza ferramentas .NET e oferece interfaces estáveis e amigáveis para o desenvolvimento de software. O Petrel só aceita *plug-ins* desenvolvidos a partir deste *framework*. A desvantagem é que um pesquisador não pode integrar seu próprio *plug-in* ao Petrel, a não ser que ele tenha o desenvolvido utilizando Ocean.

Medeiros *et al.* [40], [59] apresentaram o WASA, um *framework* arquitetural para dar suporte a aplicações científicas. O WASA possui uma arquitetura dividida em quatro camadas: (1) Interface com usuário, (2) Ferramentas internas, (3) Melhorias nas funcionalidades do banco de dados, e (4) Banco de dados. Ele é baseado no gerenciamento de workflow como um componente central, e seu objetivo é proporcionar um ambiente onde seja possível modelar atividades decorrentes de aplicações científicas e suas evoluções, os *workflows*.

Ramos *et al.* [48] propuseram uma extensão de uma abordagem de reengenharia para LPS proposta por Ramos e Penteado [49], e ilustraram como ela pode ser utilizada para evoluir sistemas legados. A abordagem apresentada é baseada em práticas de engenharia de software tradicional e as utiliza para apoiar o desenvolvimento de Sistemas de Sistemas (*Systems of Systems*, SoS), que é um tipo de sistema emergente que ainda necessita de técnicas específicas de desenvolvimento. Ela combina conceitos e técnicas de reengenharia incremental e técnicas de LPS. A reengenharia incremental é uma técnica alternativa para

a manutenção evolutiva de software por meio de iterações. Nesta abordagem, as funções úteis de um sistema legado são identificadas, modeladas como características e passam por uma reengenharia, dando origem a componentes para popularem um núcleo de artefatos. O estudo considerou sistemas legados para compor SoS por existirem vários sistemas legados que são atualmente utilizados sem compartilhamento de recursos, e mostrou que a reestruturação do código legado pode ser feita sem afetar o funcionamento do sistema legado.

O trabalho de Costa *et al.* [23] propõe uma abordagem que utiliza um conjunto de ontologias e modelos de características para o desenvolvimento de uma LPS, voltada para apoiar o desenvolvimento e/ou a execução de aplicações científicas. Através da conexão entre esses modelos, o objetivo final da LPS é a geração de um *workflow* científico com as atividades a serem realizadas de acordo com o domínio escolhido e com as necessidades do usuário. Esta abordagem, chamada de PL-Science, visa auxiliar o cientista durante o processo de especificação das tarefas necessárias para a realização de um experimento científico.

Tabela 3.1: Trabalhos Relacionados

Trabalho	LPS	Independência de Linguagem de Programação	e-Science	Sistema Real
FeatuRSEB	•			
Cout *et al.*	•			•
Ogasawara *et al.*	•		•	
MyGrid			•	•
Petrel			•	•
WASA		•		
Costa *et al.*	•		•	
Foschiani	•	•	•	•

Não foi encontrado nenhum trabalho na literatura que envolve a abordagem de LPS, juntamente com DBC e que promova independência de linguagem de programação para o desenvolvimento dos componentes.

3.6 Resumo do Capítulo

Este capítulo detalhou o método extrativo proposto para o desenvolvimento de uma LPS baseada em componentes, bem como descreveu sua utilização para a geração de produtos. Esta LPS foi desenvolvida a partir do sistema UNIPAR, e é chamada UNIPAR-LPS. O

método é baseado no arcabouço da Reengenharia Orientada a características e é dividido em quatro fases principais: (1) Engenharia Reversa da LPS, (2) Análise da LPS, (3) Projeto da LPS e (4) Implementação da LPS.

Na primeira fase foi feita uma engenharia reversa do sistema UNIPAR para a extração das funcionalidades que deveriam estar contidas na UNIPAR-LPS. Na segunda fase, utilizando técnicas de modelagem do método PLUS, foram desenvolvidos modelos de requisitos onde os requisitos funcionais do sistema foram definidos em termos de casos de uso. Na terceira fase foi aplicado o método FArM no modelo de características da UNIPAR-LPS e através de quatro transformações feitas com base neste modelo, foi gerada a arquitetura inicial baseada em componentes. Diagramas de comunicação foram utilizados para auxiliar a definição das interfaces dos componentes. Por fim, na última fase, os componentes da UNIPAR-LPS foram desenvolvidos utilizando código legado do sistema UNIPAR e utilizando o modelo de componentes COSMOS*.

Tendo os componentes da UNIPAR-LPS desenvolvidos, produtos são gerados *on the fly* através da especificação de *workflows* científicos. Um *workflow* explicita quais componentes farão parte do produto, bem como a ordem de execução desses componentes. São os *workflows* que definem a ordem de execução das tarefas dos experimentos. Ao final da execução de cada experimento, os dados de entrada e resultados obtidos são armazenados para que possam ser reutilizados.

Este capítulo também apresentou alguns trabalhos correlatos a esta dissertação, que envolvem o estudo de LPS, *workflows* científicos e reengenharia de software.

No próximo capítulo serão apresentados três estudos de caso realizados para a avaliação da viabilidade da solução apresentada.

Capítulo 4

Estudos de Caso

Este capítulo apresenta três estudos de caso e suas avaliações. Os estudos foram analisados qualitativamente.

Com o primeiro estudo de caso, Seção 4.1 deste capítulo, foi feito um estudo da viabilidade da solução apresentada, a UNIPAR-LPS. O objetivo foi avaliar se a infraestrutura disponibilizada é capaz de gerar um produto equivalente ao módulo MAI do sistema UNIPAR. O segundo estudo de caso foi realizado para avaliar a UNIPAR-LPS em relação à sua capacidade de reutilização de componentes. Para tanto, reconstruímos o módulo MEC do sistema UNIPAR utilizando os mesmos componentes utilizados para a reconstrução do MAI, apenas alterando a metodologia, passando a utilizar a que representa a análise econômica. No terceiro estudo de caso, a intenção foi avaliar a solução como um todo, principalmente a facilidade de inclusão de novas variabilidades arquiteturais, representadas por novos componentes, à UNIPAR-LPS.

Os estudos de caso foram executados no laboratório UNISIM, em uma máquina Linux 64 bits, com 2 processadores, distribuição Debian Lenny 5.0, com 8Gb de memória RAM, processador Intel Core2Duo 3.33GHz. As simulações foram distribuídas em dois nós de um cluster com máquinas Linux 64 bits, com 64 processadores, distribuição Debian Squeeze, com 128Gb de RAM, processador AMD Opteron 2.7GHz.

4.1 Estudo de Caso: Reconstrução do Produto MAI

Neste estudo de caso o Produto MAI foi reconstruído a partir dos artefatos disponibilizados pela UNIPAR-LPS e comparar sua execução com a execução do Módulo MAI do sistema UNIPAR. Para tanto, executou-se um experimento já existente feito utilizando o MAI, que trabalha com o campo de petróleo de Namorado, descrito a seguir. O experimento foi obtido no Banco de Casos, ferramenta interna do UNISIM.

Para este estudo de caso executamos o MAI utilizando a metodologia Análise de

Sensibilidade (AS). Como já dito anteriormente, no Capítulo 3, Seção 3.2.1 o MAI utiliza o conceito de atributos incertos e atributos críticos. Os atributos incertos são definidos pelo usuário com base em sua experiência, levando em conta as incertezas relacionadas a um reservatório submetido à simulação de escoamento. Assim, atributos incertos são todos aqueles que possuem alguma incerteza associada, como por exemplo, porosidade ou permeabilidade do reservatório de petróleo. Nas pesquisas na área de petróleo, a metodologia de Análise de Sensibilidade é utilizada para definir quais atributos incertos mais influenciam no reservatório, selecionando os atributos mais influentes (chamados atributos críticos) para os próximos experimentos.

4.1.1 Informações Sobre o Cenário Estudado

Descoberto em novembro de 1975, o campo de Namorado é um campo real, localizado na Bacia de Campos, a 80 quilômetros da costa. Por se tratar de um campo não mais ativo, a Agência Nacional do Petróleo (ANP) disponibilizou informações sobre o campo para as várias instituições de ensino superior que mantém cursos na área de Engenharia de Petróleo.

O campo apresenta a extensão de aproximadamente 8 km de comprimento, 3 km de largura e está situado em lâmina d'água da ordem de 170 metros. O reservatório é formado por um arenito turbidítico confinado por falhas, sendo dividido em três blocos e apresenta boas características de porosidade e permeabilidade. O volume original de óleo deste reservatório é da ordem de 100 milhões de m3 e possui uma densidade de 28o API. A malha do modelo de simulação é composta por 51 blocos na direção x, 28 blocos na direção y e 6 camadas (direção z). Cada bloco possui as dimensões de 150m x 150m x 150m.

O início da produção deste campo foi em junho de 1979 através de 4 poços para o navio P.P. Moraes. Foram instaladas no campo duas plataformas fixas, PNA-1 e PNA-2. Hoje o campo conta com quatro poços produtores e treze poços injetores.

4.1.2 Planejamento do Estudo de Caso

O propósito deste estudo de caso é verificar a capacidade da UNIPAR-LPS de reproduzir o módulo MAI do sistema legado UNIPAR. Para isso, vamos executar o MAI utilizando a metodologia de Análise de Sensibilidade, com um experimento já existente. Depois vamos especificar um *workflow* que gere o produto MAI, derivado da UNIPAR-LPS, e executar o mesmo experimento no produto gerado.

Ao final das execuções, vamos analisar os resultados obtidos e os arquivos gerados. Eles devem ser idênticos.

4.1.3 Execução do Estudo de Caso

Os arquivos de entrada de dados necessários para os módulos do UNIPAR foram mantidos na UNIPAR-LPS. Portanto, as informações que seguem referentes à preparação dos arquivos de entrada são válidas tanto para o MAI quanto para a UNIPAR-LPS.

Planejamento do Experimento Científico

Em experimentos de Análise de Incertezas e Risco, o modelo numérico que representa o reservatório é chamado de modelo base. Uma vez que se tenha definido o modelo base e os atributos incertos, é necessário criar os arquivos referentes aos atributos incertos e incluir referência a estes atributos neste modelo base.

Para este estudo foram utilizados 9 atributos incertos, com 3 níveis de incerteza: Pessimista, Provável e Otimista. A tabela exibida na Figura 4.1 ilustra os atributos utilizados e os valores associados a cada nível de incerteza. Cada nível de um atributo incerto precisa ser representado por um arquivo de inclusão. Neste caso, o usuário precisa ter, ao todo, 27 arquivos de inclusão, pois são 9 atributos incertos com 3 níveis de incerteza cada um.

	Atributos Incertos	Níveis de Incerteza		
		Pessimista	Provável	Otimista
1	Contato Óleo-Água	3080	3100	3120
2	Compressibilidade da Rocha	$20{,}0 * 10^{-6}$	$50{,}0 * 10^{-6}$	$80{,}0 * 10^{-6}$
3	Porosidade	0,8	1	1,2
4	Netpay	0,8	1	1,2
5	Permeabilidade Horizontal	0,25	0,66	1,41
6	Permeabilidade Vertical	0,25	0,66	1,41
7	Permeabilidade Relativa	Arq. kr0.inc	Arq. kr2.inc	Arq. kr1.inc
8	PVT	Arq. pvt2.inc	Arq. pvt0.inc	Arq. pvt1.inc
9	Modelo Estrutural	Menor	Barreira	Fluxo

Figura 4.1: Atributos incertos utilizados no Estudo de Caso 1

Nesta execução, de Análise de Sensibilidade, serão gerados nove arquivos de simulação baseando-se nos nomes e nos níveis dos atributos, pois será variado um atributo por vez.

Também é necessário gerar um arquivo de entrada (chamado neste estudo de "estudo1.mai") com informações tais como o simulador a ser utilizado, tempos para realização dos cálculos, e outras informações que não serão detalhadas pois não são relevantes ao estudo de caso.

Após todos os arquivos de entrada gerados, pode-se começar a execução do experimento.

Execução do Experimento Científico no módulo MAI

Apesar do sistema UNIPAR contar com uma interface gráfica para facilitar a execução dos experimentos, neste estudo vamos utilizar a interface via linha de comando para a execução do módulo MAI pois a UNIPAR-LPS ainda não conta com uma interface gráfica.

Para a escolha dos parâmetros de execução do MAI pode-se executar o comando "mai -h"e todos os parâmetros disponíveis são apresentados juntamente com uma breve descrição. A Figura 4.2 mostra parte da saída de tela do comando.

```
************************************************
*                      MAI                     *
*    Módulo de Análise de Incertezas e Risco   *
*                   5.5.2.2                    *
*                   UNISIM                     *
*        DEP - FEM - CEPETRO - UNICAMP          *
*                                               *
*               Apoio: PETROBRAS                *
*      Colaboradores: CNPq, FAPESP e FINEP      *
************************************************

MODO DE USO:

   mai  [-a] [-b] [-c] [-d] [-e <nome configuracao>] [-f] [-g] [-i] [-j
        <precisao dados>] [-k] [-l <nivel de log>] [-L] [-m] [-n] [-o] [-p
        <nome parametros>] ...  [-P] [-q] [-r] [-R <tentativas de reenvio>]
        [-s] [-v] [-w] [-W <numero de processadores>] [-h] [--] <file>


Onde:

   -a,  --mantem-arquivos-dat
     Mantém os arquivos dat normalizados

   -b,  --refaz-calculo-economico
     Apenas refaz o cálculo econômico

   -c,  --hcld
     Executa no modo Hipercubo Latino Discreto (HCLD)

   -d,  --arvore-derivacao
     Executa no modo  Árvore de Derivação

   -e <nome configuracao>,  --escalonador <nome configuracao>
     Indica qual configuração de escalonador será utilizada (simulação
     distribuída)

   -f,  --gera-fluxo-caixa
     Gera o arquivo com o fluxo de caixa para cada modelo econômico e para
     cada modelo de simulação (<modelo_simulação>.<modelo_economico>.csv)

   -g,  --apenas-gera-dat
     Apenas gera os arquivos normalizados de simulação (.tmp.dat ou
     .TMP.DATA)

   -i,  --mantem-saidas
     Mantém todos os arquivos gerados pelo MAI / MEC / DSGU, pelo simulador
     e pelo Report
```

Figura 4.2: Help do módulo MAI

Sabendo todos os parâmetros possíveis para a execução do MAI, para este estudo de caso foram selecionados apenas dois parâmetros, mostrados na Tabela 4.1 com seus respectivos valores:

Tabela 4.1: Parâmetros selecionados para o MAI, seus respectivos valores e descrição

Parâmetro	Valor	Descrição
-s	NA	Modo de AS
-e	torque	Utiliza a configuração de escalonador "torque"

Tendo os parâmetros escolhidos, para a execução do MAI foi utilizado o comando:

```
mai -s -e torque estudo1.mai
```

onde "estudo1.mai"é o nome do arquivo de entrada do MAI.

Execução do Experimento Científico utilizando a UNIPAR-LPS

Para a comparação correta da execução do MAI com a UNIPAR-LPS, é preciso considerar alguns pontos que são padrão da execução do MAI:

- O MAI sempre reaproveita todos os arquivos já simulados existentes no diretório (arquivos de saída gráfica, saída texto, entrada e saída do Report e arquivos UNIPRO);
- O MAI mantém no diretório de trabalho somente os arquivos UNIPRO;
- O MAI calcula somente as funções-objetivo técnicas (que são produções e injeções de óleo, água e gás, e fatores de recuperação);
- O MAI sempre utiliza o nível de log "médio"(representado por "2");
- Os modelos de simulação sempre são verificados antes de enviar para simular.

Estes pontos devem ser levados em consideração ao realizar a configuração do produto da UNIPAR-LPS.

Como descrito no Capítulo 3, na UNIPAR-LPS a variabilidade foi implementada tanto em nível arquitetural, ou seja, no nível dos componentes, quanto por parâmetros de execução.

Para escolher os componentes e parâmetros a fim de reproduzir exatamente a execução do sistema UNIPAR, neste estudo de caso utilizou-se o modelo chamado **Configura Produto**, gerado na Fase 4: Implementação da UNIPAR-LPS (Seção 3.3.4).

Este documento é o modelo de características após a última transformação do FArM, com informações de como a característica foi implementada na UNIPAR-LPS. Com este modelo o pesquisador pode visualizar e escolher as características que seu produto terá, ou seja, resolver a variabilidade da LPS, e gerar o arquivo de especificação do *workflow*.

A Figura 4.3 mostra o modelo com as variabilidades resolvidas (marcadas na cor cinza) para este experimento. As linhas tracejadas informam como a característica foi implementada.

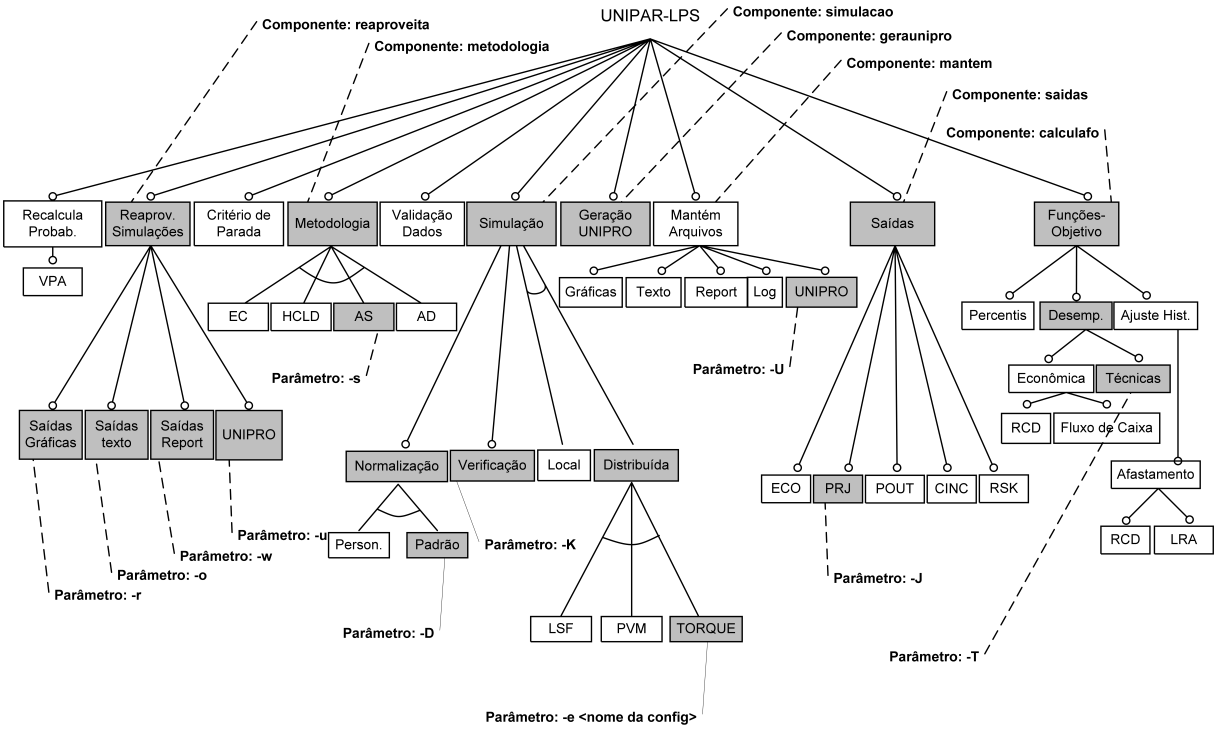


Figura 4.3: Configura Produto resolvido para o estudo de caso: Reconstrução do Produto MAI

Para utilizar as supercaracterísticas, deve-se utilizar o componente indicado. As subcaracterísticas são escolhidas através de parâmetros para os componentes de sua supercaracterística.

Para simplificar a figura, foram exibidos somente os componentes e parâmetros utilizados neste estudo de caso. A Tabela 4.2 mostra os componentes escolhidos, seus parâmetros e valores de parâmetros.

A UNIPAR-LPS precisa de um arquivo de entrada da própria linha de produtos, o arquivo que especifica o o *workflow* científico. Neste arquivo são especificados todos os componentes e parâmetros de execução que compõem o produto que será gerado. Tendo os componentes e parâmetros escolhidos, geramos o arquivo mostrado na Figura 4.4.

Depois de todos os arquivos gerados corretamente no diretório de trabalho, foi executado o comando:

```
lps exec workflow estudo1.xml
```

Tabela 4.2: Componentes e parâmetros selecionados para o *workflow*, seus valores e descrição

Componente Metodologia		
Parâmetro	Valor	Descrição
-s	NA	Modo de AS
-l	2	Nível de log médio
Componente Simulação		
Parâmetro	Valor	Descrição
-D	NA	Normaliza os arquivos de simulação utilizando a normalização padrão
-K	NA	Verifica os arquivos antes de enviar para simulação
-l	2	Nível de log médio
-e	torque	Utiliza a configuração de escalonador "torque"
Componente Gera UNIPRO		
Parâmetro	Valor	Descrição
-l	2	Nível de log médio
Componente CalculaFO		
Parâmetro	Valor	Descrição
-D	NA	Calcula FOs de desempenho
-l	2	Nível de log médio
Componente Gera Saídas		
Parâmetro	Valor	Descrição
-J	NA	Gera Arquivo de Projeto
-l	2	Nível de log médio

```
<?xml version="1.0" encoding="UTF-8"?>
<main>
    <componentsList>
            <component name="metodologia" arguments="-s -l 2" />
            <component name="reaproveitaarquivos" arguments="-u -l 2"/>
            <component name="simula" arguments="-D -K -e torque -l 2"/>
            <component name="geraunipro" arguments="-l 2 -L"/>
            <component name="calculafo" arguments="-T -l 2 -L"/>
            <component name="gerasaidas" arguments="-J -l 2"/>
        </componentsList>
</main>
```

Figura 4.4: Workflow que representa a execução do MAI

4.1.4 Resultados

Ao fim da execução do MAI e do produto derivado da UNIPAR-LPS gerado pelo *workflow* definido, encontravam-se no diretório de trabalho os seguintes arquivos:

- Arquivos do tipo UNIPRO (*.unipro);
- Arquivo de projeto do MAI (projeto_estudo1.prj).

Os arquivos de saída da execução do MAI foram comparados aos de saída do produto gerado pela UNIPAR-LPS e são idênticos.

4.2 Estudo de Caso: Reconstução do Produto MEC

Neste estudo de caso a execução da metodologia de Análise de Econômica da LPS foi comparada à execução do módulo MEC, mas o foco é o reúso dos componentes utilizados no estudo de caso anterior. Neste estudo, executou-se um experimento já existente do MEC que trabalha com o campo de petróleo PUNQ. Este experimento foi obtido no Banco de Casos, ferramenta interna do UNISIM.

4.2.1 Informações Sobre o Cenário Estudado

O MEC é o módulo do sistema UNIPAR responsável pela análise econômica dos resultados simulados de modelos de reservatório. Utilizando informações financeiras e sobre a produção, o MEC realiza cálculos de índices econômicos que permitem avaliar a viabilidade de um projeto de exploração de petróleo.

A fatia governamental inclui tributos, contribuições sociais e as participações governamentais. As participações governamentais compreendem os bônus de assinatura, royalties, participação especial, pagamento por ocupação ou retenção de área. As categorias de tributação dividem-se em:

- Impostos proporcionais à produção: royalties, PIS/Pasep e COFINS, participação especial
- Impostos proporcionais ao lucro: Imposto de Renda de Pessoa Jurídica (IRPJ) e Contribuição Social sobre Lucro Líquido (CSLL)
- Bônus fixos: bônus de assinatura e pagamento pela ocupação ou retenção de área

O caso PUNQ-S3 foi obtido de um estudo de engenharia de reservatórios sobre um campo real operado pela *Elf Exploration Production*, empresa operadora do ramo petrolífero [5]. Ele é um reservatório de petróleo, um campo novo, considerado de tamanho pequeno. O campo foi inicialmente desenvolvido através de seis poços produtores localizados em torno do contato Gás-Óleo.

4.2.2 Planejamento do Estudo de Caso

O objetivo deste estudo de caso é avaliar a capacidade da UNIPAR-LPS em relação à reutilização de componentes de software e também avaliar a reconstrução do produto MEC. Apesar dos módulos MAI e MEC do sistema UNIPAR terem partes iguais, eles possuem muito código duplicado. Neste estudo utilizou-se o *workflow* científico especificado no estudo de caso anterior (Reconstrução do Produto MAI), alterando apenas o parâmetro do componente "metodologia"para que ele execute a metodologia Econômica. Para validar o produto gerado, comparou-se sua execução com a execução do módulo MEC. Ao final das execuções, os resultados obtidos e arquivos gerados são analisados e eles devem ser idênticos.

4.2.3 Execução do Estudo de Caso

Os arquivos de entrada de dados do módulo MEC foram mantidos na UNIPAR-LPS. Sendo assim, as informações que seguem referentes à preparação dos arquivos de entrada são válidas tanto para o MEC quanto para a UNIPAR-LPS.

Planejamento do Experimento Científico

Para que o VPL seja calculado, é necessário informar aos programas dados econômicos. Neste caso os seguintes valores foram utilizados:

- Preço de venda do óleo: US$ 40,00/bbl
- Taxa de atratividade: 10% ao ano
- Alíquota de imposto de renda e contribuição social 34%

- Alíquota de PIS/Cofins: 9,25%
- Royalties: 10%
- Investimento inicial no campo: US$ 200 milhões
- Investimento na plataforma: US$ 500 milhões
- Custo médio total de abertura e completação: US$ 40 milhões/poço
- Custo de produção de óleo: US$ 5,00/bbl
- Custo de produção de água: US$ 2,00/bbl
- Custo de injeção de água: US$ 2,00/bbl
- Custo de descomissionamento do campo: US$ 100 milhões

Os valores devem ser informados em um arquivo do tipo "Seção Financeira", utilizado tanto pelo UNIPAR quanto pela UNIPAR-LPS.

Execução do Experimento Científico utilizando o MEC

Assim como o estudo de caso anterior, o MEC foi executado via linha de comando. Para a escolha dos parâmetros de execução do MEC pode-se executar o comando "mec –h"e todos os parâmetros disponíveis são apresentados juntamente com uma breve descrição. A Figura 4.5 mostra parte da saída de tela do comando.

Sabendo todos os parâmetros possíveis para a execução do MEC, para este estudo de caso foi selecionado apenas um parâmetro, relativo à escolha do tipo de simulação. Como este experimento possui apenas um modelo de simulação, a simulação foi realizada na máquina local, escolhendo o parâmetro de execução -m".

Para a execução do MEC foi utilizado o comando:

```
mec -m estudo2.mec
```

Assim como no MAI, alguns padrões serão adotados no MEC pelo sistema UNIPAR:

- O MEC sempre reaproveita todos os arquivos já simulados existentes no diretório (arquivos de saída gráfica, saída texto, entrada e saída do Report e arquivos UNIPRO);
- O MEC mantém no diretório de trabalho somente os arquivos UNIPRO;
- O MEC sempre calcula as funções-objetivo técnicas (que são produções e injeções de óleo, água e gás, e fatores de recuperação) e econômicas;

```
************************************************
*                      MEC                     *
*         Módulo de Cálculo Econômico          *
*                    5.5.2.2                   *
*                    UNISIM                    *
*       DEP - FEM - CEPETRO - UNICAMP          *
*                                              *
*               Apoio: PETROBRAS               *
*      Colaboradores: CNPq, FAPESP e FINEP     *
************************************************

MODO DE USO:

   mec  [-a] [-A] [-e <nome configuracao>] [-f] [-i] [-j <precisao dados>]
        [-k] [-l <nivel de log>] [-L] [-m] [-n] [-o] [-p <nome parametros>]
        ...  [-P] [-q] [-r] [-R <tentativas de reenvio>] [-v] [-x] [-w] [-W
        <numero de processadores>] [-z] [-h] [--] <file>


Onde:

   -a,  --mantem-arquivos-dat
     Mantém os arquivos dat normalizados

   -A,  --datas-anuais
     Insere datas anuais nos arquivos de simulação

   -e <nome configuracao>,  --escalonador <nome configuracao>
     Indica qual configuração de escalonador será utilizada (simulação
     distribuída)

   -f,  --gera-fluxo-caixa
     Gera o arquivo com o fluxo de caixa para cada modelo econômico e para
     cada modelo de simulação (<modelo_simulação>.<modelo_economico>.csv)

   -i,  --mantem-saidas
     Mantém todos os arquivos gerados pelo MAI / MEC / DSGU, pelo simulador
     e pelo Report

   -j <precisao dados>,  --precisao <precisao dados>
     Especifica a precisão dos dados de saída. O valor deve estar entre 1 e
     20.

   -k,  --nao-verifica
     Não verifica os arquivos antes de enviar para simulação

   -l <nivel de log>,  --log <nivel de log>
     Gera um arquivo de log para cada um dos módulos executados, com nível
     de detalhe especificado: 0 - Baixo; 1 - Médio; 2 - Alto

   -L,  --mantem-arquivos-log
     Mantém os arquivos de log no final da execução
```

Figura 4.5: Parte da saída de tela do comando "help"do MEC

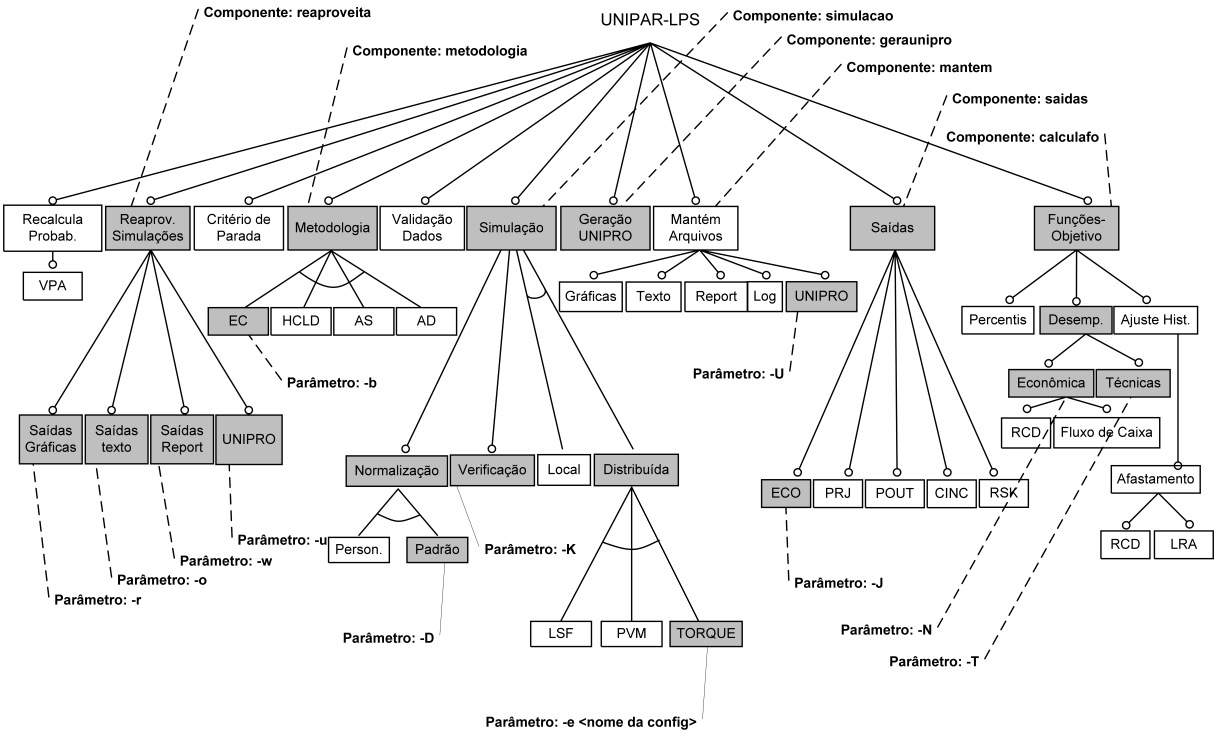


Figura 4.6: Configura Produto resolvido para o estudo de caso: Reutilização de Componentes

- O MEC sempre utiliza o nível de log "médio"(representado por "2");
- Os modelos de simulação sempre são verificados antes de enviar para simular.

Execução do Experimento Científico utilizando a UNIPAR-LPS

Foi utilizado novamente o documento **Configura Produtos** para escolher os componentes e parâmetros a fim de reproduzir exatamente a execução do módulo MEC do sistema UNIPAR. A Figura 4.6 exibe o modelo com a variabilidade resolvida para este estudo de caso.

A Tabela 4.3 mostra os componentes que foram selecionados para compor este produto, seus parâmetros, valores e uma breve descrição.

Depois de todos os arquivos gerados corretamente no diretório de trabalho, foi executado o comando:

```
lps exec workflow estudo2.xml
```

Tabela 4.3: Componentes e parâmetros selecionados para o *workflow*, seus respectivos valores e descrição

Componente Metodologia		
Parâmetro	Valor	Descrição
-b	NA	Cálculo Econômico
-l	2	Nível de log médio
Componente Simulação		
Parâmetro	Valor	Descrição
-n	D	Normaliza os arquivos de simulação utilizando a normalização padrão
-m	NA	Simula localmente
-l	2	Nível de log médio
Componente Gera UNIPRO		
Parâmetro	Valor	Descrição
-l	2	Nível de log médio
Componente CalculaFO		
Parâmetro	Valor	Descrição
-D	NA	Calcula FOs de desempenho
-C	NA	Calcula funções econômicas
-l	2	Nível de log médio
Componente Mantém Arquivos		
Parâmetro	Valor	Descrição
-u	NA	Mantém arquivos UNIPRO
-l	2	Nível de log médio
Componente Gera Saídas		
Parâmetro	Valor	Descrição
-O	NA	Gera Arquivo de Saída
-l	2	Nível de log médio

```
<?xml version="1.0" encoding="UTF-8"?>
<main>
    <componentsList>
            <component name="metodologia" arguments="-b -l 2 " />
            <component name="reaproveitaarquivos" arguments="-u -l 2"/>
            <component name="simula" arguments="-D -K -m -l 2"/>
            <component name="geraunipro" arguments="-l 2"/>
            <component name="calculafo" arguments="-T -N -l 2"/>
            <component name="gerasaidas" arguments="-C -l 2"/>
        </componentsList>
</main>
```

Figura 4.7: Workflow que representa a execução do MEC

O arquivo de especificação de *workflow* utilizado para este estudo de caso é exibido na Figura 4.7:

4.2.4 Resultados

No final da execução do MEC e do produto da UNIPAR-LPS gerado pelo *workflow* definido, foram mantidos os seguintes arquivos no diretório de trabalho:

- Arquivo do tipo UNIPRO correspondente ao modelo de simulação utilizado;
- Arquivo de saída do MEC (saida_estudo2.eco)

Os arquivos de saída do MEC foram comparados aos de saída do produto gerado pela execução do *workflow* especificado e são idênticos.

4.3 Estudo de Caso: Avaliar a Facilidade de Modificação da UNIPAR-LPS

Sistemas de software evoluem, caso contrário, seu uso pode tornar-se menos satisfatório [38]. No que diz respeito a LPS, existem diversos cenários de evolução, por exemplo, adição e remoção de características, mudança de característica opcional para obrigatória e vice-versa. Esses cenários podem modificar significativamente arquiteturas de LPS e eventualmente causar aumento de gastos com manutenção. Por isso, é importante para as organizações projetar arquiteturas de LPS fáceis de evoluir [55].

Em linhas de produtos, funcionalidades de software são mapeadas para características, representadas em um modelo de características. No caso particular da solução proposta neste trabalho, uma nova característica, como por exemplo uma nova metodologia de estudo de campos de petróleo, pode ser adicionada à LPS pelo usuário final, e pode

corresponder a um componente de software desenvolvido pelo próprio pesquisador, como também pode ser algum sistema de software desenvolvido por terceiros. O objetivo deste estudo de caso é avaliar a adição de uma nova variante à LPS e utilizá-la na execução de um experimento científico.

4.3.1 Planejamento do Estudo de Caso

A intenção deste estudo de caso é avaliar a facilidade de modificação da UNIPAR-LPS através da inserção de uma nova variante à UNIPAR-LPS. Utilizou-se o mesmo fluxo de trabalho especificado no Estudo de Caso: Reconstrução do Produto MAI, apresentado na Seção 4.1, apenas alterando a metodologia utilizada para a metodologia desenvolvida pelo novo componente.

4.3.2 Execução

Para que uma nova metodologia fosse incluída à LPS, foi necessário criar uma nova variante no modelo de características, em um ponto de variação existente, neste caso **Metodologia**, e a implementação desta característica é dada através da inclusão de um novo artefato no núcleo de artefatos da UNIPAR-LPS.

Criamos um componente ("caixa-preta") desenvolvido em MatLab que, tendo como entrada um modelo numérico de um reservatório, gera 10 modelos derivados. O intuito do componente desenvolvido é substituir o componente **Metodologia** disponível na UNIPAR-LPS por um novo componente. Este novo componente poderia corresponder à implementação de qualquer nova metodologia desenvolvida por um pesquisador ou aluno do UNISIM, em qualquer linguagem de programação. Para que o componente possa ser utilizado no experimento, é necessário que sua saída siga a especificação de interfaces definida para o próximo componente que será executado pelo *workflow*. As especificações das interfaces dos componentes estão disponíveis em: www.unisim.cepetro.unicamp.br/ fernanda. O componente criado foi chamado de "metodologia_personalizada".

O modelo **Configura Produto** exibido na Figura 4.8 mostra o ponto de variação que deve ser alterado para a utilização do componente "metodologia_personalizada".

Para que o componente possa ser executado pelo *workflow*, é necessário que seu executável seja disponibilizado no diretório "lps/bin/'arch'", onde 'arch' é a arquitetura da máquina onde o componente foi compilado e será executado. Neste caso, disponibilizamos o componente em "lps/bin/LINUX64". O arquivo de especificação do *workflow* ficou como exibido na Figura 4.9.

Depois de todos os arquivos gerados corretamente no diretório de trabalho, foi executado o comando:

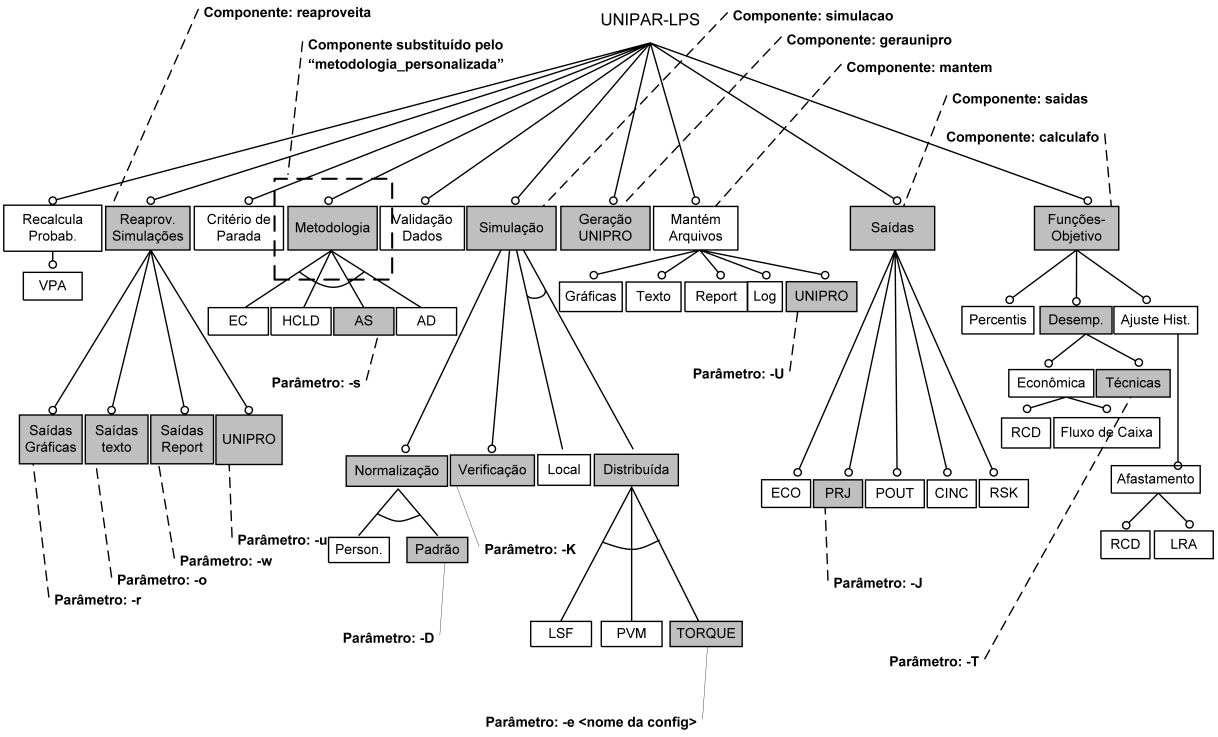


Figura 4.8: Configura Produto: utilização de um novo componente

```
<?xml version="1.0" encoding="UTF-8"?>
<main>
    <componentsList>
            <component name="metodologia_personalizada" />
            <component name="reaproveitaarquivos" arguments="-u -l 2"/>
            <component name="simula" arguments="-D -K -e torque -l 2"/>
            <component name="geraunipro" arguments="-l 2 "/>
            <component name="calculafo" arguments="-T -l 2"/>
            <component name="gerasaidas" arguments="-J -l 2"/>
        </componentsList>
</main>
```

Figura 4.9: Workflow personalizado utilizando o componente metodologia_personalizada

```
lps exec workflow estudo3.xml
```

4.3.3 Resultados

A execução do experimento ocorreu sem problemas até o fim. Foram mantidos os seguintes arquivos no diretório de trabalho:

- Arquivos do tipo UNIPRO (*.unipro);

- Arquivos de saída formato "PRJ"(projeto_estudo3.prj).

Os dados de saída não foram analisados pois a intenção era apenas testar a inserção do novo componente.

4.4 Avaliação de Resultados

Os dois primeiros estudos de caso comparavam a execução de um experimento realizado utilizando o MAI e um experimento realizado utilizando o MEC, módulos do UNIPAR, com a execução dos produtos correspondentes na UNIPAR-LPS. Estes dois estudos de caso foram utilizados para validar, além da UNIPAR-LPS, o apoio à execução de *workflows* científicos. O terceiro caso foi utilizado para analisar a facilidade ou dificuldade de inclusão de novos componentes na UNIPAR-LPS. Os itens a seguir mostram as conclusões de cada análise.

1. **LPS**

   Pelo fato dos resultados gerados na execução dos módulos do sistema UNIPAR serem idênticos aos resultados gerados na execução dos produtos gerados pela UNIPAR-LPS (produtos gerados em tempo de execução, a partir dos arquivos de configuração do *workflow*), fica claro que a UNIPAR-LPS atingiu o primeiro objetivo, substituir o sistema legado.

   Entretanto, o estudo de caso apresentado na Seção 4.1 mostrou um ponto que precisa ser analisado. Em uma execução do MAI, apenas o arquivo base é normalizado, e isto é suficiente já que todos os outros arquivos são derivados do arquivo base. Na UNIPAR-LPS, a funcionalidade de normalização foi incorporada ao componente "Simula". Desta maneira todos os arquivos que são informados a este componente são normalizados. Este passo, no caso do MAI, é desnecessário. Apesar disso, pelo fato do processo de normalização ser extremamente rápido se comparado ao tempo de demora de uma simulação, este retrabalho não chega a ser considerado um problema.

   Quanto ao desempenho de execução, à parte do fato analisado acima, as execuções do produto derivado da UNIPAR-LPS foi discretamente maior do que a execução dos módulos do MAI.

2. **Método Proposto**

   Pode-se concluir que o método proposto para o desenvolvimento de uma LPS a partir de um código legado é viável, uma vez que a partir dele conseguimos coletar as características do sistema UNIPAR e reproduzi-las na UNIPAR-LPS. No total,

18 das 32 características existentes na UNIPAR-LPS foram testadas nos estudos de caso e funcionaram corretamente.

3. **Apoio à execução de experimentos**

   Os dois primeiros estudos de caso, Reconstrução do Produto MAI, Seção 4.1, e Reutilização de Componentes, Seção 4.2, mostraram que a solução proposta para a resolução de variabilidade de cada produto gerado (escolha através dos componentes e parâmetros de execução de cada componente, em tempo de execução), ou seja, o arquivo XML de configuração do *workflow* é válido.

   Apesar da execução do *workflow* exigir um passo a mais (a geração do arquivo com a configuração do fluxo de trabalho) se comparada à execução dos módulos do sistema UNIPAR, futuramente sugere-se a implementação uma interface gráfica para facilitar este passo.

   O Estudo de Caso 3: Avaliação da adição de novas características na UNIPAR-LPS, Seção estudo-caso3, mostrou que é possível a inclusão de um novo componente à LPS, sem a intervenção da equipe de desenvolvimento do UNISIM.

4. **Reutilização de Componentes**

   Os dois primeiros estudos de caso, Reconstrução do Produto MAI, Seção 4.4, e Reutilização de Componentes, Seção 4.7, especificaram *workflows* científicos que geraram produtos derivados da UNIPAR-LPS capazes de reproduzir os módulos MAI e MEC do sistema UNIPAR. Os dois produtos gerados pela UNIPAR-LPS utilizavam o mesmo conjunto de componentes, mostrando assim a possibilidade de reutilização de componentes em diferentes produtos.

4.5 Resumo do Capítulo

Neste capítulo foram apresentados três estudos de caso e suas avaliações. A intenção com o primeiro estudo de caso, Seção 4.1, foi avaliar a viabilidade da solução apresentada, a UNIPAR-LPS, através da reconstrução do MAI do sistema UNIPAR. O segundo estudo de caso foi realizado para avaliar a UNIPAR-LPS em relação à sua capacidade de reutilização de componentes, através da reconstrução do MEC do sistema UNIPAR, utilizando os mesmos componentes utilizados para a reconstrução do MAI, apenas alterando a metodologia. No terceiro estudo de caso, a intenção foi avaliar a solução como um todo, principalmente a facilidade de inclusão de novas variabilidades, representadas por novos componentes, à UNIPAR-LPS.

Os três estudos de caso apresentaram resultados positivos, indicando que a UNIPAR-LPS é capaz de substituir o sistema UNIPAR e mostrando-se fácil de ser adaptada para a utilização de novas metodologias de pesquisa, através da adição de novos componentes.

O próximo capítulo apresenta as conclusões finais desta dissertação e destaca suas principais contribuições.

Capítulo 5

Conclusões e Trabalhos Futuros

Neste capítulo são apresentadas as conclusões deste trabalho e suas principais contribuições (Seção 5.1) e os trabalhos futuros (Seção 5.2).

5.1 Conclusões

Hoje o UNISIM conta com dois principais software que disponibilizam ferramentas de apoio à pesquisa aos engenheiros de reservatórios. São eles o sistema UNIPAR e o sistema MERO. O sistema MERO, ainda em fase inicial de desenvolvimento, não foi abordado no escopo deste projeto. O foco principal deste trabalho foi identificar e propor soluções para as deficiências encontradas no sistema UNIPAR.

O sistema UNIPAR é um sistema legado bastante utilizado por alunos e pesquisadores do UNISIM, e também por algumas unidades Petrobras. O sistema UNIPAR possui um código muito antigo, mal estruturado e possui uma pobre documentação técnica. Ele vem sendo constantemente atualizado, o que acaba por entrelaçar ainda mais o código. Alterações consideradas simples acabam tornando-se complexas pela falta de documentação técnica e má estruturação do código fonte.

A fim de solucionar o problema de alto acoplamento do código, foi proposta a adoção de uma LPS baseada em componentes utilizando uma abordagem extrativa, para que ela substitua o sistema UNIPAR. A LPS foi desenvolvida utilizando código existente do sistema UNIPAR, mas a arquitetura criada permite a substituição do código antigo por código novo (e melhor estruturado) facilmente. A LPS gerada foi utilizada como base para um ambiente de *workflow* científico, a ser implantado no UNISIM.

A abordagem extrativa proposta utiliza o arcabouço da Reengenharia Orientada a características (Seção 2.3.1), técnicas de modelagem da metodologia PLUS (Seção 2.1.8) e o método FArM (Seção 2.2) para a geração da arquitetura inicial da LPS. A variabilidade da LPS é resolvida pelo usuário final. Ou seja, os produtos são gerados *on the fly*, de

forma personalizada para o pesquisador. Não é permitido criar novos pontos de variação, mas permite-se que novos componentes sejam adicionados ao núcleo de artefatos da LPS e utilizados nos produtos, sem a necessidade de equipe de desenvolvimento para a inclusão do novo componente.

De maneira geral buscou-se de forma simples resolver alguns dos problemas existentes hoje no UNISIM. Pode-se destacar como principais contribuições deste trabalho os itens explicados a seguir:

1. **Método extrativo para adoção de LPS**

   Neste projeto aplicamos o método extrativo de adoção de LPS proposto em um sistema legado real, o sistema UNIPAR. A abordagem mostrou-se capaz de identificar todas as características do sistema legado e representá-las na LPS criada, possibilitando assim estruturar o código existente, sem perda de funcionalidade do software, tornando-o fácil de ser mantido. Assim, os produtos gerados pela LPS são capazes de substituir o sistema legado sem perda de funcionalidades.

2. **Reúso de código legado**

   Através da componentização de código legado foram gerados componentes de software, permitindo que o código seja aproveitado, mas estruturando-o para que seu alto acoplamento não prejudique a evolução da LPS.

3. **Arquitetura baseada em componentes**

   A aplicação da abordagem proposta no sistema legado UNIPAR gerou uma arquitetura baseada em componentes que será reutilizada no sistema MERO.

4. **Reúso de componentes**

   Como visto em capítulos anteriores, o sistema UNIPAR possui muito código duplicado, o que dificulta a modificabilidade e evolução do sistema. Na UNIPAR-LPS foram especificados 11 componentes e 13 subcomponentes que podem ser facilmente utilizados em diversos produtos, basta que suas interfaces sejam respeitadas.

5. **LPS para apoio ao *workflow* científico**

   A LPS gerada é capaz de gerar produtos a fim de substituir o sistema legado sistema UNIPAR e também novos produtos (gerados *on the fly* ou não) utilizando os artefatos produzidos para a LPS.

6. **Ambiente de *workflow* científico** O protótipo do ambiente proposto contempla todas as fases de um *workflow* científico, permitindo a concepção de fluxos de

trabalho (ou seja, geração de produtos membros da LPS em tempo de execução), execução dos fluxos de trabalho, compartilhamento de dados e reúso de experimentos, facilitando assim a pesquisa científica.

5.2 Trabalhos Futuros

Os resultados dos estudos de caso executados para a avaliação da solução proposta neste trabalho foram satisfatórios o suficiente para que as definições do ambiente de *workflow* científico sejam implementadas no sistema MERO.

A seguir são apresentadas algumas sugestões para trabalhos futuros que complementam o trabalho feito nessa dissertação. São elas:

- Desenvolvimento de uma interface gráfica para a geração do arquivo de configuração do fluxo de trabalho;
- Desenvolvimento de uma interface gráfica para a visualização e recuperação de dados compartilhados (informações sobre os experimentos executados anteriormente);
- Integração dos componentes desenvolvidos a partir do sistema UNIPAR no sistema MERO;
- Integração de banco de dados ao componente que trata o compartilhamento e recuperação de dados compartilhados;
- Suporte à especificação de iterações (*loop*) no *workflow*.

Apêndice A

Diagramas de Casos de Uso

Na modelagem de casos de uso para LPS, apenas alguns dos casos de uso e atores são requeridos por um determinado membro da linha de produtos. Os casos de uso que são exigidos por todos os membros da linha de produtos são referidos como casos de uso do tipo ***kernel***. Os casos de uso **opcionais** são necessários apenas por alguns membros da LPS. Os casos de uso do tipo **alternativos** representam os diferentes casos de uso que são necessários por diferentes membros da LPS [32]. As Figuras A.1 e A.2 exibem os casos de uso gerados para um cenário de execução completa do Produto MAI e Produto MEC da UNIPAR-LPS, respectivamente. A Figura A.3 exibe os casos de uso para a definição de dados econômicos, utilizado tanto pelo MAI quanto MEC.

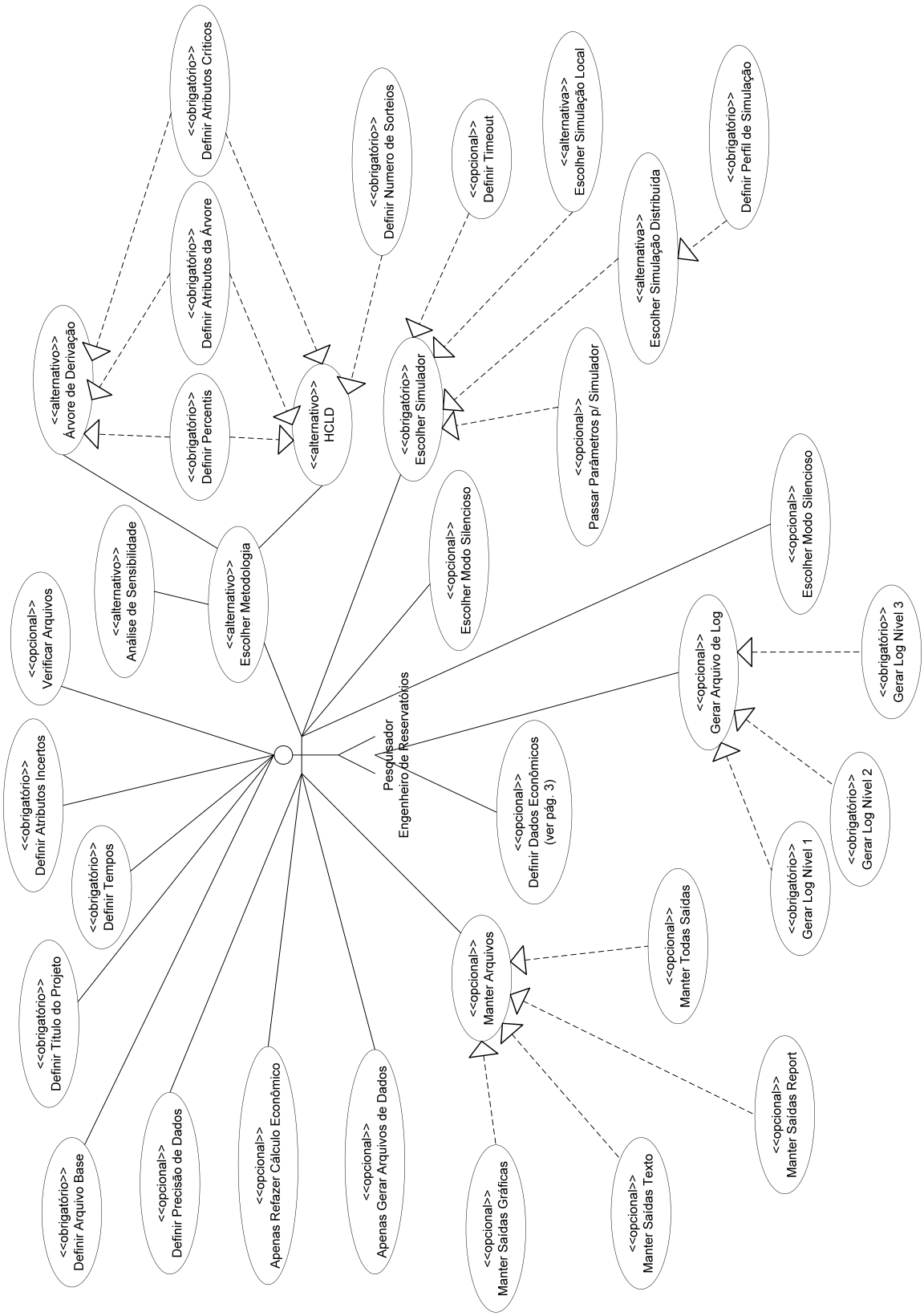


Figura A.1: Diagrama de Caso de Uso: Cenário de execução do MAI

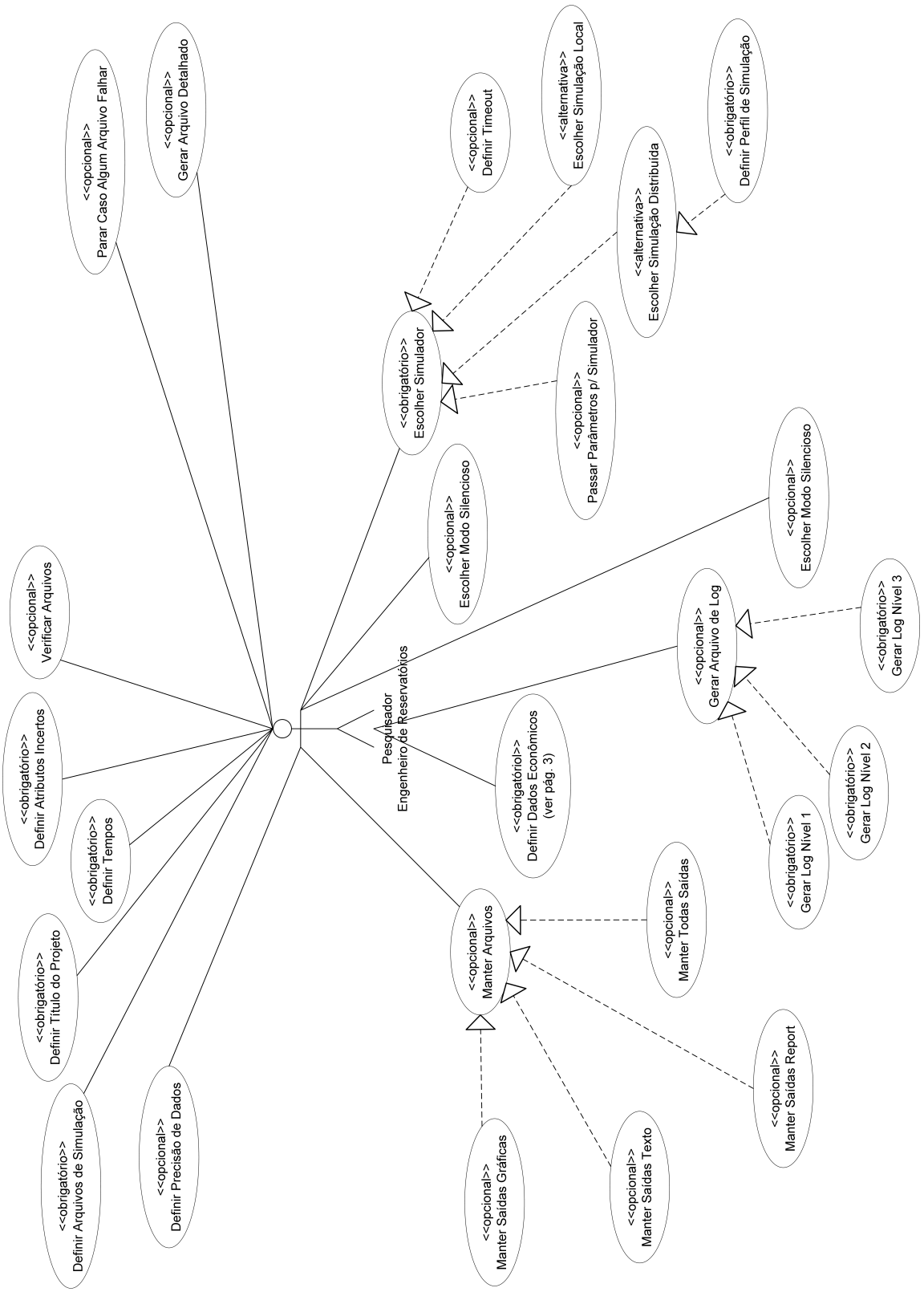


Figura A.2: Diagrama de Caso de Uso: Cenário de execução do MEC

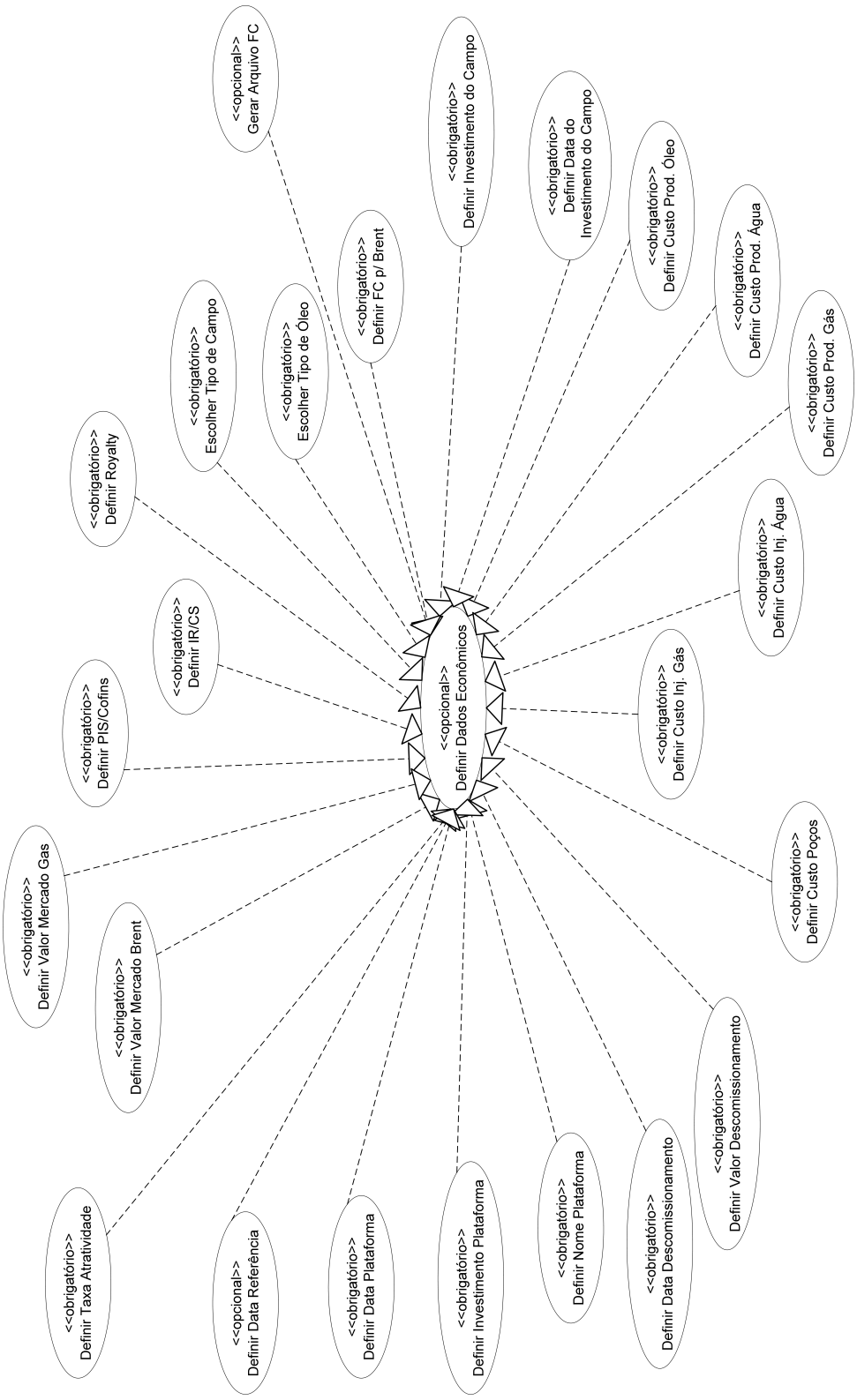


Figura A.3: Diagrama de Caso de Uso: Definição de Dados Econômicos

Apêndice B

Especificação de Interfaces de Componentes

A interface entre os componentes da UNIPAR-LPS é realizada através de arquivos XML. Para a documentação da especificação das interfaces foram gerados documentos no formato PDF e HTML, especificando cada um dos elementos e atributos dos arquivos XML. Cada componente tem sua especificação de arquivo de entrada e saída. A Figura B.1 mostra um exemplo do documento em formato pdf de especificação do componente **Reaproveita Arquivos**. A saída deste componente é o arquivo **reaproveita.xml**. Este é o arquivo especificado na figura.

Schema documentation for reaproveita.xsd

march 9, 2012

Table of Contents

Namespace: ""

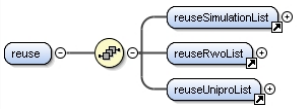
Schema(s)

Main schema `reaproveita.xsd`

Namespace	No namespace
Properties	attribute form default: `unqualified` element form default: `qualified`

Element(s)

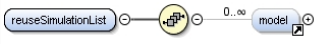
Element `reuse`

Namespace	No namespace
Diagram	
Properties	content: `complex`
Model	reuseSimulationList , reuseRwoList , reuseUniproList
Children	reuseRwoList, reuseSimulationList, reuseUniproList

Instance	`<reuse>`
	`  <reuseSimulationList>{1,1}</reuseSimulationList>`
	`  <reuseRwoList>{1,1}</reuseRwoList>`
	`  <reuseUniproList>{1,1}</reuseUniproList>`
	`</reuse>`

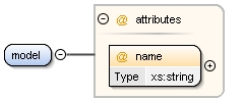
Source	`<xs:element name="reuse">`
	`  <xs:complexType>`
	`  <xs:sequence>`
	`  <xs:element ref="reuseSimulationList"/>`
	`  <xs:element ref="reuseRwoList"/>`
	`  <xs:element ref="reuseUniproList"/>`
	`  </xs:sequence>`
	`  </xs:complexType>`
	`</xs:element>`

Element `reuseSimulationList`

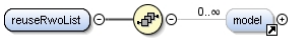
Namespace	No namespace
Diagram	
Properties	content: `complex`

Used by	Element	reuse
Model	model*	
Children	model	
Instance	<reuseSimulationList> <model name="">{0,unbounded}</model> </reuseSimulationList>	
Source	<xs:element name="reuseSimulationList"> <xs:complexType> <xs:sequence> <xs:element minOccurs="0" maxOccurs="unbounded" ref="model"/> </xs:sequence> </xs:complexType> </xs:element>	

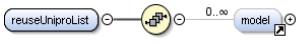
Element `model`

Namespace	No namespace				
Diagram					
Properties	content:	complex			
Used by	Elements	reuseRwoList, reuseSimulationList, reuseUniproList			
Attributes	**QName**	**Type**	**Fixed**	**Default**	**Use**
	name	xs:string			required
Source	<xs:element name="model"> <xs:complexType> <xs:attribute name="name" use="required" type="xs:string"/> </xs:complexType> </xs:element>				

Element `reuseRwoList`

Namespace	No namespace	
Diagram		
Properties	content:	complex
Used by	Element	reuse
Model	model*	
Children	model	
Instance	<reuseRwoList> <model name="">{0,unbounded}</model> </reuseRwoList>	
Source	<xs:element name="reuseRwoList"> <xs:complexType> <xs:sequence> <xs:element minOccurs="0" maxOccurs="unbounded" ref="model"/> </xs:sequence> </xs:complexType> </xs:element>	

Element `reuseUniproList`

Namespace	No namespace	
Diagram		
Properties	content:	complex
Used by	Element	reuse
Model	model*	

Children	model

Instance	`<reuseUniproList>`
	`  <model name="">{0,unbounded}</model>`
	`</reuseUniproList>`

Source	`<xs:element name="reuseUniproList">`
	`  <xs:complexType>`
	`  <xs:sequence>`
	`  <xs:element minOccurs="0" maxOccurs="unbounded" ref="model"/>`
	`  </xs:sequence>`
	`  </xs:complexType>`
	`</xs:element>`

Attribute(s)

Attribute `model / @name`

Namespace	No namespace	
Type	xs:string	
Properties	use:	required
Used by	Element	model
Source	`<xs:attribute name="name" use="required" type="xs:string"/>`	

Figura B.1: Documentação da interface do componente Reaproveita Arquivos

Apêndice C

Arquivo de Especificação de Componentes

O arquivo **Especificação de Componentes** é um arquivo no formato XML, utilizado pela UNIPAR-LPS para a validação das especificações de *workflows*.

O arquivo possui a lista de todos os componentes que fazem parte da UNIPAR-LPS. Para cada componente é especificada uma lista de um ou mais componentes antecessores aceitos, bem como a lista de seus parâmetros curtos e longos.

Cada novo componente adicionado à UNIPAR-LPS deve ser incluído neste arquivo. Parte do arquivo pode ser vista na Figura C.1. Na figura são exibidas as especificações dos componentes **validadados metodologia** e **reaproveitaarquivos**.

```
<unipar-lps>

  <componentsList>

    <component name="validadados">
      <previousList>
        <previous name=""/>
      </previousList>
      <parametersList>
        <parameter short="" long=""/>
      </parametersList>
    </component>

    <component name="metodologia">
      <previousList>
        <previous name="validadados"/>
      </previousList>
      <parametersList>
        <parameter short="s" long="sensibilidade"/>
        <parameter short="d" long="derivacao"/>
        <parameter short="c" long="hcld"/>
        <parameter short="b" long="economico"/>
        <parameter short="a" long="anterior"/>
        <parameter short="l" long="log"/>
        <parameter short="q" long="silencioso"/>
        <parameter short="v" long="version"/>
        <parameter short="h" long="help"/>
      </parametersList>
    </component>

   <component name="reaproveitaarquivos">
      <previousList>
        <previous name="metodologia"/>
      </previousList>
      <parametersList>
        <parameter short="u" long="unipro"/>
        <parameter short="w" long="rwo"/>
        <parameter short="g" long="grafica"/>
        <parameter short="o" long="simulados"/>
        <parameter short="a" long="anterior"/>
        <parameter short="l" long="log"/>
        <parameter short="q" long="silencioso"/>
        <parameter short="v" long="version"/>
        <parameter short="h" long="help"/>
      </parametersList>
    </component>
```

Figura C.1: Arquivo de Especificação de Componentes da UNIPAR-LPS

Referências Bibliográficas

[1] *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1998.

[2] *Software Design Methodology: From principles to architectural styles*. Elsevier, 2005.

[3] myGrid Project. Disponível em: http://www.mygrid.org.uk/. Acesso em: 27 jul. 2012.

[4] Petrel E&P Software Platform. Disponível em: http://www.slb.com/services/software/geo/petrel.aspx/.

[5] Imperial College, London - Department of Earth Science and Engineering, 2012. Disponível em: http://www3.imperial.ac.uk/earthscienceandengineering/research/perm/punq-s3model.

[6] Software Engineering Institute - Framework for Software Product Line Practice. Disponível em: http://www.sei.cmu.edu. Acesso em: 18 ago. 2012.

[7] Portal UNISIM. Disponível em: http://www.unisim.cepetro.unicamp.br/. Acesso em: 02 out. 2012.

[8] The Workflow Management Coalition Specification. Disponível em: http://www.wfmc.org/.

[9] M. Anastasopoulos and D. Muthig. An evaluation of aspect-oriented programming as a product line implementation technology. In *In Proceedings of the International Conference on Software Reuse (ICSR)*, pages 141–156, 2004.

[10] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wüst, and J. Zettel. *Component-based Product Line Engineering with UML*. Addison-Wesley, 2002.

[11] K. Aziz. Notes for petroleum reservoir simulation. In *Department of Petroleum Engineering, School of Earth Sciences, Stanford University*, CA, USA, 1994.

[12] F. Bachmann, L. Bass, C. Buhman, S. Comella-Dorda, F. Long, J. Robert, R. Seacord, and K. Wallnau. Volume II: Technical concepts of component-based software engineering. Technical Report CMU/SEI-2000-TR-008, Software Engineering Institute, 2000.

[13] F. Bachmann, L. Bass, J. Carriere, P. Clements, D. Garlan, J. Ivers, R. Nord, and R. Little. Software architecture documentation in practice: Documenting architectural layers, 2000.

[14] F. Bachmann and P. Clements. Variability in software product lines. In *Technical Report*, CMU/SEI-2005-TR-012 ESC-TR-2005-012, 2005.

[15] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 2003.

[16] R.L. Beltrao, C. Sombra, A. Lage, J. Fagundes Netto, and C. Henriques. Ss: Pre-salt santos basin-challenges and new technologies for the development of the pre-salt cluster, santos basin, brazil. In *Offshore Technology Conference*, 2009.

[17] K. Berg, J. Bishop, and D. Muthig. Tracing software product line variability: from problem to solution space. In *Proceedings of the Annual research Conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, pages 182–191, Republic of South Africa, 2005. South African Institute for Computer Scientists and Information Technologists.

[18] P. H. S. Brito, P. A. C. Guerra, and C. M.F. Rubira. Estudo sobre estilos arquiteturais para sistemas de software baseados em componentes. In *Technical Report - IC-07-10*, March 2007.

[19] J. Cheesman and J. Daniels, editors. *UML Components*. Addison-Wesley, 2001.

[20] E. J. Chikofsky and J. H. Cross II. Reverse engineering and design recovery: a taxonomy. *Software, IEEE*, 7(1):13 –17, January 1990.

[21] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[22] S. Cohen. Predicting when product line investment pays. In *Technical Note*, CMU/SEI-2003-TN-017, July 2003.

[23] G. C. B. Costa, R. Braga, and J. M. N. David. Apoiando uma linha de produtos de workflow científico através da associação de ontologias e modelos de features. 2012.

[24] M.V. Couto, M.T. Valente, and E. Figueiredo. Extracting software product lines: A case study using conditional compilation. In *European Conference on Software Maintenance and Reengineering*, pages 191 –200, 2011.

[25] F. de Oliveira, L. Murta, C. Werner, and M. Mattoso. Using provenance to improve workflow design. In Juliana Freire, David Koop, and Luc Moreau, editors, *Provenance and Annotation of Data and Processes*, volume 5272 of *Lecture Notes in Computer Science*, pages 136–143. Springer Berlin / Heidelberg, 2008.

[26] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[27] I.D.B.S. Designs and I.C.W. Pidgeon. Software change through design maintenance. In *1997 IEEE International Conference on Software Maintenance*, page 250. IEEE, 1997.

[28] R. C. Durscki, Spinola M. M., R. C. Burnett, and S. S. Reinehr. Linhas de produto de software: riscos e vantagens de sua implantação. In *VI Simpósio Internacional de Melhoria de Processos de Software, SEMPROS*, São Paulo, Brasil, 2004.

[29] J. Formigli, A. Capeleiro Pinto, and A. Almeida. Ss: Santos basin's pre-salt reservoirs development: The way ahead. In *Offshore Technology Conference*, 2009.

[30] D. Garlan, R. Monroe, and D. Wile. Acme: Architectural description of component-based systems. In Gary T. Leavens and Murali Sitaraman, editors, *Foundations of Component-Based Systems*, chapter 3, pages 47 – 68. Cambridge University Press, 2000.

[31] L. A. Gayard, C. M. F. Rubira, and P. A. C. Guerra. COSMOS*: a COmponent System MOdel for Software Architectures. Technical Report IC-08-04, Instituto de Computação, UNICAMP, 2008.

[32] H. Gomaa. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures.* Addison Wesley, 2004.

[33] M.L. Griss, J. Favaro, and M. d'Alessandro. Integrating feature modeling with the rseb. In *Software Reuse, 1998. Proceedings. Fifth International Conference on*, pages 76 –85, June 1998.

[34] K. Kang, M. Kim, J. Lee, and B. Kim. Feature-oriented re-engineering of legacy systems into product line assets–a case study. *Software Product Lines*, pages 45–56, 2005.

[35] K. C. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-oriented domain analysis. Technical report, CMU/SEI, 1990.

[36] C. W. Krueger. Easing the transition to software mass customization. In *International Workshop on Software Product-Family Engineering*, pages 282–293. Springer, 2002.

[37] I. H. Kruger and R. Mathew. Systematic development and exploration of service-oriented software architectures. In *Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on*, pages 177 – 187, June 2004.

[38] M. M. Lehman, J. F. Ramil, and D. E. Perry. On evidence supporting the feast hypothesis and the laws of software evolution. In *Proceedings of the 5th International Symposium on Software Metrics*, METRICS'98, Washington, DC, USA, 1998. IEEE Computer Society.

[39] M. Mattoso, C. Werner, G. Travassos, V. Braganholo, and L. Murta. Gerenciando experimentos científicos em larga escala. In *Anais do XXVIII Congresso da SBC*, page 121, 2008.

[40] C.B. Medeiros, G. Vossen, and M. Weske. WASA: A workflow-based architecture to support scientific database applications. In *Proceedings of the 6th DEXA Conference (Editors: N. Revell, A. M. Tjoa)*, Springer LNCS 978, pages 574–583, London, UK, 1995.

[41] S. F. Mello. Estudo sobre simulação composicional de reservatórios de petróleo com injeção de co2. Master's thesis, Departamento de Engenharia de Petróleo - UNICAMP, 2011.

[42] E. Ogasawara, L. Murta, C. Werner, and M. Mattoso. Linhas de experimento: Reutilização e gerência de configuração em workflows científicos. In *2 Workshop E-Science*, pages 31–40, 2008.

[43] M.R. Olsem. An incremental approach to software systems reengineering. *Journal of Software Maintenance*, 10(3):181–202, 1998.

[44] D. C. Pallazi. Qdaontology – abordagem para o desenvolvimento de ontologias em e-science: um estudo de caso em biologia. Master's thesis, Universidade Federal de Juiz de Fora, 2010.

[45] Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT SOFTWARE ENGINEERING NOTES*, 17:40–52, 1992.

[46] S. H. Pinho. Rigel: Um repositório com suporte para desenvolvimento basaeado em componentes. Master's thesis, Instituto de Computação - UNICAMP, 2011.

[47] R.S. Pressman and D. Ince. *Software engineering: a practitioner's approach*, volume 5. McGraw-hill New York, 1992.

[48] M. A. Ramos, P. C. Masiero, R. T. V. Braga, and R. A. D. Penteado. Reengineering legacy systems towards system of systems development. In *Information Reuse and Integration (IRI), 2012 IEEE 13th International Conference on*, pages 624 –630, August 2012.

[49] M. A. Ramos and R. A. D. Penteado. Embedded software revitalization through component mining and software product line techniques. *Journal of Universal Computer Science (J.UCS)*, 14(8), April 2008.

[50] C. Riva and J.V. Rodriguez. Combining static and dynamic views for architecture reconstruction. In *Software Maintenance and Reengineering, 2002. Proceedings. Sixth European Conference on*, pages 47 –55, 2002.

[51] L. H. Rosenberg and L. E. Hyatt. Software re-engineering. *Tekninen raportti, Software Assurance Technology Center, lokakuu*, 1996.

[52] J. P. M. Santos and D. J. Schiozer. Determinação de metodologia de ajuste automatizado de histórico. In *Rio Oil & Gas*, Rio de Janeiro, Brasil, 2000.

[53] P. Sochos, M. Riebisch, and I. Philippow. The feature-architecture mapping (FArM) method for feature-oriented development of software product lines. In *Proceedings of the Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems*, pages 308–318. IEEE Computer Society, 2006.

[54] M. Svahnberg, J. Van Gurp, and J. Bosch. A taxonomy of variability realization techniques. *Software - Pratice and Experience*, 35:705–754, 2001.

[55] L. P. Tizzei. *Evolução de Arquiteturas de Linhas de Produtos baseadas em Componentes e Aspectos*. PhD thesis, Institute of Computing, University of Campinas, Campinas, Brazil, July 2012.

[56] W. Van Der Aalst, K. Van Hee, Prof. Dr. Kees, M. Hee, R. De Vries, J. Rigter, E. Verbeek, and M. Voorhoeve. Workflow management: Models, methods, and systems, 2002.

[57] A. Van Deursen, C. Hofmeister, R. Koschke, L. Moonen, and C. Riva. Symphony: view-driven software architecture reconstruction. In *Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on*, pages 122 – 132, June 2004.

[58] J. Van Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture*, page 45. IEEE Computer Society, 2001.

[59] G. Vossen and M. Weske. The WASA approach to workflow management for scientific applications. In *A. Dogac, L. Kalinichenko, M.T. Özsu, A. Sheth (Eds.): Workflow Management Systems and Interoperability*, ASI NATO Series, Series F: Computer and Systems Sciences, Vol. 164, pages 145–164, 1998.

[60] C. Werner, M. Mattoso, R. Braga, M. Barros, L. Murta, and A. Dantas. Odyssey: Infra-estrutura de reutilização baseada em modelos de domínio. *XIII Simpósio Brasileiro de Engenharia de Software, Caderno de Ferramentas, Florianópolis, 1999*, 10(3):181–202, 1998.

[61] J. Zhang, X. Cai, and G. Liu. Mapping features to architectural components in aspect-oriented software product lines. In *International Conference on Computer Science and Software Engineering*, pages 94 – 97, 2008.