RAFAEL HORNUNG

UM MÉTODO PARA IDENTIFICAÇÃO ANTECIPADA DE CANDIDATOS A ASPECTO NO DESENVOLVIMENTO DE FRAMEWORKS DE DOMÍNIO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Andrey Ricardo Pimentel

CURITIBA

2010

RAFAEL HORNUNG

UM MÉTODO PARA IDENTIFICAÇÃO ANTECIPADA DE CANDIDATOS A ASPECTO NO DESENVOLVIMENTO DE FRAMEWORKS DE DOMÍNIO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Andrey Ricardo Pimentel

CURITIBA

2010

RAFAEL HORNUNG

UM MÉTODO PARA IDENTIFICAÇÃO ANTECIPADA DE CANDIDATOS A ASPECTO NO DESENVOLVIMENTO DE FRAMEWORKS DE DOMÍNIO

Dissertação aprovada como requisito parcial à obtenção do grau de Mestre no Programa de Pós-Graduação em Informática da Universidade Federal do Paraná, pela Comisso formada pelos professores:

Orientador: Prof. Dr. Andrey Ricardo Pimentel
Departamento de Informática, UFPR

Prof. Dra. Silvia Regina Vergilio
Departamento de Informática, UFPR

Prof. Dra. Simone Nasser Matos
Departamento de Informática, UTFPR

Curitiba, 23 de agosto de 2010

Para meu Pai e minha Mãe, minha Avó, meu Irmão e minha
Esposa, pelo apoio, carinho e incentivo.
Com Amor, dedico.

AGRADECIMENTOS

A Deus, pela vida, e por esta oportunidade.

Aos familiares, que sempre me apóiam e incentivam, em especial a minha mãe Gladis e a meu pai Luiz Fernando.

A minha esposa Polyanna, pela paciência, carinho e cuidados.

Ao Professor Orientador Dr. Andrey Ricardo Pimentel, pela orientação, ensinamentos e dedicação.

A Professora Dra. Simone Nasser Matos, pela contribuição no desenvolvimento deste trabalho.

A Universidade Federal do Paraná pela disponibilização de sua infraestrutura e espaço físico para a realização desta pesquisa e minha formação.

A empresa em que trabalho que sempre me liberou nos dias e horas necessárias para a realização desta pós-graduação.

A todos que contribuíram direta ou indiretamente para a realização deste trabalho.

RESUMO

Os frameworks representam atualmente uma tecnologia relevante para o desenvolvimento de aplicações. Sendo desenvolvidos para aumentar a flexibilidade e permitir o reúso tanto do código quanto do projeto, sua adoção traz, em geral, ganhos de produtividade e qualidade no desenvolvimento. Apesar destes benefícios, a implementação de frameworks possui certas limitações para abstrair interesses que afetam a aplicação como um todo. Conhecidos como interesses transversais, estes geralmente encontram-se espalhados e/ou entrelaçados por diversos módulos. Buscando formas de melhorar a modularização destes interesses, tem-se explorado o uso das técnicas propostas pelo Desenvolvimento de Software Orientado a Aspectos (DSOA). Neste contexto, esta dissertação apresenta um método que aplica os conceitos de *early-aspects*, uma das linhas de pesquisa do DSOA, nas fases iniciais do processo de desenvolvimento de frameworks de domínio. O método proposto possui dois subprocessos principais que estão inseridos na fase de análise de requisitos. O primeiro, refere-se ao procedimento para identificação e representação das responsabilidades espalhadas pelos requisitos. Enquanto que, o segundo objetiva a identificação e representação dos requisitos que apresentam características que indicam que sua implementação com aspectos é factível. O método proposto foi implementado facilitando assim, a identificação dos candidatos a aspecto por um processo apoiado por computador. A aplicação do método proposto foi ilustrada na elaboração do modelo de requisitos para os seguintes domínios: "Jogo de Corrida de Carros" e "Controle de Finanças Pessoais".

Palavras-chave: Framework de Domínio, Desenvolvimento de Software Orientado a Aspectos, Método Dirigido a Responsabilidades, *Early-Aspects*.

ABSTRACT

The frameworks represent a relevant technology to the development of applications. Being developed to improve flexibility and allow the reuse of both code and design, its adoption brings, in general, gains in productivity and quality development. Despite these benefits, its implementation has certain limitations to abstract concerns that affect the application as a whole. Known as crosscutting concerns, they usually are spread and/or entangled by various modules. Seeking ways to improve the modularization of these concerns, we have explored the use of the techniques proposed for Aspect Oriented Software Development (AOSD). In this context, this paper presents a method that applies concepts of early-aspects, a line of research of AOSD, in the early stages of the development process of domain frameworks. The proposed method has two main sub-processes that are included in the analysis phase of requirements. The first one concerns to the procedure for identification and representation of responsibilities scattered by the requirements. Meanwhile, the second sub-process is the identification and representation of requirements that have characteristics that indicate that their implementation is feasible with aspects. The proposed method has been implemented thus facilitating the identification of candidates to look for a process supported by computer. The implementation of the proposed method was illustrated in developing the model of requirements for the following fields: "Car Racing Game" and "Control of Personal Finance".

Keywords: Domain Framework, Aspect Oriented Software Development, Managed Methods to Responsibilities, Early-Aspects.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AORE	*Aspect Oriented Requirements Engineering*
CORBA	*Common Object Request Broker Architecture*
DCOM	*Distributed Component Object Model*
DCU	Diagrama de Casos de Uso
DG	Desenvolvimento Generativo
DSOA	Desenvolvimento de Software Orientado a Aspectos
DSOO	Desenvolvimento de Software Orientado a Objetos
DT	Descrição Textual
EJPs	*Extension Join Points*
FAOA	Framework de Aplicação Orientado a Aspectos
FOA	Framework Orientado a Aspectos
FOO	Framework Orientado a Objetos
FT	Framework Transversal
MDR-PEF	Método Dirigido a Responsabilidades - Pontos de Estabilidade e de Flexibilidade
MRF	Modelo de Requisitos do Framework
ODMG	*Object Database Management Group*
OMG	*Object Management Group*
OO	Orientação a Objetos
ORB	*Object Request Broker*
PDR-DFD	Processo Dirigido a Responsabilidades aplicado ao Desenvolvimento de Framework de Domínio
ProFT/PU	Processo de Desenvolvimento de Software Orientado a Aspectos apoiado por Frameworks Transversais
POA	Programação Orientada a Aspectos
UML	*Unified Modeling Language*
UML-F	*Unified Modeling Language for Object-Oriented Frameworks*

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

Atualmente, os Frameworks Orientados a Objetos (FOO), ou simplesmente frameworks, representam uma tecnologia relevante para o desenvolvimento de aplicações. Eles são construídos de forma a permitir o reúso tanto do código quanto de projeto. No desenvolvimento baseado em frameworks, aplicações similares são desenvolvidas através do reúso da arquitetura definida pelo framework e da extensão de seus pontos de flexibilidade. Consequentemente, sua adoção traz, em geral, ganhos de produtividade e qualidade no desenvolvimento de novas aplicações [38].

Apesar dos benefícios da sua adoção no desenvolvimento de frameworks, a Orientação a Objetos (OO) possui certas limitações para abstrair interesses que afetam a aplicação como um todo. Conhecidos como interesses transversais, geralmente são encontrados espalhados e/ou entrelaçados por diversos módulos da aplicação, dificultando assim, o reúso e a manutenção. Exemplos de interesses transversais são: persistência, controle de acesso, *logging*, entre outros [35, 46].

Buscando formas de melhorar a modularização dos interesses transversais encontrados em frameworks, tem-se explorado o uso das técnicas propostas pelo Desenvolvimento de Software Orientado a Aspectos (DSOA) [46, 38]. O DSOA pode ser visto como uma técnica complementar à OO, tendo como objetivo o desenvolvimento de métodos, técnicas e mecanismos que auxiliem na identificação, modularização e composição dos interesses transversais [58].

Os frameworks que utilizam as técnicas propostas pelo DSOA, ficaram conhecidos como Frameworks Orientados a Aspectos (FOA). Formado por um conjunto de unidades básicas da OO e da orientação a aspectos, assim como um FOO, um FOA pode ser definido como um sistema semi-completo e reutilizável que pode ser instanciado por um desenvolvedor de aplicações, mantendo os interesses transversais separados dos interesses

base.

No entanto, identificou-se que os trabalhos que apresentam FOA, descritos no Capítulo 2, tratam, em sua maioria, de processos evolutivos, onde os aspectos são incorporados a FOO já desenvolvidos, ou de processos que visam identificar os aspectos nas etapas de projeto e implementação. Este fato torna interessante a criação de um método, além de uma ferramenta associada a este, que permita a identificação antecipada de candidatos a aspectos no desenvolvimento de um framework de domínio.

1.1 Motivação e Justificava do Trabalho

O paradigma da orientação a aspectos surgiu inicialmente com o objetivo de solucionar problemas relativos à codificação de sistemas, motivo pelo qual, uma grande parte das pesquisas realizadas nesta área estavam focadas em atividades orientadas à implementação [58]. Entre os possíveis benefícios da utilização da orientação a aspectos no desenvolvimento de aplicações, destacam-se [35]:

- diminuição no espalhamento e entrelaçamento dos interesses transversais;
- menor complexidade do código-fonte, facilitando assim as atividades de manutenção, evolução e reutilização.

Com o amadurecimento da orientação a aspectos, a comunidade de Engenharia de Software começou a aplicar os seus fundamentos e práticas nas demais etapas do ciclo de desenvolvimento de aplicações, surgindo assim o DSOA [58].

Entre as linhas de pesquisa relacionadas ao DSOA, tem-se que a identificação e modularização dos aspectos seja realizada nas etapas iniciais do processo de desenvolvimento, ou seja, antes da codificação, nomeada de *early-aspects*. Algumas das razões que impulsionaram o interesse por essa linha de pesquisa foram [58, 61]:

- antecipar o raciocínio sobre quais são as unidades afetadas por um aspecto e como a sua composição deverá ser realizada;

- permitir que os benefícios da utilização da orientação a aspectos sejam obtidos ao longo de todo o processo de desenvolvimento e não apenas nos artefatos de implementação;

- possibilitar que o entendimento de um sistema seja realizado por meio dos modelos de análise e projeto e não unicamente da codificação;

- facilitar a identificação das unidades afetadas por eventuais mudanças realizadas nos aspectos após o inicio da sua codificação.

Jacobson e NG relatam em seu livro "*Aspect-Oriented Software Development with Use Cases*"[31], que antecipar a identificação e separação dos interesses de uma aplicação, assim como, manter estes interesses separados durante as etapas do desenvolvimento, pode resultar em um ganho de produtividade na ordem de 40%.

Nos últimos anos, muitos trabalhos vêm aplicando os conceitos do DSOA ao desenvolvimento de frameworks [16]. No entanto, estes enfocam processos onde os aspectos são incorporados a frameworks já desenvolvidos, ou são identificados nas etapas de projeto e implementação. Entre os trabalhos encontrados, apenas Nakajima [45], Camargo [16] e Kulesza [37] cogitam a possibilidade de identificar os aspectos durante a análise de requisitos, sem apresentar uma técnica para realizar tal atividade.

Neste contexto, este trabalho busca contribuir para o processo de desenvolvimento de frameworks de domínio[1] com a aplicação de conceitos e técnicas do DSOA, em fase inicial do seu processo de desenvolvimento. Formado por dois subprocessos, este trabalho propõe um método que se aplica durante a fase de análise de requisitos. No primeiro subprocesso, são identificadas as responsabilidades espalhadas pelos requisitos. Por sua vez, no segundo subprocesso identificam-se os requisitos que apresentam características que indicam que sua implementação com aspectos é factível.

[1] O termo "frameworks de domínio" é utilizado como referência aos frameworks desenvolvidos com o objetivo de cobrir as funcionalidades aplicáveis a um domínio específico, como por exemplo: telecomunicações, controle de produção, multimídia, entre outros [40].

1.2 Objetivos

Esta dissertação aborda a integração entre a linha de pesquisa *early-aspects*, proposta pelo DSOA, e o desenvolvimento de frameworks de domínio. O objetivo geral desta dissertação é:

- Propor um método que auxilie na identificação antecipada de candidatos a aspecto em fase inicial do processo de desenvolvimento de frameworks de domínio. Com isto busca-se, trazer as vantagens relacionadas ao uso dos Aspectos Antecipados (*early-aspects*), citadas na Seção 1.1, para o desenvolvimento de frameworks de domínio.

Os objetivos específicos deste trabalho são:

1. Avaliar os trabalhos relacionados ao desenvolvimento de frameworks de domínio e a linha de pesquisa *early-aspects*, buscando compreender seus princípios, técnicas e ferramentas de apoio;

2. Compreender a correspondência entre os conceitos de *early-aspects* e as atividades iniciais do processo de desenvolvimento de frameworks de domínio;

3. Aplicar o método proposto em dois estudos de caso;

4. Desenvolver uma ferramenta que forneça suporte e agilidade durante a aplicação do método proposto.

1.3 Organização do Documento

Esta dissertação apresenta o trabalho de mestrado realizado no programa de Pós-Graduação em Informática da Universidade Federal do Paraná (UFPR). Está organizada em 5 capítulos, além da introdução.

O Capítulo 2 apresenta os trabalhos relacionados aos temas abordados por esta dissertação: Framework Orientado a Objetos, o Desenvolvimento de Software Orientado a Aspectos e Framework Orientado a Aspectos.

Uma visão geral do método para identificação antecipada de candidatos a aspecto em fase inicial do processo de desenvolvimento de frameworks de domínio, proposto neste trabalho, é apresentada no Capítulo 3 através de sua aplicação no domínio "Jogo de Corrida de Carros".

O Capítulo 4 apresenta a ferramenta de apoio ao método proposto neste trabalho.

O Capítulo 5 relata o estudo de caso para a análise e avaliação do método proposto. Para isso, foi escolhido o domínio "Controle de Finanças Pessoais".

Por fim, o último capítulo descreve as conclusões da dissertação e perspectivas de trabalhos futuros.

CAPÍTULO 2

TRABALHOS RELACIONADOS

Este capítulo apresenta uma visão geral das três principais áreas relacionadas com a pesquisa conduzida por esse trabalho. A Seção 2.1 apresenta alguns conceitos relacionados a Frameworks Orientados a Objetos (FOO)[1], além de colocar e comparar algumas abordagens para o seu desenvolvimento. Em seguida, a abordagem de Desenvolvimento de Software Orientado a Aspectos (DSOA) é descrita na Seção 2.2, assim como alguns trabalhos que propõem a identificação antecipada de aspectos. Por fim, a Seção 2.3 apresenta alguns conceitos relacionados a Frameworks Orientados a Aspectos (FOA), além de trazer algumas pesquisas que propõem a identificação dos aspectos nas fases iniciais do seu processo de desenvolvimento.

2.1 Frameworks Orientados a Objetos

2.1.1 Conceitos Gerais

Umas das definições mais clássicas para frameworks foi proposta por Johnson e Foote [34], conceituando-o como um "conjunto de classes que definem um projeto abstrato de soluções para uma família de problemas relacionados". Posteriormente, Johnson [32] definiu um framework como um "projeto reutilizável de uma parte ou o todo de um sistema, que é representado por um conjunto de classes abstratas e pelo modo que elas interagem" [40].

Fayad [23, 22, 21] fez um dos trabalhos mais completos relacionados a frameworks, formado por três livros que relatam a obtenção, desenvolvimento, documentação, evolução e experiências na área de frameworks. Para Fayad um framework é uma "tecnologia promissora para materializar projetos e implementações de softwares comprovados, levando à redução de custo e ao aumento a qualidade do software" [46].

[1]Para simplificação, o termo "framework" será utilizado neste trabalho com o mesmo significado de FOO.

Para Wirfs-Brock e McKean [67], um framework é um "projeto geral para resolver um problema de software, provendo uma biblioteca de classes que desenvolvedores podem adaptar ou estender para se adequar a uma situação particular" [40].

Por sua vez, Matos [40] faz uma adaptação às definições propostas por Johnson [32] e Wirfs-Brock e McKean [67], definindo um framework como sendo um "conjunto de subsistemas[2], subframeworks[3] e componentes[4] interagindo e colaborando de forma a produzir um projeto geral para um domínio particular".

Além das definições apresentadas anteriormente, outras podem ser encontradas na literatura [68, 65, 26, 51, 9, 24]. Contudo, observa-se que alguns termos como classes e projeto reutilizáveis, entre outros, estão sempre presentes [40].

Entre os benefícios da utilização de frameworks para o desenvolvimento de aplicações, destacam-se [46]:

- **Modularidade** - a aplicação resultante possui uma alta modularidade devido ao encapsulamento dos detalhes de implementação, o que possibilita uma considerável redução do esforço necessário para o entendimento e a manutenção da aplicação;

- **Reusabilidade** - reduz o custo no desenvolvimento de novas aplicações devido a utilização de componentes já criados e testados, evitando assim, a criação de uma nova solução para um problema recorrente, além de aumentar a qualidade e confiabilidade da aplicação;

- **Extensibilidade** - a definição dos pontos de extensão permitem o desacoplamento sistemático da parte fixa do framework, presente no domínio da aplicação, da parte variável introduzida pelo processo de instanciação;

- **Inversão de controle** - a responsabilidade pelas chamadas e pelo fluxo de controle passa a ser do framework, ficando a cargo do desenvolvedor a tarefa de concretizar os métodos abstratos que são chamados pelo framework.

[2]Conjunto de classes que interagem e colaboram para a execução de uma determinada função.

[3]Parte reutilizável de um subsistema, ou conjunto deles, que permitem a sua reutilização em uma nova aplicação-exemplo.

[4]Resultado da transformação de um subframework ou de um subsistema em um projeto executável que permita ao desenvolvedor alterar apenas a parte flexível.

Apesar das vantagens demonstradas anteriormente, a utilização de frameworks também possui alguns inconvenientes:

- **Esforço de desenvolvimento** - maior complexidade no desenvolvimento devido a busca pela qualidade, reusabilidade, flexibilidade e extensibilidade. Completo domínio das técnicas de desenvolvimento orientado a objetos e do reúso de software. Conhecimento avançado no domínio ao qual se pretende desenvolver. Dificuldade em distinguir se erros que surgem são da aplicação instanciada ou do próprio framework [46, 40];

- **Curva de aprendizado** - tempo necessário para se aprender o funcionamento e o processo de reúso do framework, a fim de se obter as vantagens propostas pela sua utilização [46];

- **Integrabilidade** - como a maioria dos frameworks são desenvolvidos com o exclusivo objetivo da extensão, podem-se encontrar dificuldade ao se tentar integrar um framework com outro artefato de software, como por exemplo um outro framework [46];

- **Manutenibilidade** - assim como as aplicações convencionais, os requisitos iniciais dos frameworks podem evoluir com o tempo, tornando necessário o desenvolvimento de novas versões. No entanto, como os frameworks não são artefatos isolados, as demais aplicações instanciadas a partir dele também precisaram evoluir. Essa atividade pode ser problemática se a aplicação já estiver em produção [46];

- **Eficiência** - a utilização de frameworks usualmente causa queda na eficiência do código-fonte da aplicação, devido às chamadas adicionais a métodos para a execução de uma determinada tarefa [46].

Os frameworks podem ser classificados com relação ao seu escopo ou sua forma de reúso [7, 46]. A classificação quanto ao seu escopo está relacionada à abrangência do framework em um determinado domínio. Por sua vez, a classificação quanto a forma de

reúso está relacionada à técnica utilizada para a instanciação de novas aplicações a partir do framework.

Em relação ao escopo, um framework pode ser classificado em três grupos [40]:

- **Framework de aplicação ou de infra-estrutura** - cobre funcionalidades aplicáveis a vários domínios, como por exemplo: sistemas operacionais, sistemas portáteis, comunicação, redes, interfaces com o usuário, ferramentas de processamento de linguagem, entre outros;

- **Framework de suporte ou integração de *middleware*** - oferecem serviços de baixo nível como interfaces para periféricos e acesso a arquivos. São comumente utilizados para integrar sistemas distribuídos permitindo a troca de dados entre sistemas heterogêneos, como por exemplo: sistemas compatíveis com o modelo ORB (*Object Request Broker*), CORBA (*Common Object Request Broker Architecture*), DCOM (*Distributed Component Object Model*), ODMG (*Object Database Management Group*), entre outros, podendo incluir em sua estrutura frameworks de aplicação;

- **Framework de domínio** - cobre funcionalidades aplicáveis a um domínio específico, como por exemplo: telecomunicações, aviação, manufatura, controle de produção, multimídia, engenharia financeira, entre outros. Em geral são sistemas complexos e podem utilizar em sua estrutura outros frameworks de aplicação e suporte.

Já quanto a sua forma de reúso, um framework pode ser classificado como [40]:

- **Framework caixa branca ou dirigidos à arquitetura** - está fortemente ligado às características da orientação a objetos, sendo o reúso e a extensão obtidos através da herança e da redefinição de métodos, no entanto, demanda que o desenvolvedor conheça em detalhes a estrutura interna do framework, dificultando assim a sua utilização;

- **Framework caixa preta ou dirigidos a dados** - possibilita a extensão através da combinação de diferentes tipos de objetos, sendo o reúso obtido por meio da definição de interfaces para componentes. Possui uma menor complexidade para utilização, quando comparado ao caixa branca, no entanto, permite uma menor flexibilidade;

- **Framework caixa cinza** - híbrido das formas de reúso caixa branca e caixa preta, busca disponibilizar uma maior flexibilidade e extensibilidade, sendo o reúso das funcionalidades realizado através da herança, ligação dinâmica e interface.

De acordo com Silva [60], o ciclo de vida de um framework se diferencia de uma aplicação convencional pelo fato de não ser um artefato isolado e de sua existência estar sempre ligada a outros artefatos. Tais artefatos podem ter sido gerados a partir dele ou ter exercido alguma influência em sua formação. A Figura 2.1 apresenta o fluxo das informações dos diversos artefatos que influenciam na definição da estrutura de classes do framework, assim como das aplicações geradas a partir dele.

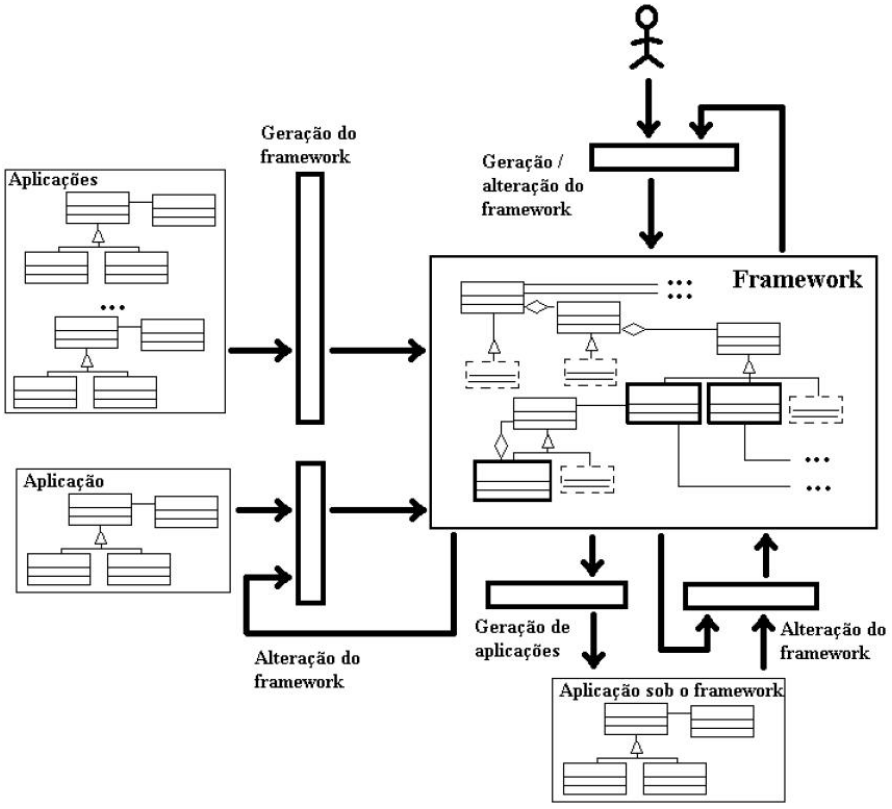

Figura 2.1: Ciclo de vida dos frameworks (figura extraída de [60]).

Como pode ser visto na Figura 2.1, inicialmente a estrutura de classes de um framework é obtida com base em um conjunto de aplicações do domínio e do conhecimento do desenvolvedor. Posteriormente, esta estrutura pode vir a ser modificada devido a obtenção de novas informações, originadas em novas aplicações ou em aplicações geradas a partir do framework.

2.1.2 Abordagens para o Desenvolvimento de Frameworks de Domínio

Segundo Matos [40], o processo de desenvolvimento de um framework de domínio é formado por uma evolução iterativa de sua estrutura de classes, que envolve atividades como identificação de classes, modelagem de cenários e identificação de estados de objetos, entre outras.

Em seu trabalho, Fayad *et al.* [22] relata que uma atividade importante durante o desenvolvimento de um framework é a identificação das partes fixas (do inglês *frozen spots*) e variáveis (do inglês *hot spots*) do domínio ao qual se pretende desenvolver:

- **partes fixas** - definem a arquitetura geral de um sistema, são projetadas para serem utilizadas sem modificação;
- **partes variáveis** - representam as características específicas de cada sistema, são criadas para serem genéricas e adaptáveis.

Na literatura especializada encontram-se diversas abordagens voltadas ao desenvolvimento de frameworks de domínio [42, 60]. A seguir, algumas destas abordagens [33, 30, 51, 60, 41] são brevemente descritas e comparadas.

2.1.2.1 Projeto Dirigido por Exemplo

Segundo Johnson [33], o desenvolvimento de um framework decorre de um processo de aprendizado a respeito do domínio através do desenvolvimento de novas aplicações ou do estudo de aplicações já existentes. A Figura 2.2 apresenta o processo apontado como ideal pelo autor, sendo este, descrito a seguir:

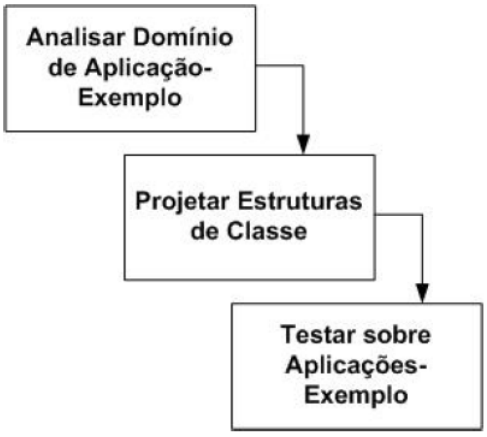

Figura 2.2: Processo de desenvolvimento proposto por Johnson (figura extraída de [42]).

- **Analisar Domínio de Aplicação-Exemplo** - composto pelo aprendizado das abstrações conhecidas, coleta e avaliação do maior número possível de aplicações-exemplo do domínio, em um mínimo de quatro;

- **Projetar Estrutura de Classe** - formada pela definição de uma hierarquia de classes que possa ser especializada a ponto de abranger os exemplos analisados na etapa anterior, nessa etapa é recomendado a utilização de padrões de projeto [26];

- **Testar sobre Aplicações-Exemplo** - responsável pela avaliação do framework desenvolvido através da implementação das aplicações exemplo analisadas na primeira etapa.

No entanto, Johnson [33] relata que este processo dificilmente é seguido devido a limitações financeiras e de tempo. Devido a isto, neste mesmo trabalho também é proposto um processo tido como adequado para o desenvolvimento de frameworks. Este processo é composto pelo desenvolvimento do framework em paralelo com duas outras aplicações similares que se necessite desenvolver, procurando maximizar a troca de informações entre os três desenvolvimentos. Ainda segundo Johnson [33], para que este processo seja realizado com sucesso, a equipe de desenvolvimento deve conter pessoas com experiência no domínio da aplicação que se está desenvolvendo, como uma forma de capturar informações de outras aplicações.

2.1.2.2 Metodologia de Projeto da Empresa Taligent

A abordagem da já extinta empresa Taligent [30], propõe que ao invés de criar um único framework que cubra as características de um domínio, sejam criados diversos frameworks menores e mais simples que, ao serem utilizados em conjunto, darão origem a aplicação desejada. Isto se justifica pelo fato de que "pequenos frameworks são mais flexíveis e podem ser reutilizados mais frequentemente", com isto o foco passa a ser desenvolvimento de pequenos frameworks que resolvam aspectos específicos do domínio.

Outra característica interessante proposta por esta metodologia é a idéia de tornar o uso do framework o mais simples possível. Esta característica pode ser alcançada atráves de [60]:

- criação de classes concretas que possam ser usadas diretamente;
- diminuir o número de classes que precisariam ser criadas;
- diminuir a quantidade de métodos que precisariam ser sobrepostos.

Na Figura 2.3 são apresentadas as fases propostas por esta abordagem para o desenvolvimento de frameworks, sendo estas, descritas a seguir:

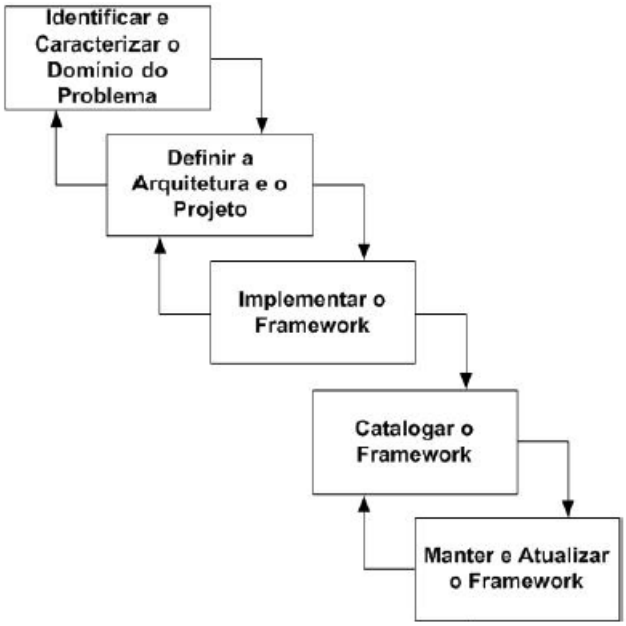

Figura 2.3: Processo de desenvolvimento proposto por Taligent (figura extraída de [42]).

- **Identificar e Caracterizar o Domínio do Problema** - etapa composta por cinco fases. Na primeira, realiza-se uma análise no domínio a fim de identificar quais são os frameworks necessários à aplicação que se pretende desenvolver, verificando para isto a possível utilização de frameworks já existentes. Na segunda, examina-se as soluções existentes para o domínio. Na terceira, identifica-se as principais abstrações do domínio. Na quarta, identifica-se o limite de responsabilidade do(s) framework(s) que se pretende desenvolver. Na quinta, valida-se as informações identificadas com o auxílio de um especialista no domínio;

- **Definir a Arquitetura e o Projeto** - nesta etapa a estrutura de classes definida no passo anterior deve ser refinada, para isso deve-se ter a preocupação em como o usuário irá interagir com o framework. É recomendado a utilização de padrões de projeto [26] a fim de aperfeiçoar o projeto, além de validar estas informações com um especialista do domínio;

- **Implementar o Framework** - etapa composta pela implementação das classes principais, testes realizados pelos desenvolvedores e por terceiros, num processo iterativo a fim de refinar o projeto;

- **Catalogar o Framework** - nesta etapa é proposta a criação da documentação na forma de diagramas e receitas contendo exemplos, incluindo o código fonte, de como se deve proceder para utilizar o framework;

- **Manter e Atualizar o Framework** - etapa responsável por manter e atualizar o framework seguindo as seguintes regras: correção imediata dos erros, adicionar ocasionalmente novas características, evitar alterar ou remover interfaces ou métodos já existentes dando sempre preferência a criação de novos.

2.1.2.3 Projeto Dirigido por Pontos de Flexibilidade

Em sua abordagem, Pree [51] propõe que a identificação dos pontos de flexibilidade seja realizada na estrutura de classes do domínio ao qual se pretende desenvolver. A Figura

2.4 apresenta as fases propostas pela abordagem de Pree, sendo estas, descritas a seguir:

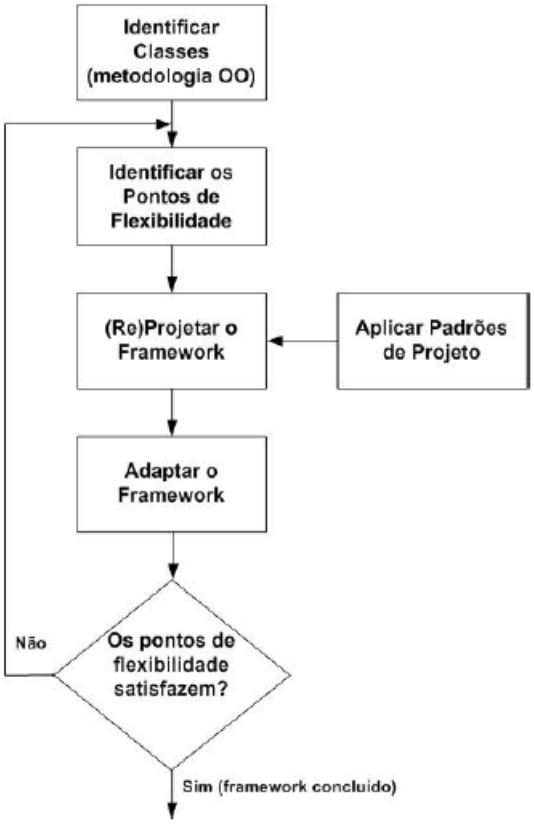

Figura 2.4: Processo de desenvolvimento proposto por Pree (figura extraída de [42]).

- **Identificar Classes (metodologia OO)** - com o auxílio de especialista do domínio, o desenvolvedor do framework define uma estrutura de classes;
- **Identificar os Pontos de Flexibilidade** - também com o auxílio de especialistas do domínio, nesta fase, são identificados os pontos de flexibilidade nas aplicações-exemplo;
- **(Re)Projetar o Framework** - esta fase consiste em melhorar a estrutura de classes inicialmente definida de modo a comportar a flexibilidade requerida. E recomendada a utilização de padrões de projeto a fim de garantir a flexibilidade;
- **Adaptar o Framework** - com o auxílio de especialista do domínio, é realizado um refinamento da estrutura do framework. Se o framework for avaliado como satisfatório, em relação aos pontos de flexibilidade identificados, está concluída uma versão do framework, caso contrário, deve-se retornar à segunda fase.

2.1.2.4 Metodologia Baseada na Fase de Projeto

Em sua abordagem, Silva [60] propõe que o desenvolvimento de um framework seja realizado por meio da execução de cinco etapas não-sequênciais, ou seja, as etapas podem ser repetidas até que a estrutura de classes do framework seja considerada satisfatória em relação aos requisitos de generalidade, alterabilidade e extensibilidade. A Figura 2.5 apresenta as etapas propostas por esta abordagem, sendo estas, descritas a seguir:

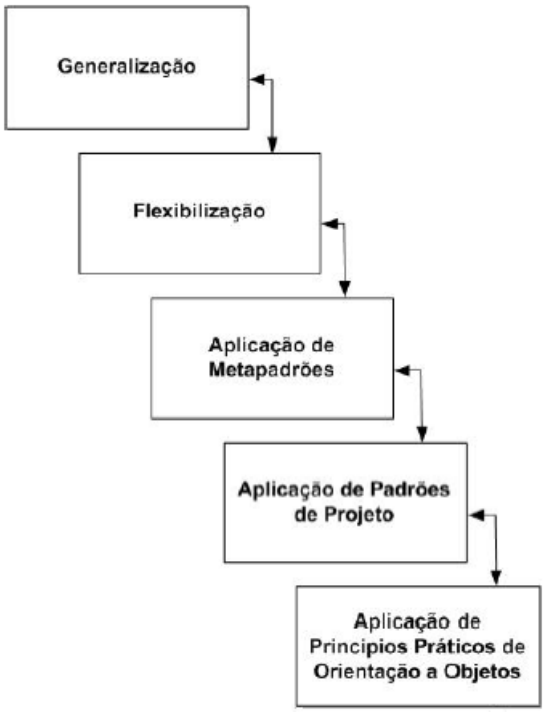

Figura 2.5: Processo de desenvolvimento proposto por Silva (figura extraída de [42]).

- **Generalização** - identificar os pontos de estabilidade nas aplicações-exemplo e gerar uma estrutura de classes que represente o domínio analisado. Nesta etapa, o desenvolvedor também deve buscar elementos prontos, como outros frameworks e componentes, que possam ser reutilizados;

- **Flexibilização** - identificar os pontos que devem ser mantidos flexíveis na estrutura de classes do framework. Esta atividade deve ser realizada através da geração de diferentes aplicações-exemplo;

- **Aplicação de Metapadrões** - identificar e adicionar à estrutura de classes do framework, os padrões que possam ser utilizados para satisfazer o requisito de fle-

xibilização. Nesta etapa, o desenvolvedor também deve criar os métodos gabarito[5] (*template*) e gancho[6] (*hook*);

- **Aplicação de Padrões de Projeto** - identificar e avaliar a necessidade de incluir as classes de um padrão de projeto na estrutura de classes do framework;

- **Aplicação de Princípios Práticos de Orientação a Objetos** - aplicar os conceitos e boas práticas da orientação a objetos com o intuito de obter um framework mais flexível.

2.1.2.5 Abordagem Dirigida a Responsabilidades

Matos e Fernandes [41] propõem uma abordagem para o desenvolvimento de frameworks de domínio chamada PDR-DFD (Processo Dirigido a Responsabilidades aplicado ao Desenvolvimento de Framework de Domínio). Formada por cinco fases, ilustradas na Figura 2.6, e criada a partir do estudo analítico [42, 40] de diversas abordagens voltadas ao desenvolvimento de frameworks. Esta abordagem está fundamentada nos métodos de Análise de Domínio [52], Projeto Dirigido por Responsabilidades [65, 66] e Arquitetura de Software [4].

A seguir, as fases que compõem a abordagem PDR-DFD, ilustradas na Figura 2.6, são brevemente apresentadas. Outras informações podem ser encontradas em [41].

- **Definir Domínio** - o projetista deve realizar um estudo do domínio com o auxílio das aplicações-exemplos identificadas;

- **Projetar Framework de Domínio** - o framework base, representando as classes comuns, e o de aplicação, representando as classes específicas, são gerados com o auxílio das aplicações-exemplo;

[5]Definem o esqueleto de um algoritmo numa operação, deixando que as subclasses redefinam certos passos sem mudar sua estrutura. Em geral, constituem uma forma de implementar os pontos de estabilidade de um framework [40].

[6]Podem possuir uma implementação inicial, a qual pode ser redefinida em subclasses. Em geral, constituem uma forma de implementar os pontos de flexibilidade de um framework [40].

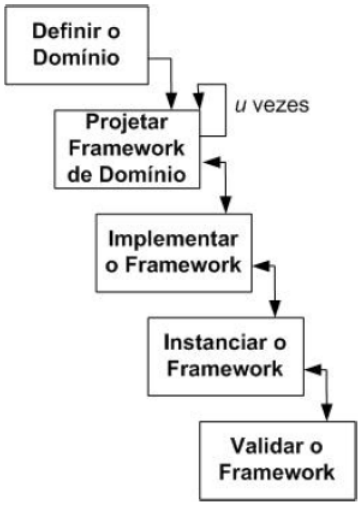

Figura 2.6: Processo de desenvolvimento proposto por Matos e Fernandes (figura extraída de [41]).

- **Implementar o Framework** - a especificação das classes, identificadas na fase anterior, são implementadas na linguagem escolhida;
- **Instanciar o Framework** - o framework gerado deve ser instanciado com a finalidade de se gerar a aplicação-exemplo para um ou mais exemplos concretos;
- **Validar o Framework** - confronta-se o que foi desenvolvido com o que foi especificado, com o objetivo de se validar o framework desenvolvido.

Nesta abordagem, a fase "Projetar Framework de Domínio" é a que possui maior destaque. Formada por duas subfases, como pode ser visto na Figura 2.7, as atividades propostas podem ser concretizadas por meio do apoio computacional provido pelo método MDR-PEF (Método Dirigido a Responsabilidades - Pontos de Estabilidade e de Flexibilidade) [40]. A seguir, as subfases que compõem a fase "Projetar Framework de Domínio" são brevemente descritas:

- **Compreender Aplicação-Exemplo** - realizada de forma iterativa, esta subfase possui a finalidade de gerar o Modelo de Requisitos, de Classes e de Arquitetura Refinada, para cada aplicação-exemplo que está sendo analisada;
- **Definir Framework Base e de Aplicação** - esta subfase possui a finalidade de gerar o modelo do framework base e de aplicação, sendo estes, utilizados como

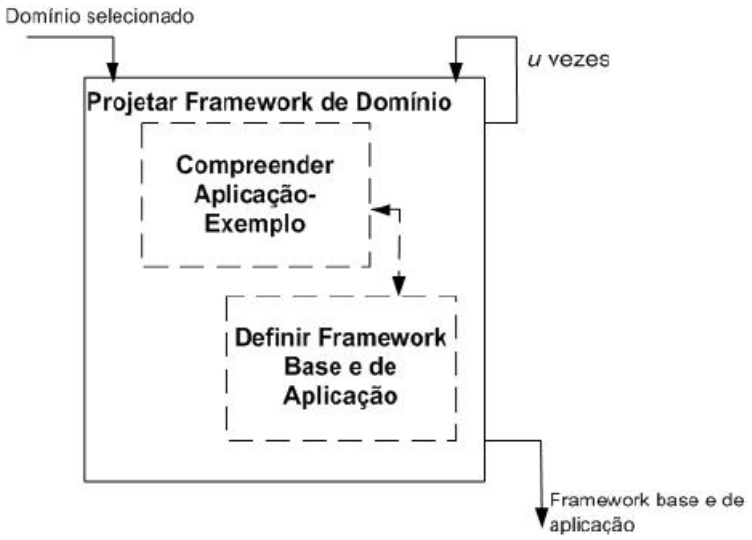

Figura 2.7: Subfases da fase "Projetar Framework de Domínio" (figura extraída de [41]).

entrada para a próxima fase ("Implementar Framework"). Assim como a anterior, esta subfase também pode ser realizada de forma iterativa.

2.1.3 Análise Comparativa entre as Abordagens Apresentadas

A comparação entre estas abordagens foi realizada utilizando como base o processo apresentado por Matos [40]. Este processo leva em consideração algumas características julgadas como essenciais para o desenvolvimento de frameworks de domínio, descritas a seguir:

- **Fase de Desenvolvimento (FD)** - define os passos do processo orientando o desenvolvedor durante a elaboração do framework;
- **Análise das Aplicações-Exemplo (AAE)** - analisa as aplicações-exemplo procurando levantar as principais abstrações do domínio;
- **Utilização de Padrões de Projeto (UPP)** - contribui para aumentar o grau de reusabilidade e evolução do framework;
- **Aplicação de Metapadrões (AM)** - melhora o grau de flexibilidade, generalidade e reusabilidade do framework;

- **Processo Iterativo (PI)** - possibilita refinar o framework à medida que novas abstrações vão sendo identificadas;
- **Produção de Subframeworks (PS)** - torna possível melhorar a reutilização, pois são direcionados a aspectos específicos do domínio;
- **Dicas de como Documentar o Framework (DDF)** - provê a documentação do framework com o objetivo de facilitar o trabalho do desenvolvedor;
- **Eliminação de Inconsistências, Ambiguidades e Contradições (EIAC)** - permite que ambiguidades possam ser identificadas em fase inicial do desenvolvimento evitando assim o retrabalho;
- **Identificação dos Pontos de Estabilidade e Flexibilidade (IPEF)** - contempla atividades que permitam a identificação dos pontos de estabilidade e de flexibilidade;
- **Criação de Subsistemas ou Componentes (CSC)** - permite a definição de uma arquitetura que facilita o entendimento do funcionamento do sistema e de seu reúso;
- **Utilização de Estilo Arquitetural (UEA)** - possibilita a aplicação de um estilo arquitetural visando facilitar a atividade de compreensão do sistema desenvolvido por outra pessoa e o seu reúso em um outro contexto, ou seja, uma nova aplicação-exemplo.

A Tabela 2.1 ilustra o relacionamento entre estas características e as abordagens apresentadas nesta dissertação:

Entre as abordagens presentes na Tabela 2.1, o que motivou a escolha deste trabalho por adaptar as atividades propostas pela abordagem PDR-DFD [41], foram:

- estar focada nas atividades iniciais do processo de desenvolvimento de frameworks de domínio;
- permitir a análise de domínio através da seleção e coleta das fontes de conhecimento;

Tabela 2.1: Comparação entre as abordagens para o desenvolvimento de frameworks (adaptado de [40]).

	FD	**AAE**	**UPP**	**AM**	**PI**	**PS**	**DDF**	**EIAC**	**IPEF**	**CSC**	**UEA**
Johnson [33]	X	X	X		X				X		
Taligent [30]	X	X	X		X	X	X			X	
Pree [51]	X	X	X	X	X				X		
Silva [60]	X	X	X	X	X				X		
Matos e Fernandes [41]	X	X	X	X	X	X		X	X	X	X

- definir a arquitetura de alto nível do sistema no início do desenvolvimento, diminuindo o retrabalho durante a confecção do framework;

- ser baseada em responsabilidades, facilitando a identificação das funcionalidades do sistema, bem como a identificação dos pontos de estabilidade e flexibilidade do framework;

- ser iterativa e incremental, pois cada aplicação-exemplo é desenvolvida iterativamente, podendo ter o incremento de novos subsistemas ao framework, deixando-o mais consistente;

- disponibilizar uma ferramenta de apoio para a identificação dos pontos de estabilidade e flexibilidade.

2.2 Desenvolvimento de Software Orientado a Aspectos

2.2.1 Conceitos Gerais

A separação de interesses (do inglês *separation of concerns*) é um princípio bem estabelecido pela comunidade de engenharia de software e refere-se à limitação humana em lidar com a complexidade. Dijkstra foi o primeiro pesquisador a utilizar o termo "*separation of concerns*" em seu livro "*A Discipline of Programming*"[19], onde é apresentada uma estratégia que consiste em focar a atenção em uma porção autocontida, ou interesse, por vez, tendo-se ciência de que se está ignorando temporariamente outras partes do problema [16].

Desde o início, a separação de interesses tem sido alvo de constantes pesquisas, em que a cada novo paradigma criado, novas construções sintáticas eram fornecidas com o intuito de melhorar a modularização dos interesses, tornando assim, os sistemas de software mais coesos e com interfaces melhor definidas [16].

Atualmente, o paradigma dominante é o Desenvolvimento de Software Orientado a Objetos (DSOO) [1]. Contudo, ele possui limitações ao tratar interesses relacionados à restrições globais de software e propriedades sistêmicas, tais como: segurança, desempenho, persistência, tratamento de exceções, *logging*, entre outros [35]. Esses interesses ficaram conhecidos como interesses transversais (do inglês *crosscutting concerns*) e costumam ficar espalhados (do inglês (*spreading*)) e/ou entrelaçados (do inglês (*tangling*)) por vários módulos do sistema [61].

A Figura 2.8 ilustra o problema do espalhamento e entrelaçamento por meio de um exemplo, em que o código relativo ao interesse "*logging*", encontra-se entrelaçado e espalhado pelas diversas classes.

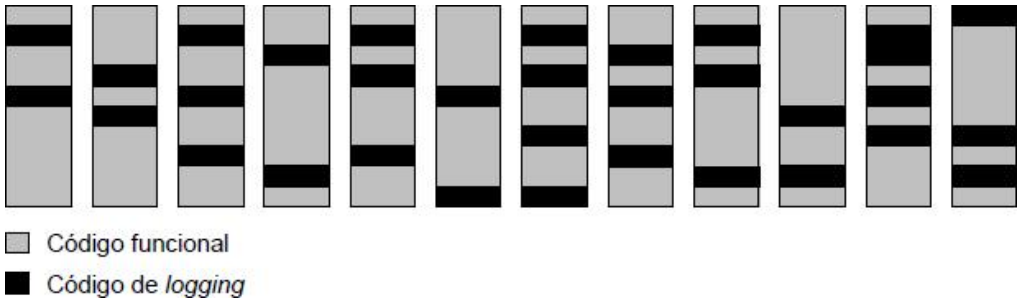

Figura 2.8: Exemplo de espalhamento e entrelaçamento de interesses transversais (figura extraída de [46]).

Com o intuito melhorar a separação dos interesses transversais, diversas abordagens foram propostas como complemento ao DSOO. Entre as abordagens, podem-se citar: a Programação Adaptativa [39], Filtros de Composição [5], Separação Multidimensional de Interesses [62] e a Programação Orientada a Aspectos [35].

Segundo Viega [64], a Programação Orientada a Aspectos (POA) proposta por Kiczales *et al.* [35], foi a abordagem usada para a separação de interesses transversais que mais se destacou. A idéia central da POA é decompor sistemas complexos em interesses-base e transversais [16]:

- **interesses-base** - referem-se às funcionalidades principais do sistema;
- **interesses transversais** - relativos aos interesses que causam os problemas de espalhamento e/ou entrelaçamento.

Dessa maneira, os interesses-base são implementados se utilizando as abordagens convencionais. Por sua vez, os interesses transversais devem ser codificados em linguagens especialmente designadas para esse propósito. Um exemplo de linguagem para a implementação dos interesses transversais é o AspectJ[7].

O AspectJ é uma extensão da linguagem de programação Java, em que os interesses transversais são modularizados e implementados em objetos denominados aspectos. Em AspectJ, um aspecto é composto por [37]:

- **pontos de junção** (do inglês *join points*) - ponto de execução que o aspecto deseja interceptar, como por exemplo, uma chamada a um método ou a ocorrência de uma exceção;
- **pontos de corte** (do inglês *pointcuts*) - local onde os pontos de junção são definidos;
- **adendos** (do inglês *advices*) - definem o comportamento que será invocado durante a ocorrência do ponto de junção, podendo ser de três tipos: *before*, *after* e *around*.

Como pode ser visto na Figura 2.9, a utilização da POA permite centralizar o código do interesse "*logging*" em um único componente, o aspecto, removendo assim seu entrelaçamento e espalhamento pelas demais classes do sistema, quando comparado a Figura 2.8.

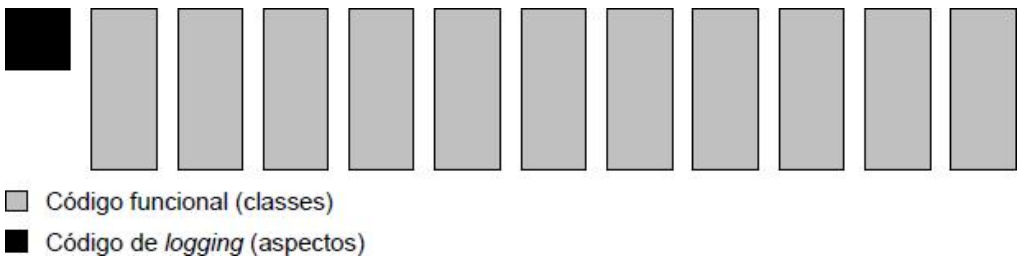

Figura 2.9: Exemplo de modularização de interesses transversais (figura extraída de [46]).

[7]AspectJ é um projeto da Eclipse Foundation. Disponível em: *http://eclipse.org/aspectj/*, julho 2010.

Inicialmente, os esforços da POA concentravam-se na fase de implementação, sendo os interesses transversais identificados em especificações de desenho orientadas por objetos e mais tarde complementados com técnicas de codificação orientada por aspectos. Porém, problemas relacionados à falta de rastreabilidade entre os aspectos e a incapacidade de prever os efeitos ocasionados por mudanças em requisitos, dificultaram a sua aplicação em sistemas de grande porte [58].

Nesse contexto, surgiu uma nova área de pesquisa conhecida como Desenvolvimento de Software Orientado a Aspectos (DSOA) [58]. O DSOA preocupa-se em desenvolver métodos, técnicas e ferramentas que permitam à identificação, modularização e composição dos interesses transversais em todas as fases do desenvolvimento, buscando assim, tornar o processo mais homogêneo, permitindo que os interesses transversais recebam a mesma importância desde a análise até a implementação.

A importância do DSOA no contexto da Engenharia de Software se mostra evidente devido a crescente quantidade de pesquisas desenvolvidas nesta área, além de algumas ferramentas tais como, o framework de integração Spring[8], o servidor de aplicações JBoss AOP[9] e framework dinâmico para programação orientado a aspectos AspectWerkz[10]. Assim, estas ferramentas têm incorporado o uso de técnicas orientadas a aspectos para endereçar interesses transversais tradicionais encontrados no desenvolvimento de aplicações corporativas ou requisitos de adaptação dinâmica [46].

Segundo Camargo [16], uma grande parte das pesquisas em DSOA focam em fases do ciclo de desenvolvimento. Tekinerdogan *et al.* [63] classificam os aspectos de acordo com a fase do desenvolvimento onde eles são identificados, como: Aspectos Antecipados (*Early Aspects*), Aspectos Intermediários (*Intermediate Aspects*) e Aspectos Tardios (*Late Aspects*). A Figura 2.10 ilustra a classificação dos tipos de aspecto de acordo com a fase do desenvolvimento de software onde eles são identificados.

[8]Spring é um projeto da SpringSource Community. Disponível em: *http://springsource.org/*, julho 2010.

[9]JBoss AOP é um projeto da JBoss Community. Disponível em: *http://jboss.org/jbossaop/*, julho 2010.

[10]AspectWerkz é um projeto de Jonas Boner e Alexandre Vasseur. Disponível em: *http://aspectwerkz.codehaus.org/*, julho 2010.

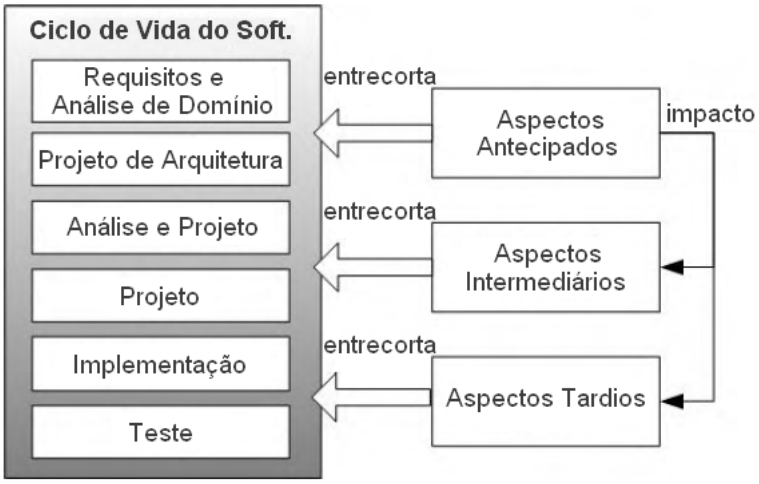

Figura 2.10: Relacionamento entre a identificação de aspectos e as fases do desenvolvimento de software (figura adaptada de [63]).

A seguir, é apresentada uma revisão bibliográfica de algumas propostas que buscam dar suporte a identificação dos aspectos nas etapas iniciais do ciclo de desenvolvimento, ou seja, antes da implementação.

2.2.2 *Early-Aspects*

As pesquisas em DSOA que buscam prover técnicas que auxiliem durante a identificação, separação, representação e composição de interesses transversais em etapas de desenvolvimento anteriores à implementação, ficaram conhecidas como *early-aspects* [58].

A seguir serão brevemente apresentadas algumas abordagens em *early-aspects* [55, 44, 43, 2, 8, 31, 3]. Mais detalhes a respeito destas abordagens podem ser encontrados em [11, 12, 61, 58, 59].

2.2.2.1 Abordagem *AORE* e Extensões

A abordagem *AORE* (*Aspect-Oriented Requirements Engineering*), inicialmente apresentada em [55], foi proposta por Rashid *et al.* com o intuito de auxiliar na separação dos interesses transversais na etapa de requisitos.

Em *AORE*, os autores enfatizam a importância de tratar os interesses transversais antecipadamente, sendo estes, inicialmente classificados como "candidato a aspecto". Nesta abordagem, os requisitos não-funcionais de alto nível de abstração como: segurança, com-

patibilidade, entre outros, são identificados como candidatos a aspecto.

Segundo o modelo *AORE*, a identificação dos candidatos a aspecto é realizada com o auxílio de uma matriz de requisitos por interesses transversais, em que os interesses que cruzam mais de um requisito são assim classificados. Posteriormente, os candidatos a aspecto identificados devem ser analisados detalhadamente, buscando possíveis conflitos e estabelecendo prioridades. Por fim, deve-se realizar a especificação do impacto dos candidatos a aspecto em termos de duas dimensões:

- **mapeamento** - estabelece como o candidato a aspecto deverá ser manipulado nas fases seguintes do desenvolvimento (função, decisão ou aspecto);
- **influência** - estabelece as fases de desenvolvimento afetadas pelo candidato a aspecto (requisitos, arquitetura, projeto ou manutenção).

Em [44], Rashid *et al.* apresenta-se uma melhoria ao modelo *AORE*, em que a principal modificação é a inclusão de duas novas atividades: a composição de candidatos a aspecto e requisitos e a manipulação de conflitos.

A primeira atividade incluída, composição de candidatos a aspecto e requisitos, é responsável por determinar como um candidato a aspecto influencia ou restringe um requisito através de regras de composição. Por sua vez, a atividade de manipulação de conflitos é formada pelo preenchimento de uma matriz de contribuição, positiva ou negativa, entre candidatos a aspecto e, posteriormente, a atribuição de peso aos candidatos que contribuem para a realização de outro.

A Figura 2.11 apresenta o modelo de processo resultante das pesquisas [55, 44].

Posteriormente, em [43, 2, 8] é proposta a utilização de uma instância simplificada do modelo *AORE* baseada em *UML*. Os requisitos funcionais são representados por meio de diagramas de casos de uso e sequência. Por sua vez, os requisitos não-funcionais são descritos usando modelos pré-definidos nos quais são inseridas referências explícitas aos requisitos funcionais afetados por eles. Para a composição dos requisitos, funcionais e não-funcionais, é proposta a utilização de uma representação gráfica, criada através da

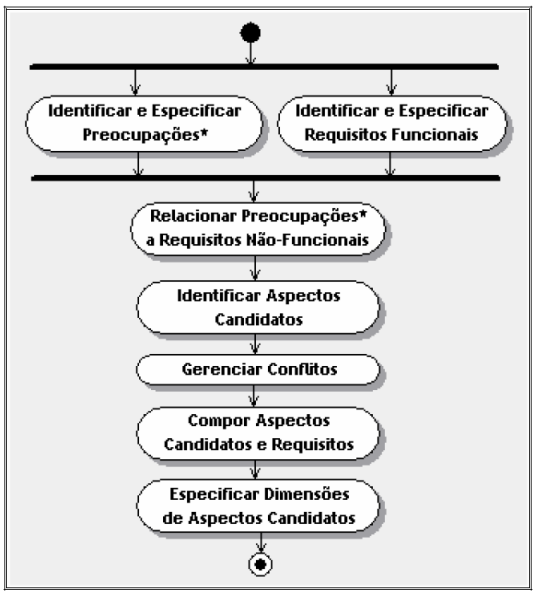

Figura 2.11: Modelo de processo *AORE* (figura extraída de [61]).

adaptação de caso de uso e diagramas de sequência, além dos operadores descritos a seguir:

- ***overlap*** - utilizado para indicar quando o comportamento do aspecto deve ser aplicado antes (*before*) ou depois (*after*) do requisito funcional que ele afeta;
- ***override*** - utilizado quando o comportamento do aspecto substituí o comportamento do requisito funcional por ele afetado;
- ***wrap*** - usado quando a realização do aspecto encapsula o comportamento do requisito funcional por ele afetado.

2.2.2.2 Abordagem Baseada em Casos de Uso

Jacobson e NG [31] propõem uma abordagem para o DSOA utilizando os conceitos de casos de uso. Segundo os autores, os casos de uso são transversais por natureza, uma vez que a sua realização pode afetar outras classes. No entanto, a principal dificuldade está em manter os interesses separados durante o processo de desenvolvimento.

Segundo esta abordagem, os interesses transversais podem ser de dois tipos:

- ***peers*** - não depende de outro interesse para existir, porém, a sua implementação pode causar o problema do espalhamento e entrelaçamento;

- ***extensions*** - representam serviços ou características adicionais, ou seja, a sua existência está diretamente relacionada a outros interesses e a sua implementação causa a inserção de forma intrusiva no componente afetado.

As Figuras 2.12 e 2.13 ilustram os problemas ocasionados pelos interesses *peers* e *extensions*, respectivamente.

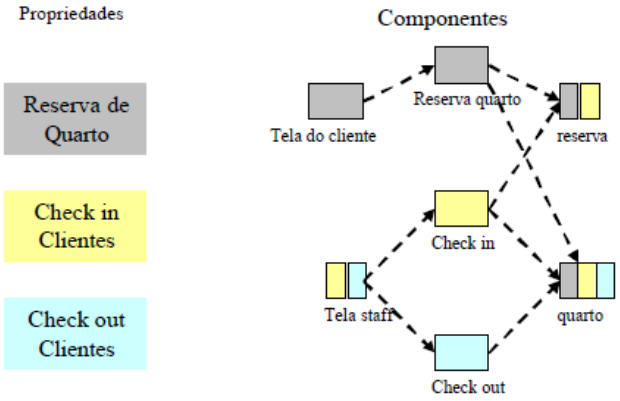

Figura 2.12: Exemplo do problema ocasionado pela realização de interesses *peers* (figura extraída de [59]).

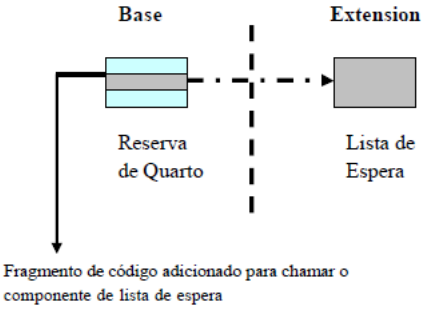

Figura 2.13: Exemplo do problema ocasionado pela realização de interesses *extensions* (figura extraída de [59]).

Os interesses transversais do tipo *peers*, ilustrados na Figura 2.12, estão relacionados ao problema de entrelaçamento e espalhamento de interesses durante a sua implementação. Por exemplo, o componente (classe) "Quarto" está envolvido na realização de três diferentes propriedades (funcionalidades): "Reserva de Quarto", "*Check in* Clientes" e "*Check out* Clientes".

Por sua vez, os interesses transversais do tipo *extensions*, ilustrados na Figura 2.13, estão relacionados ao problema de inserção de forma intrusiva de código. Por exemplo,

o código relativo a chamada do componente “Lista de Espera” está inserida no meio do código do componente base “Reserva de Quarto”.

Neste contexto, os autores propõem colecionar a especificação dos casos de uso em uma unidade modular denominado *use case slice*, atividade ilustrada na Figura 2.14.

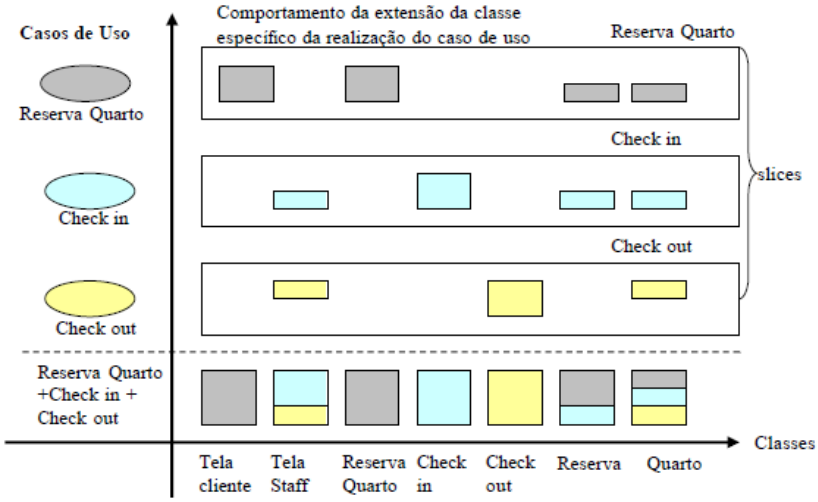

Figura 2.14: Exemplo de composição de casos de uso *peer* e *slices* (figura extraída de [59]).

Como pode ser visto na Figura 2.14, cada *use case slice* contém apenas a parte necessária à concretização do seu caso de uso. O sistema final será obtido através da união destes elementos.

Segundo Jacobson e NG [31], o processo de desenvolvimento de software se resume à construção de modelos. O primeiro destes modelos é o de casos de uso, cujo objetivo é capturar os requisitos com os *stakeholders*[11]. Em seguida, o modelo de casos de uso é refinado para dar origem ao modelo da arquitetura do sistema na fase de análise, o qual apresenta uma visão de alto nível do sistema. Logo após, a estratégia de como o sistema executará na plataforma é obtida com o modelo da arquitetura na fase de projeto. Por fim, no modelo de implementação o código-fonte do sistema é obtido.

[11] O termo *stakeholder* é utilizado para ilustrar uma pessoa, ou grupo delas, que tenha influência direta ou indireta no projeto.

2.2.2.3 Abordagem Baseada em Temas

Em [3], Baniassad e Clarke propõem a abordagem *Theme*. Constituída por duas partes, *Theme/Doc* e *Theme/UML*, esta abordagem visa a identificação dos interesses transversais em diferentes fases do desenvolvimento. O *Theme/Doc* possui a finalidade de prover um suporte ferramental para identificação e representação de interesses transversais em documentos de requisitos. Por sua vez, o *Theme/UML* possui a finalidade de auxiliar o engenheiro durante a modelagem do projeto.

A Figura 2.15 ilustra as atividades propostas por esta abordagem.

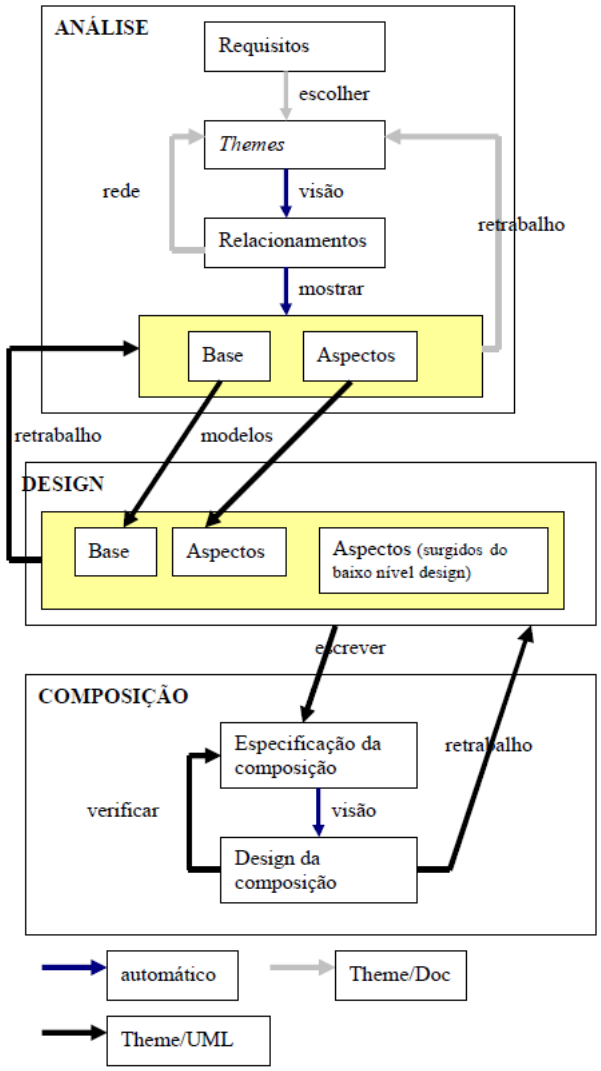

Figura 2.15: Atividades da abordagem *Theme* (figura extraída de [59]).

De maneira geral, a abordagem *Theme* é formada por três atividades principais:

- **Análise** - identificação dos *themes*[12] através da análise dos requisitos, visualização do relacionamento entre os interesses com o auxílio do *Theme/Doc*, identificação dos interesses entrelaçados e potenciais aspectos;

- **Projeto** - detalhamento do modelo de projeto dos *themes* com o auxílio do *Theme/UML*, nesta etapa, outros aspectos podem ser identificados;

- **Composição** - especificação de como os modelos *Theme/UML* devem ser combinados, podendo utilizar algumas visões apresentadas pelo *Theme/Doc* para auxiliar na determinação de como os *themes* se relacionam com outros.

2.2.3 Análise Comparativa entre as Abordagens Apresentadas

A comparação entre as abordagens foi realizada utilizando como base o processo apresentado por Silva [59]. Este processo leva em consideração uma adaptação das características de qualidade elaboradas pela ISO/IEC 9126. Em [59], se encontra a descrição completa deste processo, sendo a seguir, realizado apenas uma breve introdução:

- **Funcionalidade** - evidencia que o conjunto de funções atende às necessidades explícitas e implícitas para a finalidade a que se destina a abordagem;

- **Confiabilidade** - evidencia que o desempenho se mantém ao longo do ciclo ou fase a que se destina e em condições estabelecidas;

- **Usabilidade** - evidencia a facilidade para a utilização da abordagem;

- **Eficiência** - evidencia que os recursos e os tempos envolvidos para a aplicação da abordagem são compatíveis com o nível de desempenho requerido para a aplicação da abordagem;

- **Manutenibilidade** - evidencia que há facilidades para correções, atualizações e alterações durante a sua aplicação;

[12]Segundo os autores, *theme* é um elemento de projeto (*design*) que encapsula o comportamento de um interesse.

- **Portabilidade** - evidencia que é possível utilizar a abordagem em diversas plataformas (utilizando sua ferramenta Case) com pequeno esforço de adaptação.

O critério para comparação entre as abordagens leva em consideração três níveis de avaliação descritos a seguir:

- **fraca** - não atende ao critério ou é considerada insatisfatória;
- **regular** - atende ao critério, porém não completamente;
- **boa** - atende completamente ao critério ou é considerada satisfatória.

Na Tabela 2.2 encontra-se a comparação entre as abordagens apresentadas utilizando os criterios relatados anteriormente.

Tabela 2.2: Comparação entre as abordagens apresentadas em *early-aspects* (tabela extraída de [59]).

	***AORE* [55, 44, 43, 2, 8]**	**Casos de Uso [31]**	***Theme* [3]**
Funcionalidade	Fraca	Regular	Regular
Confiabilidade	Fraca	Boa	Boa
Usabilidade	Regular	Regular	Regular
Eficiência	Regular	Regular	Regular
Manutenibilidade	Boa	Regular	Regular
Portabilidade	Boa	Regular	Fraca

As abordagens descritas anteriormente contribuiram para o desenvolvimento do método proposto, sendo utilizadas como base para a criação de seus conceitos, critérios e técnicas.

2.3 Frameworks Orientados a Aspectos

2.3.1 Conceitos Gerais

Com o avanço da Programação Orientada a Aspectos (POA) e de suas linguagens, como AspectJ, surgiram trabalhos como [45, 27, 29, 17, 16, 46, 37], entre outros, onde os novos conceitos da POA foram aplicados no contexto de Frameworks Orientados a Objetos (FOO) [13].

Conhecido como Framework Orientado a Aspectos (FOA) [16], possui a sua estrutura formada por um conjunto de aspectos e opcionalmente classes. Ou seja, um FOA pode ser composto exclusivamente de aspectos, embora isso não seja comum. Este conjunto de aspectos e classes representam o projeto abstrato de soluções para uma família de problemas relacionados [46].

Inicialmente, a definição utilizada de forma intuitiva por diversos autores caracterizava um FOA como um framework convencional que também possuía estruturas de aspectos em sua implementação [17].

Um dos primeiros autores a propor uma definição para um FOA foi Hanenberg [27], definindo-o como uma coleção de aspectos concretos e abstratos. Ele ainda afirma que além dos métodos-gancho (do inglês *hook methods*), que são conceitos amplamente utilizados em frameworks tradicionais, os conjuntos de junção (do inglês *pointcuts*) também podem ser especializados, permitindo que um interesse seja acoplado a inúmeros módulos de implementação [27, 29].

Por sua vez, Camargo e Masiero [17] vieram propor uma definição para um FOA, considerando a sua estrutura e o seu propósito. Quanto a sua estrutura, eles definem um FOA como um conjunto formado por aspectos e unidades básicas da programação orientada a objetos, como classes e interfaces. Já quanto ao seu propósito, assim como um framework convencional, um FOA pode ser definido como um sistema semi-completo e reutilizável que pode ser instanciado por um desenvolvedor de aplicações, mantendo os interesses transversais separados dos interesses-base.

Quanto a sua classificação, Camargo e Masiero [17] dividem um FOA em dois grupos:

- **Frameworks Transversais (FT)** - possui mecanismos de composição abstratos e variabilidades correspondentes a um único interesse transversal;
- **Frameworks de Aplicação Orientado a Aspectos (FAOA)** - implementa uma arquitetura genérica para um domínio.

Quanto aos seus benefícios, além dos encontrados em um framework convencional, um FOA, quando bem desenvolvido, melhora a modularidade, reusabilidade e extensibilidade,

se comparado a um framework convencional. Isto se deve a uma melhor organização do código e separação dos interesses envolvidos em sua estrutura, característica esta, própria da orientação a aspectos. Outro ponto, é que dependendo da sua natureza, um FOA pode ser utilizado em diferentes domínios, acoplado a aplicações já existentes ou a outros frameworks. Ponto este, que o difere de um framework convencional [46].

Porém, a sua utilização requer um maior cuidado, pois assim como um aspecto, um FOA pode modificar totalmente um método, por exemplo, através da substituição do seu código, possivelmente aumentando assim a sua curva de aprendizado. Quanto ao desenvolvedor, existe a dificuldade em manter a extensibilidade e restringir a sua atuação, o que causa impacto no tempo total de desenvolvimento [46].

O reúso de um FOA é mais abrangente do que um framework convencional, existindo três formas possíveis de reúso: instanciação, composição, instanciação e composição, e muitas vezes, sua origem, FT ou FAOA, determina a forma de reúso. Esta classificação não é válida para um FOA caixa preta, já que não importa como foi projetada a arquitetura da parte variável, pois todas as variabilidades já foram concretizadas e trata-se apenas de um processo de escolha que geralmente é feito com auxílio automatizado [16].

Embora a maior parte das pesquisas concentre-se no desenvolvimento de FOA de um único interesse, como persistência, segurança, distribuição, concorrência, entre outros, alguns trabalhos [49, 48, 50, 28] têm aplicado os aspectos em frameworks de aplicação orientados a objetos [16].

2.3.2 *Early-Aspects* e o Desenvolvimento de Frameworks Orientados a Aspectos

Entre os trabalhos pesquisados, apenas Nakajima [45], Camargo [16] e Kulesza [37] fazem referência à identificação de aspectos em etapas iniciais do processo de desenvolvimento de frameworks. Nestes trabalhos a identificação antecipada de aspectos é tratada como uma possibilidade, não sendo apresentada, proposta ou referenciada nenhuma técnica para realizar tal atividade. A seguir é realizada uma breve introdução a estes trabalhos.

2.3.2.1 Proposta de Nakajima

Em seu trabalho intitulado "*Separation of Concerns in Early Stage of Framework Development*" [45], Nakajima descreve sobre a importância dos FOO e as dificuldades encontradas pelas metodologias durante o seu desenvolvimento.

Nakajima também relata que as metodologias de desenvolvimento de FOO são em geral eficazes, porém, possuem grandes inconveniências como:

- serem desenvolvidas com base na orientação a objetos, consideram assim que um sistema é formado unicamente por objetos;
- apresentarem apenas orientações gerais de como os sistemas podem ser decompostos em objetos, porém, não apresentando dicas concretas;
- apesar de os padrões de projeto (do inglês *design patterns*) serem extremamente úteis, na maioria dos casos, servem mais às etapas de projeto e de implementação, portanto, não são aplicáveis em etapas iniciais do desenvolvimento.

O foco central do trabalho de Nakajima é apresentar um método que trata da separação de interesses em etapas iniciais do processo de desenvolvimento de FOO. Este método foi desenvolvido com base na experiência obtida com o desenvolvimento de um FOO para a implementação de servidores de negociação *OMG*.

O método proposto por Nakajima consiste basicamente de duas fases:

- **Modelagem de Aspectos (do inglês *Aspect Design*)** - como entrada, esta fase utiliza os documentos da OMG e os requisitos do sistema, procurando-se identificar um conjunto de aspectos distintos e a partir destes, gerar uma descrição semi-formal do problema. Os dados obtidos nesta fase são utilizados como entrada para a próxima fase;
- **Modelagem Orientada a Objetos (do inglês *Object-Oriented Design*)** - nesta fase, o framework e a sua documentação são produzidos de forma incremental, para isso são utilizados métodos já conhecidos, como: padrões de projeto, a linguagens como Java e a biblioteca JavaIDL.

Em seu trabalho, Nakajima apresenta um método baseado na identificação de um conjunto de aspectos distintos que são utilizados como entrada para o desenvolvimento de frameworks. Também são apresentadas as vantagens, dificuldades e possíveis áreas de pesquisa a respeito deste assunto.

2.3.2.2 Proposta de Camargo

Em sua tese [16], Camargo apresenta algumas definições e classificações para frameworks desenvolvidos no contexto da POA. Também é apresentado pelo autor um processo de desenvolvimento de software baseado no Processo Unificado [6], denominada ProFT/PU. Este processo é iterativo e incremental, além de ser fortemente baseado na identificação e acompanhamento de interesses transversais ao longo do processo de desenvolvimento.

Com relação a modelagem de aspectos durante o desenvolvimento de FOA, Camargo relata a possibilidade de utilizar três abordagens, sendo estas, brevemente apresentadas a seguir:

- **Baniassad e Clarke [3]** - apresentam um método para o desenvolvimento de software orientado a aspectos. São apresentadas duas abordagens específicas para as fases de análise e composição: *Theme/Doc* e *Theme/UML*, respectivamente. O *Theme/Doc* está relacionado com a identificação e representação de características transversais em documentos de requisitos e, por sua vez, o *Theme/UML* possui a finalidade de auxiliar o engenheiro durante a modelagem da arquitetura;

- **Rausch *et al.* [56]** - esta abordagem propõe uma extensão da *UML* para modelar separadamente o framework, a aplicação e as suas regras de composição. O objetivo central desta abordagem é deixar claro para o engenheiro quais são os pontos de acoplamento do framework, sendo estes chamados pelos autores de *hot spots* e modelados como aspectos;

- **Krechetov *et al.* [36]** - esta abordagem foi desenvolvida com base na união de outras abordagens notacionais para arquitetura de software. O seu objetivo central

é disponibilizar uma visão de alto nível da arquitetura em termos de seus módulos-base e aspectuais. Para isso, são utilizados a *UML* e algumas notações específicas para os componentes aspectuais e seu relacionamento com os demais componentes.

Camargo relata a utilização destas abordagens na fase de projeto, não sendo citada a fase de análise e especificação de requisitos.

2.3.2.3 Abordagem de Kulesza

Em sua tese [37], Kulesza apresenta uma abordagem sistemática para o desenvolvimento de frameworks usando técnicas do DSOA e do Desenvolvimento Generativo (DG) [15, 14]. O objetivo central desta abordagem é melhorar a modularização de interesses transversais encontrados em frameworks de forma a facilitar a sua customização para diferentes cenários. Para isto, é proposto que o framework seja obtido através da composição de uma estrutura núcleo e de um conjunto de aspectos de extensão, como pode ser visto na Figura 2.16.

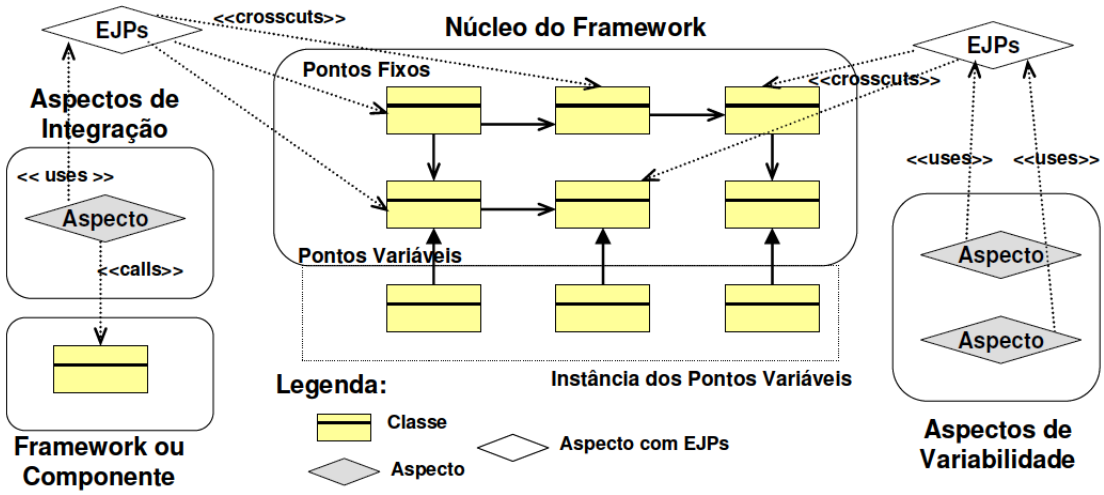

Figura 2.16: Elementos de implementação da abordagem de Kulesza (figura extraída de [37]).

A seguir, os elementos ilustrados na Figura 2.16 são brevemente apresentados:

- **núcleo do framework** - contém a implementação de classes que representam os pontos de estabilidade e flexibilidade não transversais do domínio;

- **aspectos do núcleo** - contém os interesses transversais existentes nas classes do núcleo do framework, são implementados utilizando as técnicas tradicionais da POA;

- **aspectos de variabilidade** - contém os interesses opcionais e alternativas transversais existentes no núcleo do framework, estendem os EJPs[13] do framework;

- **aspectos de integração** - definem composições transversais entre o núcleo do framework e outras extensões existentes, como: bibliotecas, componentes ou outro framework, também estendem os EJPs do framework.

As atividades para obtenção da estrutura proposta por esta abordagem estão organizadas sob duas perspectivas: engenharia de domínio e engenharia de aplicação. Porém, estas atividades não são descritas pelo presente trabalho por não fazerem parte de seu escopo.

Apesar de Kulesza propor uma abordagem para implementação de interesses transversais encontrados em FOO, atualmente sua abordagem endereça as etapas de projeto, implementação e instanciação. Sendo a etapa de análise e especificação de requisitos, uma atividade apresentada como trabalho futuro.

2.4 Conclusões

Este capítulo apresentou uma visão geral das três principais abordagens relacionadas com a pesquisa conduzida por esse trabalho. Na Seção 2.1, alguns conceitos e definições a cerca de FOO foram brevemente apresentados, assim como, algumas abordagens para o desenvolvimento de frameworks de domínio. Na Seção 2.2, o DSOA e algumas abordagens para identificação de aspectos em etapas iniciais do processo de desenvolvimento foram brevemente apresentadas. Por fim, a Seção 2.3 apresentou alguns conceitos relacionados a FOA, assim como, algumas pesquisas que propõem a identificação dos aspectos nas fases iniciais do seu processo de desenvolvimento.

[13]Os EJPs (do inglês *Extension Join Points*) expõem pontos específicos da execução do framework com a finalidade de facilitar a implementação de variabilidades transversais e de integração.

O capítulo seguinte apresenta o método proposto por esta dissertação para identificação de candidatos a aspectos em etapas iniciais do processo de desenvolvimento de frameworks de domínio, atividade essa, não abordada pelos trabalhos descritos neste capítulo.

CAPÍTULO 3

UM MÉTODO PARA IDENTIFICAÇÃO ANTECIPADA DE CANDIDATOS A ASPECTO

Neste capítulo é feita a descrição do método para identificação de candidatos a aspecto em fase inicial do processo de desenvolvimento de frameworks de domínio. Este método foi desenvolvido com base em conceitos, critérios e técnicas de trabalhos que propõem o desenvolvimento de frameworks de domínio [41] e a identificação antecipada de aspectos (*early-aspects*) [61, 31, 16]. A Seção 3.1 apresenta uma visão geral do método proposto. Na Seção 3.2 o método proposto é apresentado. A Seção 3.3 apresenta o contexto de aplicação do método proposto.

3.1 Visão Geral do Método Proposto

Como citado na Seção 2.3.1, com o avanço da Programação Orientada a Aspectos (POA) e de suas linguagens, como AspectJ, surgiram trabalhos em que estes novos conceitos foram aplicados na implementação de Frameworks Orientados a Objetos (FOO), surgindo assim, os Frameworks Orientados a Aspectos (FOA). No entanto, tratam-se de processos evolutivos, onde os aspectos são incorporados a FOO já desenvolvidos, ou de processos que visam a identificar os aspectos nas etapas de projeto e implementação. Entre os trabalhos encontrados, apenas Nakajima [45], Camargo [16] e Kulesza [37] relatam a possibilidade de identificar os aspectos durante a análise de requisitos, entretanto, são apenas propostas. Na Seção 2.3.2, estes trabalhos foram brevemente apresentados.

Neste contexto, um dos principais objetivos deste trabalho é propor um método que permita antecipar a identificação dos candidatos a aspecto para as atividades iniciais do processo de desenvolvimento de frameworks de domínio. Com isso, pretende-se trazer algumas das vantagens atribuídas ao Desenvolvimento de Software Orientado a Aspectos (DSOA) para o desenvolvimento de frameworks de domínio, tais como: diminuição

no entrelaçamento e espalhamento dos interesses, rastreabilidade dos interesses entre as etapas do desenvolvimento, diminuição do esforço de desenvolvimento e manutenção de aplicações, entre outras.

O processo geral de funcionamento do método proposto está ilustrado na Figura 3.1. Como entrada, o método necessita do Modelo de Requisitos do Framework (MRF). O MRF é constituído por um ou mais Diagramas de Casos de Uso (DCU), notação *UML-F* [25], e a Descrição Textual (DT). Como saída, o método produz um MRF com os candidatos a aspectos identificados.

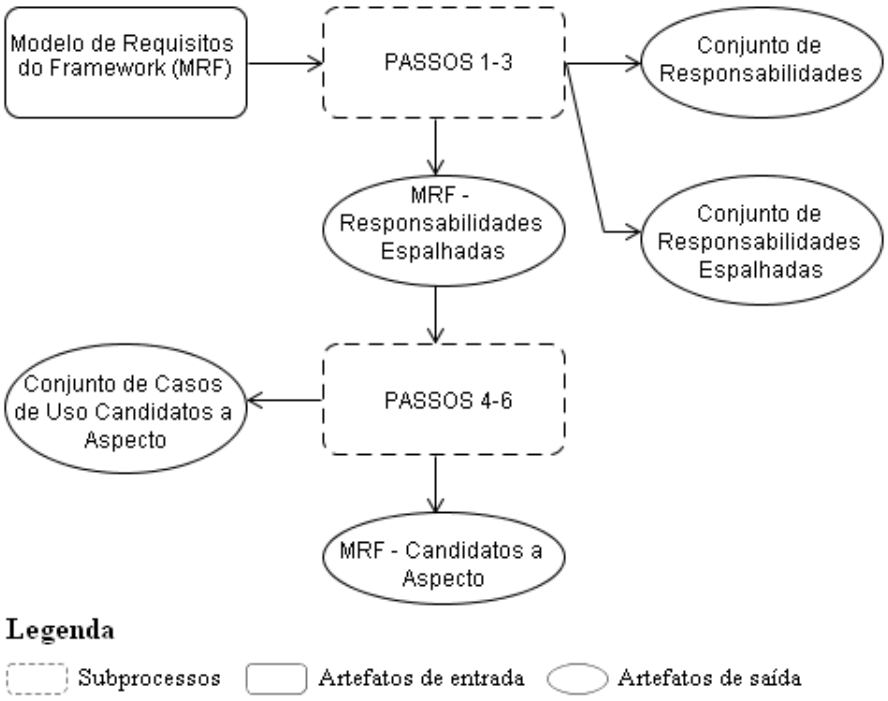

Figura 3.1: Processo geral do método proposto.

O método proposto possui dois subprocessos principais que estão representados na Figura 3.1 por retângulos com linhas pontilhadas. O primeiro subprocesso, PASSOS 1-3, refere-se ao procedimento para identificação e representação das responsabilidades[1] espalhadas pelos casos de uso. Já no segundo subprocesso, PASSOS 4-6, objetiva-se a identificação e a representação dos casos de uso que apresentam características que indicam que sua implementação com aspectos é factível.

[1]O termo "responsabilidades" está relacionado ao trabalho de Wirfs-Brock [65], são declarações gerais sobre as ações que um objeto executa e mantém, e sobre as decisões principais que um objeto produz ao afetar outros [40].

Com o método proposto, pretende-se antecipar a identificação dos candidatos a aspecto para o começo do processo de desenvolvimento de frameworks de domínio, usando como base, a abordagem para o desenvolvimento de frameworks de domínio PDR-DFD (Processo Dirigido a Responsabilidades aplicado ao Desenvolvimento de Framework de Domínio), proposta por Matos e Fernandes [41]. Na Seção 2.1.2.5 as fases e atividades desta abordagem foram brevemente descritas.

Com isso, proporciona-se às fases posteriores mais tempo para o refinamento do framework, inclusive para a definição de novos candidatos a aspecto obtidos a partir da experiência do projetista ou da estrutura de classes do framework em desenvolvimento, se for o caso.

A utilização do método proposto, contudo, não prescinde da experiência do projetista, uma vez que com o método fornece-se a ele um conjunto de casos de uso que apresentam características que permitem a sua implementação utilizando a POA. Desta forma, um projetista mais experiente poderá fazer um melhor uso das informações obtidas com a aplicação do método, em comparação com projetistas não tão experientes.

3.2 Descrição do Método Proposto

O método proposto foi construído para funcionar como uma passo da abordagem dirigida a responsabilidades PDR-DFD [41]. Neste caso, considera-se que o projetista já gerou os elementos que constituem o Modelo de Requisitos do Framework (MRF).

Um dos elementos que compõem o MRF é a Descrição Textual (DT) dos casos de uso. Nesta dissertação, a DT segue o modelo proposto por Matos e Fernandes [41]. A Tabela 3.2 apresenta um exemplo de DT.

A seguir, os dois subprocessos do método proposto serão aplicados ao domínio "Jogo de Corrida de Carros", para ilustrar a descrição do método. Os elementos que constituem o MRF deste domínio foram retirados do trabalho de Matos [40]. Alguns dos artefatos utilizados encontram-se no Apêndice A.

3.2.1 PASSOS 1-3 - Identificar e Modelar Responsabilidades Espalhadas

Este subprocesso utiliza o algoritmo ilustrado na Tabela 3.1 com a finalidade de levantar e representar as responsabilidades que encontram-se espalhadas pelos casos de uso, ou seja, estão presentes em mais de uma DT.

Tabela 3.1: Algoritmo do Subprocesso PASSOS 1-3.

Entrada
- Modelo de Requisitos do Framework (MRF)
Passos do Método
- PASSO 1: Definir o conjunto de responsabilidades do framework
- PASSO 2: Definir o conjunto de responsabilidades espalhadas
- PASSO 3: Representar as responsabilidades espalhadas
Saídas
- MRF - Responsabilidades Espalhadas
- Conjunto de Responsabilidades do Framework
- Conjunto de Responsabilidades Espalhadas

Como entrada, este subprocesso necessita dos elementos que compõem o Modelo de Requisitos do Framework (MRF). A partir da análise da Descrição Textual (DT) dos casos de uso, as responsabilidades são obtidas, PASSO 1. Em seguida, são identificadas as responsabilidades espalhadas entre os casos de uso, PASSO 2. Por fim, o Diagrama de Casos de Uso (DCU) é adaptado a fim de melhor representar as responsabilidades identificadas, PASSO 3. Como resultado deste subprocesso, tem-se o Conjunto de Responsabilidades do Framework, o Conjunto de Responsabilidades Espalhadas e o MRF onde as responsabilidades espalhadas foram identificadas e representadas. A seguir, os passos que formam este subprocesso serão detalhados e exemplificados para o domínio de "Jogo de Corrida de Carros".

PASSO 1 - Definir o conjunto de responsabilidades do framework

O objetivo neste passo é gerar as responsabilidades dos casos de uso. Elas são obtidas através da análise da DT, na parte "Responsabilidades do Sistema". Para isto, utiliza-se o critério [40]:

Responsabilidade = Verbo + Complemento

Por exemplo, a análise da DT do caso de uso "Processar Opção", ilustrada na Tabela 3.2, tem como resultado as responsabilidades apresentadas na Tabela 3.3.

Tabela 3.2: Descrição textual do caso de uso "Processar Opção" (adaptado de [40]).

Nome do Caso de Uso	Processar Opção
Estabilidade [x]	**Flexibilidade** [ ]
Ações do Usuário	**Responsabilidades do Sistema**
1. Selecionar opção	2. Verificar a opção selecionada
	3. Executar uma ação a partir da opção escolhida
4. Apresentar resultado sobre a opção escolhida	

Tabela 3.3: Responsabilidades do caso de uso "Processar Opção" (adaptado de [40]).

Verbo	**Pergunta ao verbo**	**Complemento**	**Responsabilidade**
Verificar...	O quê?	Opção	Verificar opção
Executar...	O quê?	Ação	Executar ação

Após gerar todas as responsabilidades, o projetista deve realizar um pré-processamento em busca de problemas de semântica. O problema de semântica ocorre, por exemplo, quando têm-se responsabilidades com nomes diferentes, mas semanticamente possuem a mesma finalidade [40].

A Tabela A.1 ilustra as responsabilidades obtidas por Matos [40] ao analisar a DT dos casos de uso do domínio "Jogo de Corrida de Carros".

PASSO 2 - Definir o conjunto de responsabilidades espalhadas

Neste passo, o objetivo é identificar as responsabilidades que encontram-se espalhadas pelos casos de uso, ou seja, estão presentes em mais de uma DT, bem como a quantidade de vezes em que aparece.

Por exemplo, ao analisar as responsabilidades do domínio "Jogo de Corrida de Carros", ilustradas na Tabela A.1, pode-se identificar que as responsabilidades: *Mostrar Imagens*, *Permitir Navegabilidade* e *Requisitar "Buscar Valor"* estão espalhadas por mais de um caso de uso.

Após esta identificação, o projetista deve analisar as responsabilidades em busca de inconsistências, como por exemplo, a responsabilidade *Requisitar "Buscar Valor"* que

está relacionada com a chamada para o caso de uso *Buscar Valor*, sendo assim, pode vir a ser desconsiderada pelo projetista. Com isso, tem-se como resultado deste passo as responsabilidades presentes na Tabela 3.4.

Tabela 3.4: Conjunto de responsabilidades espalhadas do domínio "Jogo de Corrida de Carros".

Responsabilidade	**Nr. de Casos de Uso**
Mostrar Imagens	10
Permitir Navegabilidade	10

Como pode ser visto na Tabela 3.4, cada uma das responsabilidades identificadas: *Mostrar Imagens* e *Permitir Navegabilidade*, encontram-se espalhadas por dez (10) casos de uso.

PASSO 3 - Representar as responsabilidades espalhadas

O objetivo deste passo é representar no DCU as responsabilidades identificadas no PASSO 2. Para isto, deve-se criar um novo caso de uso para cada uma destas responsabilidades. Estes casos de uso devem ser associados a seus casos de uso originais pelo estereótipo ≪*crosscuts*≫. Assim como em Camargo [16], este estereótipo é utilizado para indicar que o seu comportamento entrecorta o caso de uso associado. Por fim, a DT dos casos de uso afetados deve ser adaptada, ou criada, se for o caso.

Para o domínio "Jogo de Corrida de Carros", por exemplo, deve-se criar um novo caso de uso, assim como sua DT, para cada responsabilidade presente na Tabela 3.4. Os casos de uso que foram afetados por esta modificação também devem ter sua DT adaptada. Esta adaptação deve ser realizada através da substituição da parte relacionada com a responsabilidade espalhada, pela chamada ao novo caso de uso criado.

A Figura 3.2 ilustra o DCU após as alterações propostas por este passo. Nesta figura os casos de uso classificados como de "Estabilidade" e "Flexibilidade" para o domínio estão representados através da cor de fundo branco e cinza, respectivamente.

Como pode ser visto na Figura 3.2, dois novos casos de uso foram criados: *Permitir Navegabilidade* e *Mostrar Imagens*. Estes casos de uso representam as responsabilidades identificadas como espalhadas pelo PASSO 2.

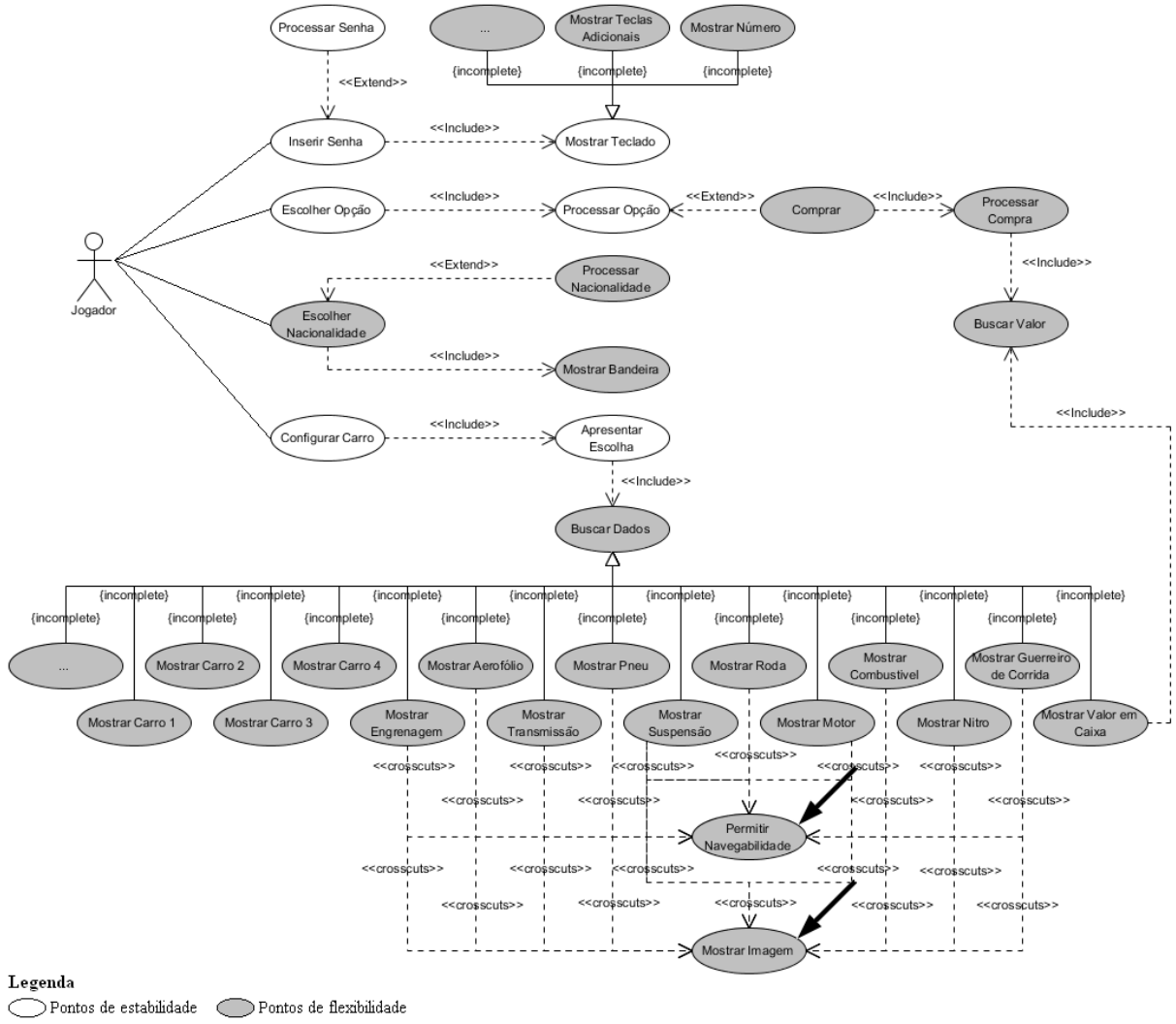

Figura 3.2: Diagrama de casos de uso para o domínio "Jogo de Corrida de Carros" - PASSOS 1-3.

3.2.2 PASSOS 4-6 - Identificar Casos de Uso Candidatos a Aspecto

Este subprocesso utiliza o algoritmo ilustrado na Tabela 3.5 com a finalidade de identificar e representar os casos de uso cujas características indicam que sua implementação com aspectos[2] é factível. Assim como Camargo [16], esta dissertação denomina estes casos de uso de "candidatos a aspecto". O termo "candidato" é utilizado para indicar que nesta fase do desenvolvimento, a análise de requisitos, o projetista ainda não pode garantir que o caso de uso poderá ser implementado utilizando a tecnologia, ficando esta decisão para a fase de projeto.

[2]O termo "aspectos" está relacionado à Programação Orientada a Aspectos.

Tabela 3.5: Algoritmo do Subprocesso PASSOS 4-6.

Entrada
- Modelo de Requisitos do Framework (MRF) - Responsabilidades Espalhadas
Passos do Método
- PASSO 4: Identificar os casos de uso colaboradores
- PASSO 5: Definir o conjunto de casos de uso candidatos a aspecto
- PASSO 6: Representar os casos de uso candidatos a aspecto
Saídas
- MRF - Candidatos a Aspecto
- Conjunto de Casos de Uso Candidatos a Aspecto

Como entrada, este subprocesso necessita dos elementos que compõem o Modelo de Requisitos do Framework (MRF). A partir da análise do Diagrama de Casos de Uso (DCU), identificam-se os casos de uso colaboradores, PASSO 4. Em seguida, com o auxílio da Descrição Textual (DT), os casos de uso candidatos a aspecto são identificados, PASSO 5. Por fim, o DCU é adaptado a fim de melhor representar os casos de uso candidatos a aspecto, PASSO 6. Como resultado deste subprocesso, têm-se o Conjunto de Casos de Uso Candidatos a Aspecto e o MRF onde os casos de uso candidatos a aspecto foram identificados e representados. A seguir, os passos que formam este subprocesso serão detalhados.

PASSO 4 - Identificar os casos de uso colaboradores

O objetivo deste passo é identificar e representar a associação entre os casos de uso base e seus colaboradores. O termo "colaborador" é utilizado para representar um caso de uso que contribui de alguma forma para a realização de um outro caso de uso, sendo este segundo, representado pelo termo "base". Na *UML*, a associação entre os casos de uso é representada pelos estereótipos «*include*» e «*extend*». Esta dissertação também utiliza o estereótipo «*crosscuts*», adicionado ao DCU no PASSO 3.

Para auxiliar o projetista durante a identificação dos casos de uso colaboradores, é proposta a criação de duas tabelas. A primeira tabela, apresenta uma matriz de associação entre os casos de uso base e seus colaboradores, por sua vez, a segunda ilustra os estereótipos utilizados nestas associações.

Por exemplo, a análise do DCU do domínio "Jogo de Corrida de Carros", ilustrado na

Figura 3.2, tem como resultado as Tabelas 3.7 e 3.8. Estas tabelas apresentam, respectivamente, uma matriz de associação entre os casos de uso base e colaboradores e o tipo de associação utilizado. A Tabela 3.6 apresenta o nome dos casos de uso ilustrados na Tabela 3.7.

Tabela 3.6: Nome dos casos de uso da Tabela 3.7.

Nr.	**Nome do Caso de Uso**	**Nr.**	**Nome do Caso de Uso**
01	Inserir Senha	02	Escolher Opção
03	Escolher Nacionalidade	04	Configurar Carro
05	Processar Senha	06	Mostrar Teclado
07	Processar Opção	08	Processar Nacionalidade
09	Mostrar Bandeira	10	Apresentar Escolha
11	Buscar Dados	12	Mostrar Teclas Adicionais
13	Mostrar Número	14	Comprar
15	Mostrar Carro 1	16	Mostrar Carro 2
17	Mostrar Carro 3	18	Mostrar Carro 4
19	Mostrar Engrenagem	20	Mostrar Aerofólio
21	Mostrar Transmissão	22	Mostrar Pneu
23	Mostrar Suspensão	24	Mostrar Roda
25	Mostrar Motor	26	Mostrar Combustível
27	Mostrar Nitro	28	Mostrar Guerreiro de Corrida
29	Mostrar Valor em Caixa	30	Processar Compra
31	Buscar Valor	32	Permitir Navegabilidade
33	Mostrar Imagem		

Tabela 3.7: Matriz de associação entre os casos de uso base e colaboradores do domínio “Jogo de Corrida de Carros”.

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
01					X	X																											
02							X																										
03								X	X																								
04										X																							
05																																	
06												X	X																				
07														X																			
08																																	
09																																	
10											X																						
11															X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
12																																	
13																																	
14																														X			
15																																	
16																																	
17																																	
18																																	
19																																X	X
20																																X	X
21																																X	X
22																																X	X
23																																X	X
24																																X	X
25																																X	X
26																																X	X
27																																X	X
28																																X	X
29																															X		
30																															X		
31																																	
32																																	
33																																	

Tabela 3.8: Tipo da associação entre os casos de uso da Tabela 3.7.

Nr.	**Caso de Uso Base**	**Nr.**	**Caso de Uso Colaborador**	**Tipo da Associação**
01	Inserir Senha	06	Mostrar Teclado	*include*
01	Inserir Senha	05	Processar Senha	*extend*
02	Escolher Opção	07	Processar Opção	*include*
03	Escolher Nacionalidade	08	Processar Nacionalidade	*extend*
03	Escolher Nacionalidade	09	Mostrar Bandeira	*include*
04	Configurar Carro	10	Apresentar Escolha	*include*
06	Mostrar Teclado	12	Mostrar Teclas Adicionais	*generalization*
06	Mostrar Teclado	13	Mostrar Número	*generalization*
07	Processar Opção	14	Comprar	*extend*
10	Apresentar Escolha	11	Buscar Dados	*include*
11	Buscar Dados	15	Mostrar Carro 1	*generalization*
11	Buscar Dados	16	Mostrar Carro 2	*generalization*
11	Buscar Dados	17	Mostrar Carro 3	*generalization*
11	Buscar Dados	18	Mostrar Carro 4	*generalization*
11	Buscar Dados	19	Mostrar Engrenagem	*generalization*
11	Buscar Dados	20	Mostrar Aerofólio	*generalization*
11	Buscar Dados	21	Mostrar Transmissão	*generalization*
11	Buscar Dados	22	Mostrar Pneu	*generalization*
11	Buscar Dados	23	Mostrar Suspensão	*generalization*
11	Buscar Dados	24	Mostrar Roda	*generalization*
11	Buscar Dados	25	Mostrar Motor	*generalization*
11	Buscar Dados	26	Mostrar Combustível	*generalization*
11	Buscar Dados	27	Mostrar Nitro	*generalization*
11	Buscar Dados	28	Mostrar Guerreiro de Corrida	*generalization*
11	Buscar Dados	29	Mostrar Valor em Caixa	*generalization*
14	Comprar	30	Processar Compra	*include*
30	Processar Compra	31	Buscar Valor	*include*
29	Mostrar Valor em Caixa	31	Buscar Valor	*include*
19	Mostrar Engrenagem	32	Permitir Navegabilidade	*crosscuts*
20	Mostrar Aerofólio	32	Permitir Navegabilidade	*crosscuts*
21	Mostrar Transmissão	32	Permitir Navegabilidade	*crosscuts*
22	Mostrar Pneu	32	Permitir Navegabilidade	*crosscuts*
23	Mostrar Suspensão	32	Permitir Navegabilidade	*crosscuts*
24	Mostrar Roda	32	Permitir Navegabilidade	*crosscuts*
25	Mostrar Motor	32	Permitir Navegabilidade	*crosscuts*
26	Mostrar Combustível	32	Permitir Navegabilidade	*crosscuts*
27	Mostrar Nitro	32	Permitir Navegabilidade	*crosscuts*
28	Mostrar Guerreiro de Corrida	32	Permitir Navegabilidade	*crosscuts*
19	Mostrar Engrenagem	33	Mostrar Imagem	*crosscuts*
20	Mostrar Aerofólio	33	Mostrar Imagem	*crosscuts*
21	Mostrar Transmissão	33	Mostrar Imagem	*crosscuts*
22	Mostrar Pneu	33	Mostrar Imagem	*crosscuts*
23	Mostrar Suspensão	33	Mostrar Imagem	*crosscuts*
24	Mostrar Roda	33	Mostrar Imagem	*crosscuts*
25	Mostrar Motor	33	Mostrar Imagem	*crosscuts*
26	Mostrar Combustível	33	Mostrar Imagem	*crosscuts*
27	Mostrar Nitro	33	Mostrar Imagem	*crosscuts*
28	Mostrar Guerreiro de Corrida	33	Mostrar Imagem	*crosscuts*

PASSO 5 - Definir o conjunto de casos de uso candidatos a aspecto

Este passo tem por objetivo identificar os casos de uso colaboradores candidatos a

aspecto. Para isto, é proposta a utilização dos quatro critérios descritos a seguir [61, 16]:

1. Estar associado a mais de um caso de uso. Em analogia ao problema de espalhamento, em que possivelmente, a implementação deste caso de uso possuirá trechos de código, relacionados a sua funcionalidade, espalhados por vários locais do framework [16];

2. Ser um caso de uso do tipo não-funcional. Os casos de uso deste tipo, em geral, são implementados como aspecto [16];

3. Estar associado a um caso de uso através do estereótipo «*extend*». Este tipo de associação geralmente inclui funcionalidades extras ao comportamento do caso de uso base [16];

4. Alterar o fluxo de execução do caso de uso base, porém, não estar diretamente relacionado com o seu objetivo, permitindo assim, que a sua remoção não impeça a sua realização [61, 16].

Ressalta-se que estes critérios são independentes, ou seja, basta que um seja satisfeito para que o caso de uso seja classificado como candidato a aspecto [16]. Após realizar esta avaliação, os casos de uso colaboradores que se enquadram nos critérios propostos, passam a ser classificados como candidatos a aspecto.

Para auxiliar na avaliação dos casos de uso colaboradores, o projetista deve utilizar as Tabelas 3.7 e 3.8. Por exemplo, com o auxilio da Tabela 3.7, pode-se identificar que o caso de uso colaborador *Buscar Valor* esta associado a dois outros casos de uso: *Processar Compra* e *Mostrar Valor em Caixa*, sendo assim é classificado como candidato a aspecto pelo 1ºcritério. Por sua vez, com o auxilio da Tabela 3.8, pode-se identificar que o caso de uso colaborador *Processar Senha* esta associado ao caso de uso *Inserir Senha* pelo estereótipo «*extend*», sendo assim também é classificado como candidato a aspecto pelo 3ºcritério.

Por sua vez, o 2ºcritério não foi utilizado neste domínio pelo fato do mesmo não possuir casos de uso não-funcionais claramente representados entre seus artefatos. Um exemplo

de caso de uso não-funcional pode ser o caso de uso “Processar Senha” com relação ao requisito “Autenticação”, classificado como não-funcional por Camargo [16].

Para o domínio “Jogo de Corrida de Carros”, por exemplo, tem-se que os casos de uso ilustrados na Tabela 3.9 são candidatos a aspecto.

Tabela 3.9: Casos de uso candidatos a aspecto do domínio “Jogo de Corrida de Carros”.

Nome do Caso de Uso	**Critério Utilizado**
Buscar Valor	1º
Mostrar Imagens	1º
Permitir Navegabilidade	1º
Processar Senha	3º
Comprar	3º
Processar Nacionalidade	3º
Mostrar Teclado	4º
Mostrar Bandeira	4º

PASSO 6 - Representar os casos de uso candidatos a aspecto

O objetivo deste passo é representar no DCU e na DT os casos de uso classificados como candidatos a aspecto no PASSO 5. A fim de representar esta classificação, é proposta uma adaptação a DT descrita em [41], como ilustrado na Tabela 3.10, onde foi incluída a informação “Candidato a Aspecto”.

Tabela 3.10: Descrição textual proposta.

Nome do Caso de Uso	...
Estabilidade ☐ Flexibilidade ☐ Candidato a Aspecto ☐	
Ações do Usuário	**Responsabilidades do Sistema**
1. ...	2. ...
3. ...	4. ...

Para o domínio “Jogo de Corrida de Carros”, por exemplo, a Figura 3.3 ilustra o DCU com os casos de uso classificados como candidatos a aspecto (ver Tabela 3.9) destacados pelo estereótipo ≪CA≫.

O estereótipo ≪CA≫ é utilizado por este trabalho em analogia ao trabalho Camargo [16]. Em seu trabalho, Camargo [16] utiliza o estereótipo ≪NF≫ para representar os casos de uso classificados como não-funcionais.

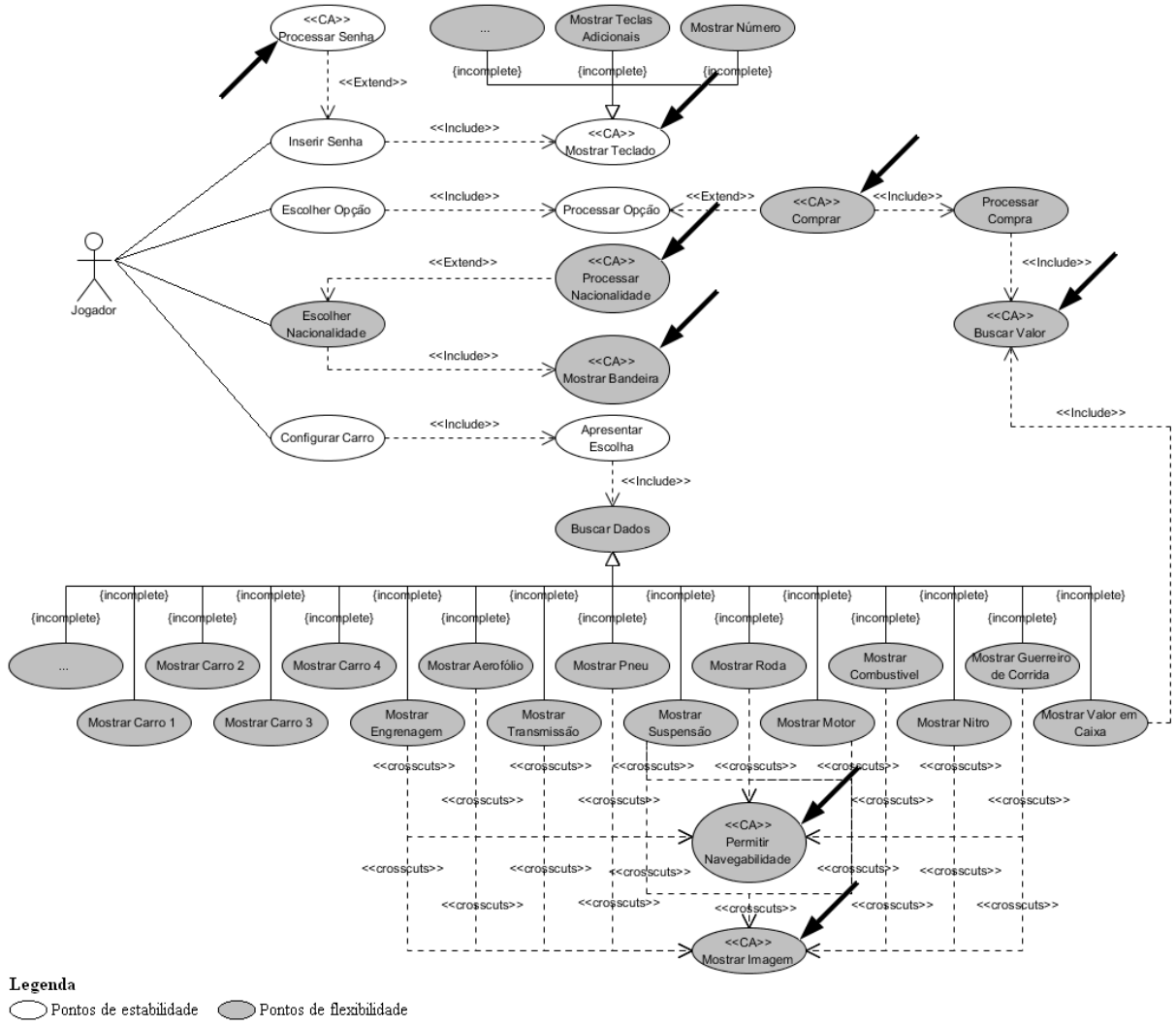

Figura 3.3: Diagrama de casos de uso para o domínio "Jogo de Corrida de Carros" - PASSOS 4-6.

3.3 Contexto de Aplicação do Método Proposto

Com já citado, o método proposto utiliza como base a abordagem PDR-DFD (Processo Dirigido a Responsabilidades aplicado ao Desenvolvimento de Framework de Domínio) [41]. No contexto desta abordagem, o método proposto deve ser utilizado após a atividade "Refinar Requisitos" da subfase "Definir Framework Base e de Aplicação" (ver Figura 2.7). Ao fim desta atividade, "Refinar Requisitos", o projetista tem os artefatos que compõem o Modelo de Requisitos do Framework (MRF), entrada para o método.

O método proposto nesta dissertação cobre apenas a fase de análise de requisitos. Com relação as demais fases do desenvolvimento, como a fase de projeto por exemplo, o projetista pode utilizar os critérios apresentados no trabalho de Camargo [16], a fim

de identificar quais casos de uso classificados como candidatos a aspecto serão modelados utilizando a tecnologia de aspectos. A seguir, os critérios propostos por Camargo [16] são brevemente apresentados:

- **Não-funcionais clássico** - verifica se o caso de uso candidato a aspecto é um exemplo clássico de requisito não-funcional. Em seu trabalho, Camargo [16] apresenta uma lista de requisitos não-funcionais clássicos;
- **Atomicidade** - compara os casos de uso para identificar se o compartilhamento é fruto de um requisito transversal ou um requisito mal redigido;
- **Volatilidade** - verifica se o caso de uso candidato a aspecto envolve um requisito volátil, ou seja, que possui uma alta probabilidade de mudança;
- **Framework transversal** - deve-se verificar se existe um framework transversal disponível que possa ser utilizado para a implementação do caso de uso candidato a aspecto.

3.4 Conclusões

Este capítulo apresentou um método para identificação antecipada de candidatos a aspecto durante o desenvolvimento de frameworks de domínio, que é o objetivo principal desta dissertação. A Seção 3.1 apresentou uma visão geral do método proposto. Na Seção 3.2 o método proposto foi apresentado. A Seção 3.3 apresentou o contexto de aplicação do método proposto.

Utilizando a abordagem PDR-DFD como base, o método proposto possui dois subprocessos. O primeiro, com o objetivo de identificar e representar as responsabilidades que se encontram espalhadas pelos casos de uso. Por sua vez, o segundo visa a identificar e representar os casos de uso que apresentam características que indicam que sua implementação com aspectos é factível.

No próximo capítulo é apresentada a ferramenta de apoio ao método proposto neste trabalho.

CAPÍTULO 4

FERRAMENTA DE APOIO

Neste capítulo apresenta-se a implementação do método proposto descrito no capítulo anterior. Na Seção 4.1 apresenta-se uma visão geral do protótipo proposto, relatando-se as linguagens e ferramentas utilizadas para seu desenvolvimento. Na Seção 4.2 descreve-se brevemente as funcionalidades gráficas do protótipo proposto.

4.1 Visão Geral do Protótipo Proposto

O processo geral de funcionamento do protótipo proposto está ilustrado na Figura 4.1. Como entrada, o protótipo utiliza os artefatos gerados pela abordagem PDR-DFD (Processo Dirigido a Responsabilidades aplicado ao Desenvolvimento de Framework de Domínio), proposta por Matos e Fernandes [41]. Como saída, o protótipo disponibiliza os casos de uso classificados como candidatos a aspecto, sendo estes, utilizados pelo projetista para a criação do diagrama de casos de uso do domínio.

Na Figura 4.1 as atividades relacionadas com a abordagem PDR-DFD estão destacadas por uma linha cinza, por sua vez, as atividades do protótipo proposto estão destacados por uma linha preta.

Este protótipo foi desenvolvido por meio da linguagem Java, o framework de integração Spring[1]. A interface gráfica foi desenvolvida com o framework *Ext-JS*[2].

4.2 Protótipo Proposto

Como já citado, o protótipo proposto utiliza-se dos artefatos gerados pela ferramenta da abordagem PDR-DFD. Esta ferramenta (abordagem PDR-DFD), disponibiliza o nome

[1]Spring é um projeto da SpringSource Community. Disponível em: *http://springsource.org/*, julho 2010.

[2]*Ext-JS* é um framework para desenvolvimento de aplicações web. Disponível em: *http://www.sencha.com/products/js/*, julho 2010.

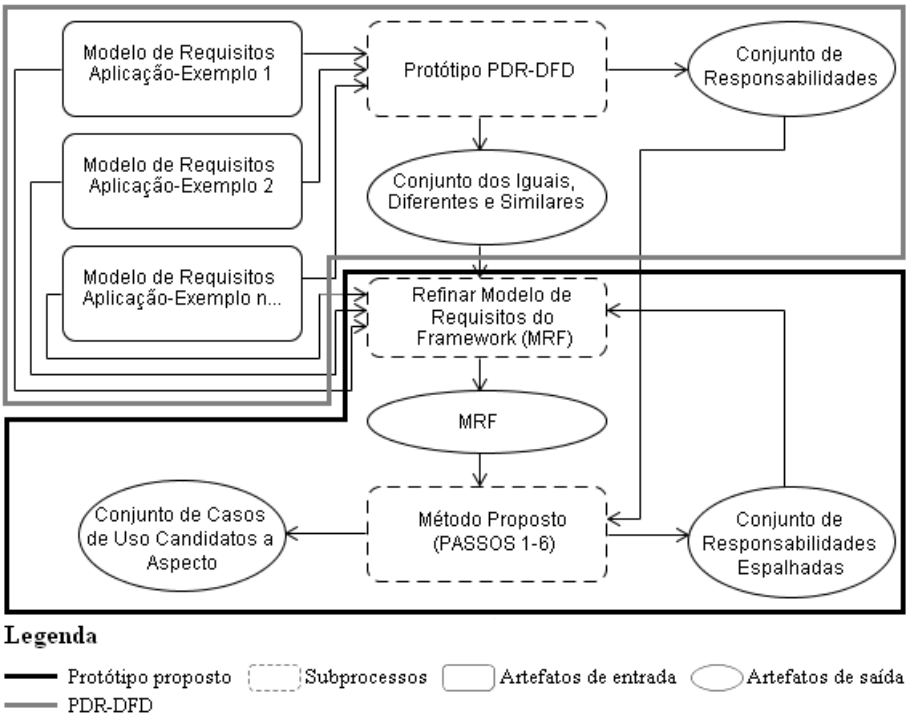

Figura 4.1: Processo geral do protótipo proposto.

dos casos de uso do framework, suas associações, sua descrição textual e responsabilidades.

O protótipo disponibiliza duas abas principais: "Refinar Modelo de Requisitos do Framework (MRF)" e "Identificar Casos de uso Candidatos a Aspecto". A primeira aba tem por objetivo auxiliar o projetista durante a atividade de refinamento do MRF (ver Seção 4.2.1). Por sua vez, a segunda objetiva a identificação dos casos de uso candidatos a aspecto (ver Seção 4.2.2).

Afim de facilitar a compreensão do funcionamento do protótipo proposto, os artefatos do domínio "Jogo de Corrida de Carros" foram utilizados para sua ilustração.

4.2.1 Refinar o Modelo de Requisitos do Framework (MRF)

A ferramenta disponibilizada pela abordagem PDR-DFD classifica os casos de uso como: "Igual", "Diferente" ou "Similar", ficando a cargo do projetista o refinamento destes casos de uso. Este refinamento tem por objetivo identificar os casos de uso de estabilidade e de flexibilidade do domínio em desenvolvimento. Os critérios que o projetista pode utilizar para a atividade de refinamento dos casos de uso podem ser encontrados em [40].

Segundo a abordagem PDR-DFD, os casos de uso classificados como "Igual" e "Di-

ferente", podem ser automaticamente classificados como de "Estabilidade" e "Flexibilidade", respectivamente. Por sua vez, os casos de uso classificados como "Similar" devem ser refinados e classificados. A Figura 4.2 ilustra a tela onde o projetista pode realizar este processo de refinamento. Esta tela apresenta todos os casos de uso do framework que foram classificados pela ferramenta da abordagem PDR-DFD.

Figura 4.2: Tela que apresenta os casos de uso classificados pela abordagem PDR-DFD.

Para o processo de refinamento dos casos de uso, o protótipo disponibiliza ao projetista uma interface para criação, ou alteração, dos casos de uso do domínio, ilustrada nas Figuras 4.3, 4.4 e 4.5.

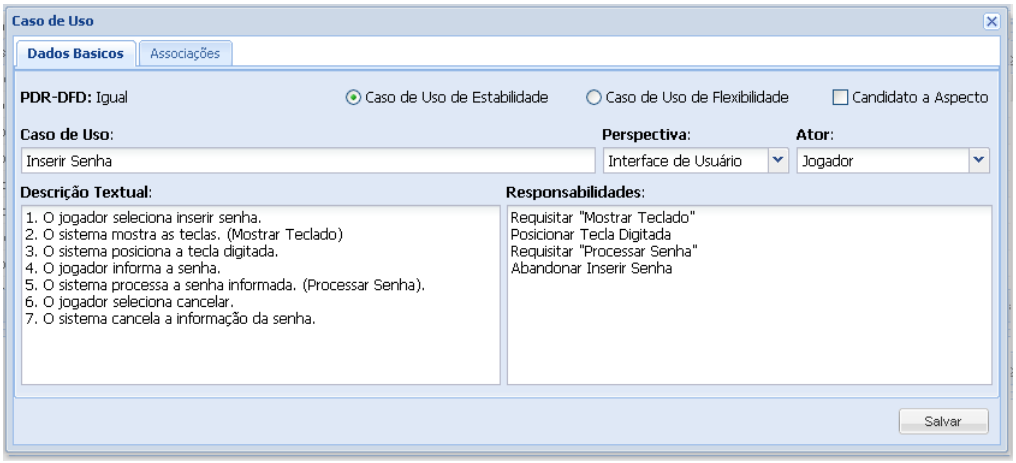

Figura 4.3: Tela que apresenta os dados básicos do caso de uso.

Após realizar o refinamento dos casos de uso, o projetista deve adaptar o diagrama de casos de uso do domínio a fim de representar as alterações. Esta atividade é necessária pois este artefato será utilizado nas etapas seguintes do método proposto.

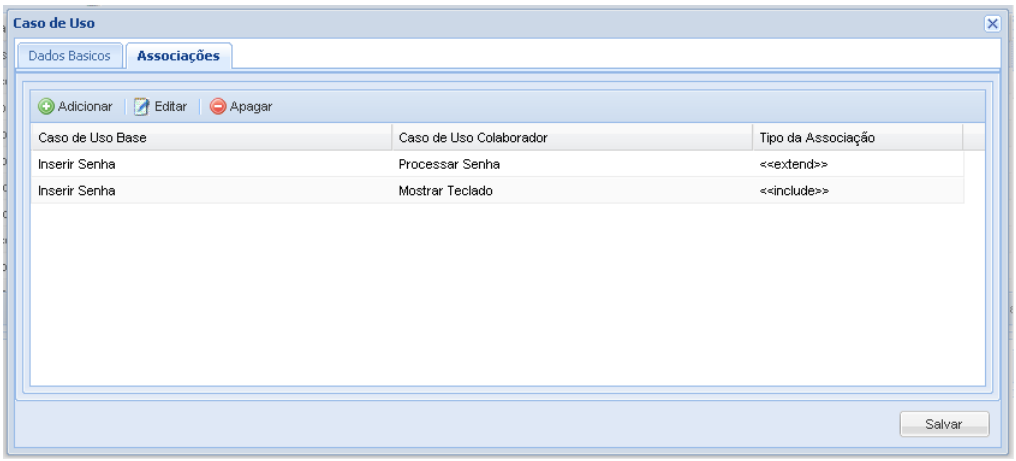

Figura 4.4: Tela que apresenta as associações do caso de uso.

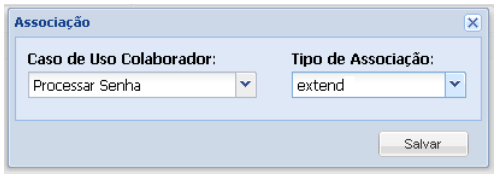

Figura 4.5: Cadastro de associação entre os casos de uso.

4.2.2 Método Proposto (PASSOS 1-6)

Após o refinamento dos casos de uso, o projetista deve identificar os casos de uso classificados como candidatos a aspecto. Para isto, o protótipo disponibiliza duas atividades: "Responsabilidades Espalhadas" e "Casos de Uso Colaboradores". Estas atividades estão representadas no protótipo através de abas, como ilustra a Figura 4.6.

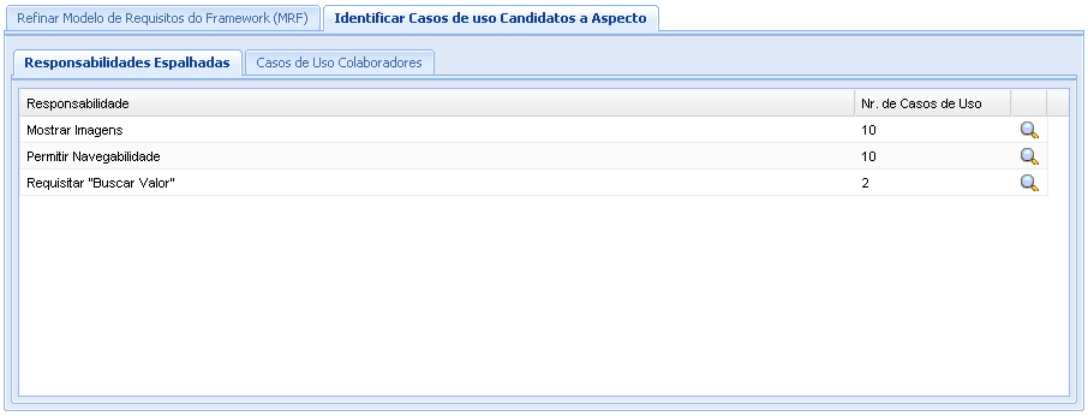

Figura 4.6: Tela para aplicação do método proposto.

Ao entrar na aba "Responsabilidades Espalhadas", o protótipo disponibiliza as responsabilidades que encontram-se em mais de um caso de uso. Atividade esta realizada pelo

protótipo através de uma comparação aos pares entre as responsabilidades do domínio.

Após avaliar as responsabilidades ditas como espalhadas, o projetista deve retornar a aba "Refinar Modelo de Requisitos do Framework (MRF)" e criar um novo caso de uso para cada uma destas responsabilidades, assim como, adaptar os casos de uso afetados, PASSOS 1-2 do método proposto. Ao fim, o projetista deve realizar as adaptação necessárias ao diagrama de casos de uso do domínio, PASSO 3 do método proposto.

Na aba "Casos de Uso Colaboradores" (ver Figura 4.7), o protótipo disponibiliza os casos de uso colaboradores. Estes casos de uso são identificados com o auxílio do relacionamento entre os casos de uso (ver Figura 4.4).

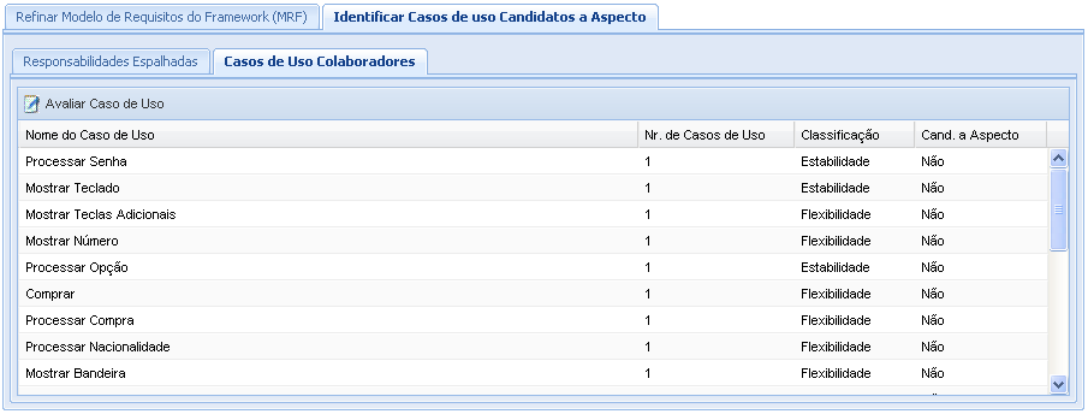

Figura 4.7: Tela que apresenta casos de uso colaboradores.

Após a identificação dos casos de uso colaboradores, o projetista deve avaliar estes casos de uso individualmente, e se assim os considerar, classificá-los como candidatos a aspecto. A Figura 4.8 ilustra as telas de avaliação dos casos de uso.

Após identificar os casos de uso candidatos a aspecto, o projetista deve adaptar o diagrama de casos de uso do domínio a fim de representar esta classificação.

4.3 Conclusões

Neste capítulo apresentou-se a implementação de uma ferramenta computacional para apoiar a aplicação do método proposto por meio de um protótipo. A Seção 4.1 apresentou uma visão geral deste protótipo, relatando-se as linguagens e ferramentas utilizadas para seu desenvolvimento. Na Seção 4.2, as funcionalidades do protótipo foram apresentadas.

Figura 4.8: Telas de avaliação dos casos de uso colaboradores.

No próximo capítulo é apresentado um estudo de caso para a análise e avaliação do método proposto. Para isto, foi escolhido o domínio “Controle de Finanças Pessoais”.

CAPÍTULO 5

ESTUDO DE CASO - DOMÍNIO DE CONTROLE DE FINANÇAS PESSOAIS

Neste capítulo apresenta-se a aplicação do método proposto no domínio "Controle de Finanças Pessoais". A Seção 5.1 apresenta os artefatos obtidos com a aplicação da abordagem PDR-DFD (Processo Dirigido a Responsabilidades aplicado ao Desenvolvimento de Framework de Domínio), proposta por Matos e Fernandes [41]. Estes artefatos são utilizados como entrada para o método proposto, sendo sua aplicação, descrita na Seção 5.2. A Seção 5.3 apresenta os resultados obtidos com a aplicação do método proposto no domínio "Controle de Finanças Pessoais". Por fim, a Seção 5.4 apresenta uma avaliação do método proposto.

5.1 Aplicação da Abordagem PDR-DFD

A abordagem PDR-DFD tem por objetivo auxiliar o projetista durante o desenvolvimento de um framework de domínio. Composta por cinco fases, descritas na Seção 2.1.2.5, esta abordagem disponibiliza uma ferramenta para auxiliar o projetista durante a identificação dos pontos de estabilidade e flexibilidade.

A aplicação da abordagem PDR-DFD [41] no domínio "Controle de Finanças Pessoais", tem por objetivo a obtenção dos artefatos que são utilizados como entrada para o método proposto. Como o método proposto cobre apenas a etapa de análise de requisitos, apenas as atividades da abordagem PDR-DFD relacionadas a esta etapa foram aplicadas.

Para o domínio "Controle de Finanças Pessoais" as aplicações-exemplo escolhidas foram: "*Buddi*", "GFP" e "*jGnash*". Durante a análise destas aplicações diversos subsistemas foram identificados, no entanto, apenas alguns serão apresentados. Pelo fato destas aplicações não possuírem em sua documentação os requisitos do sistema. Os mesmos foram obtidos por este trabalho através da utilização dos softwares disponibilizados.

Com isso, pode-se levantar alguns requisitos que compõem estas aplicações-exemplo, no entanto, os requisitos não-funcionais não puderam ser identificados.

5.1.1 Aplicação-Exemplo *Buddi*

"... *Buddi* é um programa de finanças pessoais e orçamento, tendo em vista aqueles que têm pouco ou nenhum fundo financeiro. Ao fazer este software, eu tenho tentado fazer as coisas do modo mais simples possível, mantendo as funções de forma a satisfazer a maioria dos usuários domésticos. ..."[1].

Desenvolvida utilizando a linguagem Java, esta ferramenta é distribuída sobre a licença *GNU General Public License - Version 2*.

A Figura 5.1 ilustra o diagrama de casos de uso para o subsistema "Transação" desta aplicação-exemplo. Os demais artefatos desta aplicação-exemplo, encontram-se no Apêndice B.

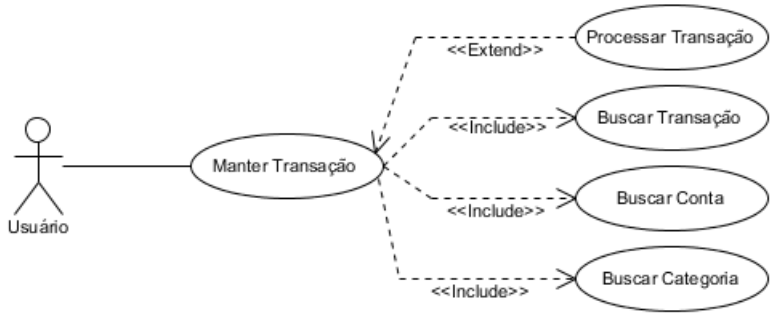

Figura 5.1: Diagrama de casos de uso para o subsistema "Transação" da aplicação-exemplo *Buddi*.

5.1.2 Aplicação-Exemplo GFP

"... GFP - Gerenciador de Finanças Pessoais - é um software que auxilia na gestão das finanças de uma pessoa fornecendo uma gama de relatórios ou filtros em tela de onde é possível extrair informações importantes para a boa administração de seu rico dinheirinho. ..."[2].

[1]Tradução livre do texto retirado do site principal da ferramenta. Disponível em: *http://buddi.digitalcave.ca/*, julho 2010.

[2]Texto retirado do site principal da ferramenta. Disponível em: *http://gfd.sourceforge.net/*, julho 2010.

A aplicação GFP é *open source* e foi desenvolvida utilizando a linguagem Java.

A Figura 5.2 ilustra o diagrama de casos de uso do subsistema "Transação". Os demais artefatos desta aplicação-exemplo, encontram-se no Apêndice C.

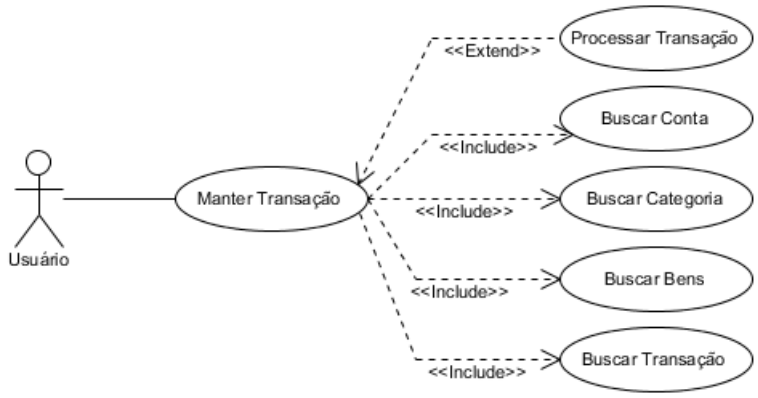

Figura 5.2: Diagrama de casos de uso para o subsistema "Transação" da aplicação-exemplo GFP.

5.1.3 Aplicação-Exemplo *jGnash*

"... *jGnash* é um gerenciador gratuito de finanças pessoais com muitos dos mesmos recursos presentes em softwares disponíveis no mercado. Ela foi criada para tornar o controle de suas finanças pessoais fácil, mas também fornece a funcionalidade necessária para usuários avançados. ..."[3].

Desenvolvida utilizando a linguagem Java, esta ferramenta é distribuída sobre a licença *GNU General Public License - Version 3*.

A Figura 5.3 ilustra o diagrama de casos de uso do subsistema "Transação". Os demais artefatos desta aplicação-exemplo, encontram-se no Apêndice D.

5.1.4 Artefatos do Modelo de Requisitos do Framework

Com o auxílio dos artefatos das aplicações-exemplo, deve-se criar o Modelo de Requisitos do Framework (MRF). O MRF é constituído por um ou mais Diagramas de Casos de Uso (DCU), notação *UML-F* [25], e sua Descrição Textual (DT).

[3]Tradução livre do texto retirado do site principal da ferramenta. Disponível em: *http://sourceforge.net/apps/mediawiki/jgnash/*, julho 2010.

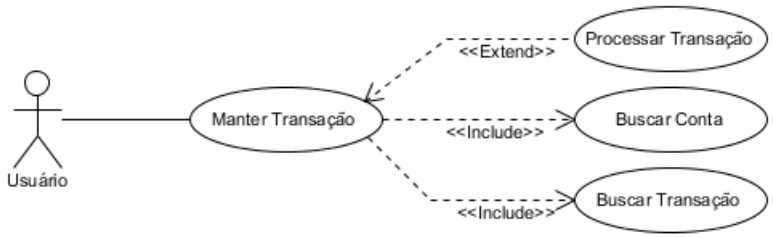

Figura 5.3: Diagrama de casos de uso para o subsistema "Transação" da aplicação-exemplo *jGnash*.

Os artefatos gerados com o auxílio da abordagem PDR-DFD para o domínio "Controle de Finanças Pessoais" encontram-se ilustrados no Apêndice E.

5.2 Aplicação do Método Proposto

O método proposto tem por objetivo identificar e representar os casos de uso que apresentam características que indiquem que sua implementação com aspectos é viável. Esta identificação é realizada através de dois subprocessos, PASSOS 1-3 e PASSOS 4-6. As atividades descritas a seguir foram realizadas com o apoio do protótipo proposto nesta dissertação.

5.2.1 PASSOS 1-3

Como entrada deste subprocesso são utilizados dois artefatos que compõem o MRF do domínio "Controle de Finanças Pessoais". O DCU (ver Figura E.1) e as responsabilidades do domínio (ver Tabela E.1).

O PASSO 1 do método proposto tem por objetivo gerar o conjunto de responsabilidades do domínio. Este conjunto é obtido a partir da análise da DT dos casos de uso. A abordagem PDR-DFD já disponibiliza estas responsabilidades para a continuidade do processo. Na abordagem PDR-DFD, estas responsabilidades são obtidas durante a atividade de refinamento dos casos de uso identificados durante a análise das aplicações-exemplo.

PASSO 2 - Definir o conjunto de responsabilidades espalhadas

Neste passo as responsabilidades que encontram-se espalhadas pelos casos de uso de-

vem ser identificadas. Ao analisar as responsabilidades do domínio "Controle de Finanças Pessoais", presentes na Tabela E.1, pode-se verificar que as responsabilidades ilustradas na Figura 5.4 estão espalhadas por mais de um caso de uso.

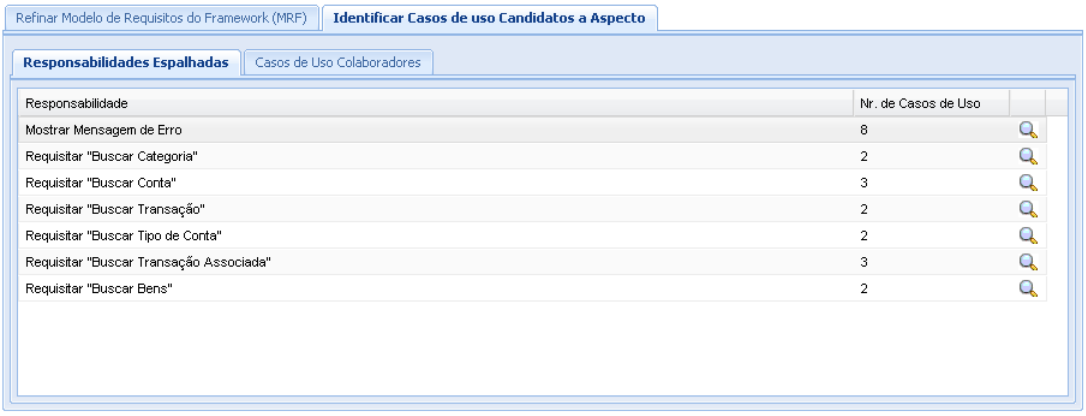

Figura 5.4: Responsabilidades espalhadas pelo domínio "Controle de Finanças Pessoais".

Após a identificação, estas responsabilidades devem ser avaliadas em busca de inconsistências, como por exemplo, as responsabilidades: *Requisitar "Buscar Categoria"*, *Requisitar "Buscar Conta"*, *Requisitar "Buscar Transação"*, *Requisitar "Buscar Tipo de Conta"*, *Requisitar "Verificar Transação Associada"* e *Requisitar "Buscar Bens"*. Estas responsabilidades encontram-se espalhadas por mais de um caso de uso, no entanto, representam chamadas para outros casos de uso, sendo assim, podem vir a ser desconsideradas.

PASSO 3 - Representar as responsabilidades espalhadas

Neste passo a responsabilidade *Mostrar Mensagem de Erro*, identificada como espalhada no passo anterior, deve ser representada no DCU do domínio. Para isso, deve-se criar um novo caso de uso que associa-se aos demais pelo estereótipo «*crosscuts*». Também deve-se criar a DT para este caso de uso, assim como, adaptar a dos casos de uso afetados.

A Figura 5.5 ilustra o DCU após as alterações realizadas neste passo. Nesta figura os casos de uso classificados como de "Estabilidade" e "Flexibilidade" para o domínio estão representados através da cor de fundo branco e cinza, respectivamente.

Como pode ser visto na Figura 5.5, o caso de uso *Exibir Erro* foi criado para para representar a responsabilidade *Mostrar Mensagem de Erro*.

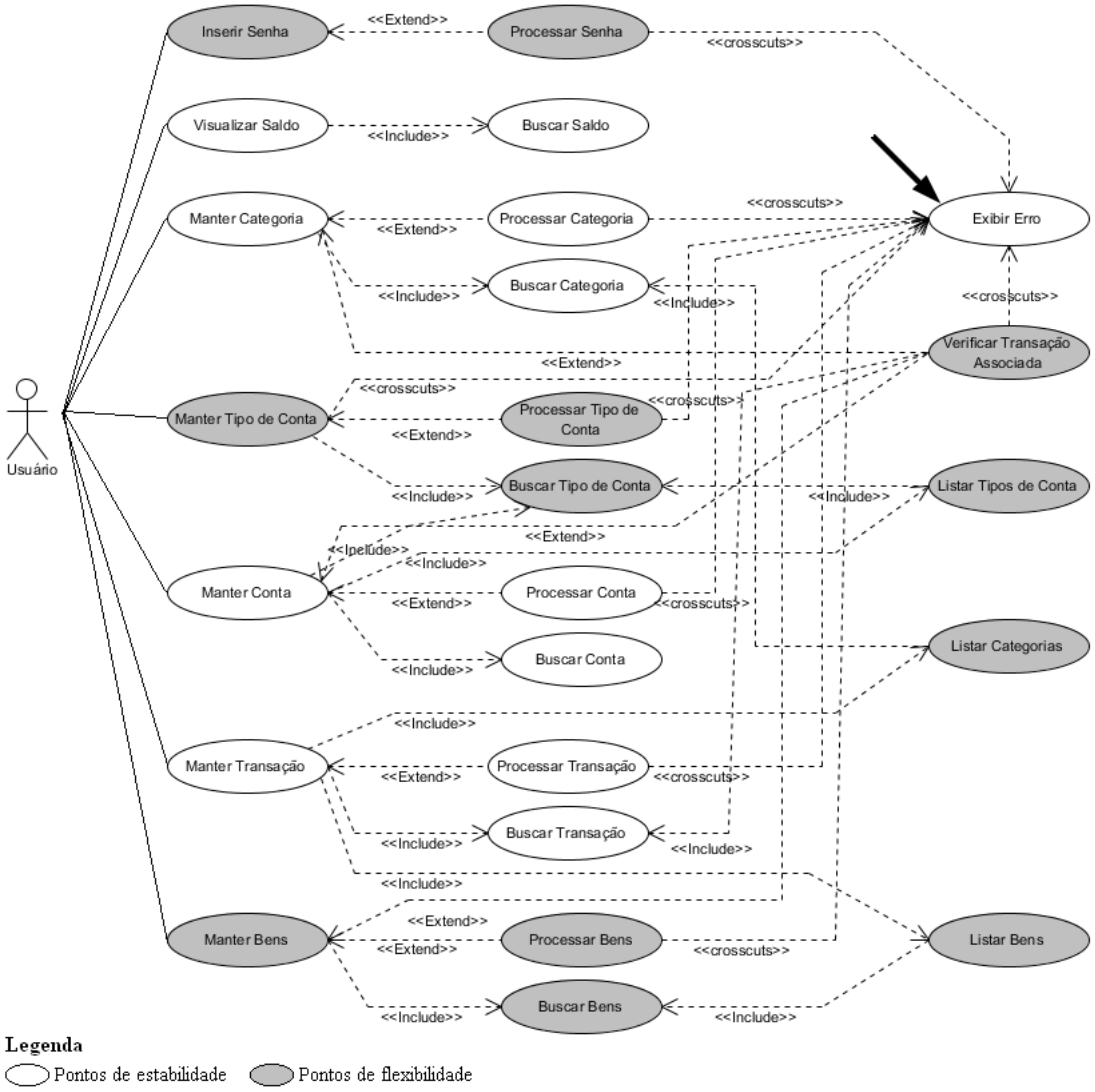

Figura 5.5: Diagrama de casos de uso para o domínio "Controle de Finanças Pessoais" - PASSOS 1-3.

5.2.2 PASSOS 4-6

Como entrada, este subprocesso utiliza dois artefatos que compõem o MRF do domínio "Controle de Finanças Pessoais". O DCU (ver Figura 5.5), onde as responsabilidades espalhadas foram identificadas e representadas, e a DT dos casos de uso.

O PASSO 4 do método proposto tem por objetivo identificar os casos de uso colaboradores do domínio. Estes casos de uso são obtidos a partir da análise das associações entre os casos de uso. O protótipo proposto já disponibiliza estes casos de uso para a continuidade do processo. A Figura 5.6 ilustra os casos de uso colaboradores do domínio "Controle de Finanças Pessoais".

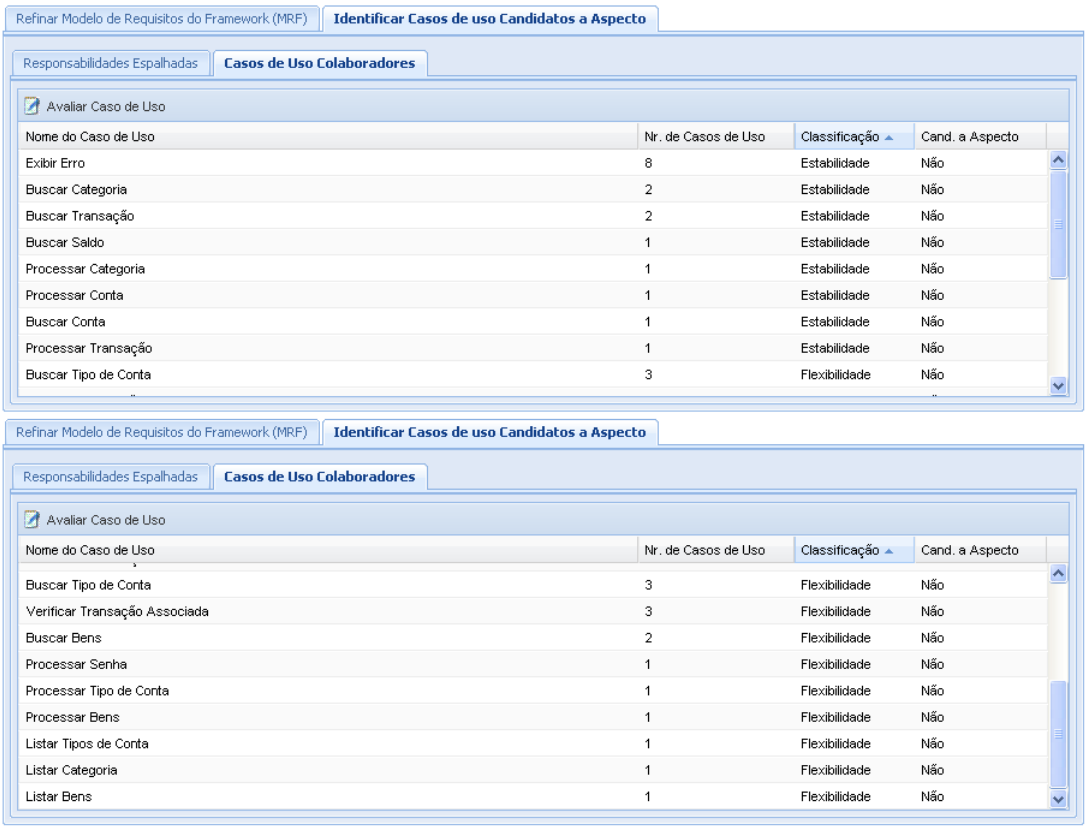

Figura 5.6: Casos de uso colaboradores do domínio "Controle de Finanças Pessoais".

PASSO 5 - Definir o conjunto de casos de uso candidatos a aspecto

Neste passo os casos de uso candidatos a aspecto devem ser identificados. Estes casos de uso são identificados a partir da avaliação dos casos de uso colaboradores (ver Figura 5.6). A Figura 5.7 ilustra a tela do protótipo proposto em que o projetista pode realizar a avaliação do caso de uso colaborador *Exibir Erro*, por exemplo.

Ao fim da avaliação dos casos de uso colaboradores, os casos de uso ilustrados na Figura 5.8 foram classificados como candidatos a aspecto para o domínio "Controle de Finanças Pessoais".

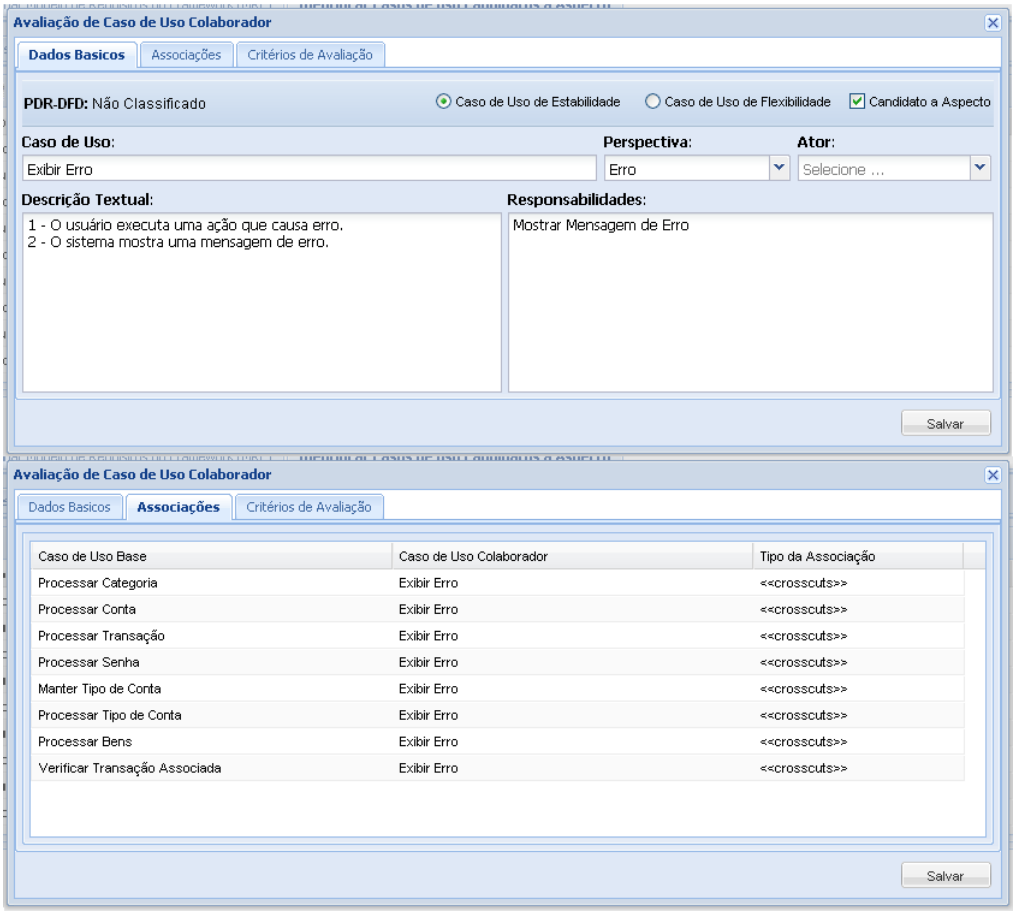

Figura 5.7: Avaliação do caso de uso colaborador *Exibir Erro*.

PASSO 6 - Representar os casos de uso candidatos a aspecto

Neste passo o DCU deve ser adaptado a fim de representar os casos de uso classificados como candidatos a aspecto (ver Figura 5.8). Esta representação deve ser realizada através do estereótipo «CA». A Figura 5.9 ilustra o DCU do domínio "Controle de Finanças Pessoais" com os casos de uso classificados como candidatos a aspecto representados.

Ao fim deste passo o projetista deve dar sequência às demais atividades da abordagem PDR-DFD, no entanto, como não fazem parte do escopo deste trabalho, estas atividades não foram realizadas.

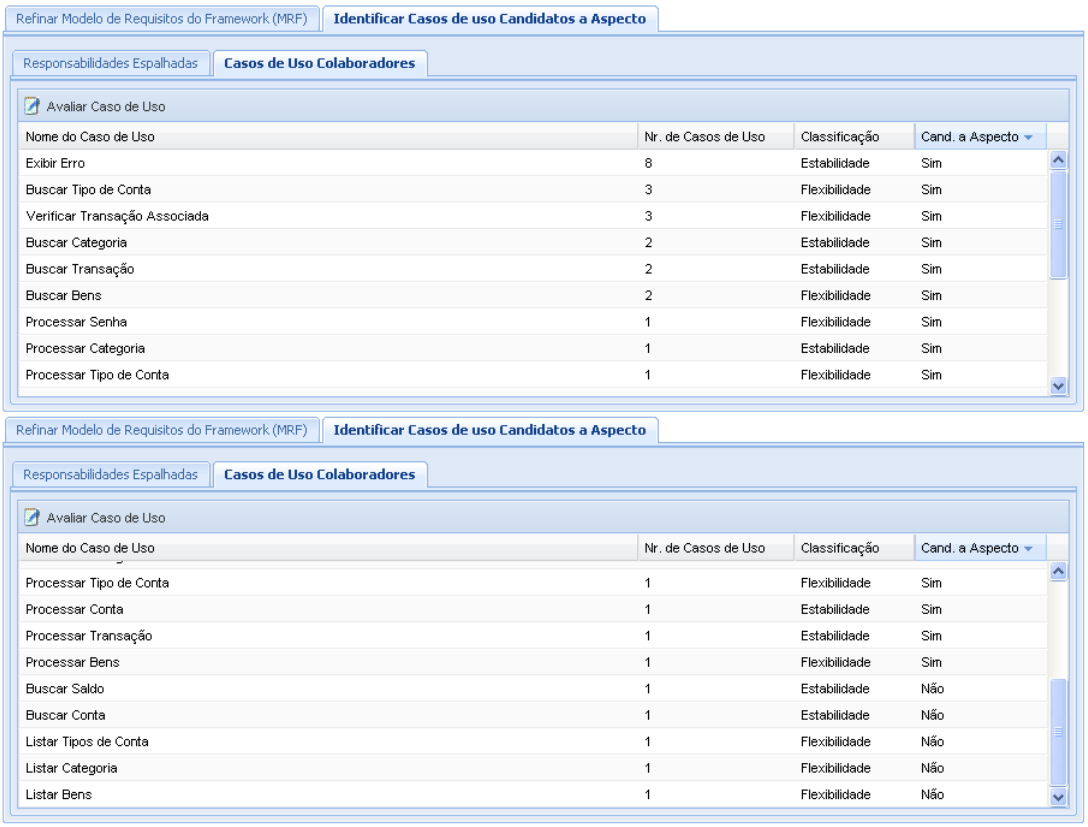

Figura 5.8: Casos de uso candidatos a aspecto do domínio "Controle de Finanças Pessoais".

5.3 Avaliação dos Resultados

Com a aplicação do método proposto no domínio "Controle de Finanças Pessoais", pode-se identificar e representar antecipadamente uma (1) responsabilidade espalhada (PASSOS 1-3) e doze (12) casos de uso candidatos a aspecto (PASSOS 4-6).

A identificação desta responsabilidade espalhada permitiu de forma antecipada identificar e representar (estereótipo «*crosscuts*») no DCU um elemento que possivelmente teria sua implementação, ou seja, seu código-fonte, espalhado e entrelaçado pelos interesses de oito (8) outros casos de uso.

Com relação aos casos de uso identificados e representados (estereótipo «CA») como candidatos a aspecto no DCU, pode-se de forma antecipada identificar os elementos afetados por estes casos de uso candidatos a aspecto, permitindo assim que o projetista possua mais informações a respeito do relacionamento entre os casos de uso para realizar o refinamento das classes do framework.

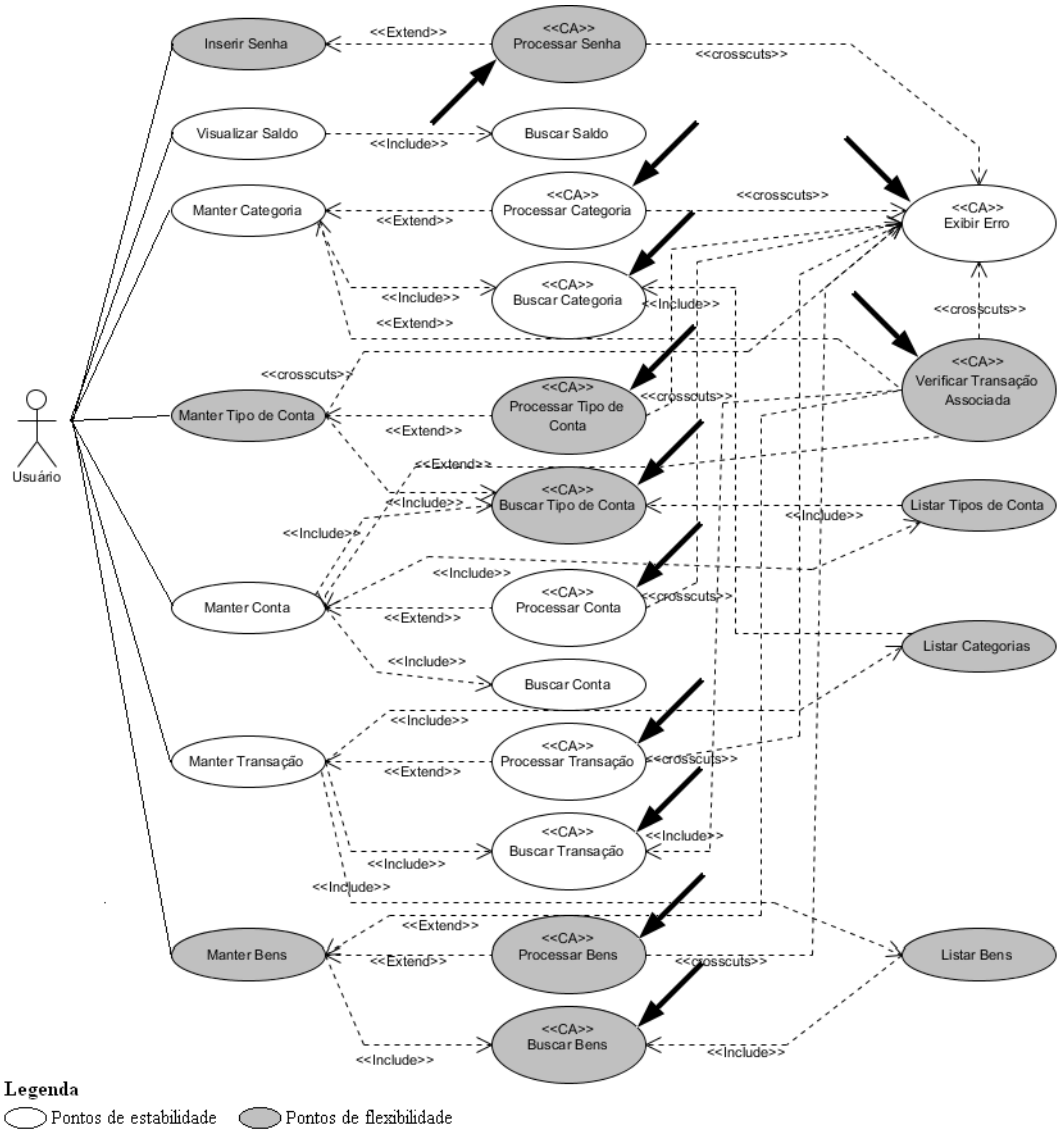

Figura 5.9: Diagrama de casos de uso para o domínio "Controle de Finanças Pessoais" - PASSOS 4-6.

Além destes elementos, vale ressaltar que outros poderiam ter sido identificados. Isto porque nem todos os subsistemas (perspectivas) do domínio foram avaliados, assim como, não foram adicionados requisitos não-funcionais ao domínio, atividade esta comumente realizada no desenvolvimento de aplicações.

5.4 Avaliação do Método Proposto

A aplicação do método proposto por esta dissertação nos domínios "Jogo de Corrida de Carros" e "Controle de Finanças Pessoais", ilustra que este método pode vir a ser utilizado com a finalidade de identificar de forma antecipada, os casos de uso que apresentam

características que indicam que sua implementação com aspectos é factível (PASSOS 4-6).

Para o domínio de "Jogo de Corrida de Carros", por exemplo, pode-se identificar antecipadamente oito (8) casos de uso candidatos a aspecto. Por sua vez, para o domínio de "Controle de Finanças Pessoais" foram identificados doze (12) casos de uso candidatos a aspecto.

Segundo Jacobson e NG [31], a identificação antecipada dos candidatos a aspecto pode refletir na arquitetura, gerando uma implementação menos acoplada, da aplicação em desenvolvimento. Esta diminuição no acoplamento ocorre devido a implementação dos interesses transversais utilizando os aspectos.

Além dos casos de uso candidatos a aspecto, o método proposto por esta dissertação, também propõe que o projetista analise o conjunto de responsabilidades dos casos de uso do domínio. Esta análise tem por objetivo identificar as responsabilidades que encontram-se espalhadas por mais de um caso de uso (PASSOS 1-3).

No domínio de "Jogo de Corrida de Carros", por exemplo, em um conjunto de sessenta e seis (66) responsabilidades, foi identificado que duas (2) destas encontravam-se espalhadas pelos casos de uso do domínio. Por sua vez, para o domínio de "Controle de Finanças Pessoais" foi identificada uma (1) responsabilidade espalhada, em um conjunto de noventa e oito (98) responsabilidades.

5.5 Conclusões

Este capítulo apresentou a aplicação do método proposto no domínio "Controle de Finanças Pessoais". Na Seção 5.1 foi descrita a aplicação da abordagem PDR-DFD com o objetivo de se obter os artefatos de entrada do método proposto. A Seção 5.2 apresentou a aplicação do método proposto nesta dissertação. Na Seção 5.3 foi apresentado os resultados obtidos com a aplicação do método proposto no domínio "Controle de Finanças Pessoais". Por fim, a Seção 5.4 apresentou uma avaliação do método proposto.

O próximo capítulo apresenta as conclusões desta dissertação, bem como, as possíveis pesquisas a serem realizadas como continuação a este trabalho.

CAPÍTULO 6

CONCLUSÕES E TRABALHOS FUTUROS

Com o avanço da Programação Orientada a Aspectos (POA) e de linguagens como AspectJ, pesquisadores passaram a aplicar seus conceitos no desenvolvimento de Frameworks Orientados a Objetos (FOO), surgindo assim, os Frameworks Orientados a Aspectos (FOA). Formado por um conjunto de unidades básicas da programação orientada a objetos e da POA, assim como um FOO, um FOA pode ser definido como um sistema semi-completo e reutilizável que pode ser instanciado por um desenvolvedor de aplicações, mantendo os interesses transversais separados dos interesses base.

No entanto, identificou-se que os trabalhos que apresentam FOA tratam, em sua maioria, de processos evolutivos, onde os aspectos são incorporados a FOO já desenvolvidos, ou de processos que visam identificar os aspectos nas etapas de projeto e implementação. Entre os trabalhos encontrados, apenas Nakajima [45], Camargo [16] e Kulesza [37] relatam a possibilidade de identificar os aspectos durante a análise de requisitos, entretanto, são apenas propostas.

Neste contexto, esta dissertação tem como principal contribuição a descrição de um método para auxiliar na identificação de candidatos a aspecto durante as atividades iniciais do processo de desenvolvimento de frameworks de domínio. Este método foi desenvolvido com base em conceitos de trabalhos que propõem o desenvolvimento de frameworks de domínio [41] e a identificação antecipada de aspectos (*early-aspects*) [61, 31, 16].

Formado por dois subprocessos, este método se aplica durante a fase de análise de requisitos. No primeiro subprocesso, são identificadas as responsabilidades espalhadas pelos requisitos. Por sua vez, no segundo subprocesso identificam-se os requisitos que apresentam características que indicam que sua implementação com aspectos é factível.

O processo de avaliação do método proposto foi ilustrado por meio da sua aplicação nos domínio de “Jogo de Corrida de Carros” e “Controle de Finanças Pessoais”. Com o

auxílio do método as responsabilidades espalhadas e os casos de uso candidatos a aspecto, puderam ser identificados durante a análise dos requisitos do domínio.

O método proposto nesta dissertação possui quatro contribuições principais. A primeira contribuição é que ele oferece um processo em que a identificação de candidatos a aspecto é realizada no modelo de requisitos. Dessa forma, o projetista não necessita previamente realizar uma análise completa das classes para conseguir classificá-las, basta apenas que utilize o resultado obtido na análise do modelo de requisitos. Além disso, como esta classificação é antecipada para as atividades iniciais do processo de desenvolvimento, tem-se uma visão antecipada dos elementos afetados pelos candidatos a aspecto identificados.

A segunda contribuição é que como a identificação dos candidatos a aspecto ocorre durante as atividades iniciais do processo de desenvolvimento de frameworks de domínio, pode-se antecipar o raciocínio sobre quais serão as unidades afetadas por um aspecto. Permitindo assim, que o entendimento de um sistema, assim como das unidades afetadas por eventuais modificações, seja realizado através dos modelos de análise e não unicamente da codificação do framework.

A terceira contribuição é que identificam-se os interesses transversais no modelo de requisitos do framework, utilizando para isso o conceito de responsabilidades de Wirfs-Brock [65]. Isto facilita o processo de identificação destes interesses permitindo inclusive a sua automação. Como esta identificação ocorre durante as etapas iniciais do processo de desenvolvimento, tem-se uma diminuição no acoplamento dos interesses do framework, contribuindo para que as atividades de implementação, manutenção e reúso sejam realizadas mais facilmente.

A quarta contribuição é que foi implementado um processo apoiado por computador para auxiliar na identificação de candidatos a aspecto. Com isso, tem-se um ganho de tempo e esforço no processo de identificação, quando comparado a forma manual de análise. Este ganho ocorre pois a quantidade de elementos que compõem o modelo de requisitos do framework pode ser grande, o que torna a identificação de forma manual um processo custoso. Além disso, os dados que compõem estes elementos ficam armazenados digitalmente para uma posterior manutenção ou refinamento.

6.1 Trabalhos Futuros

Há diversas pesquisas que podem ser realizadas como continuação ou melhoria ao método proposto. Por exemplo, aplicar o método a outros domínios, permitindo assim, que os critérios utilizados para identificar as responsabilidades espalhadas e os candidatos a aspecto sejam validados e melhorados.

Atualmente, o método cobre apenas a fase de análise de requisitos, sendo assim, torna-se importante dar continuidade para as demais fases do desenvolvimento. Também pode-se investigar a possibilidade de aplicar o conceito de aspectos para tratar dos pontos de flexibilidade do framework de domínio [57, 18, 38]. Propor critérios e técnicas que auxiliem durante a composição e o tratamento de conflito entre os casos de uso identificados como candidatos a aspecto [61, 16], é outra possibilidade de continuidade do desenvolvimento do tema.

Hoje, o método não disponibiliza de nenhum mecanismo para avaliação dos resultados obtidos. Por isso, torna-se interessante buscar formas de realizar esta atividade [10, 54, 53, 47].

Com relação ao protótipo proposto, há diversos pontos que podem ser melhorados ou implementados, no entanto, o principal deles é disponibilizar uma única ferramenta que auxilie o projetista na obtenção antecipada dos pontos de estabilidade, flexibilidade e candidatos a aspecto. Atualmente, esta atividade é realizada em duas aplicações distintas, o protótipo proposto por este trabalho, e o apresentado pela abordagem PDR-DFD (Processo Dirigido a Responsabilidades aplicado ao Desenvolvimento de Framework de Domínio) [41].

A melhora da interface com o usuário do protótipo é um outro ponto a ser considerado, visto que neste trabalho procurou-se mostrar apenas que o método proposto poderia ser implementado. Não preocupando-se com questões de usabilidade, por exemplo.

Como o método proposto possui um processo apoiado por computador, onde têm-se disponível as informações do modelo de requisitos das aplicações-exemplo e do framework, é interessante avaliar a possibilidade de exportar estes dados para ambientes de modelagem de casos de uso nas notações UML e UML-F.

BIBLIOGRAFIA

[1] S. Ambler. *The object primer: the application developers guide to objected orientation and the UML.* Cambridge University Press, 2001.

[2] J. Araújo, A. Moreira, I. Brito, e A. Rashid. Aspect-oriented requirements with UML. *Workshop on Aspect-Oriented Modeling with UML*, 7, 2002.

[3] E. Baniassad e S. Clarke. *Aspect-Oriented Analysis and Design: The Theme Approach.* Addison-Wesley, 2005.

[4] L. Bass, P. Clements, e R. Kazman. *Software Architecture in Practice.* New York: Addison-Wesley, 2003.

[5] L. Bergmans e M. Aksit. Composing crosscutting concerns using composition filters. *Communications of the ACM*, 44(10):51–57, 2001.

[6] G. Booch, J. Rumbaugh, e I. Jacobson. *The Unified Software Development Process.* Addison Wesley Longman, 1998.

[7] R.T.V. Braga. *Um processo para construção e instanciação de frameworks baseados em uma linguagem de padrões para um domínio específico.* Tese de Doutorado, ICMC/USP, 2003.

[8] I. Brito e A. Moreira. Towards a composition process for aspect-oriented requirements. *Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, March*, 2003.

[9] T. Budd e R.D. Design. *Introduction to Object Oriented Programming 2E.* Addison-Wesley, 1997.

[10] M. Ceccato e P. Tonella. Measuring the effects of software aspectization. *Workshop on Aspect Reverse Engineering*, 2004.

[11] R. Chitchyan, A. Rashid, e P. Sawyer. Comparing requirements engineering approaches for handling crosscutting concerns. *Workshop on Requirements Engineering (held with CAiSE), Porto, Portugal, June 12*, 14, 2005.

[12] R. Chitchyan, A. Rashid, P. Sawyer, A. Garcia, M.P. Alarcon, J. Bakker, B. Tekinerdogan, S. Clarke, e A. Jackson. Survey of aspect-oriented analysis and design approaches. *AOSD-Europe-ULANC-9, AOSD-EUROPE network of excellence*, 2005.

[13] C.F.M. Couto. Um arcabouço orientado por aspectos para implementação automatizada de persistência. Dissertação de Mestrado, UFMG, 2006.

[14] K. Czarnecki. Overview of generative software development. *Unconventional Programming Paradigms*, páginas 326–341, 2004.

[15] K. Czarnecki e U.W. Eisenecker. Generative Programming: Methods, Tools, and Applications. *Boston: Addison Wesley*, 26:832, 2000.

[16] V.V. de Camargo. *Frameworks transversais: definições, classificações, arquitetura e utilização em um processo de desenvolvimento de software*. Tese de Doutorado, USP - São Carlos, agosto de 2006.

[17] V.V. de Camargo e P. Masiero. Frameworks Orientados a Aspectos. *Anais do 19º Simpósio Brasileiro de Engenharia de Software (SBES2005)*, páginas 200–215, 2005.

[18] V.V. de Camargo, R.A. Ramos, e P.C. Masiero. Implementação de Variabilidades em Frameworks Orientados a Aspecto desenvolvidos em AspectJ. *Workshop de Desenvolvimento de Software Orientado a Aspectos, Brasília*, páginas 8, 2004.

[19] E.W. Dijkstra. *A Discipline of Programming*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1997.

[20] Oniria Enternainment. Disponível em: http://www.oniriagames.com.br/, outubro de 2005.

[21] M. Fayad e R.E. Johnson. *Domain-specific application frameworks: frameworks experience by industry*. Wiley USA;, 2000.

[22] M.E. Fayad, D.C. Schmidt, e R.E. Johnson. *Building application frameworks: object-oriented foundations of framework design*. John Wiley & Sons, Inc. New York, NY, USA, 1999.

[23] M.E. Fayad, D.C. Schmidt, e R.E. Johnson. *Implementing application frameworks: object-oriented frameworks at work*. John Wiley & Sons, Inc. New York, NY, USA, 1999.

[24] M. Fontoura. *A systematic approach to framework development*. Tese de Doutorado, PUC - Rio de Janeiro, 1999.

[25] M. Fontoura, W. Pree, e B. Rumpe. UML-F: A modeling language for object-oriented frameworks. *ECOOP 2000-Object-Oriented Programming*, páginas 63–82, 2000.

[26] E. Gamma, R. Helm, R. Johnson, e J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, 1995.

[27] S. Hanenberg. Multi-Design Application Frameworks. *In: Proc. of the Generative and Component-Based Software Engineering Young Researchers Workshop, Erfurt*, 2000.

[28] S. Hanenberg, R. Hirschfeld, R. Unland, e K. Kawamura. Applying Aspect-Oriented Composition to Framework Development - A Case Study. *1st International Workshop on Foundations of Unanticipated Software Evolution, Barcelona, Spain*, 2004.

[29] S. Hanenberg e A. Schmidmeier. Idioms for building software frameworks in AspectJ. *AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software*, 2003.

[30] T. Inc. Building object-oriented frameworks. *A Taligent White Paper*, 1994.

[31] I. Jacobson e P.W. Ng. *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley Professional, 2005.

[32] R.E. Johnson. Frameworks=(components+ patterns). *ACM New York, NY, USA*, 1997.

[33] R.E. Johnson e P. Deutsch. How to design frameworks. *Urbana*, 51:61801, 1993.

[34] R.E. Johnson e B. Foote. Designing reusable classes. *Journal of object-oriented programming*, 1(2):22–35, 1988.

[35] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.V. Lopes, J.M. Loingtier, e J. Irwin. Aspect-Oriented Programming. *In proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland*, páginas 220–242, 1997.

[36] I. Krechetov, B. Tekinerdogan, A. Garcia, C. Chavez, e U. Kulesza. Towards an integrated aspect-oriented modeling approach for software architecture design. *8th Workshop on Aspect-Oriented Modelling (AOM. 06), AOSD*, 6, 2006.

[37] U. Kulesza. *Uma Abordagem Orientada a Aspectos para o Desenvolvimento de Frameworks.* Tese de Doutorado, PUC-Rio, 2007.

[38] U. Kulesza, V. Alves, A. Garcia, C.J.P.D. Lucena, e P. Borba. Improving extensibility of object-oriented frameworks with aspect-oriented programming. *Proceedings of 9th International Conference on Software Reuse, Icsr 2006 Turin, Italy, June 12-15, 2006. Lecture Notes in Computer Science: Reuse of Off-the-Shelf Components*, páginas 231–245, 2006.

[39] K.J. Lieberherr. Adaptive object-oriented software: the demeter method with propagation patterns. *PWS Boston*, 1996.

[40] S.N. Matos. *Um Método Dirigido por Responsabilidades para Obtenção Antecipada de Pontos de Estabilidade e de Flexibilidade no Desenvolvimento de Frameworks de Domínio.* Tese de Doutorado, ITA - São José dos Campos, 2008.

[41] S.N. Matos e C.T. Fernandes. Abordagem dirigida por responsabilidades voltada ao desenvolvimento de frameworks de domímio. *Relatório Técnico Cientifico ITA, São Jose dos Campos SP*, 2007.

[42] S.N. Matos e C.T. Fernandes. Um panorama dos processos de desenvolvimento de frameworks de domínio. *Relatório Técnico Cientifico ITA, São Jose dos Campos SP*, 2007.

[43] A. Moreira, J. Araújo, e I. Brito. Crosscutting quality attributes for requirements engineering. *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, páginas 167–174, 2002.

[44] A. Moreira, A. Rashid, e J. Araujo. Modularization and composition of aspectual requirements. *2nd International Conference on Aspect-Oriented Software Development*, páginas 11–20, 2003.

[45] S. Nakajima. Separation of concerns in early stage of framework development. *OOPSLA 99 Workshop on Multi-Dimensional Separation of Concerns in Object-Oriented Systems*, 1999.

[46] L. Penczek. AFR: uma abordagem para a sistematização do reúso de frameworks orientados a aspectos. Dissertação de Mestrado, PUC-RS, 2006.

[47] A.R. Pimentel. *Uma abordagem para projeto de software orientado a objetos baseado na teoria de projeto axiomático*. Tese de Doutorado, UTFPR, 2007.

[48] M. Pinto, M. Amor, L. Fuentes, e J.M. Troya. Collaborative virtual environment development: an aspect-orientedapproach. *Distributed Computing Systems Workshop, 2001 International Conference on*, páginas 97–102, 2001.

[49] M. Pinto, L. Fuentes, M.E. Fayad, e J.M. Troya. Towards an aspect-oriented framework in the design of collaborative virtual environments. *Proceedings of FTDCS*, 1, 2001.

[50] M. Pinto, L. Fuentes, M.E. Fayad, e J.M. Troya. Separation of coordination in a dynamic aspect oriented framework. *Proceedings of the 1st international conference on Aspect-oriented software development*, páginas 134–140, 2002.

[51] W. Pree. *Design Patterns for Object-Oriented Software Development.* Addison-Wesley, 1994.

[52] R. Prieto-Diaz. Domain analysis for reusability. *Proceedings of COMPSAC*, volume 87, páginas 23–29, 1987.

[53] RA Ramos, A. Carvalho, C. Monteiro, C. Silva, JFB Castro, F. Alencar, e R. Afonso. Avaliação da Qualidade de um Documento de Requisitos Orientado a Aspectos. *IX IDEAS*, 6, 2006.

[54] R.A. Ramos e J.F.B. Castro. Avaliação de uma Metodologia de Medição da Qualidade em um Documento de Requisitos Orientado a Aspectos. *WER, Porto-Portugal*, 2005.

[55] A. Rashid, P. Sawyer, A. Moreira, e J. Araujo. Early aspects: a model for aspect-oriented requirements engineering. *IEEE Joint International Conference on Requirements Engineering, 2002. Proceedings*, páginas 199–202, 2002.

[56] A. Rausch, B. Rumpe, e L. Hoogendoorn. Aspect-oriented framework modeling. *4th AOM Workshop at UML*, 3, 2003.

[57] A. Rocha, V.V. de Camargo, e P.C. Masiero. Uso de aspectos para verificar regras de instanciação de frameworks. *Workshop de Desenvolvimento de Software Orientado a Aspectos, Brasília*, páginas 8, 2004.

[58] L. F. Silva. *Uma Estratégia Orientada a Aspectos para Modelagem de Requisitos.* Tese de Doutorado, PUC-Rio, março de 2006.

[59] L.P. Silva. A engenharia de requisitos orientada a aspectos: a abordagem DAORE. Dissertação de Mestrado, UEM, 2007.

[60] R.P. Silva. *Suporte ao desenvolvimento e uso de frameworks e componentes.* Tese de Doutorado, UFRS, 2000.

[61] G. Sousa. Uma Abordagem Direcionada a Casos de Uso para o Desenvolvimento de Software Orientado a Aspectos. Dissertação de Mestrado, UFPE, maio de 2004.

[62] P. Tarr, H. Ossher, W. Harrison, e S.M. Sutton Jr. N degrees of separation: Multi-dimensional separation of concerns. 1999.

[63] B. Tekinerdogan, A. Moreira, J. Araújo, e P. Clements. Early aspects: Aspect-oriented requirements engineering and architecture design. Disponível em: http://doc.utwente.nl/56986/, julho de 2010.

[64] J. Viega e J. Voas. Can aspect-oriented programming lead to more reliable software? *IEEE SOFTWARE*, páginas 19–21, 2000.

[65] A. Wirfs-Brock. Designing reusable designs: Experiences designing object-oriented frameworks. *European Conference on Object-Oriented Programming and Conference on Object-Oriented Programming: Systems, Languages and Applications*, páginas 19–24, 1990.

[66] A. Wirfs-Brock, B. Wilkerson, e L. Wiener. Design Object-Oriented Software. *NJ: Prentice Hall*, páginas 341, 1990.

[67] R. Wirfs-Brock e A. McKean. *Object design: roles, responsibilities, and collaborations*. Pearson Education, 2002.

[68] R.J. Wirfs-Brock e R.E. Johnson. Surveying current research in object-oriented design. *Communications of the ACM*, 33(9):124, 1990.

APÊNDICE A

ARTEFATOS DO DOMÍNIO "JOGO DE CORRIDA DE CARROS"

Os artefatos do domínio "Jogo de Corrida de Carros", ilustrados a seguir, foram adaptados do trabalho de Matos [40]. Em seu trabalho, Matos utiliza a abordagem PDR-DFD (Processo Dirigido a Responsabilidades aplicado ao Desenvolvimento de Framework de Domínio) para realizar a análise das aplicações-exemplo escolhidas.

As aplicações-exemplo escolhidas foram as seguintes: "Nigel Mansell´s", "Top Gear" e "Rock N´ Roll Racing". Onde, apenas os subsistemas (perspectivas): "Interface de Usuário", "Setar Carro", "Senha" e "Nacionalidade", foram analisados.

Sobre o Domínio

O desenvolvimento de jogos é um dos ramos da informática que mais impulsiona a evolução tecnológica da industria ([20] apud [40]). Sendo um dos tipos de aplicação mais usados em computadores pessoais, assim como, um dos tipos mais implementados. Cerca de 60% dos programas escritos são jogos [40].

Os jogos de corrida de carros podem ser classificados como simuladores. Normalmente, buscam levar em consideração um ambiente real que, pode ou não, ter a sua interface implementada 3D. Em termos de lógica operacional, possuem uma alta complexidade. Geralmente, este tipo de jogo trás informações das pistas, carros, jogadores, acessórios, fases, entre outros recursos [40]. Mais detalhes acerca deste domínio pode ser encontrado em [40].

Diagrama de Casos de Uso do Domínio

A Figura A.1 ilustra o Diagrama de Casos de Uso do domínio "Jogo de Corrida de Carros". Este diagrama faz parte do Modelo de Requisitos do Framework (MRF). Ele foi

criado com o auxílio de artefatos presentes no trabalho de Matos [40].

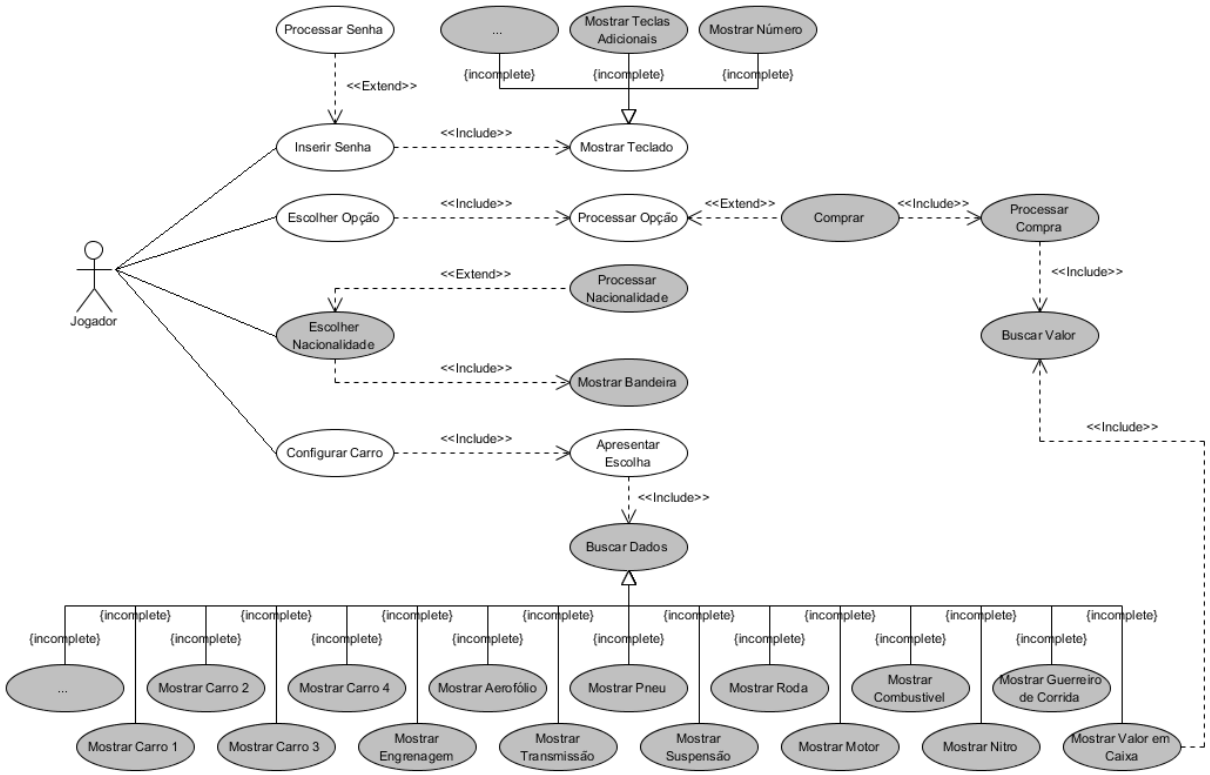

Figura A.1: Diagrama de casos de uso para o domínio “Jogo de Corrida de Carros”.

Na Figura A.1 os casos de uso classificados como de “Estabilidade” e “Flexibilidade” para o domínio estão representados através da cor de fundo branco e cinza, respectivamente.

Responsabilidades do Domínio

A Tabela A.1 apresenta o conjunto de responsabilidades do domínio “Jogo de Corrida de Carros”. Este conjunto foi criado a partir da análise da descrição textual dos casos de uso ilustrados na Figura A.1.

Tabela A.1 Responsabilidades do domínio “Jogo de Corrida de Carros”.

Ator	**Perspectiva**	**Responsabilidade**	**Caso de Uso**	**PDR-DFD**
Jogador	Int. de Usuário	Abandonar Inserir Senha	Inserir Senha	Estabilidade
Jogador	Int. de Usuário	Posicionar Tecla Digitada	Inserir Senha	Estabilidade
Jogador	Int. de Usuário	Requisitar “Mostrar Teclado”	Inserir Senha	Estabilidade
Jogador	Int. de Usuário	Requisitar “Processar Senha”	Inserir Senha	Estabilidade
Jogador	Int. de Usuário	Requisitar “Processar Opção”	Escolher Opção	Estabilidade
Jogador	Int. de Usuário	Requisitar “Apresentar Escolha”	Configurar Carro	Estabilidade
Continua na próxima página ...				

Tabela A.1 – continuação da página anterior.

Ator	**Perspectiva**	**Responsabilidade**	**Caso de Uso**	**PDR-DFD**
Jogador	Int. de Usuário	Exibir Letra Alfabeto	Mostrar Teclado	Estabilidade
Int. de Usuário	Setar Carro	Requisitar “Buscar Dados”	Apresentar Escolha	Estabilidade
Int. de Usuário	Senha	Validar Senha	Processar Senha	Estabilidade
Int. de Usuário	Senha	Salvar Senha	Processar Senha	Estabilidade
Int. de Usuário	Setar Carro	Verificar Opção	Processar Opção	Estabilidade
Int. de Usuário	Setar Carro	Executar Ação	Processar Opção	Estabilidade
Int. de Usuário	Setar Carro	Requisitar “Comprar”	Processar Opção	Estabilidade
Jogador	Int. de Usuário	Requisitar “Mostrar Bandeira”	Escolher Nacionalidade	Flexibilidade
Jogador	Int. de Usuário	Navegar Imagens	Escolher Nacionalidade	Flexibilidade
Jogador	Int. de Usuário	Requisitar “Processar Nacionalidade”	Escolher Nacionalidade	Flexibilidade
Jogador	Int. de Usuário	Abandonar Escolher Nacionalidade	Escolher Nacionalidade	Flexibilidade
Jogador	Int. de Usuário	Exibir Bandeira	Mostrar Bandeira	Flexibilidade
Jogador	Int. de Usuário	Exibir Nome Pais	Mostrar Bandeira	Flexibilidade
Jogador	Int. de Usuário	Exibir Número	Mostrar Número	Flexibilidade
Jogador	Int. de Usuário	Exibir Teclas Adicionais	Mostrar Teclas Adicionais	Flexibilidade
Int. de Usuário	Setar Carro	Requisitar “Processar Compra”	Comprar	Flexibilidade
Int. de Usuário	Nacionalidade	Salvar Nacionalidade	Processar Nacionalidade	Flexibilidade
Int. de Usuário	Setar Carro	Verificar Valor Item	Processar Compra	Flexibilidade
Int. de Usuário	Setar Carro	Requisitar “Buscar Valor”	Processar Compra	Flexibilidade
Int. de Usuário	Setar Carro	Debitar Valores	Processar Compra	Flexibilidade
Int. de Usuário	Setar Carro	Atualizar Valor	Processar Compra	Flexibilidade
Int. de Usuário	Setar Carro	Buscar Valor	Buscar Valor	Flexibilidade
Int. de Usuário	Setar Carro	Retornar Valor	Buscar Valor	Flexibilidade
Int. de Usuário	Setar Carro	Buscar Dados Item	Buscar Dados	Flexibilidade
Int. de Usuário	Setar Carro	Exibir Dados Carro-1	Mostrar Carro-1	Flexibilidade
Int. de Usuário	Setar Carro	Exibir Dados Carro-2	Mostrar Carro-2	Flexibilidade
Int. de Usuário	Setar Carro	Exibir Dados Carro-3	Mostrar Carro-3	Flexibilidade
Int. de Usuário	Setar Carro	Exibir Dados Carro-4	Mostrar Carro-4	Flexibilidade
Int. de Usuário	Setar Carro	Setar Aerofólio Padrão	Mostrar Aerofólio	Flexibilidade
Int. de Usuário	Setar Carro	Mostrar Imagens	Mostrar Aerofólio	Flexibilidade
Int. de Usuário	Setar Carro	Permitir Navegabilidade	Mostrar Aerofólio	Flexibilidade
Int. de Usuário	Setar Carro	Mostrar Dados Combustível	Mostrar Combustível	Flexibilidade
Int. de Usuário	Setar Carro	Mostrar Imagens	Mostrar Combustível	Flexibilidade
Int. de Usuário	Setar Carro	Permitir Navegabilidade	Mostrar Combustível	Flexibilidade
Int. de Usuário	Setar Carro	Setar Engrenagem Padrão	Mostrar Engrenagem	Flexibilidade
Int. de Usuário	Setar Carro	Mostrar Imagens	Mostrar Engrenagem	Flexibilidade
Int. de Usuário	Setar Carro	Permitir Navegabilidade	Mostrar Engrenagem	Flexibilidade
Int. de Usuário	Setar Carro	Mostrar Dados Guerreiro de Corrida	Mostrar Guerreiro	Flexibilidade
Int. de Usuário	Setar Carro	Mostrar Imagens	Mostrar Guerreiro	Flexibilidade
Int. de Usuário	Setar Carro	Permitir Navegabilidade	Mostrar Guerreiro	Flexibilidade

Continua na próxima página ...

Tabela A.1 – continuação da página anterior.

Ator	**Perspectiva**	**Responsabilidade**	**Caso de Uso**	**PDR-DFD**
Int. de Usuário	Setar Carro	Mostrar Dados Motor	Mostrar Motor	Flexibilidade
Int. de Usuário	Setar Carro	Mostrar Imagens	Mostrar Motor	Flexibilidade
Int. de Usuário	Setar Carro	Permitir Navegabilidade	Mostrar Motor	Flexibilidade
Int. de Usuário	Setar Carro	Mostrar Dados Nitro	Mostrar Nitro	Flexibilidade
Int. de Usuário	Setar Carro	Mostrar Imagens	Mostrar Nitro	Flexibilidade
Int. de Usuário	Setar Carro	Permitir Navegabilidade	Mostrar Nitro	Flexibilidade
Int. de Usuário	Setar Carro	Setar Pneu Padrão	Mostrar Pneu	Flexibilidade
Int. de Usuário	Setar Carro	Mostrar Imagens	Mostrar Pneu	Flexibilidade
Int. de Usuário	Setar Carro	Permitir Navegabilidade	Mostrar Pneu	Flexibilidade
Int. de Usuário	Setar Carro	Mostrar Dados Roda	Mostrar Roda	Flexibilidade
Int. de Usuário	Setar Carro	Mostrar Imagens	Mostrar Roda	Flexibilidade
Int. de Usuário	Setar Carro	Permitir Navegabilidade	Mostrar Roda	Flexibilidade
Int. de Usuário	Setar Carro	Mostrar Dados Suspensão	Mostrar Suspensão	Flexibilidade
Int. de Usuário	Setar Carro	Mostrar Imagens	Mostrar Suspensão	Flexibilidade
Int. de Usuário	Setar Carro	Permitir Navegabilidade	Mostrar Suspensão	Flexibilidade
Int. de Usuário	Setar Carro	Setar Transmissão Padrão	Mostrar Transmissão	Flexibilidade
Int. de Usuário	Setar Carro	Mostrar Imagens	Mostrar Transmissão	Flexibilidade
Int. de Usuário	Setar Carro	Permitir Navegabilidade	Mostrar Transmissão	Flexibilidade
Int. de Usuário	Setar Carro	Requisitar “Buscar Valor”	Mostrar Valor em Caixa	Flexibilidade
Int. de Usuário	Setar Carro	Exibir Valor em Caixa	Mostrar Valor em Caixa	Flexibilidade

APÊNDICE B

ARTEFATOS DA APLICAÇÃO-EXEMPLO *BUDDI*

Os artefatos da aplicação-exemplo *Buddi* do domínio "Controle de Finanças Pessoais", ilustrados a seguir, foram gerados usando as atividades propostas pela abordagem PDR-DFD (Processo Dirigido a Responsabilidades aplicado ao Desenvolvimento de Framework de Domínio) [41].

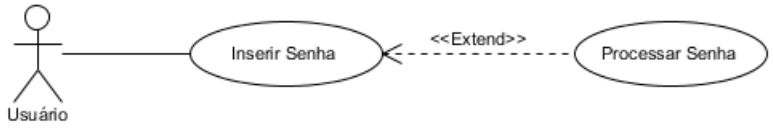

Figura B.1: Diagrama de casos de uso para o subsistema "Senha" da aplicação-exemplo *Buddi*.

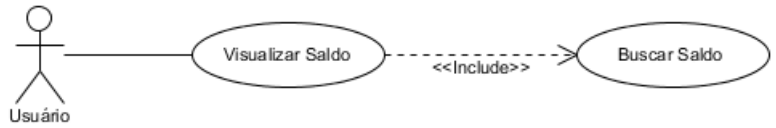

Figura B.2: Diagrama de casos de uso para o subsistema "Saldo" da aplicação-exemplo *Buddi*.

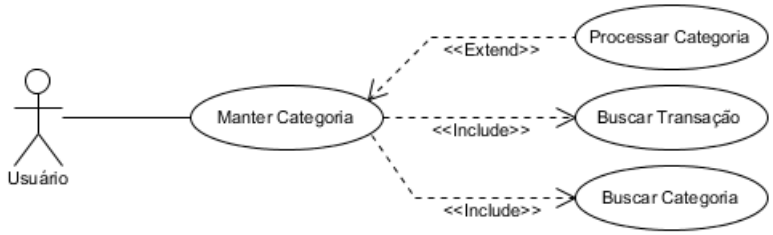

Figura B.3: Diagrama de casos de uso para o subsistema "Categoria" da aplicação-exemplo *Buddi*.

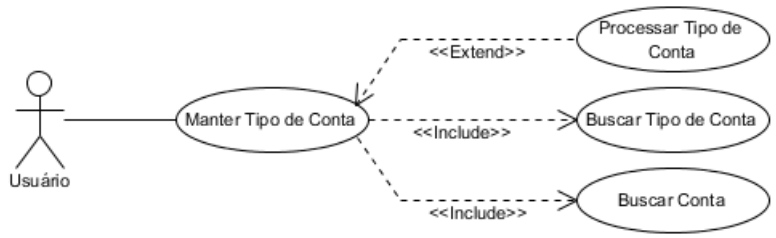

Figura B.4: Diagrama de casos de uso para o subsistema "Tipo de Conta" da aplicação-exemplo *Buddi*.

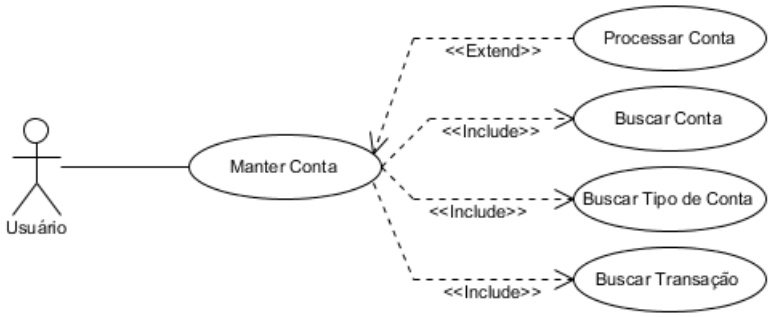

Figura B.5: Diagrama de casos de uso para o subsistema "Conta" da aplicação-exemplo *Buddi*.

Tabela B.1: Descrição textual do caso de uso "Inserir Senha" da aplicação-exemplo *Buddi*.

Nome do Caso de Uso	Inserir Senha
Ações do Ator	**Responsabilidades do Sistema**
1. O usuário seleciona inserir senha. **3.** O usuário informa a senha.	**2.** O sistema mostra a tela de "Inserir Senha". **4.** O sistema processa a senha informada (Processar Senha).

Tabela B.2: Descrição textual do caso de uso "Processar Senha" da aplicação-exemplo *Buddi*.

Nome do Caso de Uso	Processar Senha
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa a senha.	**2.** O sistema valida a senha informada. **3.** O sistema exibe a mensagem de sucesso. **4.** O sistema exibe a mensagem de erro.

Tabela B.3: Descrição textual do caso de uso "Visualizar Saldo" da aplicação-exemplo *Buddi*.

Nome do Caso de Uso	Visualizar Saldo
Ações do Ator	**Responsabilidades do Sistema**
1. O usuário seleciona visualizar saldo.	**2.** O sistema busca o saldo (Buscar Saldo). **3.** O sistema mostra o saldo.

Tabela B.4: Descrição textual do caso de uso "Buscar Saldo" da aplicação-exemplo *Buddi*.

Nome do Caso de Uso	Buscar Saldo
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário seleciona buscar saldo.	**2.** O sistema busca o saldo. **3.** O sistema retorna o saldo.

Tabela B.5: Descrição textual do caso de uso "Manter Categoria" da aplicação-exemplo *Buddi*.

Nome do Caso de Uso	Manter Categoria
Ações do Ator	**Responsabilidades do Sistema**
1. O usuário seleciona a opção "Categoria".	**2.** O sistema busca as categorias existentes (Buscar Categoria). **3.** O sistema apresenta a lista de categorias existentes, juntamente com opções para cadastrar, consultar, editar e excluir.
4. Após o passo 3 o usuário seleciona cadastrar categoria.	**5.** O sistema apresenta a tela de "Cadastrar Categoria".
6. O usuário informa os dados da categoria.	**7.** O sistema processa os dados informados (Processar Categoria). **8.** O sistema salva a categoria.
9. Após o passo 3 o usuário seleciona consultar categoria.	**10.** O sistema apresenta a tela de "Consultar Categoria".
11. O usuário informa os dados de busca.	**12.** O sistema consulta os dados da categoria (Buscar Categoria). **13.** O sistema apresenta a lista de categorias encontradas.
14. Após o passo 3 o usuário seleciona editar categoria.	**15.** O sistema apresenta a tela de "Editar Categoria". **16.** O sistema consulta os dados da categoria (Buscar Categoria). **17.** O sistema apresenta os dados da categoria.
18. O usuário atualiza os dados da categoria.	**19.** O sistema processa os dados informados (Processar Categoria). **20.** O sistema atualiza a categoria.
21. Após o passo 3 o usuário seleciona excluir categoria.	**22.** O sistema consulta os dados da categoria (Buscar Categoria). **23.** O sistema apresenta os dados da categoria, o sistema pede para o usuário confirmar a exclusão.
24. O usuário confirma a exclusão.	**25.** O sistema exclui a categoria. **26.** Antes do passo 25 o sistema verifica se tem transação vinculada a categoria (Buscar Transação). **27.** O sistema mostra mensagem de erro se tiver transação vinculada a categoria.

Tabela B.6: Descrição textual do caso de uso "Processar Categoria" da aplicação-exemplo *Buddi*.

Nome do Caso de Uso	Processar Categoria
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados da categoria.	**2.** O sistema valida os dados da categoria. **3.** O sistema retorna os dados da categoria. **4.** O sistema exibe a mensagem de erro.

Tabela B.7: Descrição textual do caso de uso “Buscar Categoria” da aplicação-exemplo *Buddi*.

Nome do Caso de Uso	Buscar Categoria
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados de busca.	**2.** O sistema valida os dados da categoria. **3.** O sistema busca a categoria. **4.** O sistema retorna a categoria.

Tabela B.8: Descrição textual do caso de uso “Manter Tipo de Conta” da aplicação-exemplo *Buddi*.

Nome do Caso de Uso	Manter Tipo de Conta
Ações do Ator	**Responsabilidades do Sistema**
1. O usuário seleciona a opção “Tipo de Conta”.	**2.** O sistema busca os tipos de conta existentes (Buscar Tipo de Conta). **3.** O sistema apresenta a lista de tipos de conta existentes, juntamente com opções para cadastrar, consultar, editar e excluir.
4. Após o passo 3 o usuário seleciona cadastrar tipo de conta.	**5.** O sistema apresenta a tela de “Cadastrar Tipo de Conta”.
6. O usuário informa os dados do tipo de conta.	**7.** O sistema processa os dados informados (Processar Tipo de Conta). **8.** O sistema salva o tipo de conta.
9. Após o passo 3 o usuário seleciona consultar tipo de conta.	**10.** O sistema apresenta a tela de “Consultar Tipo de Conta”.
11. O usuário informa os dados de busca.	**12.** O sistema consulta os dados do tipo de conta (Buscar Tipo de Conta). **13.** O sistema apresenta a lista de tipo de conta encontradas.
14. Após o passo 3 o usuário seleciona editar tipo de conta.	**15.** O sistema apresenta a tela de “Editar Tipo de Conta”. **16.** O sistema consulta os dados do tipo de conta (Buscar Tipo de Conta). **17.** O sistema apresenta os dados do tipo de conta.
18. O usuário atualiza os dados do tipo de conta.	**19.** O sistema processa os dados informados (Processar Tipo de Conta). **20.** O sistema atualiza o tipo de conta.
21. Após o passo 3 o usuário seleciona excluir tipo de conta.	**22.** O sistema consulta os dados do tipo de conta (Buscar Tipo de Conta). **23.** O sistema apresenta os dados do tipo de conta, o sistema pede para o usuário confirmar a exclusão.
24. O usuário confirma a exclusão.	**25.** O sistema exclui o tipo de conta. **26.** Antes do passo 25 o sistema verifica se tem cona vinculada ao tipo de conta (Buscar Conta). **27.** O sistema mostra mensagem de erro se tiver conta vinculada ao tipo de conta.

Tabela B.9: Descrição textual do caso de uso “Processar Tipo de Conta” da aplicação-exemplo *Buddi*.

Nome do Caso de Uso	Processar Tipo de Conta
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados do tipo de conta.	**2.** O sistema valida os dados do tipo de conta. **3.** O sistema retorna os dados do tipo de conta. **4.** O sistema exibe a mensagem de erro.

Tabela B.10: Descrição textual do caso de uso “Buscar Tipo de Conta” da aplicação-exemplo *Buddi*.

Nome do Caso de Uso	Buscar Tipo de Conta
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados de busca.	**2.** O sistema busca o tipo de conta. **3.** O sistema retorna o tipo de conta.

Tabela B.11: Descrição textual do caso de uso “Processar Conta” da aplicação-exemplo *Buddi*.

Nome do Caso de Uso	Processar Conta
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados da conta.	**2.** O sistema valida os dados da conta. **3.** O sistema retorna os dados da conta. **4.** O sistema exibe a mensagem de erro.

Tabela B.12: Descrição textual do caso de uso “Buscar Conta” da aplicação-exemplo *Buddi*.

Nome do Caso de Uso	Buscar Conta
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados de busca.	**2.** O sistema busca a conta. **3.** O sistema retorna a conta.

Tabela B.13: Descrição textual do caso de uso “Manter Conta” da aplicação-exemplo *Buddi*.

Nome do Caso de Uso	Manter Conta
Ações do Ator	**Responsabilidades do Sistema**
1. O usuário seleciona a opção “Conta”.	**2.** O sistema busca as contas existentes (Buscar Conta). **3.** O sistema apresenta a lista de contas existentes, juntamente com opções para cadastrar, consultar, editar e excluir.
4. Após o passo 3 o usuário seleciona cadastrar conta.	**5.** O sistema apresenta a tela de “Cadastrar Conta”. **6.** O sistema apresenta a lista de tipo de contas existentes (Buscar Tipo de Conta).
7. O usuário informa os dados da conta.	**8.** O sistema processa os dados informados (Processar Conta). **9.** O sistema salva a conta.
10. Após o passo 3 o usuário seleciona consultar conta.	**11.** O sistema apresenta a tela de “Consultar Conta”.
12. O usuário informa os dados de busca.	**13.** O sistema consulta os dados da conta (Buscar Conta). **14.** O sistema apresenta a lista de contas encontradas.
15. Após o passo 3 o usuário seleciona editar conta.	**16.** O sistema apresenta a tela de “Editar Conta”. **17.** O sistema consulta os dados da conta (Buscar Conta). **18.** O sistema apresenta a lista de tipo de contas existentes (Buscar Tipo de Conta). **19.** O sistema apresenta os dados da conta.
20. O usuário atualiza os dados da conta	**21.** O sistema processa os dados informados (Processar Conta). **22.** O sistema atualiza a conta.
23. Após o passo 3 o usuário seleciona excluir conta.	**24.** O sistema consulta os dados da conta (Buscar Conta). **25.** O sistema apresenta os dados da conta, o sistema pede para o usuário confirmar a exclusão.
26. O usuário confirma a exclusão.	**27.** O sistema exclui a conta. **28.** Antes do passo 27 o sistema verifica se tem transação vinculada a conta (Buscar Transação). **29.** O sistema mostra mensagem de erro se tiver transação vinculada a conta.

Tabela B.14: Descrição textual do caso de uso “Buscar Transação” da aplicação-exemplo *Buddi*.

Nome do Caso de Uso	Buscar Transação
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados de busca.	**2.** O sistema busca a transação. **3.** O sistema retorna a transação.

Tabela B.15: Descrição textual do caso de uso “Manter Transação” da aplicação-exemplo *Buddi*.

Nome do Caso de Uso	Manter Transação
Ações do Ator	**Responsabilidades do Sistema**
1. O usuário seleciona a opção “Transação”.	**2.** O sistema busca as transações existentes (Buscar Transação). **3.** O sistema apresenta a lista de transações existentes, juntamente com opções para cadastrar, consultar, editar e excluir.
4. Após o passo 3 o usuário seleciona cadastrar transação.	**5.** O sistema apresenta a tela de “Cadastrar Transação”. **6.** O sistema apresenta a lista de contas existentes (Buscar Conta). **7.** O sistema apresenta a lista de categorias existentes (Buscar Categoria).
8. O usuário informa os dados da transação.	**9.** O sistema processa os dados informados (Processar Transação). **10.** O sistema salva a transação.
11. Após o passo 3 o usuário seleciona consultar transação.	**12.** O sistema apresenta a tela de “Consultar Transação”.
13. O usuário informa os dados de busca.	**14.** O sistema consulta os dados da transação (Buscar Transação). **15.** O sistema apresenta a lista de transações encontradas.
16. Após o passo 3 o usuário seleciona editar transação.	**17.** O sistema apresenta a tela de “Editar Transação”. **18.** O sistema apresenta a lista de contas existentes (Buscar Conta). **19.** O sistema apresenta a lista de categorias existentes (Buscar Categoria). **20.** O sistema consulta os dados da transação (Buscar Transação). **21.** O sistema apresenta os dados da transação.
22. O usuário atualiza os dados da transação.	**23.** O sistema processa os dados informados (Processar Transação). **24.** O sistema atualiza a transação.
25. Após o passo 3 o usuário seleciona excluir transação.	**26.** O sistema consulta os dados da transação (Buscar Transação). **27.** O sistema apresenta os dados da transação, o sistema pede para o usuário confirmar a exclusão.
28. O usuário confirma a exclusão.	**29.** O sistema exclui a transação.

Tabela B.16: Descrição textual do caso de uso “Processar Transação” da aplicação-exemplo *Buddi*.

Nome do Caso de Uso	Processar Transação
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados da transação.	**2.** O sistema valida os dados da transação. **3.** O sistema retorna os dados da transação. **4.** O sistema exibe a mensagem de erro.

APÊNDICE C

ARTEFATOS DA APLICAÇÃO-EXEMPLO GFP

Os artefatos da aplicação-exemplo GFP do domínio "Controle de Finanças Pessoais", ilustrados a seguir, foram gerados usando as atividades propostas pela abordagem PDR-DFD (Processo Dirigido a Responsabilidades aplicado ao Desenvolvimento de Framework de Domínio) [41].

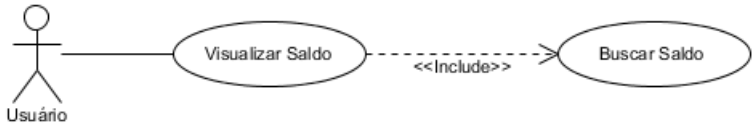

Figura C.1: Diagrama de casos de uso para o subsistema "Saldo" da aplicação-exemplo GFP.

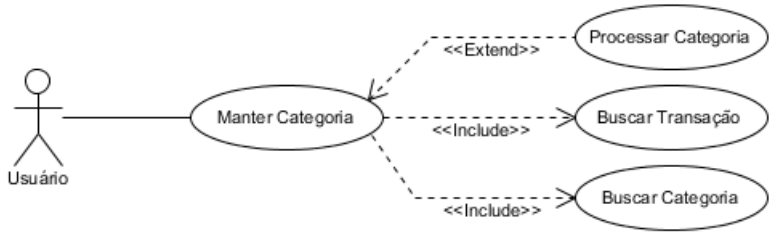

Figura C.2: Diagrama de casos de uso para o subsistema "Categoria" da aplicação-exemplo GFP.

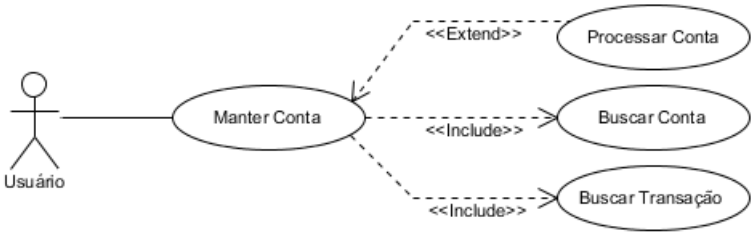

Figura C.3: Diagrama de casos de uso para o subsistema "Conta" da aplicação-exemplo GFP.

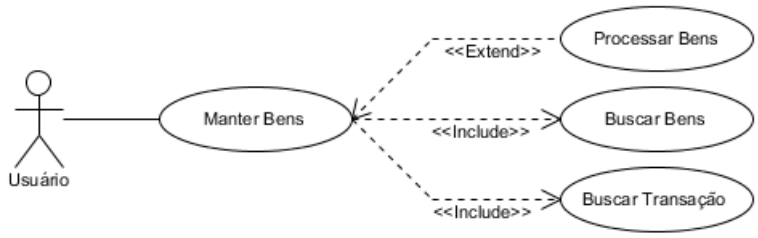

Figura C.4: Diagrama de casos de uso para o subsistema “Bens‘” da aplicação-exemplo GFP.

Tabela C.1: Descrição textual do caso de uso “Visualizar Saldo” da aplicação-exemplo GFP.

Nome do Caso de Uso	Visualizar Saldo
Ações do Ator	**Responsabilidades do Sistema**
1. O usuário seleciona visualizar saldo.	**2.** O sistema busca o saldo (Buscar Saldo). **3.** O sistema mostra o saldo.

Tabela C.2: Descrição textual do caso de uso “Buscar Saldo” da aplicação-exemplo GFP.

Nome do Caso de Uso	Buscar Saldo
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário seleciona buscar saldo.	**2.** O sistema busca o saldo. **3.** O sistema retorna o saldo.

Tabela C.3: Descrição textual do caso de uso “Processar Categoria” da aplicação-exemplo GFP.

Nome do Caso de Uso	Processar Categoria
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados da categoria.	**2.** O sistema valida os dados da categoria. **3.** O sistema retorna os dados da categoria. **4.** O sistema exibe a mensagem de erro.

Tabela C.4: Descrição textual do caso de uso “Buscar Categoria” da aplicação-exemplo GFP.

Nome do Caso de Uso	Buscar Categoria
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados de busca.	**2.** O sistema valida os dados da categoria. **3.** O sistema busca a categoria. **4.** O sistema retorna a categoria.

Tabela C.5: Descrição textual do caso de uso “Manter Categoria” da aplicação-exemplo GFP.

Nome do Caso de Uso	Manter Categoria
Ações do Ator	**Responsabilidades do Sistema**
1. O usuário seleciona a opção “Categoria”.	**2.** O sistema busca as categorias existentes. (Buscar Categoria) **3.** O sistema apresenta a lista de categorias existentes, juntamente com opções para cadastrar, consultar, editar e excluir.
4. Após o passo 3 o usuário seleciona cadastrar categoria.	**5.** O sistema apresenta a tela de “Cadastrar Categoria”.
6. O usuário informa os dados da categoria.	**7.** O sistema processa os dados informados. (Processar Categoria) **8.** O sistema salva a categoria.
9. Após o passo 3 o usuário seleciona consultar categoria.	**10.** O sistema apresenta a tela de “Consultar Categoria”.
11. O usuário informa os dados de busca.	**12.** O sistema consulta os dados da categoria. (Buscar Categoria) **13.** O sistema apresenta a lista de categorias encontradas.
14. Após o passo 3 o usuário seleciona editar categoria.	**15.** O sistema apresenta a tela de “Editar Categoria”. **16.** O sistema consulta os dados da categoria. (Buscar Categoria) **17.** O sistema apresenta os dados da categoria.
18. O usuário atualiza os dados da categoria.	**19.** O sistema processa os dados informados. (Processar Categoria) **20.** O sistema atualiza a categoria.
21. Após o passo 3 o usuário seleciona excluir categoria.	**22.** O sistema consulta os dados da categoria. (Buscar Categoria) **23.** O sistema apresenta os dados da categoria, o sistema pede para o usuário confirmar a exclusão.
24. O usuário confirma a exclusão.	**25.** O sistema exclui a categoria.
26. Antes do passo 25 o sistema verifica se tem transação vinculada a categoria. (Buscar Transação)	**27.** O sistema mostra mensagem de erro se tiver transação vinculada a categoria.

Tabela C.6: Descrição textual do caso de uso “Processar Conta” da aplicação-exemplo GFP.

Nome do Caso de Uso	Processar Conta
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados da conta.	**2.** O sistema valida os dados da conta. **3.** O sistema retorna os dados da conta. **4.** O sistema exibe a mensagem de erro.

Tabela C.7: Descrição textual do caso de uso “Manter Conta” da aplicação-exemplo GFP.

Nome do Caso de Uso	Manter Conta
Ações do Ator	**Responsabilidades do Sistema**
1. O usuário seleciona a opção “Conta”.	**2.** O sistema busca as contas existentes (Buscar Conta). **3.** O sistema apresenta a lista de contas existentes, juntamente com opções para cadastrar, consultar, editar e excluir.
4. Após o passo 3 o usuário seleciona cadastrar conta.	**5.** O sistema apresenta a tela de “Cadastrar Conta”.
6. O usuário informa os dados da conta.	**7.** O sistema processa os dados informados (Processar Conta). **8.** O sistema salva a conta.
9. Após o passo 3 o usuário seleciona consultar conta.	**10.** O sistema apresenta a tela de “Consultar Conta”.
11. O usuário informa os dados de busca.	**12.** O sistema consulta os dados da conta (Buscar Conta). **13.** O sistema apresenta a lista de contas encontradas.
14. Após o passo 3 o usuário seleciona editar conta.	**15.** O sistema apresenta a tela de “Editar Conta”. **16.** O sistema consulta os dados da conta (Buscar Conta). **17.** O sistema apresenta os dados da conta.
18. O usuário atualiza os dados da conta.	**19.** O sistema processa os dados informados (Processar Conta). **20.** O sistema atualiza a conta.
21. Após o passo 3 o usuário seleciona excluir conta.	**22.** O sistema consulta os dados da conta. (Buscar Conta). **23.** O sistema apresenta os dados da conta, o sistema pede para o usuário confirmar a exclusão.
24. O usuário confirma a exclusão.	**25.** O sistema exclui a conta.
26. Antes do passo 25 o sistema verifica se tem transação vinculada a conta (Buscar Transação).	**27.** O sistema mostra mensagem de erro se tiver transação vinculada a conta.

Tabela C.8: Descrição textual do caso de uso “Buscar Conta” da aplicação-exemplo GFP.

Nome do Caso de Uso	Buscar Conta
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados de busca.	**2.** O sistema busca a conta. **3.** O sistema retorna a conta.

Tabela C.9: Descrição textual do caso de uso “Processar Bens” da aplicação-exemplo GFP.

Nome do Caso de Uso	Processar Bens
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados do bem.	**2.** O sistema valida os dados do bem. **3.** O sistema retorna os dados do bem. **4.** O sistema exibe a mensagem de erro.

Tabela C.10: Descrição textual do caso de uso “Manter Bens” da aplicação-exemplo GFP.

Nome do Caso de Uso	Manter Bens
Ações do Ator	**Responsabilidades do Sistema**
1. O usuário seleciona a opção “Bens”.	**2.** O sistema busca as bens existentes (Buscar Bens). **3.** O sistema apresenta a lista de bens existentes, juntamente com opções para cadastrar, consultar, editar e excluir.
4. Após o passo 3 o usuário seleciona cadastrar bem.	**5.** O sistema apresenta a tela de “Cadastrar Bens”.
6. O usuário informa os dados do bem.	**7.** O sistema processa os dados informados (Processar Bens). **8.** O sistema salva o bem.
9. Após o passo 3 o usuário seleciona consultar bens.	**10.** O sistema apresenta a tela de “Consultar Bens”.
11. O usuário informa os dados de busca.	**12.** O sistema consulta os dados do bem (Buscar Bens). **13.** O sistema apresenta a lista de bens encontrados.
14. Após o passo 3 o usuário seleciona editar bem.	**15.** O sistema apresenta a tela de “Editar Bens”. **16.** O sistema consulta os dados do bem (Buscar Bens). **17.** O sistema apresenta os dados do bem.
18. O usuário atualiza os dados do bem.	**19.** O sistema processa os dados informados (Processar Bens). **20.** O sistema atualiza o bem.
21. Após o passo 3 o usuário seleciona excluir bem.	**22.** O sistema consulta os dados do bem (Buscar Bens). **23.** O sistema apresenta os dados do bem, o sistema pede para o usuário confirmar a exclusão.
24. O usuário confirma a exclusão.	**25.** O sistema exclui o bem.
26. Antes do passo 25 o sistema verifica se tem transação vinculada ao bem (Buscar Transação).	**27.** O sistema mostra mensagem de erro se tiver transação vinculada ao bem.

Tabela C.11: Descrição textual do caso de uso “Buscar Bens” da aplicação-exemplo GFP.

Nome do Caso de Uso	Buscar Bens
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados de busca.	**2.** O sistema busca o bem. **3.** O sistema retorna bem.

Tabela C.12: Descrição textual do caso de uso “Processar Transação” da aplicação-exemplo GFP.

Nome do Caso de Uso	Processar Transação
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados da transação.	**2.** O sistema valida os dados da transação. **3.** O sistema retorna os dados da transação. **4.** O sistema exibe a mensagem de erro.

Tabela C.13: Descrição textual do caso de uso “Manter Transação” da aplicação-exemplo GFP.

Nome do Caso de Uso	Manter Transação
Ações do Ator	**Responsabilidades do Sistema**
1. O usuário seleciona a opção “Transação”.	**2.** O sistema busca as transações existentes (Buscar Transação). **3.** O sistema apresenta a lista de transações existentes, juntamente com opções para cadastrar, consultar, editar e excluir.
4. Após o passo 3 o usuário seleciona cadastrar transação.	**5.** O sistema apresenta a tela de “Cadastrar Transação”. **6.** O sistema apresenta a lista de contas existentes (Buscar Conta). **7.** O sistema apresenta a lista de categorias existentes. (Buscar Categoria). **8.** O sistema apresenta a lista de bens existentes (Buscar Bens).
9. O usuário informa os dados da transação.	**10.** O sistema processa os dados informados (Processar Transação). **11.** O sistema salva a transação.
12. Após o passo 3 o usuário seleciona consultar transação.	**13.** O sistema apresenta a tela de “Consultar Transação”.
14. O usuário informa os dados de busca.	**15.** O sistema consulta os dados da transação (Buscar Transação). **16.** O sistema apresenta a lista de transações encontradas.
17. Após o passo 3 o usuário seleciona editar transação.	**18.** O sistema apresenta a tela de “Editar Transação”. **19.** O sistema apresenta a lista de contas existentes (Buscar Conta). **20.** O sistema apresenta a lista de categorias existentes (Buscar Categoria). **21.** O sistema apresenta a lista de bens existentes (Buscar Bens). **22.** O sistema consulta os dados da transação (Buscar Transação). **23.** O sistema apresenta os dados da transação.
24. O usuário atualiza os dados da transação.	**25.** O sistema processa os dados informados (Processar Transação). **26.** O sistema atualiza a transação.
27. Após o passo 3 o usuário seleciona excluir transação.	**28.** O sistema consulta os dados da transação (Buscar Transação). **29.** O sistema apresenta os dados da transação, o sistema pede para o usuário confirmar a exclusão.
30. O usuário confirma a exclusão.	**31.** O sistema exclui a transação.

Tabela C.14: Descrição textual do caso de uso “Buscar Transação” da aplicação-exemplo GFP.

Nome do Caso de Uso	Buscar Transação
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados de busca.	**2.** O sistema busca a transação. **3.** O sistema retorna a transação.

APÊNDICE D

ARTEFATOS DA APLICAÇÃO-EXEMPLO *JGNASH*

Os artefatos da aplicação-exemplo *jGnash* do domínio "Controle de Finanças Pessoais", ilustrados a seguir, foram gerados usando as atividades propostas pela abordagem PDR-DFD (Processo Dirigido a Responsabilidades aplicado ao Desenvolvimento de Framework de Domínio) [41].

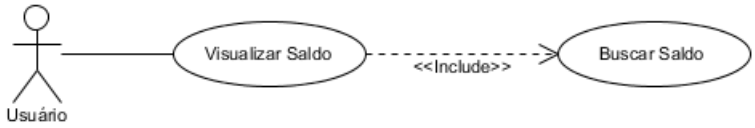

Figura D.1: Diagrama de casos de uso para o subsistema "Saldo" da aplicação-exemplo *jGnash*.

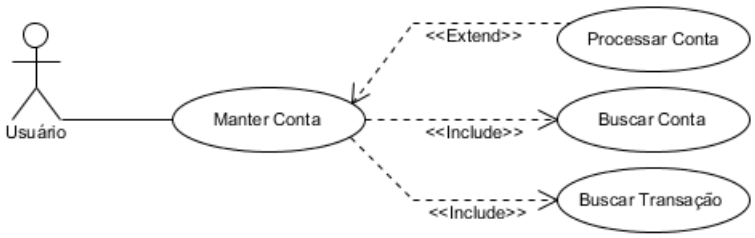

Figura D.2: Diagrama de casos de uso para o subsistema "Conta" da aplicação-exemplo *jGnash*.

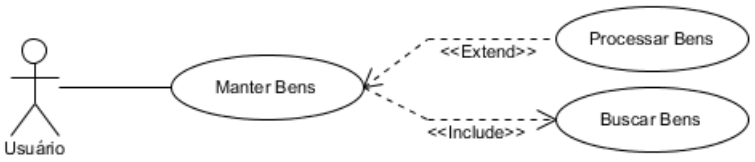

Figura D.3: Diagrama de casos de uso para o subsistema "Bens" da aplicação-exemplo *jGnash*.

Tabela D.1: Descrição textual do caso de uso "Visualizar Saldo" da aplicação-exemplo *jGnash*.

Nome do Caso de Uso	Visualizar Saldo
Ações do Ator	**Responsabilidades do Sistema**
1. O usuário seleciona visualizar saldo.	**2.** O sistema busca o saldo (Buscar Saldo). **3.** O sistema mostra o saldo.

Tabela D.2: Descrição textual do caso de uso "Buscar Saldo" da aplicação-exemplo *jGnash*.

Nome do Caso de Uso	Buscar Saldo
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário seleciona buscar saldo.	**2.** O sistema busca o saldo. **3.** O sistema retorna o saldo.

Tabela D.3: Descrição textual do caso de uso "Manter Conta" da aplicação-exemplo *jGnash*.

Nome do Caso de Uso	Manter Conta
Ações do Ator	**Responsabilidades do Sistema**
1. O usuário seleciona a opção "Conta".	**2.** O sistema busca as contas existentes (Buscar Conta). **3.** O sistema apresenta a lista de contas existentes, juntamente com opções para cadastrar, consultar, editar e excluir.
4. Após o passo 3 o usuário seleciona cadastrar conta.	**5.** O sistema apresenta a tela de "Cadastrar Conta".
6. O usuário informa os dados da conta.	**7.** O sistema processa os dados informados (Processar Conta). **8.** O sistema salva a conta.
9. Após o passo 3 o usuário seleciona consultar conta.	**10.** O sistema apresenta a tela de "Consultar Conta".
11. O usuário informa os dados de busca.	**12.** O sistema consulta os dados da conta (Buscar Conta). **13.** O sistema apresenta a lista de contas encontradas.
14. Após o passo 3 o usuário seleciona editar conta.	**15.** O sistema apresenta a tela de "Editar Conta". **16.** O sistema consulta os dados da conta (Buscar Conta). **17.** O sistema apresenta os dados da conta.
18. O usuário atualiza os dados da conta.	**19.** O sistema processa os dados informados (Processar Conta). **20.** O sistema atualiza a conta.
21. Após o passo 3 o usuário seleciona excluir conta.	**22.** O sistema consulta os dados da conta (Buscar Conta). **23.** O sistema apresenta os dados da conta, o sistema pede para o usuário confirmar a exclusão.
24. O usuário confirma a exclusão. **26.** Antes do passo 25 o sistema verifica se tem transação vinculada a conta (Buscar Transação).	**25.** O sistema exclui a conta. **27.** O sistema mostra mensagem de erro se tiver transação vinculada a conta.

Tabela D.4: Descrição textual do caso de uso "Processar Conta" da aplicação-exemplo *jGnash*.

Nome do Caso de Uso	Processar Conta
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados da conta.	**2.** O sistema valida os dados da conta. **3.** O sistema retorna os dados da conta. **4.** O sistema exibe a mensagem de erro.

Tabela D.5: Descrição textual do caso de uso "Buscar Conta" da aplicação-exemplo *jGnash*.

Nome do Caso de Uso	Buscar Conta
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados de busca.	**2.** O sistema busca a conta. **3.** O sistema retorna a conta.

Tabela D.6: Descrição textual do caso de uso "Manter Bens" da aplicação-exemplo *jGnash*.

Nome do Caso de Uso	Manter Bens
Ações do Ator	**Responsabilidades do Sistema**
1. O usuário seleciona a opção "Bens".	**2.** O sistema busca as bens existentes (Buscar Bens). **3.** O sistema apresenta a lista de bens existentes, juntamente com opções para cadastrar, consultar, editar e excluir.
4. Após o passo 3 o usuário seleciona cadastrar bem.	**5.** O sistema apresenta a tela de "Cadastrar Bens".
6. O usuário informa os dados do bem.	**7.** O sistema processa os dados informados (Processar Bens). **8.** O sistema salva o bem.
9. Após o passo 3 o usuário seleciona consultar bens.	**10.** O sistema apresenta a tela de "Consultar Bens".
11. O usuário informa os dados de busca.	**12.** O sistema consulta os dados do bem (Buscar Bens). **13.** O sistema apresenta a lista de bens encontrados.
14. Após o passo 3 o usuário seleciona editar bem.	**15.** O sistema apresenta a tela de "Editar Bens". **16.** O sistema consulta os dados do bem (Buscar Bens). **17.** O sistema apresenta os dados do bem.
18. O usuário atualiza os dados do bem.	**19.** O sistema processa os dados informados (Processar Bens). **20.** O sistema atualiza o bem.
21. Após o passo 3 o usuário seleciona excluir bem.	**22.** O sistema consulta os dados do bem (Buscar Bens). **23.** O sistema apresenta os dados do bem, o sistema pede para o usuário confirmar a exclusão.
24. O usuário confirma a exclusão.	**25.** O sistema exclui o bem.

Tabela D.7: Descrição textual do caso de uso "Processar Bens" da aplicação-exemplo *jGnash*.

Nome do Caso de Uso	Processar Bens
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados do bem.	**2.** O sistema valida os dados do bem. **3.** O sistema retorna os dados do bem. **4.** O sistema exibe a mensagem de erro.

Tabela D.8: Descrição textual do caso de uso "Buscar Bens" da aplicação-exemplo *jGnash*.

Nome do Caso de Uso	Buscar Bens
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados de busca.	**2.** O sistema busca o bem. **3.** O sistema retorna bem.

Tabela D.9: Descrição textual do caso de uso "Manter Transação" da aplicação-exemplo *jGnash*.

Nome do Caso de Uso	Manter Transação
Ações do Ator	**Responsabilidades do Sistema**
1. O usuário seleciona a opção "Transação".	**2.** O sistema busca as transações existentes (Buscar Transação). **3.** O sistema apresenta a lista de transações existentes, juntamente com opções para cadastrar, consultar, editar e excluir.
4. Após o passo 3 o usuário seleciona cadastrar transação.	**5.** O sistema apresenta a tela de "Cadastrar Transação". **6.** O sistema apresenta a lista de contas existentes (Buscar Conta).
7. O usuário informa os dados da transação.	**8.** O sistema processa os dados informados (Processar Transação). **9.** O sistema salva a transação.
10. Após o passo 3 o usuário seleciona consultar transação.	**11.** O sistema apresenta a tela de "Consultar Transação".
12. O usuário informa os dados de busca.	**13.** O sistema consulta os dados da transação (Buscar Transação). **14.** O sistema apresenta a lista de transações encontradas.
15. Após o passo 3 o usuário seleciona editar transação.	**16.** O sistema apresenta a tela de "Editar Transação". **17.** O sistema apresenta a lista de contas existentes (Buscar Conta). **18.** O sistema consulta os dados da transação (Buscar Transação). **19.** O sistema apresenta os dados da transação.
20. O usuário atualiza os dados da transação.	**21.** O sistema processa os dados informados (Processar Transação). **22.** O sistema atualiza a transação.
23. Após o passo 3 o usuário seleciona excluir transação.	**24.** O sistema consulta os dados da transação (Buscar Transação). **25.** O sistema apresenta os dados da transação, o sistema pede para o usuário confirmar a exclusão.
26. O usuário confirma a exclusão.	**27.** O sistema exclui a transação.

Tabela D.10: Descrição textual do caso de uso "Processar Transação" da aplicação-exemplo *jGnash*.

Nome do Caso de Uso	Processar Transação
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados da transação.	**2.** O sistema valida os dados da transação. **3.** O sistema retorna os dados da transação. **4.** O sistema exibe a mensagem de erro.

Tabela D.11: Descrição textual do caso de uso "Buscar Transação" da aplicação-exemplo *jGnash*.

Nome do Caso de Uso	Buscar Transação
Ações do Ator	**Responsabilidades do Sistema**
1. A interface do usuário informa os dados de busca.	**2.** O sistema busca a transação. **3.** O sistema retorna a transação.

APÊNDICE E

ARTEFATOS DO DOMÍNIO "CONTROLE DE FINANÇAS PESSOAIS"

Os artefatos do domínio "Controle de Finanças Pessoais", ilustrados a seguir, foram gerados usando as atividades propostas pela abordagem PDR-DFD (Processo Dirigido a Responsabilidades aplicado ao Desenvolvimento de Framework de Domínio) [41]. As aplicações-exemplo do domínio escolhidas foram as seguintes: "*Buddi*", "GFP" e "*jGnash*". Onde, apenas algus de subsistemas (perspectivas) foram analisados.

Diagrama de Casos de Uso do Domínio

A Figura E.1 ilustra o Diagrama de Casos de Uso do domínio "Controle de Finanças Pessoais". Este diagrama faz parte do Modelo de Requisitos do Framework (MRF). Ele foi criado com o auxílio de artefatos gerados a partir da análise das aplicações-exemplo do domínio.

Na Figura E.1 os casos de uso classificados como de "Estabilidade" e "Flexibilidade" para o domínio estão representados através da cor de fundo branco e cinza, respectivamente.

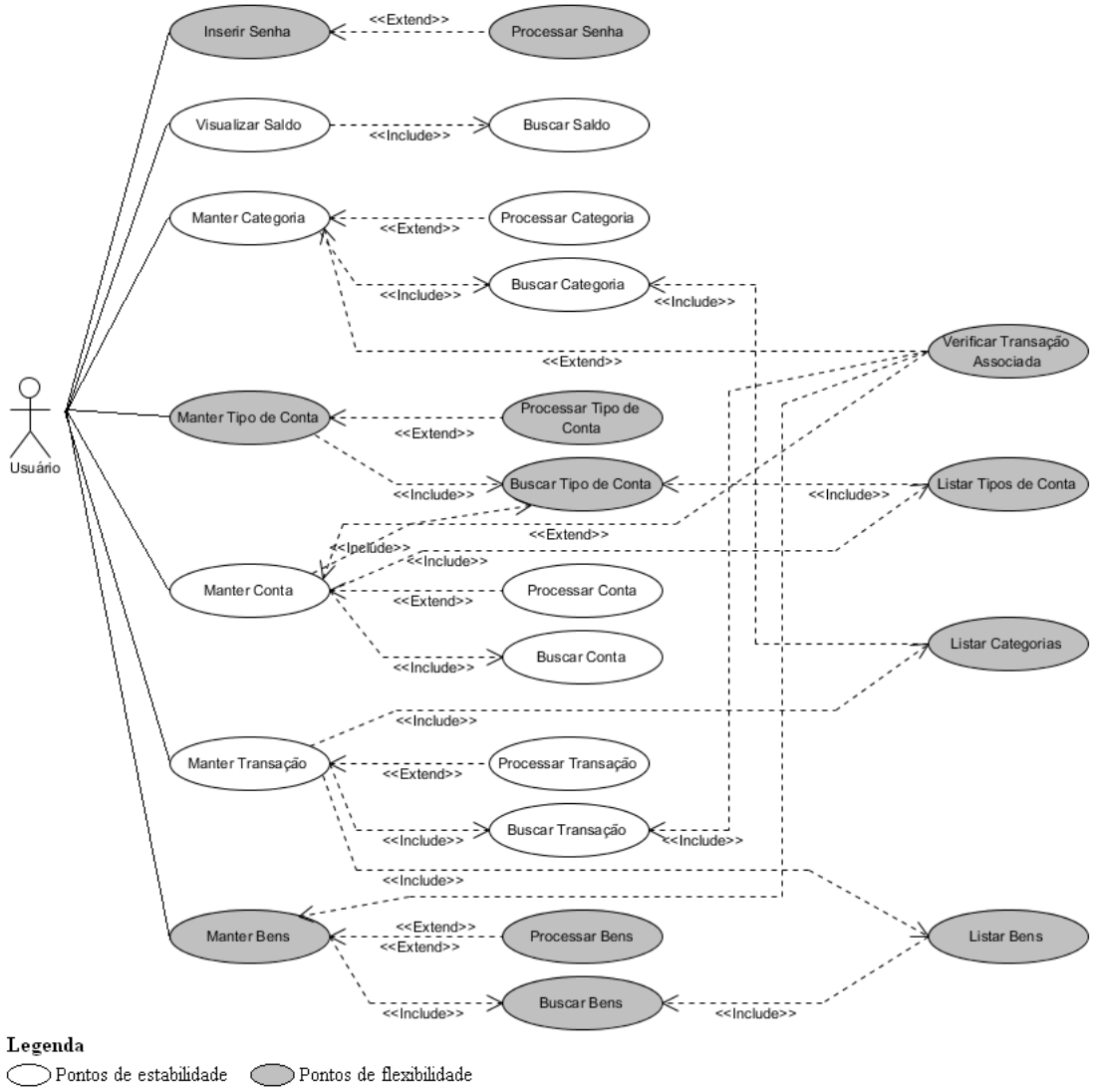

Figura E.1: Diagrama de casos de uso para o domínio "Controle de Finanças Pessoais".

Responsabilidades do Domínio

A Tabela E.1 apresenta o conjunto de responsabilidades do domínio "Controle de Finanças Pessoais". Este conjunto foi criado a partir da análise da descrição textual dos casos de uso ilustrados na Figura E.1.

Tabela E.1 Responsabilidades do domínio "Controle de Finanças Pessoais".

Ator	Perspectiva	Responsabilidade	Caso de Uso	PDR-DFD
Usuário	Int. de Usuário	Requisitar "Buscar Saldo"	Visualizar Saldo	Estabilidade
Usuário	Int. de Usuário	Mostrar Saldo	Visualizar Saldo	Estabilidade
Int. de Usuário	Saldo	Buscar Saldo	Buscar Saldo	Estabilidade
Int. de Usuário	Saldo	Retornar Saldo	Buscar Saldo	Estabilidade
Int. de Usuário	Categoria	Validar Dados Categoria	Processar Categoria	Estabilidade
Int. de Usuário	Categoria	Retornar Dados Categoria	Processar Categoria	Estabilidade
Int. de Usuário	Categoria	Mostrar Mensagem de Erro	Processar Categoria	Estabilidade

Continua na próxima página ...

Tabela E.1 – continuação da página anterior.

Ator	Perspectiva	Responsabilidade	Caso de Uso	PDR-DFD
Int. de Usuário	Categoria	Buscar Categoria	Buscar Categoria	Estabilidade
Int. de Usuário	Categoria	Retornar Categoria	Buscar Categoria	Estabilidade
Int. de Usuário	Conta	Validar Dados Conta	Processar Conta	Estabilidade
Int. de Usuário	Conta	Retornar Dados Conta	Processar Conta	Estabilidade
Int. de Usuário	Conta	Mostrar Mensagem de Erro	Processar Conta	Estabilidade
Int. de Usuário	Conta	Buscar Conta	Buscar Conta	Estabilidade
Int. de Usuário	Conta	Retornar Conta	Buscar Conta	Estabilidade
Int. de Usuário	Transação	Validar Dados Transação	Processar Transação	Estabilidade
Int. de Usuário	Transação	Retornar Dados Transação	Processar Transação	Estabilidade
Int. de Usuário	Transação	Mostrar Mensagem de Erro	Processar Transação	Estabilidade
Int. de Usuário	Transação	Buscar Transação	Buscar Transação	Estabilidade
Int. de Usuário	Transação	Retornar Transação	Buscar Transação	Estabilidade
Usuário	Int. de Usuário	Requisitar “Buscar Categoria”	Manter Categoria	Estabilidade
Usuário	Int. de Usuário	Listar Categorias	Manter Categoria	Estabilidade
Usuário	Int. de Usuário	Mostrar Tela Cadastrar Categoria	Manter Categoria	Estabilidade
Usuário	Int. de Usuário	Requisitar “Processar Categoria”	Manter Categoria	Estabilidade
Usuário	Int. de Usuário	Salvar Categoria	Manter Categoria	Estabilidade
Usuário	Int. de Usuário	Mostrar Tela Consultar Categoria	Manter Categoria	Estabilidade
Usuário	Int. de Usuário	Mostrar Tela Editar Categoria	Manter Categoria	Estabilidade
Usuário	Int. de Usuário	Mostrar Dados Categoria	Manter Categoria	Estabilidade
Usuário	Int. de Usuário	Atualizar Categoria	Manter Categoria	Estabilidade
Usuário	Int. de Usuário	Excluir Categoria	Manter Categoria	Estabilidade
Usuário	Int. de Usuário	Requisitar “Verificar Transação Associada”	Manter Categoria	Estabilidade
Usuário	Int. de Usuário	Requisitar “Buscar Conta”	Manter Conta	Estabilidade
Usuário	Int. de Usuário	Listar Conta	Manter Conta	Estabilidade
Usuário	Int. de Usuário	Mostrar Tela Cadastrar Conta	Manter Conta	Estabilidade
Usuário	Int. de Usuário	Requisitar “Processar Conta”	Manter Conta	Estabilidade
Usuário	Int. de Usuário	Salvar Conta	Manter Conta	Estabilidade
Usuário	Int. de Usuário	Mostrar Tela Consultar Conta	Manter Conta	Estabilidade
Usuário	Int. de Usuário	Mostrar Tela Editar Conta	Manter Conta	Estabilidade
Usuário	Int. de Usuário	Mostrar Dados Conta	Manter Conta	Estabilidade
Usuário	Int. de Usuário	Atualizar Conta	Manter Conta	Estabilidade
Usuário	Int. de Usuário	Excluir Conta	Manter Conta	Estabilidade
Usuário	Int. de Usuário	Requisitar “Verificar Transação Associada”	Manter Conta	Estabilidade
Usuário	Int. de Usuário	Requisitar “Listar Tipos de Conta”	Manter Conta	Estabilidade
Usuário	Int. de Usuário	Requisitar “Buscar Transação”	Manter Transação	Estabilidade
Usuário	Int. de Usuário	Listar Transação	Manter Transação	Estabilidade
Usuário	Int. de Usuário	Mostrar Tela Cadastrar Transação	Manter Transação	Estabilidade
Usuário	Int. de Usuário	Requisitar “Buscar Conta”	Manter Transação	Estabilidade
Usuário	Int. de Usuário	Requisitar “Processar Transação”	Manter Transação	Estabilidade
Usuário	Int. de Usuário	Salvar Transação	Manter Transação	Estabilidade
Usuário	Int. de Usuário	Mostrar Tela Consultar Transação	Manter Transação	Estabilidade
Usuário	Int. de Usuário	Mostrar Tela Editar Transação	Manter Transação	Estabilidade
Usuário	Int. de Usuário	Mostrar Dados Transação	Manter Transação	Estabilidade
Usuário	Int. de Usuário	Atualizar Transação	Manter Transação	Estabilidade
Usuário	Int. de Usuário	Excluir Transação	Manter Transação	Estabilidade
Usuário	Int. de Usuário	Requisitar “Listar Bens”	Manter Transação	Estabilidade
Usuário	Int. de Usuário	Requisitar “Listar Categorias”	Manter Transação	Estabilidade
Usuário	Int. de Usuário	Mostrar Tela Inserir Senha	Inserir Senha	Flexibilidade
Usuário	Int. de Usuário	Requisitar “Processar Senha”	Inserir Senha	Flexibilidade
Int. de Usuário	Senha	Validar Senha	Processar Senha	Flexibilidade
Int. de Usuário	Senha	Mostrar Mensagem de Sucesso	Processar Senha	Flexibilidade
Int. de Usuário	Senha	Mostrar Mensagem de Erro	Processar Senha	Flexibilidade
Usuário	Int. de Usuário	Requisitar “Buscar Tipo de Conta”	Manter Tipo de Conta	Flexibilidade
Usuário	Int. de Usuário	Listar Tipo de Conta	Manter Tipo de Conta	Flexibilidade

Continua na próxima página ...

Tabela E.1 – continuação da página anterior.

Ator	**Perspectiva**	**Responsabilidade**	**Caso de Uso**	**PDR-DFD**
Usuário	Int. de Usuário	Mostrar Tela Cadastrar Tipo de Conta	Manter Tipo de Conta	Flexibilidade
Usuário	Int. de Usuário	Requisitar “Processar Tipo de Conta”	Manter Tipo de Conta	Flexibilidade
Usuário	Int. de Usuário	Salvar Tipo de Conta	Manter Tipo de Conta	Flexibilidade
Usuário	Int. de Usuário	Mostrar Tela Consultar Tipo de Conta	Manter Tipo de Conta	Flexibilidade
Usuário	Int. de Usuário	Mostrar Tela Editar Tipo de Conta	Manter Tipo de Conta	Flexibilidade
Usuário	Int. de Usuário	Mostrar Dados Tipo de Conta	Manter Tipo de Conta	Flexibilidade
Usuário	Int. de Usuário	Atualizar Tipo de Conta	Manter Tipo de Conta	Flexibilidade
Usuário	Int. de Usuário	Excluir Tipo de Conta	Manter Tipo de Conta	Flexibilidade
Usuário	Int. de Usuário	Requisitar “Buscar Conta”	Manter Tipo de Conta	Flexibilidade
Usuário	Int. de Usuário	Mostrar Mensagem de Erro	Manter Tipo de Conta	Flexibilidade
Int. de Usuário	Tipo de Conta	Validar Dados Tipo de Conta	Processar Tipo de Conta	Flexibilidade
Int. de Usuário	Tipo de Conta	Retornar Dados Tipo de Conta	Processar Tipo de Conta	Flexibilidade
Int. de Usuário	Tipo de Conta	Mostrar Mensagem de Erro	Processar Tipo de Conta	Flexibilidade
Int. de Usuário	Tipo de Conta	Buscar Tipo de Conta	Buscar Tipo de Conta	Flexibilidade
Int. de Usuário	Tipo de Conta	Retornar Tipo de Conta	Buscar Tipo de Conta	Flexibilidade
Usuário	Int. de Usuário	Requisitar “Buscar Bens”	Manter Bens	Flexibilidade
Usuário	Int. de Usuário	Listar Bens	Manter Bens	Flexibilidade
Usuário	Int. de Usuário	Mostrar Tela Cadastrar Bens	Manter Bens	Flexibilidade
Usuário	Int. de Usuário	Requisitar “Processar Bens”	Manter Bens	Flexibilidade
Usuário	Int. de Usuário	Salvar Bem	Manter Bens	Flexibilidade
Usuário	Int. de Usuário	Mostrar Tela Consultar Bens	Manter Bens	Flexibilidade
Usuário	Int. de Usuário	Mostrar Tela Editar Bens	Manter Bens	Flexibilidade
Usuário	Int. de Usuário	Mostrar Dados Bem	Manter Bens	Flexibilidade
Usuário	Int. de Usuário	Atualizar Bem	Manter Bens	Flexibilidade
Usuário	Int. de Usuário	Excluir Bem	Manter Bens	Flexibilidade
Usuário	Int. de Usuário	Requisitar “Verificar Transação Associada”	Manter Bens	Flexibilidade
Int. de Usuário	Bens	Validar Dados Bem	Processar Bens	Flexibilidade
Int. de Usuário	Bens	Retornar Dados Bem	Processar Bens	Flexibilidade
Int. de Usuário	Bens	Mostrar Mensagem de Erro	Processar Bens	Flexibilidade
Int. de Usuário	Bens	Buscar Bem	Buscar Bens	Flexibilidade
Int. de Usuário	Bens	Retornar Bem	Buscar Bens	Flexibilidade
Int. de Usuário	Bens	Requisitar “Buscar Bens”	Listar Bens	Flexibilidade
Int. de Usuário	Categoria	Requisitar “Buscar Categoria”	Listar Categorias	Flexibilidade
Int. de Usuário	Tipo de Conta	Requisitar “Buscar Tipo de Conta”	Listar Tipos de Conta	Flexibilidade
Int. de Usuário	Transação	Requisitar “Buscar Transação”	Verificar Transação Associada	Flexibilidade
Int. de Usuário	Transação	Mostrar Mensagem de Erro	Verificar Transação Associada	Flexibilidade