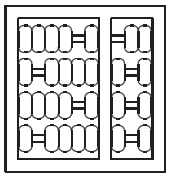

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Um Método de Desenvolvimento e Testes para Sistemas Confiáveis Baseados em Componentes: Um Estudo de Caso

Camila Ribeiro Rocha
Patrick Henrique da Silva Brito
Eliane Martins *Cecília Mary Fischer Rubira*

Technical Report - IC-07-09 - Relatório Técnico

March - 2007 - Março

Um Método de Desenvolvimento e Testes para Sistemas Confiáveis Baseados em Componentes: Um Estudo de Caso

Camila Ribeiro Rocha* Patrick Henrique da Silva Brito† Eliane Martins
Cecília Mary Fischer Rubira

Resumo

O projeto, a implementação e os testes do comportamento excepcional de um software são tarefas complexas que normalmente não recebem a atenção necessária das metodologias de desenvolvimento existentes. Este trabalho apresenta uma maneira sistemática de lidar com o tratamento de exceções desde a especificação de requisitos, especificação, implementação e testes, no desenvolvimento de sistemas baseados em componentes. Essa sistemática é apresentada através da execução de um estudo de caso de um sistema financeiro. As atividades de teste são executadas desde os estágios iniciais do desenvolvimento, proporcionando um aumento da qualidade do sistema produzido. Nossa solução refina a Metodologia para Definição do Comportamento Excepcional, MDCE, nas fases de projeto arquitetural, implementação, e testes. O método resultante desse refinamento foi denominado MDCE+. Além de refinar a MDCE, o método MDCE+ foi adaptado ao processo UML Components. Do ponto de vista do projeto CompGov, este estudo de caso foi utilizado para avaliar parcialmente o processo de desenvolvimento baseado em componentes com maximização do reuso. A parte avaliada do processo consistiu na reutilização de componentes a partir das especificações das suas interfaces providas e requeridas. Em um trabalho futuro, as demais atividades de reuso deverão ser avaliadas: (i) negociação dos requisitos a partir dos componentes já existentes; e (ii) reuso de *framework* e de componentes arquiteturais.

1 Introdução

Cada vez mais o desenvolvimento de sistemas computacionais é permeado por prazos curtos e exigência de alta qualidade. Em busca do cumprimento de objetivos tão antagônicos, o desenvolvimento baseado em componentes vem sendo sendo adotado atualmente, por proporcionar reutilização de código e conseqüente redução no custo e tempo de desenvolvimento.

O desenvolvimento baseado em componentes define que os sistemas devem ser formados por "unidades de composição com interfaces especificadas através de contratos e de-

*Financiada pela CAPES
†Financiado pelo curso de Especialização em Engenharia de Software IC/Unicamp

pendências de contexto explícitas´´ [Szy00]. Por favorecer a alta coesão e o baixo acoplamento entre os componentes, essa estrutura facilita a reutilização, permitindo que até mesmo componentes produzidos pro terceiros sejam integrados a um sistema.

As vantagens do desenvolvimento baseado em componentes têm sido apoveitadas até mesmo no desenvolvimento de sistemas para controle de atividades críticas, o que evidencia a necessidade de aliar este paradigma à técnicas para construção e testes de sistemas confiáveis, que considerem seu comportamento excepcional desses sistemas.

Tratamento de exceções [Goo75] é uma técnica bastante conhecida para a estruturação do comportamento excepcional em sistemas de software. Além disso, é implementado por diversas linguagens de programação populares, como C++, Java e C#. Apesar dessa popularidade, o projeto e a implementação do comportamento excepcional de um sistema são tarefas muito complexas que não recebem a atenção devida de metodologias de desenvolvimento existentes [dLR99, RdLeFCF04]. A situação é ainda mais crítica se levarmos em consideração os métodos para DBC. Conseqüentemente, os desenvolvedores sentem dificuldade tanto para usar os mecanismos de tratamento de exceções, quanto para projetar e testar o comportamento excepcional, o que compromete a confiabilidade dos sistemas construídos [RS03].

Este trabalho propõe um método integrado para o projeto, implementação e teste do comportamento excepcional de sistemas baseados em componentes. O método proposto é uma adaptação do processo de desenvolvimento baseado em componentes *UMLComponents* [CD00], que possui uma estruturação simples, de fácil compreensão e utilização prática. Essa característica o torna mais acessível ao mercado corporativo, principalmente quando comparado a outros processos de desenvolvimento baseado em componentes, como o Catalysis [DW99].

O método de desenvolvimento proposto inclui novas atividades ao *UMLComponents*, que conduzem a definição do comportamento excepcional do sistema e sua inclusão no código fonte de maneira organizada. Paralelamente ao desenvolvimento, são propostas atividades de teste, que auxiliam a criação e execução automatizada de casos de teste. Por enquanto, apenas a fase de testes de componentes unitários é considerada.

Este relatório técnico apresenta um estudo de caso no qual o método completo foi experimentado. O estudo foi realizado em um ambiente real, uma empresa brasileira de desenvolvimento de software de médio porte, que trabalha com aplicações de sistema financeiro. Em conjunto com a equipe da empresa, foi construído e testado em sistema financeiro de cadastro e controle de emissão de cheques e limite de crédito, no qual a segurança no funcionamento é uma das principais características.

O restante do relatório técnico está organizado de forma a explicitar a integração das atividades de desenvolvimento e testes de componentes, apresentando os documentos produzidos durante a modelagem e os resultados finais. Na Seção 2 são apresentados separadamente os métodos *UMLComponents*, MDCE+ (Método para Definição do Comportamento Excepcional) e de testes. As Seções 3 e 4 apresentam a descrição do sistema desenvolvido e do estudo de caso respectivamente. As Seções 5 a 8, são divididas em duas partes: na primeira, são detalhadas as atividades de cada uma das fases do método, explicitando os documentos necessários para o início da fase e os documentos gerados após a execução das atividades, e com a classificação em atividade de desenvolvimento ou teste (facilitando a

divisão de responsabilidades entre a equipe); na segunda, são apresentados os produtos obtidos durante o estudo de caso. Na Seção 9 os resultados obtidos são analisados de maneira crítica, tanto pela equipe criadora do método quanto pela equipe da empresa. Na Seção 10 são apresentadas algumas conclusões e trabalhos futuros. No Apêndice A são apresentados outros artefatos produzidos durante a modelagem.

2 Métodos

Por não ser um processo completo, o método proposto é baseado no processo *UMLComponents*, acrescentando à suas fases atividades para a definição do comportamento excepcional e atividades para a construção de casos de teste para componentes.

Apesar de estar detalhadamente descrito ao longo deste relatório técnico, nesta Seção é apresentada uma visão geral do método proposto. As subseções a seguir descrevem resumidamente cada uma das fases do *UMLComponents*, do método de desenvolvimento (MDCE+) e do método de testes. Ao final da Seção é apresentada um esquema com as atividades de desenvolvimento e testes integradas (Figura 3).

2.1 *UMLComponents*

O *UMLComponents* [CD00] é um processo de desenvolvimento para sistemas baseados em componentes que utiliza diagramas da UML em suas especificações. É um processo simples, de fácil compreensão e utilização prática [BFR04]. Por simplicidade, propõe a estruturação do sistema em quatro camadas: interface de usuário; diálogo com a interface, que faz a ponte entre as telas e a lógica da aplicação; serviços do sistema, que implementa as funcionalidades; e serviços de negócio, que armazena os dados da aplicação.

O processo *UMLComponents* guia o desenvolvimento das duas últimas camadas de forma iterativa, e é dividido em seis fases:

1. *Especificação de Requisitos*: desenvolvimento dos casos de uso, definindo os requisitos da aplicação; modelo do conceito do negócio (*Business Concept Model*), um esboço inicial das entidades básicas do sistema.

2. *Especificação dos Componentes*: fase mais importante do processo, na qual os componentes são especificados. É dividida em três subfases:

   (a) *Identificação dos Componentes*: identificação das interfaces providas dos componentes da camada de sistema a partir dos casos de uso; identificação dos componentes de negócio.

   (b) *Interação dos Componentes*: identificação das interfaces requeridas dos componentes de sistema e das interfaces providas dos componentes de negócio, a partir da criação de um diagrama de colaboração para cada operação das interfaces providas dos componentes de sistema.

   (c) *Especificação Final dos Componentes*: refatoração das interfaces de cada componente (divisão ou agrupamento), mantendo a coesão e evitando o excesso de

interfaces; formalização dos contratos das interfaces providas dos componentes de sistema.

3. *Provisionamento (Provisioning)*: implementação ou aquisição dos componentes especificados.

4. *Montagem*: integração dos componentes de acordo com a arquitetura especificada. Caso seja necessário, nesta fase são construídos conectores para adaptações entre interfaces providas e requeridas.

5. *Testes*: aplicação dos testes de sistema.

6. *Instalação*: início da operação do sistema no ambiente do usuário.

Com as fases bem definidas, o processo facilita a utilização de componentes na construção de sistemas computacionais. Sua principal limitação, porém, é a pequena preocupação com a reutilização de componentes: apenas na fase de provisionamento a aquisição de componentes é considerada. Como as interfaces requeridas já foram especificadas, geralmente é necessário um uso maciço de conectores e envólucros (*wrappers)* para a adaptação das interfaces.

2.2 MDCE+

O método MDCE+ [BFR04] sistematiza a especificação e implementação do comportamento excepcional nas diversas fases do desenvolvimento de um sistema baseado em componentes. É um refinamento da MDCE [Fer01], cujo enfoque eram as fases de especificação de requisitos e análise. O MDCE+ tem como enfoque as fases de projeto arquitetural e implementação.

Todos os componentes especificados adotam a estrutura do componente ideal tolerante a falhas [AL90] como forma de separação entre os comportamentos normal e excepcional. O método foi incorporado ao *UMLComponents*, como ilustra a Figura 1. Além das atividades propostas pelo *UMLComponents* para cada uma de suas fases, foram acrescentadas funções específicas para determinação do comportamento excepcional pelo MDCE+:

1. *Especificação de Requisitos*: definição dos cenários excepcionais; especificação de pré, pós-condições e invariantes para cada caso de uso; classificação das entidades componentes do modelo do conceito do negócio em relação à criticidade, para escolha de quais terão mecanismos de tolerância a falhas implementados.

2. *Especificação dos Componentes*:

   (a) *Identificação dos Componentes*: juntamente com as interfaces providas dos componentes de sistema, identificação das exceções que podem ser lançadas por essas interfaces; criação de uma interface para cada exceção, encapsulando seus diversos tratadores; criação dos componentes ideais: cada componente normal é associado ao seu componente excepcional específico, que funciona como um conector entre o componente normal e os tratadores das suas possíveis exceções.

(b) *Interação dos Componentes*: descoberta das exceções surgidas a partir da das interações entre os componentes e especificação de seus tratadores. O diagrama de colaboração da *UMLComponents* foi substituído pelo diagrama de atividades, que possibilita tanto a descoberta das operações das interfaces requeridas quanto das exceções lançadas por estas interfaces.

(c) *Especificação Final dos Componentes*: formalização dos contratos definidos na fase de especificação de requisitos (opcional), e avaliação para a criação de conectores que tratem possíveis exceções através de conversão de tipos.

3. *Provisionamento (Provisioning)*: implementação/aquisição dos envólucros (*wrappers*) para adaptação dos componentes adquiridos, com a contextualização das exceções para as especificidades da aplicação.

4. *Montagem*: identificação do comportamento excepcional dos conectores e sua implementação.

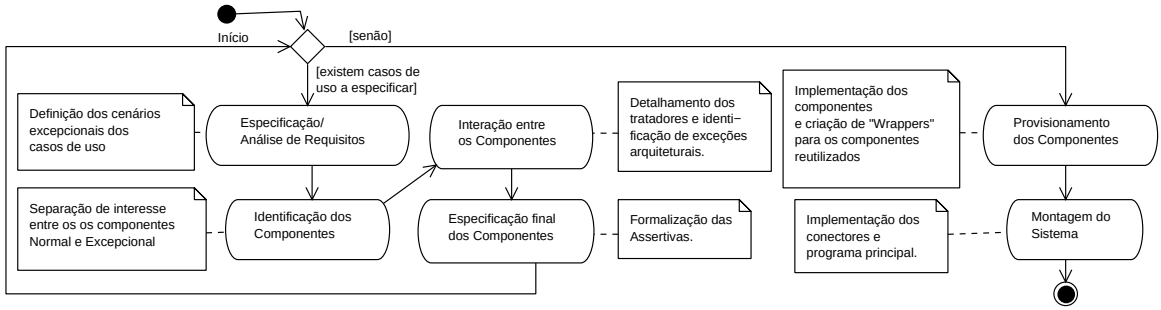

Figura 1: Extensão do *UMLComponents* pelo método MDCE+ [BFR04].

O *overhead* para a especificação detalhada das exceções e dos tratadores é compensado pela melhoria da qualidade no tratamento das situações excepcionais, com a distribuição das atividades durante todo o processo de desenvolvimento. Esta contextualização das exceções facilita a identificação, fazendo com que um maior número de situações excepcionais seja prevista e, conseqüentemente, aumentando a segurança no funcionamento (*dependability*), essencial para sistemas críticos.

Maiores detalhes sobre cada uma das fases são fornecidos nas Seções seguintes.

2.3 Método de Testes

O método de testes fornece diretrizes para testes de componentes, espalhando as atividades nas diversas fases do desenvolvimento. Inicialmente, o método abrange apenas a fase de testes de componentes. Futuramente, as fases de testes de integração e sistema também serão cobertas.

A base do método é a combinação da melhoria da testabilidade do componente com a automatização da geração e execução dos casos de teste. A melhoria da testabilidade é

obtida com a implementação da arquitetura do componente testável [RM04], que pode ser vista na Figura 2.

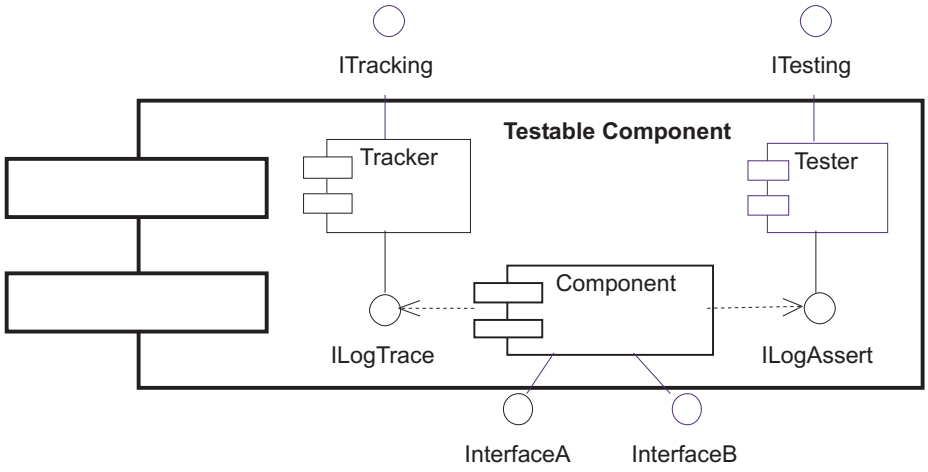

Figura 2: Arquitetura do componente testável.

A arquitetura do componente testável que inclui mecanismos de monitoração e de verificação de assertivas no componente sob teste com a utilização de programação orientada a aspectos [K+97], que permite a instrumentação mesmo quando o código fonte não está disponível. São empacotados o componente sob teste, o componente Tracker, responsável pelos mecanismos de monitoração, e o componente Tester, responsável pelos mecanismos de verificação de assertivas.

Também fazem parte da arquitetura do componente testável as especificações contratual e comportamental do componente. A especificação contratual contém o contrato do componente na linguagem OCL (*Object Constraint Language*), e é utilizada para geração automática das assertivas a serem incluídas no código do componente sob teste.

A especificação comportamental tem a forma do diagrama de atividades da UML [**?**] e é utilizada na geração automática de casos de teste. Os casos de teste consistem de percursos no diagrama de atividades, e são automaticamente gerados em uma linguagem intermediária que pode ser traduzida para *scripts* de testes em linguagens específicas.

A criação da arquitetura do componente testável, das especificações e da execução dos casos de testes é disposta ao longo das fases do MDCE+, aproveitando os documentos produzidos durante o desenvolvimento, minimizando o tempo para criação dos casos de teste. As atividades são:

1. *Especificação de Requisitos*: por cobrir apenas a fase de testes de componentes, na fase de especificação de requisitos há apenas a construção do plano de testes, onde são definidos, dentre outros aspectos, os recursos a serem alocados para os testes, cronograma, e abrangência dos testes. A abrangência consiste em quais componentes transformados em componentes testáveis, com base no modelo do conceito do negócio produzido nesta fase.

2. *Especificação dos Componentes*:
   (a) *Identificação dos Componentes*: nesta fase a definição dos componentes a serem tornados testáveis é finalizada, com a definição dos componentes do sistema.
   (b) *Interação dos Componentes*: na fase de interação inicia-se a criação das especificações comportamentais, ilustrando o relacionamento do componente com suas interfaces requeridas.
   (c) *Especificação Final dos Componentes*: as atividade de teste para esta fase são a produção da especificação contratual, com a formalização dos contratos dos componentes (presente também nas atividades propostas pelo MDCE+, e por isso realizada em conjunto com o desenvolvimento); e a finalização da especificação comportamental, com a inclusão da especificação do fluxo de execução das interfaces requeridas.
3. *Provisionamento (Provisioning)*: paralelamente à implementação dos componentes, são gerados os componentes Tester e Tracker, os casos de teste no formato intermediário, e na linguagem de programação específica. Após a implementação, os testes são executados e defeitos encontrados são encaminhados para a manutenção.
4. *Montagem*: como o método ainda não cobre os testes de integração, nesta fase o ciclo é repetido para os conectores que foram especificados, gerando a arquitetura do componente testável e os casos de teste automatizados para os conectores de maior complexidade. A aplicação dos testes também é realizada nesta fase.

Maiores detalhes sobre cada uma das fases são fornecidos nas Seções seguintes.

3 Descrição do Sistema

O sistema desenvolvido no estudo de caso pertence ao domínio de sistemas financeiros bancários. Foram especificados para ele seis funcionalidades básicas: (i)requisição de talão de cheques; (ii) entrega de talão de cheques; (iii) sustação de cheques; (iv) captura de cheque para compensação; (v) cancelamento do contrato da conta; e (vi) cadastramento de limite adicional.

Duas dessas funcionalidades: (i) a responsável pelo cancelamento do contrato da conta; e (ii) a responsável pela sustação de cheques, devem estar disponíveis o máximo de tempo possível. Essa preocupação tem o objetivo de evitar a utilização de um crédito que já não esteja disponível, quer seja por parte do cliente (utilização do limite) ou de terceitos (resgate de cheques cancelados). Por essa razão, assume-se que essas funcionalidades possuem requisitos críticos de disponibilidade.

Por apresentar uma preocupação maior com os atributos de qualidade do sistema, no contexto dessa dissertação, será descrito o desenvolvimento da quinta funcionalidade citada (cancelamento do contrato da conta). Mas antes do detalhamento dos procedimentos de especificação e implementação, a Seção 4 descreve o estudo de caso em si, incluindo o ambiente de trabalho utilizado para a sua condução. Essa descrição do ambiente abrange tanto a descrição ferramental, quanto do perfil dos executores.

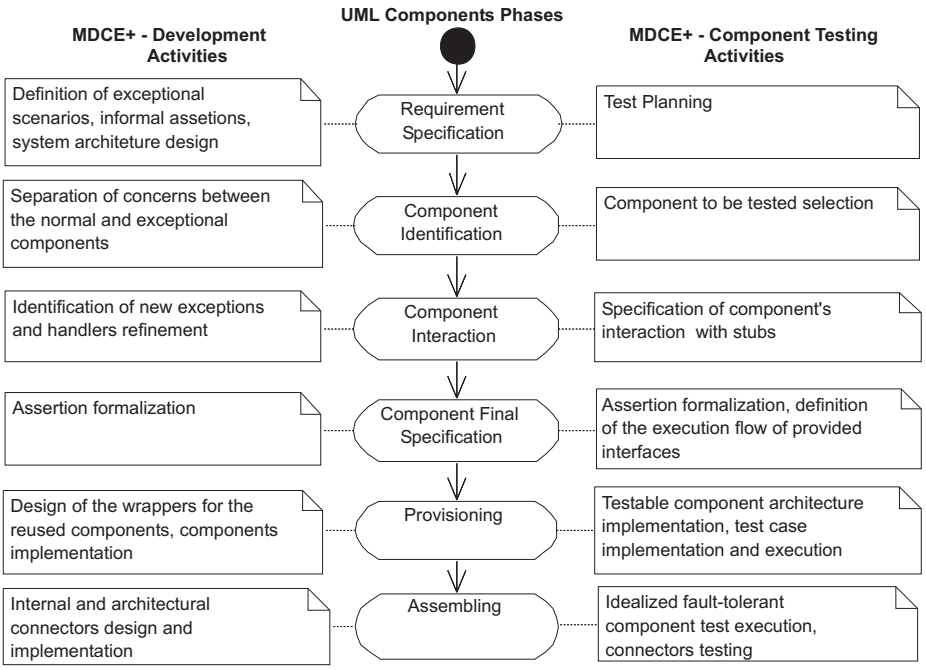

Figura 3: Atividades de desenvolvimento e testes de exceções no processo *UML components*

4 Descrição do Estudo de Caso

O estudo de caso consistiu basicamente da aplicação do processo UML Components adaptado com a especificação do comportamento excepcional no desenvolvimento do sistema financeiro descrito na Seção 3. O estudo foi planejado pelos autores e executado por duas pessoas distintas, através de uma parceria firmada com uma empresa cituada em São Paulo. Por considerar que a parte mais crítica do desenvolvimento, do ponto de vista da análise do processo, é a especificação do sistema, ela foi desempenhada por dois funcionários da empresa. Apesar do contato anterior de um dos executores com o UML Components, por se tratar de um método novo, a familiaridade com o MDCE+ não existia.

Uma característica importante da equipe de desenvolvimento é o fato de uma das pessoas envolvidas ser um especialista no domínio do negócio, tendo mais de 10 anos de experiência. Esse fato facilitou a identificação de um número maior de exceções relativas às regras do negócio.

Com o intuito de proporcionar uma maior interação entre os integrantes da equipe de desenvolvimento, durante a fase de especificação os modelos foram desenhados manualmente, isto é, sem o auxílio de ferramentas CASE.

Finalizada a especificação, a próxima etapa foi a etapa de implementação dos componentes do sistema. Por ser considerada uma etapa de tradução dos modelos, ela foi desempenhada pelo próprio autor da dissertação.

Com o sistema implementado, chegou a hora de se aplicar os testes unitários, que foram especificados paralelamente ao desenvolvimento.

5 Especificação de Requisitos

A fase de especificação dos requisitos descreve as funcionalidades que devem ser oferecidas pelo sistema através de casos de uso. Para a execução desta fase, é imprescindível a interação com o cliente interessado no sistema, uma vez que a principal variável para o seu sucesso é o conhecimento da lógica do negócio e a definição clara dos objetivos do sistema.

Relativo ao desenvolvimento, a especificação de requisitos é composta por duas fases:

1. Modelo do conceito do negócio:

   O modelo do conceito do negócio representa as principais entidades do negócio e o relacionamento entre elas, e é construído de maneira livre, sem a preocupação com boas práticas de modelagem como alta coesão e baixo acoplamento, por exemplo. Esse modelo é útil para promover a compreensão do domínio que é necessária para a especificação do sistema. Após a especificação desse modelo, são identificadas as entidades críticas, isto é, as entidades que são consideradas essenciais para a execução das principais atividades do negócio.

2. Elaboração dos casos de uso:

   Cada caso de uso, além do diagrama, é composto por:

   (a) Descrição do caso de uso

   (b) Lista dos dados de entrada (informal)

   (c) Lista de atores participantes

   (d) Restrições, especificadas através de assertivas, isto é, pré-, pós-condições, invariantes e restrições gerais de negócio

   (e) Cenários principais, alternativos e excepcionais: os cenário excepcionais são classificados em recuperáveis, quando representam uma tentativa de recuperar o estado normal do sistema, e de falha, quando representam um comportamento paliativo sem a possibilidade de recuperação do estado normal.

   (f) Exceções agendadas: lista das exceções a serem lançadas por esse caso de uso, obtidas a partir da análise de possíveis violações das assertivas e dos cenários. As exceções são agendadas a fim de serem especificadas na próxima fase (Seção 6.1).

Como o método de testes por enquanto cobre apenas o teste de componentes, nesta fase não há atividades de teste. Um trabalho futuro seria a criação de atividades para a especificação dos testes de sistema.

5.1 Modelo Conceitual do Negócio (*Business Concept Model*)

O modelo conceitual do negócio pode ser visto na figura 4. Como mostrado na Seção 2.1, esse modelo representa as entidades básicas da lógica do negócio e os relacionamentos entre elas.

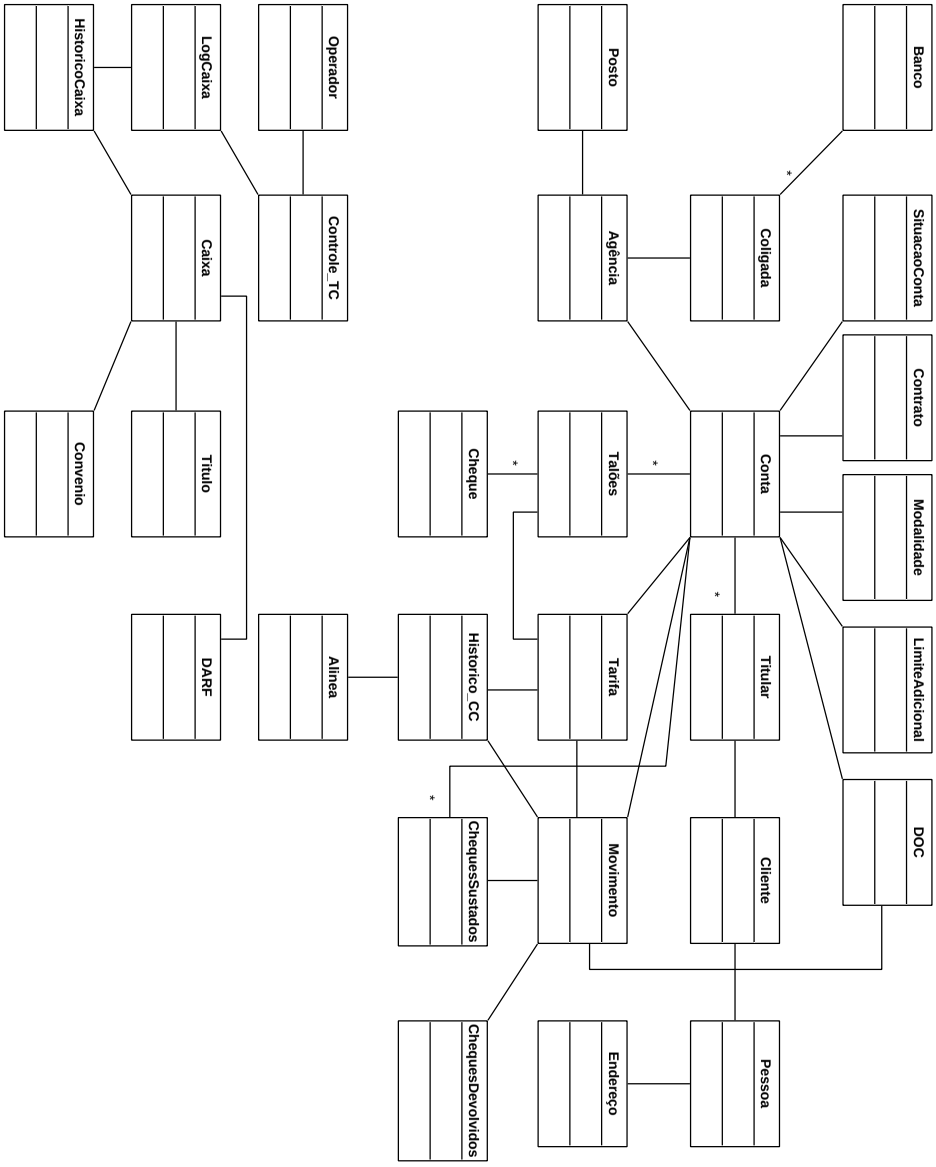

Figura 4: Modelo Conceitual do Negócio

5.2 Casos de Uso

O diagrama de casos de uso, que contém a lista de funcionalidades desejadas ao sistema, pode ser visto na figura 5. As subseções a seguir trazem a especificação de cada caso de uso separadamente.

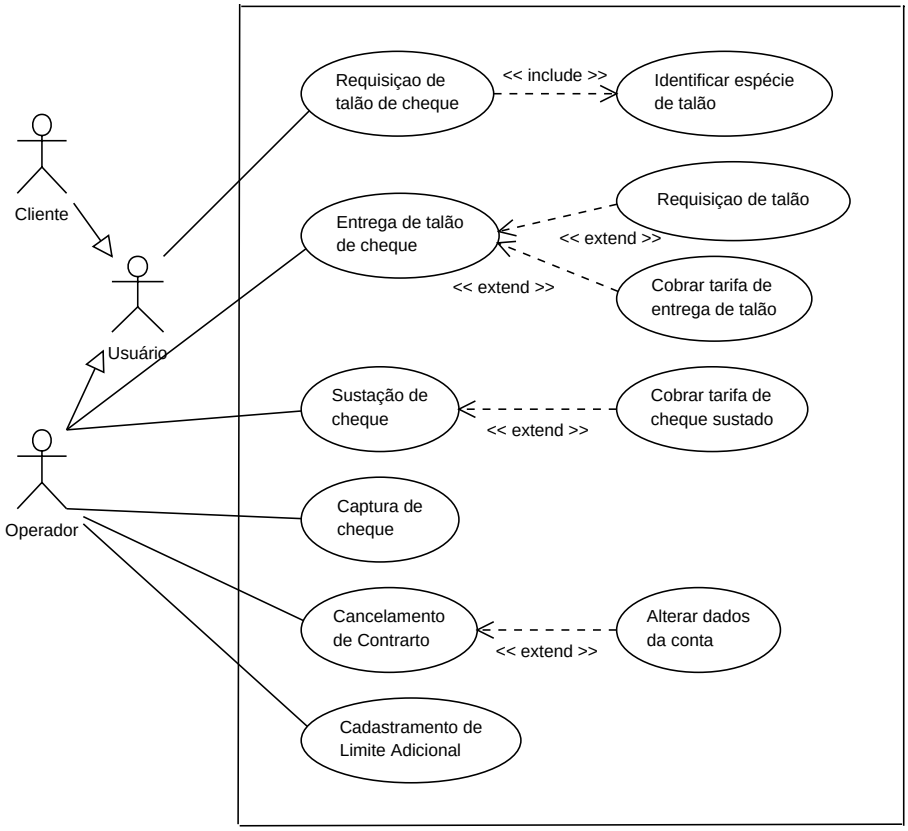

Figura 5: Diagrama de Casos de Uso

5.2.1 Casos de Uso: Requisição de Talão de Cheque

Descrição: Registra a solicitação de talão para o cliente.

Dados de Entrada: Nome do usuário, código da agência, número da conta, código do cliente1, código do cliente2, quantidade de talões, tipo do talão (normal/tb/sanfonado), tipo de solicitação (automática/manual).

Atores: Usuário (cliente ou operador).

Pré-Condições:

- A agência da conta tem que estar aberta;
- A agência da conta deve ser != NULL E != ' ' E tem que estar cadastrada;
- A conta deve ser != NULL E != ' ' E tem que estar cadastrada;

- O nome do usuário deve ser != NULL E != ' ';
- O código do cliente1 deve ser != NULL E != ' ';
- O cliente1 deve ser o titular da conta;
- A quantidade deve ser ¿ 0;
- O tipo é obrigatório E deve ser um dos seguntes valores (normal/tb/sanfonado);
- O tipo de solicitação é obrigatório E deve ser um dos seguintes valores (automático/manual);
- O código do cliente2 deve ser == NULL OU == ' ' OU (!= NULL E != ' ' E o cliente2 ser titular da conta);
- O código do cliente2 deve ser != do código do ciente1;
- A Conta não pode estar bloqueada para requisição de talão;
- Coligada deve estar cadastrada;
- Modalidade deve estar cadastrada;

Cenário Principal:

- 01- Recuperar coligada;
- 02- Recuperar a agência da conta;
- 03- Recuperar a conta;
- 04- Recuperar a modalidade da conta;
- 05- <<include>> Identificar especie de Talão;
- 06- Identificar tipo de talão;
- 07- Incluir registro da requisição de talão;

Invariantes:

- A agência da conta deve estar aberta;

Pós-Condições:

- O cliente possui uma requisição de talão válida;

Outras restrições de negócio

- Deve-se verificar se a conta pode requisitar talão (depende da especificação da espécie do talão);

Cenário Excepcional 1:

- *Tipo de Cenário:* Recuperável;
- *Condição excepcional:* Se (existir talão E este estiver excluído)
  - 07.1- Atualizar registro de talões com os novos dados da requisição;

Cenário Excepcional 2:

- *Tipo de Cenário:* De falha;
- *Condição excepcional:* 'depende de cada exceção (cenário genérico)'
  - X.1- Lançar exceção;

Exceções Agendadas

***E1 - Agência não está aberta* - nova**
Passo onde pode ser detectada: 2-2º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: Código da agência;
***E2 - Agência inválida* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: Código da agência;
***E3 - Agência Não cadastrada* - nova**
Passo onde pode ser detectada: 2-1º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: Código da agência;
***E4 - Conta inválida* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;

Classificação: Cenário excepcional (2);
Informação de contexto: Número da conta;

***E5 - Conta não cadastrada* - nova**
Passo onde pode ser detectada: 3-1º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: Código da agência; Número da conta;

***E6 - Usuário inválido* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: Nome do usuário;

***E7 - Código do cliente inválido* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: Código do cliente;

***E8 - Cliente não é titular da conta* - nova**
Passo onde pode ser detectada: 3-3º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: Código do cliente; Código da agência; Número da conta;

***E9 - A quantidade de talão requisitada deve ser maior que zero* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: NENHUMA;

***E10 - Tipo de talao invalido* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: Tipo de talão;

***E11 - Tipo de solicitação inválido* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: Tipo de solicitação;

***E12 - Clientes iguais* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: Código do cliente1; Código do cliente2;

***E13 - Requisição de talão bloqueada para esta conta* - nova**
Passo onde pode ser detectada: 3-2º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: NENHUMA;

***E14 - Requisição de talão impossibilitada para conta interna* - nova**
Passo onde pode ser detectada: 4-2º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: NENHUMA;

***E15 - Transação não foi concluída* - nova**
Passo onde pode ser detectada: 7.1 E 7.1.1;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: Nome da transação;

E16

E17

***E18 - Coligada não cadastrada* - nova**
Passo onde pode ser detectada: 1;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: Código da coligada;

***E19 - Modalidade não cadastrada* - nova**
Passo onde pode ser detectada: 4-1º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: Código da modalidade;

***E57 - Requisição já existente* - nova**
Passo onde pode ser detectada: 7;
Gravidade: Moderada. A execução pode ser recuperada;
Local de identificação: Requisitos;

Classificação: Cenário excepcional (1);
Informação de contexto: Código da modalidade;

5.2.2 Casos de Uso: Entrega de talão de cheque

Descrição: Registra a entrega do talão ao cliente.

Dados de Entrada: Nome do usuário, código da agência, código da agência local, número da conta, espécie do talão, número do primeiro cheque, indicação se é tarifado.

Atores: Operador.

Pré-Condições:

- A agência da conta tem que estar aberta;
- A agência da conta deve ser != NULL E != ' ' E tem que estar cadastrada;
- A conta deve ser != NULL E != ' ' E tem que estar cadastrada;
- A especie deve ser != NULL E != ' ';
- O número do primeiro cheque deve ser != NULL E != ' ';
- A agência local deve ser != NULL E != ' ';
- O nome do usuário deve ser != NULL E != ' ';
- Tem que existir o talão em estoque;
- A conta não pode estar bloqueada para a entrega de talão;

Cenário Principal:

- 01- Recuperar a agência da conta;
- 02- Recuperar a conta;
- 03- Executar a entrega do talão;
- 04- Se o talão estiver bloqueado
- 04.1- Desbloquear talão;
- 05- Se a conta permite solicitação automática de talão E é necessário a solicitação (estoque menor do que o indicado na conta)
- 05.1- <<extend>> Requisição de talão;
- 06- Se é indicado como tarifado (parâmetro)

- 06.1- <<extend>> Cobrar tarifa de entrega de talão

Invariantes:

- A agência da conta deve estar aberta;

Pós-Condições:

- A quantidade de talão em estoque foi reduzita de 1;
- Se ocorreu a solicitação automática, o cliente possuirá 'N' talões requisitados, onde 'N=Quantidade de talões em estoque registrado na conta - quantidade de talões em estoque realmente';

Cenário Excepcional 1:

- *Tipo de Cenário:* De falha;
- *Condição excepcional:* 'depende de cada exceção (cenário genérico)'
  - X.1- Lançar exceção;

Exceções Agendadas

***E1 - Agência não está aberta* - já agendada**
Passo onde pode ser detectada: 1-2º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da agência;

***E2 - Agência inválida* - já agendada**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da agência;

***E3 - Agência Não cadastrada* - já agendada**
Passo onde pode ser detectada: 1-1º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da agência;

***E4 - Conta inválida* - já agendada**
Passo onde pode ser detectada: 0;

Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Número da conta;

***E5 - Conta não cadastrada* - já agendada**
Passo onde pode ser detectada: 2-1º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da agência; Número da conta;

***E6 - Usuário inválido* - já agendada**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Nome do usuário;

***E15 - Transação não foi concluída* - já agendada**
Passo onde pode ser detectada: 7(final);
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Nome da transação;

***E20 - Espécie Inválida* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Espécie;

***E21 - Número do primeiro cheque inválido* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Número do primeiro cheque;

***E22 - Não existe talão em estoque* - nova**
Passo onde pode ser detectada: 2-3º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: NENHUMA;

***E23 - Talão já foi entregue* - nova**
Passo onde pode ser detectada: 2-4º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;

Classificação: Cenário excepcional (1);
Informação de contexto: NENHUMA;

***E24 - Conta bloqueada para entrega de talão* - nova**
Passo onde pode ser detectada: 2-2º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: NENHUMA;

5.2.3 Casos de Uso: Sustação de cheque

Descrição: Registra a susteção de um ou mais cheques.

Dados de Entrada: Código da agência, código da agência local, número da conta, espécie do talão, número do primeiro cheque, quantidade de cheques, motivo da sustação (roubo matole / outros / perda / roubo), situação do cheque (em branco / preenchido e datado / preenchido e não datado), data da sustação, indicador se é para tarifar.

Atores: Operador.

Pré-Condições:

- A agência da conta tem que estar aberta;
- A agência da conta deve ser != NULL E != ' ' E tem que estar cadastrada;
- A conta deve ser != NULL E != ' ' E tem que estar cadastrada;
- O número do primeiro cheque deve ser != NULL E != ' ';
- A agência local deve ser != NULL E != ' ';
- A quantidade de cheque a sustar deve ser maior que zero;
- O motivo da sustação deve ser != NULL E != ' ' E deve ser válido (roubo malote / outros / perda / roubo);
- A situação dos cheques deve ser != NULL E != ' ' E deve ser válida (em branco / preenchido e datado / preenchido e não datado);
- A data da sustação deve ser != NULL E != ' ' E deve ser válida;
- O cheque não pode estar sustado;

Cenário Principal:

- 01- Recuperar a agência da conta;

- 02- Recuperar a conta;
- 03- Para cada cheque
- 03.1- Verifiar se o cheque está sustado (REFINADO NA ESPECIFICAÇÃO);
- 03.2- Inserir na lista de cheques sustados;
- 03.3- Se há a indicação de que o transação é tqarifada
- 03.3.1- <<extend>> Cobrar tarifa de cheque sustado;

Invariantes:

- A agência da conta deve estar aberta;

Pós-Condições:

- Os cheques entre 'número inicial' e 'número inicial + quantidade - 1' estão sustados;

Cenário Excepcional 1:

- *Tipo de Cenário:* De falha;
- *Condição excepcional:* 'depende de cada exceção (cenário genérico)'
  - X.1- Lançar exceção;

Exceções Agendadas

***E1 - Agência não está aberta* - já agendada**
Passo onde pode ser detectada: 1-2º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da agência;
***E2 - Agência inválida* - já agendada**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da agência;
***E3 - Agência Não cadastrada* - já agendada**
Passo onde pode ser detectada: 1-1º;
Gravidade: Grave. A execução não pode prosseguir;

Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da agência;

***E4 - Conta inválida* - já agendada**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Número da conta;

***E5 - Conta não cadastrada* - já agendada**
Passo onde pode ser detectada: 2-1º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da agência; Número da conta;

***E15 - Transação não foi concluída* - já agendada**
Passo onde pode ser detectada: 4(final);
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Nome da transação;

***E21 - Número do primeiro cheque inválido* - já agendada**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Número do primeiro cheque;

***E25 - Quantidade de cheques inválida* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Quantidade de cheques;

***E26 - Motivo de sustação inválido* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Motivo de sustação;

***E27 - Situação de sustação inválida* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);

Informação de contexto: Situação de sustação;

***E28 - Data inválida* - nova**

Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Data;

***E29 - Cheque já está sustado* - nova**

Passo onde pode ser detectada: 2-2º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Número do cheque;

5.2.4 Casos de Uso: Captura de cheque

Descrição: Registra os cheques capturados em uma operação de depósito em cheque.
Dados de Entrada: Data do movimento, número da captura, origem, valor, comp, banco, agência, C1DV2, conta, C1DV1, número do cheque, C3DV3, tipo, indicação do cheque conveniado.
Atores: Operador.
Pré-Condições:

- Data de movimento válida;
- O número de captura deve ser != ' 'E != NULL;
- O valor deve ser > 0;
- Comp. deve ser != ' ' E != NULL;
- O banco deve ser != ' ' E != NULL;
- A agência deve ser != ' ' E != NULL;
- O C1DV2 deve ser != ' ' E != NULL;
- A conta deve ser != ' ' E != NULL;
- O C2DV1 deve ser != ' ' E != NULL;
- O C3DV3 deve ser != ' ' E != NULL;
- O número do cheque deve ser != ' ' E != NULL;
- O cheque não pode ter sido capturado na data do de movimento indicada;
- A instituição financeira do cheque deve estar 'funcionando' no sistema de compensação;

- Os dados do cheque devem estar consistentes (verificação dos campos de verificação);

Cenário Principal:

- 01- Calcular a seqüência do cheque;
- 02- Inserir o cheque na lista de cheques capturados;

Invariantes:

- NÃO TEM;

Pós-Condições:

- A quantidade de cheques capturados foi incrementada em 1;

Cenário Excepcional 1:

- *Tipo de Cenário:* De falha;
- *Condição excepcional:* 'depende de cada exceção (cenário genérico)'
  - X.1- Lançar exceção;

Exceções Agendadas

***E15 - Transação não foi concluída* - já agendada**
Passo onde pode ser detectada: 2(final);
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Nome da transação;
***E28 - Data inválida* - já agendada**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Data;
***E30 - Número da captura inválido* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Número da captura;

***E31 - Valor do cheque inválido* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Valor; Número do cheque;

***E32 - Comp do cheque inválido* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Comp; Número do cheque;

***E33 - Banco do cheque inválido* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Banco; Número do cheque;

***E34 - Agência do cheque inválida* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Agência; Número do cheque;

***E35 - C1DV2 do cheque inválido* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: C1DV2; Número do cheque;

***E36 - Conta do cheque inválida* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Conta; Número do cheque;

***E37 - C2DV1 do cheque inválido* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: C2DV1; Número do cheque;

***E38 - Número do cheque inválido* - nova**
Passo onde pode ser detectada: 0;

Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Número do cheque;

E39 - Cheque já capturado neste dia (data do movimento) **- nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Número do cheque; Data de movimento;

E40 - Instituição financeira do cheque não está funcionando no sistema de compensação **- nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código do banco;

E41 - C3DV3 do cheque inválido **- nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: C3DV3; Número do cheque;

E42 - C1DV2 do cheque inconsistente **- nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: C1DV2; Número do cheque;

E43 - C2DV1 do cheque inconsistente **- nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: C2DV1; Número do cheque;

E44 - C3DV3 do cheque inconsistente **- nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: C3DV3; Número do cheque;

5.2.5 Casos de Uso: Cancelamento de Contrato

Descrição: Efetua o cancelamento do contrato da conta, mas se a conta foi cadastrada no dia, efetua-se apenas uma alteração da modalidade e das taxas dessa nova modalidade.

Dados de Entrada: Nome do usuário, código da agência, número da conta, código da modalidade ori., indicativo se o cadastro foi feito no dia, indicativo se é isento de jurAdp, indicativo se é isento de jurSaq, indicativo se é a taxa padrão Adp, indicativo se é a taxa Padrão Saq, taxa Adp, taxa Saq.

Atores: Operador.

Pré-Condições:

- A agência da conta tem que estar aberta;
- O nome do usuário deve ser != NULL E != ' ';
- A agência da conta deve ser != NULL E != ' ' E tem que estar cadastrada;
- A conta deve ser != NULL E != ' ' E tem que estar cadastrada;
- O código da modalidade ori. deve ser != NULL E != ' ' E tem que estar cadastrado;
- O código da modalidade ori. deve ser != da que está na conta;
- O código da modalidade ori. deve ser != NULL E != ' ' e dentro das regras de negócio;
- O tipo de pessoa da modalidade ori. tem que corresponder ao mesmo tipo de pessoa do TIPO DE DEPÓSITO;
- As taxas devem ser != NULL E != ' ' E devem estar dentro dos limites da conta;
- O contrato não pode estar cancelado;

Cenário Principal:

- 01- Recuperar a agência da conta;
- 02- Recuperar a conta;
- 03- Recuperar a modalidade Ori;
- 04- Recuperar tipo de depósito;
- 05- Se NÃO cadastrado no dia, então
- 05.1- Cancelar contrato;

Cenário Alternativo 1:

- *Condição de ativação:* Se cadastrado o dia
  - 05.1- <<extend>> Alterar dados da conta;

Invariantes:

- A agência da conta deve estar aberta;

Pós-Condições:

- Se cadastrado no dia, alguns campos da conta serão atualizados;
- Senão, deverá ser alterado o estado do contrato para cancelado;

Cenário Excepcional 1:

- *Tipo de Cenário:* De falha;
- *Condição excepcional:* 'depende de cada exceção (cenário genérico)'
  - X.1- Lançar exceção;

Cenário Excepcional 2: (IDENTIFICADO NA ESPECIFICAÇÃO - interação)

- *Tipo de Cenário:* Recuperável;
- *Condição de ativação:* Se não for possível o cancelamento do contrato / alteração dos cados da conta
  - 05.1.1- Reconfigurar os componentes do sistema;
  - 05.1.2- Tentar a execução novamente (N vezes);
  - 05.1.3- Se não conseguir
  - 05.1.3.1- Lançar exceção E56
- *Localização de tratamento:* Arquitetural

Exceções Agendadas

***E1 - Agência não está aberta* - já agendada**
Passo onde pode ser detectada: 1-2º;

Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da agência;

***E2 - Agência inválida* - já agendada**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da agência;

***E3 - Agência Não cadastrada* - já agendada**
Passo onde pode ser detectada: 1-1º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da agência;

***E4 - Conta inválida* - já agendada**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Número da conta;

***E5 - Conta não cadastrada* - já agendada**
Passo onde pode ser detectada: 2-1º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da agência; Número da conta;

***E6 - Usuário inválido* - já agendada**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Nome do usuário;

***E15 - Transação não foi concluída* - já agendada**
Passo onde pode ser detectada: 6 (final);
Gravidade: Moderada. Há a tentativa de recuperar o estado do sistema;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (2);
Informação de contexto: Nome da transação;

***E19 - Modalidade não cadastrada* - já agendada**
Passo onde pode ser detectada: 3;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;

Classificação: Cenário excepcional (1);
Informação de contexto: Código da modalidade;

***E45 - Modalidade inválida* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: modalidade;

***E46 - Modalidade igual a modalidade da conta* - nova**
Passo onde pode ser detectada: 2-3º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: NENHUMA;

***E47 - Tipo de pessoa da modalidade não corresponde ao tipo de pessoa da conta* - nova**
Passo onde pode ser detectada: 4-2º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: NENHUMA;

***E48 - Contrato já está cancelado* - nova**
Passo onde pode ser detectada: 2-2º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: NENHUMA;

***E50 - A taxa deve sr maior que zero* - já agendada**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Valor da taxa;

***E53 - Tipo de depósito não cadastrado* - nova**
Passo onde pode ser detectada: 4-1º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Especificação;
Classificação: Cenário excepcional (1);
Informação de contexto: Tipo de depósito;

***E54 - Taxa inválida* - nova**
Passo onde pode ser detectada: 2;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Especificação;
Classificação: Cenário excepcional (1);

Informação de contexto: Valor da taxa; Identificador da taxa;
***E56 - Reconfiguração Impossibilitada* - nova**
Passo onde pode ser detectada: Excepcional 2 - 05.1.3;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Especificação;
Classificação: Cenário excepcional (1);
Informação de contexto: Nome da operação; Nome do componente falho; Nome do componente alternativo; Causa do erro;

5.2.6 Casos de Uso: Cadastramento de Limite Adicional

Descrição: Cadastrar um limite de crédito adicional para a conta do cliente.

Dados de Entrada: Nome do usuário, código da agência, número da conta, limite adicional, taxa.

Atores: Operador.

Pré-Condições:

- A agência da conta tem que estar aberta;
- A agência da conta deve ser != NULL E != ' ' E tem que estar cadastrada;
- A conta deve ser != NULL E != ' ' E tem que estar cadastrada;
- O limite deve ser ¿ 0;
- A taxa deve ser ¿ 0;
- A modalidade deve permitir o uso de limite adicional;
- O usuário deve ser != NULL E != ' ';
- A conta não pode ter outro limite adicional;

Cenário Principal:

- 01- Recuperar a agência da conta;
- 02- Recuperar a conta;
- 03- Recuperar a modalidade da conta;
- 04- Se NÃO existe limite adicional
- 04.1- Incluir o limite adicional;
- 04.2- Atualizar o limite da conta;

Invariantes:

- A agência da conta deve estar aberta;

Pós-Condições:

- É atribuido um limite adicional na conta;

Cenário Excepcional 1:

- *Tipo de Cenário:* De falha;
- *Condição excepcional:* 'depende de cada exceção (cenário genérico)'
  - X.1- Lançar exceção;

Exceções Agendadas

***E1 - Agência não está aberta* - já agendada**
Passo onde pode ser detectada: 1-2º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da agência;
***E2 - Agência inválida* - já agendada**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da agência;
***E3 - Agência Não cadastrada* - já agendada**
Passo onde pode ser detectada: 1-1º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da agência;
***E4 - Conta inválida* - já agendada**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Número da conta;

***E5 - Conta não cadastrada* - já agendada**
Passo onde pode ser detectada: 2-1º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da agência; Número da conta;

***E6 - Usuário inválido* - já agendada**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Nome do usuário;

***E15 - Transação não foi concluída* - já agendada**
Passo onde pode ser detectada: 5 E 4.1;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Nome da transação;

***E19 - Modalidade não cadastrada* - já agendada**
Passo onde pode ser detectada: 3-1º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Especificação;
Classificação: Cenário excepcional (1);
Informação de contexto: Código da modalidade;

***E49 - O limite deve ser maior que zero* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Valor do limite;

***E50 - A taxa deve sr maior que zero* - nova**
Passo onde pode ser detectada: 0;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Valor da taxa; Identificador da taxa;

***E51 - A modalidade não permite uso de limite adicional* - nova**
Passo onde pode ser detectada: 3-2º;
Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: Modalidade;

***E52 - A conta já possui um limite adicional* - nova**
Passo onde pode ser detectada: 4;

Gravidade: Grave. A execução não pode prosseguir;
Local de identificação: Requisitos;
Classificação: Cenário excepcional (1);
Informação de contexto: NENHUMA;

6 Especificação do Sistema

Na fase de especificação são realizados blablabla

Essa fase é dividida em três etapas: (i) identificação dos componentes, na qual os componentes do sistema são identificados a partir das suas interfaces; (ii) interação dos componentes, na qual a interação entre os componentes de negócio e de sistema é definida; e (iii) especificação final dos componentes, onde as especificações são refinadas e, se possível, representadas de uma maneira formal.

As interações do ciclo de desenvolvimento foram realizadas a cada subfase: na fase de Identificação dos Componentes, todos os casos de uso foram analisados e a fase foi finalizada, e assim nas outras duas subfases.

6.1 Identificação dos Componentes

Nesta fase são identificadas as interfaces e operações dos componentes da camada de sistema (atividade de desenvolvimento). Além disso, as exceções que foram agendadas anteriormente são finalmente especificadas através de classes para as exceções e de interfaces e operações para seus tratadores. Em relação aos componentes da camada de negócio, suas interfaces também são identificadas, ainda sem operações.

As subfases 1 e 2 são executadas separadamente para cada caso de uso. As demais são executadas apenas uma vez. São elas:

1. Interfaces e Operações de Sistema: *Atividade de desenvolvimento*

   Entradas: casos de uso completos (diagramas e detalhamentos).

   As interfaces de sistema são identificadas baseando-se nos casos de uso: para cada caso de uso, é criada uma interface. As operações, por sua vez, são deduzidas tanto a partir dos passos dos respectivos cenários normais (cenário principal e cenários alternativos), quanto das possíveis verificações das assertivas (pré-, pós-condições, invariantes e demais restrições do negócio).

   Saídas: interfaces de sistema e suas operações.

2. Especificação das Exceções Agendadas: *Atividade de desenvolvimento*

   Entradas: lista de exceções agendadas em cada caso de uso.

   Cada exceção agendada anteriormente é especificada. A especificação abrange tanto a criação de classes (para as exceções), quanto a criação de interfaces e métodos dos componentes excepcionais, onde serão implementados os tratadores. Neste estudo de caso, os executores optaram por criar um método tratador sempre que uma exceção é lançada. A principal vantagem de adotar essa política é o fato de simplificar a inserção de mecanismos de monitoramento (*logging*) das exceções com o uso de técnicas de pré-processamento, como por exemplo orientação a aspectos.

   Saídas: Classes para cada exceção, interfaces e métodos dos componentes excepcionais.

3. Modelo de Tipos do Negócio *(Business Type Model)*: *Atividade de desenvolvimento*

   Entradas: modelo do conceito do negócio.

   O modelo de tipos do negócio é baseado no modelo conceitual produzido da fase de requisitos. O objetivo deste novo modelo é refinar o modelo conceitual, restringindo-o apenas às entidades relevantes para o sistema. Essa informação pode ser obtida baseado nos casos de uso a serem implementados.

   Após esse refinamento, são destacadas as entidades principais *(core types)*, que são mais relevantes de acordo com o contexto específico do sistema. Essas entidades são identificadas pelo espeteótipo <<core>>. Todos os *core types* relacionados com alguma entidade crítica serão foco da metodologia no tocante à confiabilidade.

   Saídas: modelo conceitual refinado, com destaque para as entidades críticas.

4. Identificação das Interfaces de Negócio: *Atividade de desenvolvimento*

   Entradas: modelo de tipos do negócio.

   A descoberta das interfaces de negócio é baseada no modelo de tipos especificado anteriormente. Para cada entidade principal é criada uma interface, que futuramente conterá as operações requeridas dos componentes da camada de negócio.

   Saídas: interfaces dos componentes de negócio (sem as operações).

5. Criação dos Componentes Normais e Excepcionais: *Atividade de desenvolvimento*

   Entradas: interfaces de sistema, interfaces de negócio e modelo de tipos do negócio.

   Com as interfaces de sistema e de negócio criadas, o próximo passo é a criação dos componentes normais (de sistema e de negócio) e dos componentes excepcionais. A criação dos componentes consiste basicamente no agrupamento das interfaces coesas de uma mesma camada arquitetural em componentes.

   Saídas: arquitetura inicial do sistema.

6. Definição dos componentes a serem testados: *Atividade de testes*

   Entradas: modelo de tipos do negócio, arquitetura inicial do sistema.

   Nesta fase começam as atividades para o teste de componentes. Como geralmente o número de componentes é alto, não é possível testar todos. Alguns serão testados apenas durante os testes de integração e/ou sistema.

   Nesta fase são escolhidos, a partir da arquitetura inicial do sistema, quais componentes se tornarão testáveis. Recomenda-se que sejam escolhidos de acordo com os critérios:

   - Criticidade: componentes críticos devem ter seu funcionamento garantido;
   - Reusabilidade: como componentes reutilizáveis são testados a cada integração em um novo sistema, o *overhead* para a automação dos testes é compensado pelo número de repetições da execução dos testes;

- Volatilidade de requisitos: componentes com requisitos com alta probabilidade de serem modificados ao longo de seu ciclo de vida, exigindo a freqüente realização de testes de regressão.

Outra consideração a ser feita é relativa ao isolamento do componente durante os testes. Devem ser analisadas suas dependências relativas às interfaces requeridas, e para quais dessas interfaces vale a pena a criação de *stubs*. Caso o componente requerido seja muito simples e suas saídas sejam facilmente controláveis, não é justificado o *overhead* para a crição de um stub: é mais vantajoso que o componente requerido seja integrado ao componente sob teste durante a realização dos testes.

É importante ressaltar que as decisões tomadas durante esta fase ainda podem ser ajustadas, especialmente em relação à escolha de tratadores para se tornarem componentes testáveis, pois o seu comportamento detalhado ainda não é conhecido nesta fase, apenas durante a interação dos componentes.

Saídas: Lista de componentes a serem testados.

6.1.1 Caso de Uso: Requisição de Talão de Cheque

Interfaces e Operações de Sistema

A interface e as operações provenientes deste caso de uso podem ser vistos na Tabela 1.

Interface de Sistema: ISBReqTalaoMgr	Passo
Operações:	
? requisitarTalao(?)	0
? identificarTipoTalao(?)	6

Tabela 1: Interface e Operações – Caso de Uso *Requisição de Talão de Cheque*

Especificação das Exceções Agendadas

As classes de exceção e as interfaces dos tratadores criadas nesta fase podem ser vistas nas Tabelas 50 e 3.

6.1.2 Caso de Uso: Entrega de Talão de Cheque

Interfaces e Operações de Sistema

A interface e as operações provenientes deste caso de uso podem ser vistos na Tabela 4.

Especificação das Exceções Agendadas

	Exceção:	Interface do Tratador
E1	TEAgenciaNaoEstaAbertaException Atributos: codAgencia:String	ITEAgenciaNaoEstaAbertaException Operações: void lancar(E1 e) throws E1
E2	TEAgenciaInvalidaException Atributos: codAgencia:String	ITEAgenciaInvalidaException Operações: void lancar(E2 e) throws E2
E3	TEAgenciaNaoCadastradaException Atributos: codAgencia:String	ITEAgenciaNaoCadastradaException Operações: void lancar(E3 e) throws E3
E4	TEContaInvalidaException Atributos: numConta:String	ITEContaInvalidaException Operações: void lancar(E4 e) throws E4
E5	TEContaNaoCadastradaException Atributos: codAgencia:String numConta:String	ITEContaNaoCadastradaException Operações: void lancar(E5 e) throws E5
E6	TEUsuarioInvalidoException Atributos: nomeUsuario:String	ITEUsuarioInvalidoException Operações: void lancar(E6 e) throws E6
E7	TECodClienteInvalidoException Atributos: codCliente:String	ITECodClienteInvalidoException Operações: void lancar(E7 e) throws E7
E8	TEClienteNaoETitularContaException Atributos: codCliente:String codAgencia:String numConta:String	ITEClienteNaoETitularContaException Operações: void lancar(E8 e) throws E8
E9	TEQtdeTalaoDeveSerMaiorZeroException Atributos: –	ITEQtdeTalaoDeveSerMaiorZeroException Operações: void lancar(E9 e) throws E9

Tabela 2: Exceções, Interfaces e Operações – Caso de Uso *Requisição de Talão de Cheque*

As classes de exceção e as interfaces dos tratadores existentes que são relevantes para este estudo de caso podem ser vistas na Tabela 5. E as novas classes de exceções e interfaces de tratadores podem ser vistas na Tabela 6.

6.1.3 Caso de Uso: Sustação de Cheque

Interfaces e Operações de Sistema

A interface e as operações provenientes deste caso de uso podem ser vistos na Tabela 7.

Especificação das Exceções Agendadas

As classes de exceção e as interfaces dos tratadores existentes que são relevantes para este estudo de caso podem ser vistas na Tabela 8. E as novas classes de exceções e interfaces de tratadores podem ser vistas na Tabela 9.

	Exceção:	Interface do Tratador
E10	TETipoTalaoInvalidoException Atributos: tipoTalao:int	ITETipoTalaoInvalidoException Operações: void lancar(E10 e) throws E10
E11	TETipoSolicitacaoTalaoInvalidoException Atributos: tipoSolicitacao:int	ITETipoSolicitacaoTalaoInvalidoException Operações: void lancar(E11 e) throws E11
E12	TEClientesDevemSerDistintosException Atributos: codCliente1:String codCliente2:String	ITEClientesDevemSerDistintosException Operações: void lancar(E12 e) throws E12
E13	TERequisicaoTalaoEstaBloqueadaContaException Atributos: –	ITERequisicaoTalaoEstaBloqueadaContaException Operações: void lancar(E13 e) throws E13
E14	TERequisicaoTalaoImpossibilitadaContaInternaException Atributos: –	ITERequisicaoTalaoImpossibilitadaContaInternaException Operações: void lancar(E14 e) throws E14
E15	TETransacaoNaoFoiCohcluidaException Atributos: nomeTransacao:String	ITETransacaoNaoFoiCohcluidaException Operações: void lancar(E15 e) throws E15
E18	TEColigadaNaoCadastradaException Atributos: codColigada:String	ITEColigadaNaoCadastradaException Operações: void lancar(E18 e) throws E18
E19	TEModalidadeNaoCadastradaException Atributos: codModalidade:String	ITEModalidadeNaoCadastradaException Operações: void lancar(E19 e) throws E19
E57 *(interna)*	TERegistroJaExistenteException Atributos: codModalidade:String (interação)	ITERegistroJaExistenteException Operações: *(int)* void requisicaoExistente(E57 e) throws E15 void lancar (E57 e) throws E57

Tabela 3: Exceções, Interfaces e Operações – Caso de Uso *Requisição de Talão de Cheque*

Interface de Sistema: ISBEntregaTalaoMgr	Passo
Operações:	
? entregarTalao(?)	0

Tabela 4: Interface e Operações – Caso de Uso *Entrega de Talão de Cheque*

6.1.4 Caso de Uso: Captura de Cheque

Interfaces e Operações de Sistema

A interface e as operações provenientes deste caso de uso podem ser vistos na Tabela 10.

Especificação das Exceções Agendadas

As classes de exceção e as interfaces dos tratadores existentes que são relevantes para este estudo de caso podem ser vistas na Tabela 11. E as novas classes de exceções e interfaces de tratadores podem ser vistas nas Tabelas 12 e 13.

	Exceção:	Interface do Tratador
E1	TEAgenciaNaoEstaAbertaException Atributos: codAgencia:String	ITEAgenciaNaoEstaAbertaException Operações: void lancar(E1 e) throws E1
E2	TEAgenciaInvalidaException Atributos: codAgencia:String	ITEAgenciaInvalidaException Operações: void lancar(E2 e) throws E2
E3	TEAgenciaNaoCadastradaException Atributos: codAgencia:String	ITEAgenciaNaoCadastradaException Operações: void lancar(E3 e) throws E3
E4	TEContaInvalidaException Atributos: numConta:String	ITEContaInvalidaException Operações: void lancar(E4 e) throws E4
E5	TEContaNaoCadastradaException Atributos: codAgencia:String numConta:String	ITEContaNaoCadastradaException Operações: void lancar(E5 e) throws E5
E6	TEUsuarioInvalidoException Atributos: nomeUsuario:String	ITEUsuarioInvalidoException Operações: void lancar(E6 e) throws E6
E15	TETransacaoNaoFoiCohcluidaException Atributos: nomeTransacao:String	ITETransacaoNaoFoiCohcluidaException Operações: void lancar(E15 e) throws E15

Tabela 5: Exceções, Interfaces e Operações (já existentes) – Caso de Uso *Entrega de Talão de Cheque*

	Exceção:	Interface do Tratador
E20	TEEspecieTalaoInvalidaException Atributos: especie:String	ITEEspecieTalaoInvalidaException Operações: void lancar(E20 e) throws E20
E21	TENroPriChequeInvalidoException Atributos: –	ITENroPriChequeInvalidoException Operações: void lancar(E21 e) throws E21
E22	TENaoExisteTalaoEstoqueException Atributos: –	ITENaoExisteTalaoEstoqueException Operações: void lancar(E22 e) throws E22
E23	TETalaoJaEntregueException Atributos: –	ITETalaoJaEntregueException Operações: void lancar(E23 e) throws E23
E24	TEContaBloqueadaParaEntregaTalaoException Atributos: –	ITEContaBloqueadaParaEntregaTalaoException Operações: void lancar(E24 e) throws E24

Tabela 6: Exceções, Interfaces e Operações (novas) – Caso de Uso *Entrega de Talão de Cheque*

Interface de Sistema: ISBSustarChequeMgr	Passo
Operações:	
? sustarCheque(?)	0
? inserirCheque(?)	03.2

Tabela 7: Interface e Operações – Caso de Uso *Sustação de Cheque*

6.1.5 Caso de Uso: Cancelamento de Contrato

Interfaces e Operações de Sistema

	Exceção:	Interface do Tratador
E1	TEAgenciaNaoEstaAbertaException Atributos: codAgencia:String	ITEAgenciaNaoEstaAbertaException Operações: void lancar(E1 e) throws E1
E2	TEAgenciaInvalidaException Atributos: codAgencia:String	ITEAgenciaInvalidaException Operações: void lancar(E2 e) throws E2
E3	TEAgenciaNaoCadastradaException Atributos: codAgencia:String	ITEAgenciaNaoCadastradaException Operações: void lancar(E3 e) throws E3
E4	TEContaInvalidaException Atributos: numConta:String	ITEContaInvalidaException Operações: void lancar(E4 e) throws E4
E5	TEContaNaoCadastradaException Atributos: codAgencia:String numConta:String	ITEContaNaoCadastradaException Operações: void lancar(E5 e) throws E5
E15	TETransacaoNaoFoiConcluidaException Atributos: nomeTransacao:String	ITETransacaoNaoFoiConcluidaException Operações: void lancar(E15 e) throws E15

Tabela 8: Exceções, Interfaces e Operações (já existentes) – Caso de Uso *Sustação de Cheque*

	Exceção:	Interface do Tratador
E25	TEQtdeChequesInvalidaException Atributos: quantCheques:int	ITEQtdeChequesInvalidaException Operações: void lancar(E25 e) throws E25
E26	TEMotivoSustacaoInvalidoException Atributos: motivoSustacao:int	ITEMotivoSustacaoInvalidoException Operações: void lancar(E26 e) throws E26
E27	TESituacaoSustacaoInvalidaException Atributos: situacaoSustacao:String	ITESituacaoSustacaoInvalidaException Operações: void lancar(E27 e) throws E27
E28	TEDataInvalidaException Atributos: data:java.util.Date	ITEDataInvalidaException Operações: void lancar(E28 e) throws E28
E29	TEChequeJaEstaSustadoException Atributos: numCheque:String	ITEChequeJaEstaSustadoException Operações: void lancar(E29 e) throws E29

Tabela 9: Exceções, Interfaces e Operações (novas) – Caso de Uso *Sustação de Cheque*

Interface de Sistema: ISBCapturaChequeMgr	Passo
Operações:	
? capturarCheque(?)	0
? calcularSequenciaCheque(?)	01
? checarConsistenciaDadosCheque(?)	Pré

Tabela 10: Interface e Operações – Caso de Uso *Captura de Cheques*

A interface e as operações provenientes deste caso de uso podem ser vistos na Tabela 14.

Especificação das Exceções Agendadas

	Exceção:	Interface do Tratador
E15	TETransacaoNaoFoiCohcluidaException Atributos: nomeTransacao:String	ITETransacaoNaoFoiCohcluidaException Operações: void lancar(E15 e) throws E15
E28	TEDataInvalidaException Atributos: data:java.util.Date	ITEDataInvalidaException Operações: void lancar(E28 e) throws E28

Tabela 11: Exceções, Interfaces e Operações (já existentes) – Caso de Uso *Captura de Cheque*

	Exceção:	Interface do Tratador
E30	TENroCapturaInvalidoException Atributos: numCaptura:String	ITENroCapturaInvalidoException Operações: void lancar(E30 e) throws E30
E31	TEValorChequeInvalidoException Atributos: valor:BigDecimal numCheque:String	ITEValorChequeInvalidoException Operações: void lancar(E31 e) throws E31
E32	TECompChequeInvalidoException Atributos: comp:String numCheque:String	ITECompChequeInvalidoException Operações: void lancar(E32 e) throws E32
E33	TEBancoChequeInvalidoException Atributos: banco:String numCheque:String	ITEBancoChequeInvalidoException Operações: void lancar(E33 e) throws E33
E34	TEAgenciaChequeInvalidaException Atributos: agencia:String numCheque:String	ITEAgenciaChequeInvalidaException Operações: void lancar(E34 e) throws E34
E35	TEC1DV2ChequeInvalidoException Atributos: c1DV2:String numCheque:String	ITEC1DV2ChequeInvalidoException Operações: void lancar(E35 e) throws E35
E36	TEContaChequeInvalidaException Atributos: conta:String numCheque:String	ITEContaChequeInvalidaException Operações: void lancar(E36 e) throws E36
E37	TEC2DV1ChequeInvalidoException Atributos: c2DV1:String numCheque:String	ITEC2DV1ChequeInvalidoException Operações: void lancar(E37 e) throws E37

Tabela 12: Exceções, Interfaces e Operações (novas) – Caso de Uso *Captura de Cheque*

As classes de exceção e as interfaces dos tratadores existentes que são relevantes para este estudo de caso podem ser vistas na Tabela 15. E as novas classes de exceções e interfaces de tratadores podem ser vistas na Tabela 16.

6.1.6 Caso de Uso: Cadastramento de Limite Adicional

Interfaces e Operações de Sistema

	Exceção:	Interface do Tratador
E38	TENroChequeInvalidoException Atributos: numCheque:String	ITENroChequeInvalidoException Operações: void lancar(E38 e) throws E38
E39	TEChequeJaCapturadoNoDiaException Atributos: numCheque:String dataMovimento:java.util.Date	ITEChequeJaCapturadoNoDiaException Operações: void lancar(E29 e) throws E39
E40	TEInstituicaoChequeNaoEstaCompException Atributos: codBanco:String	ITEInstituicaoChequeNaoEstaCompException Operações: void lancar(E40 e) throws E40
E41	TEC3DV3ChequeInvalidoException Atributos: c3DV3:String numCheque:String	ITEC3DV3ChequeInvalidoException Operações: void lancar(E41 e) throws E41
E42	TEC1DV2ChequeInconsistenteException Atributos: c1DV2:String numCheque:String	ITEC1DV2ChequeInconsistenteException Operações: void lancar(E42 e) throws E42
E43	TEC2DV1ChequeInconsistenteException Atributos: C2DV1:String numCheque:String	ITEC2DV1ChequeInconsistenteException Operações: void lancar(E43 e) throws E43
E44	TEC3DV3ChequeInconsistenteException Atributos: c3DV3:String numCheque:String	ITEC3DV3ChequeInconsistenteException Operações: void lancar(E44 e) throws E44

Tabela 13: Exceções, Interfaces e Operações (novas) – Caso de Uso *Captura de Cheque*

Interface de Sistema: ISBCancelaContratoContaMgr	Passo
Operações:	
? cancelarContratoConta(?)	0

Tabela 14: Interface e Operações – Caso de Uso *Cancelamento de Contrato*

A interface e as operações provenientes deste caso de uso podem ser vistos na Tabela 17.

Especificação das Exceções Agendadas

As classes de exceção e as interfaces dos tratadores existentes que são relevantes para este estudo de caso podem ser vistas na Tabela 18. E as novas classes de exceções e interfaces de tratadores podem ser vistas na Tabela 19.

6.1.7 Modelo de Tipos do Negócio *(Business Type Model)*

O Modelo de tipos de negócio representa as entidades do negócio que são relevantes para o sistema. Entre essas entidades, existem as entidades principais, que são indicadas pelo esteriótipo <<core>>. Esse modelo é mostrado na figura 6. Devido à sua importância no funcionamento do sistema, todas as entidades principais foram consideradas críticas.

	Exceção:	Interface do Tratador
E1	TEAgenciaNaoEstaAbertaException Atributos: codAgencia:String	ITEAgenciaNaoEstaAbertaException Operações: void lancar(E1 e) throws E1
E2	TEAgenciaInvalidaException Atributos: codAgencia:String	ITEAgenciaInvalidaException Operações: void lancar(E2 e) throws E2
E3	TEAgenciaNaoCadastradaException Atributos: codAgencia:String	ITEAgenciaNaoCadastradaException Operações: void lancar(E3 e) throws E3
E4	TEContaInvalidaException Atributos: numConta:String	ITEContaInvalidaException Operações: void lancar(E4 e) throws E4
E5	TEContaNaoCadastradaException Atributos: codAgencia:String numConta:String	ITEContaNaoCadastradaException Operações: void lancar(E5 e) throws E5
E6	TEUsuarioInvalidoException Atributos: nomeUsuario:String	ITEUsuarioInvalidoException Operações: void lancar(E6 e) throws E6
E15 *interna*	TETransacaoNaoFoiCohcluidaException Atributos: nomeTransacao:String	ITETransacaoNaoFoiCohcluidaException Operações: void reconfigurar(E15 e) throws E56
E19	TEModalidadeNaoCadastradaException Atributos: codModalidade:String	ITEModalidadeNaoCadastradaException Operações: void lancar(E19 e) throws E19
E50	TETaxaDeveSerMaiorZeroException Atributos: valorTaxa:BigDecimal	ITETaxaDeveSerMaiorZeroException Operações: void lancar(E50 e) throws E50

Tabela 15: Exceções, Interfaces e Operações (já existentes) – Caso de Uso *Cancelamento de Contrato*

6.1.8 Identificação das Interfaces de Negócio

Para cada entidade principal do modelo de tipos do negócio (Seção 6.1.7), é criada uma interface. Essas interfaces de negócio podem ser vistas na Tabela 20.

6.1.9 Identificação dos Componentes Normais e Excepcionais

Antes da estruturação dos componentes de acordo com o modelo proposto pelo componente tolerante a falhas ideal é necessário criá-los; a criação dos componentes é baseada no agrupamento das respectivas interfaces, que já foram especificadas anteriormente. A ordem de criação sugerida pelo método é a que segue: (1) Componentes de Sistema; (2) Componentes de Negócio; (3) Componentes Excepcionais.

Componentes de Sistema:

As interfaces de sistema foram agrupadas em quatro componentes:

1. O componente *operacoesTalao* (Figura 7) oferece as interfaces *ISBReqTalaoMgr*, proveniente do caso de uso *Requisição de Talão de Cheque*, e *ISBEntregaTalaoMgr*, proveniente do caso de uso *Entrega de Talão de Cheque*.

2. O componente *operacoesChequeSustado* (Figura 8) oferece a interface *ISBSustarChequeMgr*, proveniente do caso de uso *Sustação de Cheque*.

	Exceção:	Interface do Tratador
E45	TEModalidadeInvalidaException Atributos: modalidade:String	ITEModalidadeInvalidaException Operações: void lancar(E45 e) throws E45
E46	TEModalidadeIgualModalidadeContaException Atributos: –	ITEModalidadeIgualModalidadeContaException Operações: void lancar(E46 e) throws E46
E47	TETipoPessoaNaoCorrespondeTipoPessoaContaException Atributos: –	ITETipoPessoaNaoCorrespondeTipoPessoaContaException Operações: void lancar(E47 e) throws E47
E48	TEContratoINEstaCanceladoException Atributos: –	ITEContratoINEstaCanceladoException Operações: void lancar(E48 e) throws E48
E53	TETipoDepositoNaoCadastradoException Atributos: tipoDeposito:String	ITETipoDepositoNaoCadastradoException Operações: void lancar(E53 e) throws E53
E54	TETaxaInvalidaException Atributos: valorTaxa:BigDecimal idTaxa:String	ITETaxaInvalidaException Operações: void lancar(E54 e) throws E54
E56	TEReconfiguracaoSistemaImpossibilitadaException Atributos: nomeOperacao:String nomeCompFalhou:String nomeCompAlternativo:String causaErro:String	ITEReconfiguracaoSistemaImpossibilitadaException Operações: void lancar(E56 e) throws E56

Tabela 16: Exceções, Interfaces e Operações (novas) – Caso de Uso *Cancelamento de Contrato*

Interface de Sistema: ISBCadLimiteAdcMgr	Passo
Operações:	
? cadastrarLimiteAdc(?)	0

Tabela 17: Interface e Operações – Caso de Uso *Cadastramento de Limite Adicional*

3. O componente *operacoesChequeCapturado* (Figura 9) oferece a interface *ISBCapturaChequeMgr*, proveniente do caso de uso *Captura de Cheque*.

4. O componente *operacoesConta* (Figura 10) oferece as interfaces *ISBCancelaContratoContaMgr*, proveniente do caso de uso *Cancelamento de Contrato*, e *ISBCadLimiteAdcMgr*, proveniente do caso de uso *Cadastramento de Limite Adicional*.

Componentes de Negócio:
As interfaces de negócio foram agrupadas em quatro componentes:

1. O componente *gerenciamentoConta* (Figura 11) oferece a interface *ISBContaMgrReq*, proveniente do *core type 'Conta'*.

2. O componente *gerenciamentoColigada* (Figura 12) oferece a interface *ISBColigadaMgrReq*, proveniente do *core type 'Coligada'*.

3. O componente *gerenciamentoMovimento* (Figura 13) oferece a interface *ISBMovimentoMgrReq*, proveniente do *core type 'Movimento'*.

	Exceção:	Interface do Tratador
E1	TEAgenciaNaoEstaAbertaException Atributos: codAgencia:String	ITEAgenciaNaoEstaAbertaException Operações: void lancar(E1 e) throws E1
E2	TEAgenciaInvalidaException Atributos: codAgencia:String	ITEAgenciaInvalidaException Operações: void lancar(E2 e) throws E2
E3	TEAgenciaNaoCadastradaException Atributos: codAgencia:String	ITEAgenciaNaoCadastradaException Operações: void lancar(E3 e) throws E3
E4	TEContaInvalidaException Atributos: numConta:String	ITEContaInvalidaException Operações: void lancar(E4 e) throws E4
E5	TEContaNaoCadastradaException Atributos: codAgencia:String numConta:String	ITEContaNaoCadastradaException Operações: void lancar(E5 e) throws E5
E6	TEUsuarioInvalidoException Atributos: nomeUsuario:String	ITEUsuarioInvalidoException Operações: void lancar(E6 e) throws E6
E15	TETransacaoNaoFoiCohcluidaException Atributos: nomeTransacao:String	ITETransacaoNaoFoiCohcluidaException Operações: void reconfigurar(E15 e) throws E56

Tabela 18: Exceções, Interfaces e Operações (já existentes) – Caso de Uso *Cadastramento de Limite Adicional*

	Exceção:	Interface do Tratador
E49	TELimiteDeveSerMaiorZeroException Atributos: limite:BigDecimal	ITELimiteDeveSerMaiorZeroException Operações: void lancar(E49 e) throws E49
E50	TETaxaDeveSerMaiorZeroException Atributos: taxa:BigDecimal	ITETaxaDeveSerMaiorZeroException Operações: void lancar(E50 e) throws E50
E51	TEModalidadeNaoPermiteUsoLimiteAdcException Atributos: modalidade:String	ITEModalidadeNaoPermiteUsoLimiteAdcException Operações: void lancar(E51 e) throws E51
E52	TEContaJaPossuiLimiteAdcException Atributos: –	ITEContaJaPossuiLimiteAdcException Operações: void lancar(E52 e) throws E52

Tabela 19: Exceções, Interfaces e Operações (novas) – Caso de Uso *Cadastramento de Limite Adicional*

Interfaces de Negócio:
ISBContaMgrReq
ISBColigadaMgrReq
ISBMovimentoMgrReq
ISBControleAgenciaMgrReq

Tabela 20: Interfaces de Negócio Identificadas

4. O componente *gerenciamentoControleAgencia* (Figura 14) oferece a interface *ISBControleAgenciaMgrReq*, proveniente do *core type 'ControleAgencia'*.

Componentes Excepcionais:

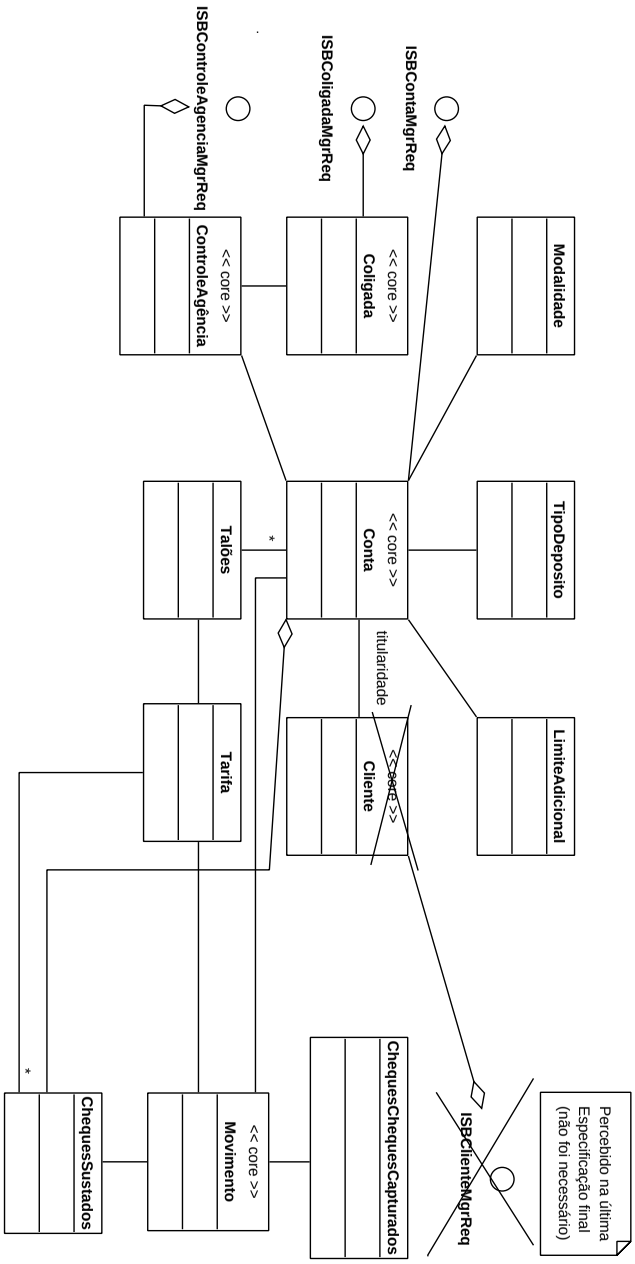

Figura 6: Modelo de Tipos do Negócio

Os componentes excepcionais implementam as interfaces dos tratadores das exceções. Foram criados 11 componentes:

1. O componente *tratadorAgencia* (Figura 15) oferece as seguintes interfaces:

   - *ITEAgenciaNaoEstaAbertaException*, que contém os tratadores para a exceção

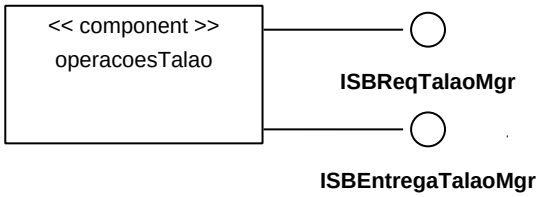

Figura 7: Componente de Sistema: operacoesTalao

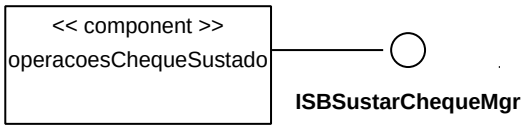

Figura 8: Componente de Sistema: operacoesChequeSustado

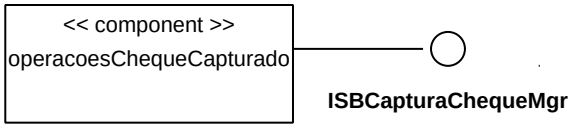

Figura 9: Componente de Sistema: operacoesChequeCapturado

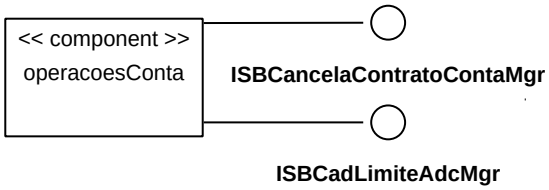

Figura 10: Componente de Sistema: operacoesConta

E1 - TEAgenciaNaoEstaAbertaException;

- *ITEAgenciaInvalidaException*, que contém os tratadores para a exceção *E2 - TEAgenciaInvalidaException*;
- *ITEAgenciaNaoCadastradaException*, que contém os tratadores para a exceção *E3 - TEAgenciaNaoCadastradaException*.

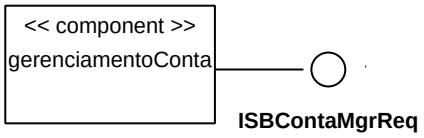

Figura 11: Componente de Negócio: gerenciamentoConta

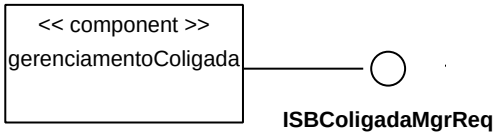

Figura 12: Componente de Negócio: gerenciamentoColigada

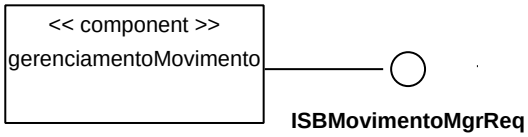

Figura 13: Componente de Negócio: gerenciamentoMovimento

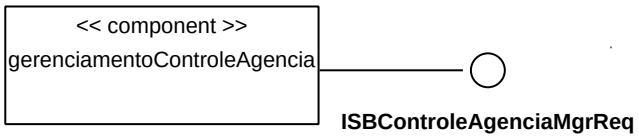

Figura 14: Componente de Negócio: gerenciamentoControleAgencia

2. O componente *tratadorConta* (Figura 16) oferece as seguintes interfaces:

   - *ITEContaInvalidaException*, que contém os tratadores para a exceção *E4 - TEContaInvalidaException*;
   - *ITEContaNaoCadastradaException*, que contém os tratadores para a exceção *E5 - TEContaNaoCadastradaException*;
   - *ITEModalidadeIgualModalidadeContaException*, que contém os tratadores para a exceção *E46 - TEModalidadeIgualModalidadeContaException*;

- *ITETipoPessoaNaoCorrespondeTipoPessoaContaException*, que contém os tratadores para a exceção *E47 - TETipoPessoaNaoCorrespondeTipoPessoaContaException*;
- *ITEContratoINEstaCanceladoException*, que contém os tratadores para a exceção *E48 - TEContratoINEstaCanceladoException*;
- *ITELimiteDeveSerMaiorZeroException*, que contém os tratadores para a exceção *E49 - TELimiteDeveSerMaiorZeroException*;
- *ITEModalidadeNaoPermiteUsoLimiteAdcException*, que contém os tratadores para a exceção *E51 - TEModalidadeNaoPermiteUsoLimiteAdcException*;
- *ITEContaJaPossuiLimiteAdcException*, que contém os tratadores para a exceção *E52 - TEContaJaPossuiLimiteAdcException*;
- *ITETipoDepositoNaoCadastradoException*, que contém os tratadores para a exceção *E53 - ITETipoDepositoNaoCadastradoException*.

3. O componente *tratadorUtil* (Figura 17) oferece as seguintes interfaces:

   - *ITEUsuarioInvalidoException*, que contém os tratadores para a exceção *E6 - TEUsuarioInvalidoException*;
   - *ITEDataInvalidaException*, que contém os tratadores para a exceção *E28 - TEDataInvalidaException*;
   - *ITETaxaDeveSerMaiorZeroException*, que contém os tratadores para a exceção *E50 - TETaxaDeveSerMaiorZeroException*;
   - *ITETaxaInvalidaException*, que contém os tratadores para a exceção *E54 - TETaxaInvalidaException*.

4. O componente *tratadorCliente* (Figura 18) oferece as seguintes interfaces:

   - *ITECodClienteInvalidoException*, que contém os tratadores para a exceção *E7 - TECodClienteInvalidoException*;
   - *ITEClienteNaoETitularContaException*, que contém os tratadores para a exceção *E8 - TEClienteNaoETitularContaException*;
   - *ITEClientesDevemSerDistintosException*, que contém os tratadores para a exceção *E12 - TEClientesDevemSerDistintosException*.

5. O componente *tratadorTalao* (Figura 19) oferece as seguintes interfaces:

   - *ITEQtdeTalaoDeveSerMaiorZeroException*, que contém os tratadores para a exceção *E9 - TEQtdeTalaoDeveSerMaiorZeroException*;
   - *ITETipoTalaoInvalidoException*, que contém os tratadores para a exceção *E10 - TETipoTalaoInvalidoException*;
   - *ITETipoSolicitacaoTalaoInvalidoException*, que contém os tratadores para a exceção *E11 - TETipoSolicitacaoTalaoInvalidoException*;

- *ITERequisicaoTalaoEstaBloqueadaContaException*, que contém os tratadores para a exceção *E13 - TERequisicaoTalaoEstaBloqueadaContaException*;
- *ITERequisicaoTalaoImpossibilitadaContaInternaException*, que contém os tratadores para a exceção *E14 - TERequisicaoTalaoImpossibilitadaContaInternaException*;
- *ITEEspecieTalaoInvalidaException*, que contém os tratadores para a exceção *E20 - TEEspecieTalaoInvalidaException*;
- *ITENroPriChequeInvalidoException*, que contém os tratadores para a exceção *E21 - TENroPriChequeInvalidoException*;
- *ITENaoExisteTalaoEstoqueException*, que contém os tratadores para a exceção *E22 - TENaoExisteTalaoEstoqueException*; *ITETalaoJaEntregueException*, que contém os tratadores para a exceção *E23 - TETalaoJaEntregueException*;
- *ITEContaBloqueadaParaEntregaTalaoException*, que contém os tratadores para a exceção *E24 - TEContaBloqueadaParaEntregaTalaoException*; e
- *ITERegistroJaExistenteException*, que contém os tratadores para a exceção *E57 - TERegistroJaExistenteException*.

6. O componente *tratadorTransacao* (Figura 20) oferece as seguintes interfaces:
   - *ITETransacaoNaoFoiCohcluidaException*, que contém os tratadores para a exceção *E15 - TETransacaoNaoFoiCohcluidaException*;
   - *ITEReconfiguracaoSistemaImpossibilitadaException*, que contém os tratadores para a exceção *E56 - TEReconfiguracaoSistemaImpossibilitadaException*.
7. O componente *tratadorColigada* (Figura 21) oferece a interface *ITEColigadaNaoCadastradaException*, que contém os tratadores para a exceção *E18 - TEColigadaNaoCadastradaException*.
8. O componente *tratadorModalidade* (Figura 22) oferece as seguintes interfaces:
   - *ITEModalidadeNaoCadastradaException*, que contém os tratadores para a exceção *E19 - ITEModalidadeNaoCadastradaException*;
   - *ITEModalidadeInvalidaException*, que contém os tratadores para a exceção *E45 - TEModalidadeInvalidaException*.
9. O componente *tratadorChequeSustado* (Figura 23) oferece as seguintes interfaces:
   - *ITEQtdeChequesInvalidaException*, que contém os tratadores para a exceção *E25 - TEQtdeChequesInvalidaException*;
   - *ITEMotivoSustacaoInvalidoException*, que contém os tratadores para a exceção *E26 - TEMotivoSustacaoInvalidoException*;
   - *ITESituacaoSustacaoInvalidaException*, que contém os tratadores para a exceção *E27 - TESituacaoSustacaoInvalidaException*; e

- *ITEChequeJaEstaSustadoException*, que contém os tratadores para a exceção *E29 - TEChequeJaEstaSustadoException*.

10. O componente *tratadorChequesCapturados* (Figura 24) oferece as seguintes interfaces:

    - *ITENroCapturaInvalidoException*, que contém os tratadores para a exceção *E30 - TENroCapturaInvalidoException*;
    - *ITEValorChequeInvalidoException*, que contém os tratadores para a exceção *E31 - TEValorChequeInvalidoException*;
    - *ITECompChequeInvalidoException*, que contém os tratadores para a exceção *E32 - TECompChequeInvalidoException*;
    - *ITEBancoChequeInvalidoException*, que contém os tratadores para a exceção *E33 - TEBancoChequeInvalidoException*;
    - *ITEAgenciaChequeInvalidaException*, que contém os tratadores para a exceção *E34 - TEAgenciaChequeInvalidaException*;
    - *ITEC1DV2ChequeInvalidoException*, que contém os tratadores para a exceção *E35 - TEC1DV2ChequeInvalidoException*;
    - *ITEContaChequeInvalidaException*, que contém os tratadores para a exceção *E36 - TEContaChequeInvalidaException*;
    - *ITEC2DV1ChequeInvalidoException*, que contém os tratadores para a exceção *E37 - TEC2DV1ChequeInvalidoException*;
    - *ITENroChequeInvalidoException*, que contém os tratadores para a exceção *E38 - TENroChequeInvalidoException*;
    - *ITEChequeJaCapturadoNoDiaException*, que contém os tratadores para a exceção *E39 - TEChequeJaCapturadoNoDiaException*;
    - *ITEInstituicaoChequeNaoEstaCompException*, que contém os tratadores para a exceção *E40 - TEInstituicaoChequeNaoEstaCompException*;
    - *ITEC3DV3ChequeInvalidoException*, que contém os tratadores para a exceção *E41 - TEC3DV3ChequeInvalidoException*;
    - *ITEC1DV2ChequeInconsistenteException*, que contém os tratadores para a exceção *E42 - TEC1DV2ChequeInconsistenteException*;
    - *ITEC2DV1ChequeInconsistenteException*, que contém os tratadores para a exceção *E43 - TEC2DV1ChequeInconsistenteException*; e
    - *ITEC3DV3ChequeInconsistenteException*, que contém os tratadores para a exceção *E44 - TEC3DV3ChequeInconsistenteException*.

6.1.10 Definição dos componentes a serem testados

Seguindo os critérios para escolha dos componente a se tornarem os componentes de sistema operacoesTalao, operacoesConta, operacoesChequesCapturados e operacoesChequeSustado foram escolhidos devido à sua criticidade. Os componentes de negócio, apesar de serem

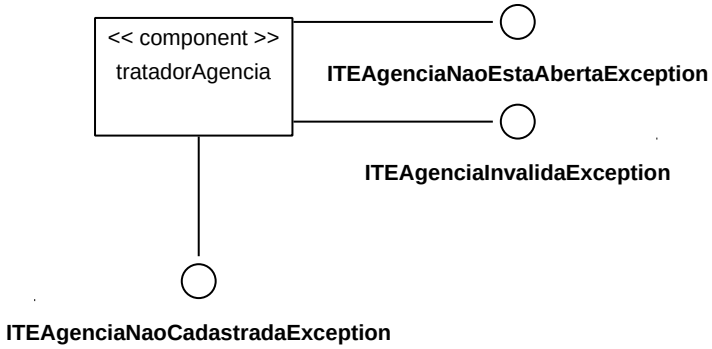

Figura 15: Componente Excepcional: tratadorAgencia

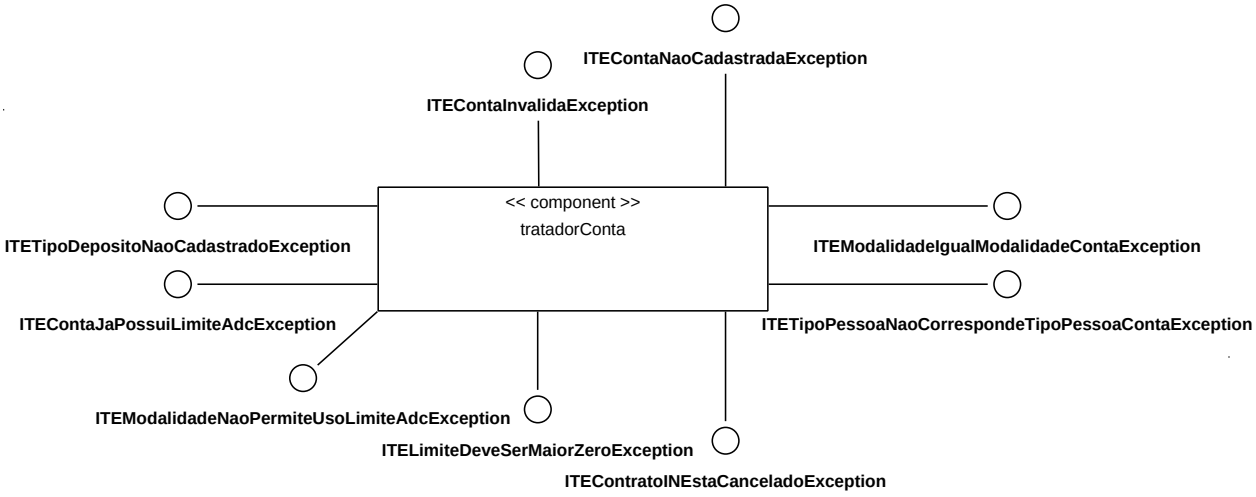

Figura 16: Componente Excepcional: tratadorConta

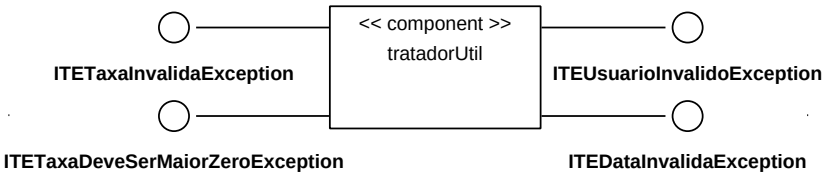

Figura 17: Componente Excepcional: tratadorUtil

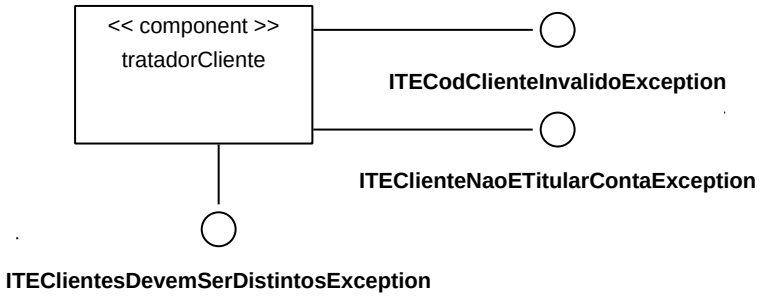

Figura 18: Componente Excepcional: tratadorCliente

fortes candidatos devido ao critério de reutilização, não foram escolhidos por não fazerem parte do escopo do sistema a ser testado. Os componentes de negócio serão reutilizados de outros projetos da mesma empresa, e para realização deste trabalho, foram utilizados apenas stubs nas simulações deste sistema.

Os tratadores, apesar de críticos, não foram escolhidos por sua lógica ser muito simples. A simplicidade também torna injustificável a criação de stubs para a substituição destes tratadores durante os testes, e por isso a decisão foi testar não apenas a parte normal dos componentes de sistema, mas testá-los já na forma de componentes ideais. Apenas os componentes de negócio serão substituídos por stubs.

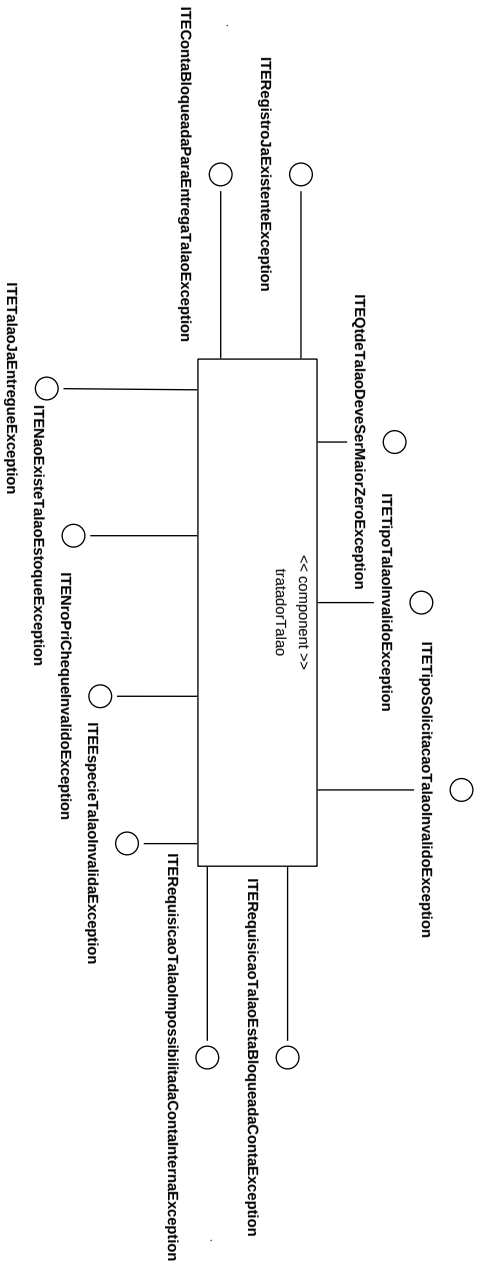

Figura 19: Componente Excepcional: tratadorTalao

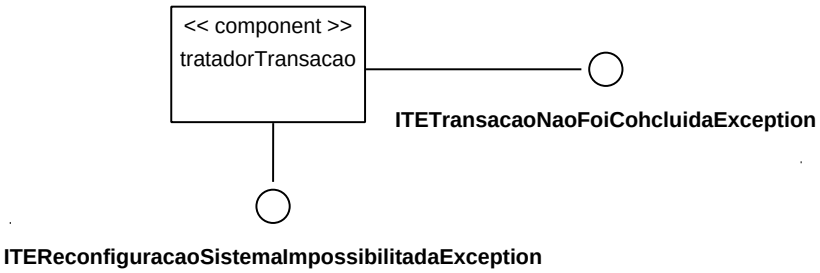

Figura 20: Componente Excepcional: tratadorTransacao

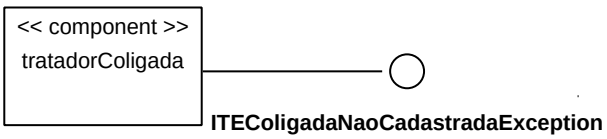

Figura 21: Componente Excepcional: tratadorColigada

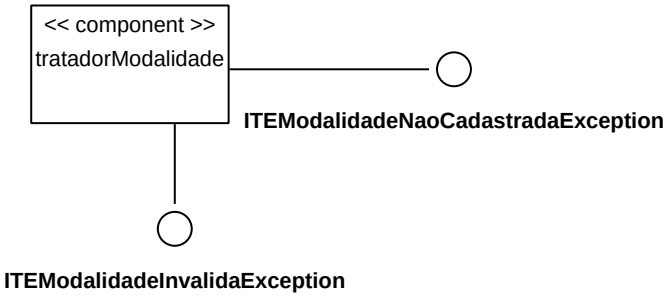

Figura 22: Componente Excepcional: tratadorModalidade

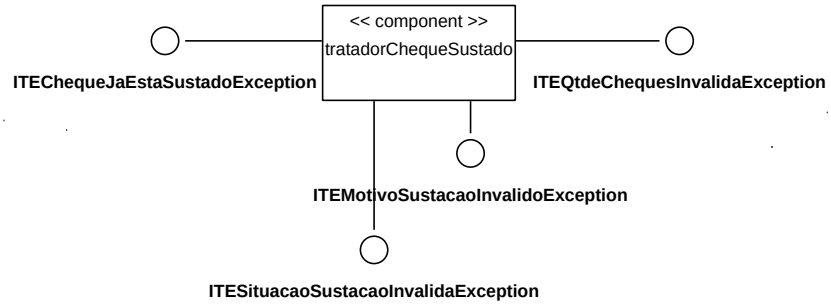

Figura 23: Componente Excepcional: tratadorChequeSustado

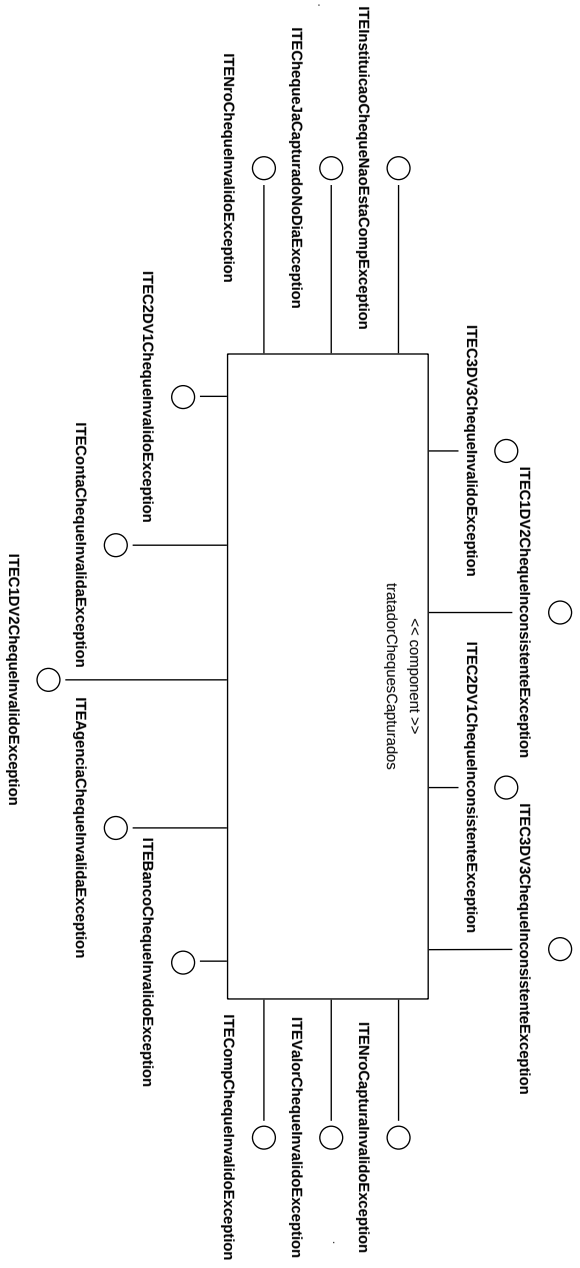

Figura 24: Componente Excepcional: tratadorChequesCapturados

6.2 Interação entre os Componentes

Nesta fase são analisadas as interações necessárias entre os componentes de sistema e os de negócio. Essas interações são derivadas dos passos dos casos de uso, e são descobertas as operações das interfaces de negócio e novas exceções possíveis.

Suas subfases são:

1. Identificação das Operações de Negócio (Operações Requeridas pelo Sistema): *Atividade de Desenvolvimento*

   Entradas: operações das interfaces de sistema e interfaces de negócio.

   As operações das interfaces de negócio são descobertas a partir da construção de um diagrama de atividades da UML para cada operação dos componentes de sistema, que ilustra a interação entre essa operação e os componentes de negócio. O diagrama é dividido em raias, que representam as diferentes interfaces de negócio com as quais o componente de sistema interage.

   Inicialmente, o diagrama é construído desconsiderando-se as exceções. As atividades representam as operações das interfaces de negócio, na ordem em que são chamadas. As operações são representadas pelas assinaturas, criando-se novos *data types* se necessário.

   Originariamente, as operações de negócio eram descobertas com a construção de um diagrama de colaboração da UML. Mas com a integração com o método de testes, buscou-se um formato que seria genérico para os propósitos de especificação e testes, decidindo-se pelo diagrama de atividades. Ele será modificado ao longo do ciclo do desenvolvimento, adaptando-se às diversas fases e não gerando retrabalho.

   Saídas: operações das interfaces de negócio (no formato de diagramas de atividades)

2. Definição do Comportamento Excepcional (Exceções Lançadas pelos Componente de Negócio): *Atividade de Desenvolvimento e Testes*

   Entradas: diagramas de atividades das operações de negócio.

   Após a construção das interações, as exceções retornadas pelas interfaces de negócio são incorporadas ao diagrama de atividades, e também o possível comportamento do componente de sistema frente ao recebimento dessas exceções.

   As possíveis saídas de cada uma das operações de negócio são incluídas nas arestas do diagrama, na forma de restrições. As restrições são especificadas na linguagem OCL, e representam desde os tipos dos retornos dos métodos (normais e excepcionais) até restrições para a execução de determinados caminhos do diagrama.

   O comportamento do método do componente de sistema frente aos retornos das interfaces requeridas também é ilustrado no diagrama, através dos nós finais. Quando o valor do retorno não é determinístico, o nó final não recebe um rótulo. Quando seu valor pode ser determinado, esse valor é acrescentado como rotulo ao nó final correspondente. Para o lançamento das exceções, por exemplo, é incluído o tipo da exceção lançada.

Saídas: diagramas de atividades explicitando o comportamento das operações das interfaces de negócio, incluindo as exceções lançadas.

3. Revisão dos componentes a serem testados: *Atividade de Testes*

   Entradas: diagramas de atividades explicitando o comportamento das operações das interfaces de negócio, incluindo as exceções lançadas.

   Após a definição do comportamento excepcional, é possível avaliar a real complexidade e controlabilidade dos tratadores, pois apenas com a construção dos diagramas de atividades seu comportamento é definido. Assim, nesta fase é realizada uma revisão da lista dos componentes a serem testados e dos *stubs* a serem criados elaborada na fase de Identificação de Componentes.

   Saídas: componentes a serem tornados testáveis.

4. Definição de Restrições Dinâmicas: *Atividade de Desenvolvimento*

   Nesta atividade, para cada componente especificado no sistema, deve-se definir restrições que possam representar violações não checáveis em tempo de compilação. Um exemplo dessas violações pode ser a definição de uma cardinalidade máxima para o número de instâncias de cada componente.

5. Definição das Interfaces Requeridas: *Atividade de Desenvolvimento*

   Nesta subfase as interfaces requeridas são agrupadas de acordo com a camada da arquitetura que ela se refere. Dessa forma, deverá existir uma interface requerida para cada camada da arquitetura que o componente dependa. Como será percebido na fase de montagem (Seção 8), essa redução do número de interfaces requeridas proporcionará uma redução no número de conectores construídos.

6. Estruturação dos Componentes Tolerantes a Falhas Ideais: *Atividade de Desenvolvimento*

   A estruturação dos componentes do sistema segundo o modelo do componente tolerante a falhas ideal [AL90] consiste basicamente na associação dos componentes normais e excepcionais criados, através da criação de conectores internos específicos. Nesta etapa, são identificadas todas as exceções (internas ou externas) lançadas por cada operação das interfaces normais (de sistema e de negócio).

6.2.1 Caso de Uso Requisição de Talão

Identificação das Operações de Negócio (Figura 25)

Definição Comportamento Excepcional (Figura 26)

Revisão dos componentes a serem testados

A partir do diagrama para definição comportamento excepcional (Figura 26), foi observado que o tratador ITETalaoJaCadastradoException não se restringe a propagação de exceções, realizando um cenário recuperável (Figura DIAGRAMA DO TRATADOR).

Por ser um tratamento simples, foi decidido que o tratador não seria tornado um componente testável. Além disso, por sua baixa controlabilidade, já que sua saída depende do comportamento de um método requerido, foi decidido pela criação de um *stub* para substitui-lo durante os testes do componente operacoesTalao.

Definição de Restrições Dinâmicas

Não há restrições dinâmicas.

Definição Interfaces Requeridas

Componente: operacaoTalao (Interfaces Requeridas: IOperacaoTalaoNegReq e IOperacaoTalaoSisReq)

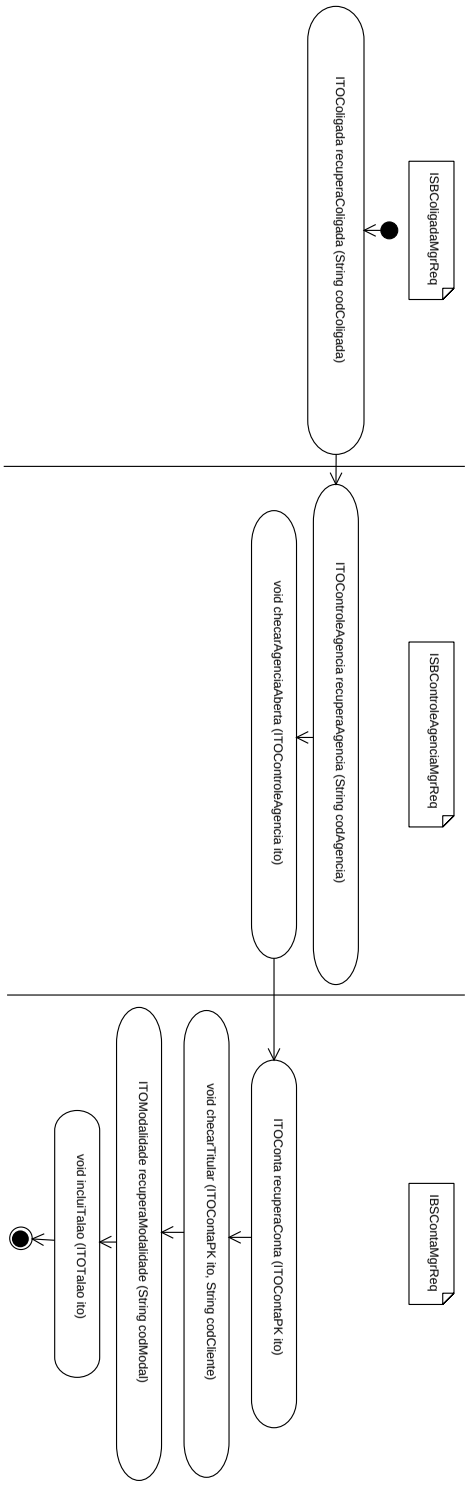

Figura 25: Diagrama para Identificação das Operações de Negócio: Caso de Uso Requisição de Talão

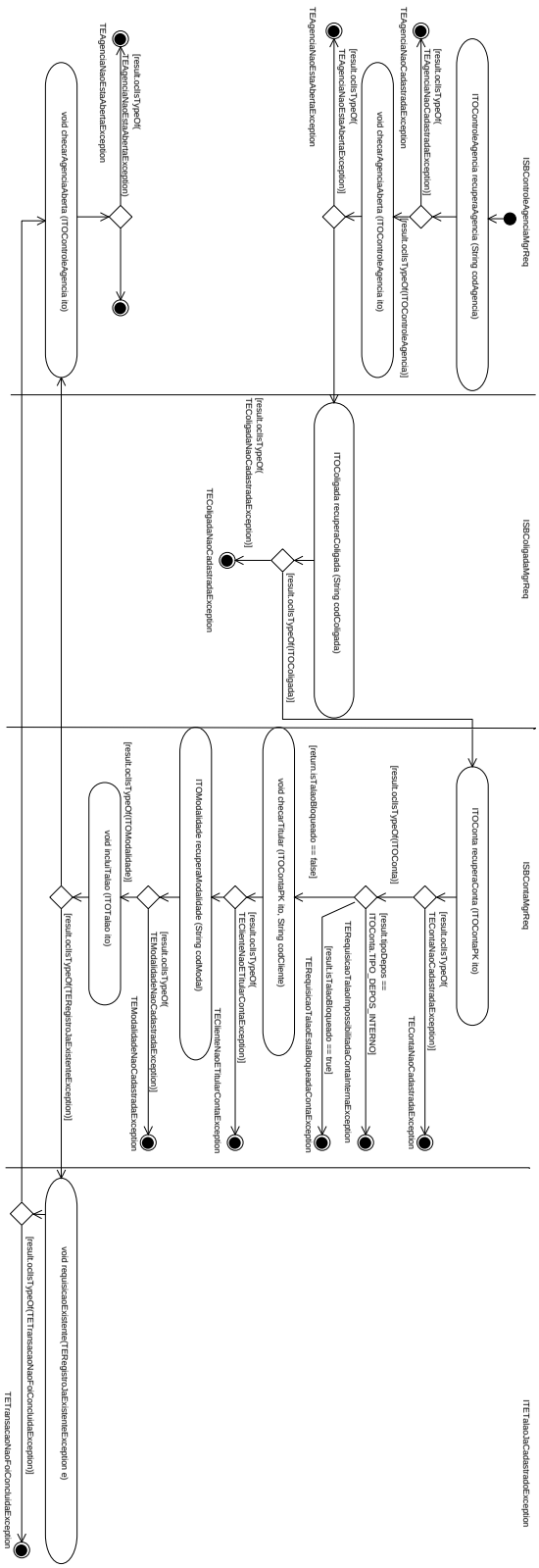

Figura 26: Diagrama para Definição Comportamento Excepcional: Caso de Uso Requisição de Talão

6.2.2 Caso de Uso Entrega de Talão

Identificação das Operações de Negócio (Figura 27)

Definição Comportamento Excepcional (Figura 6.2.2)

Revisão dos componentes a serem testados

Nenhuma modificação.

Definição de Restrições Dinâmicas

Não há restrições dinâmicas.

Definição Interfaces Requeridas

Componente: operacaoTalao (Interface IOperacaoTalaoNegReq)

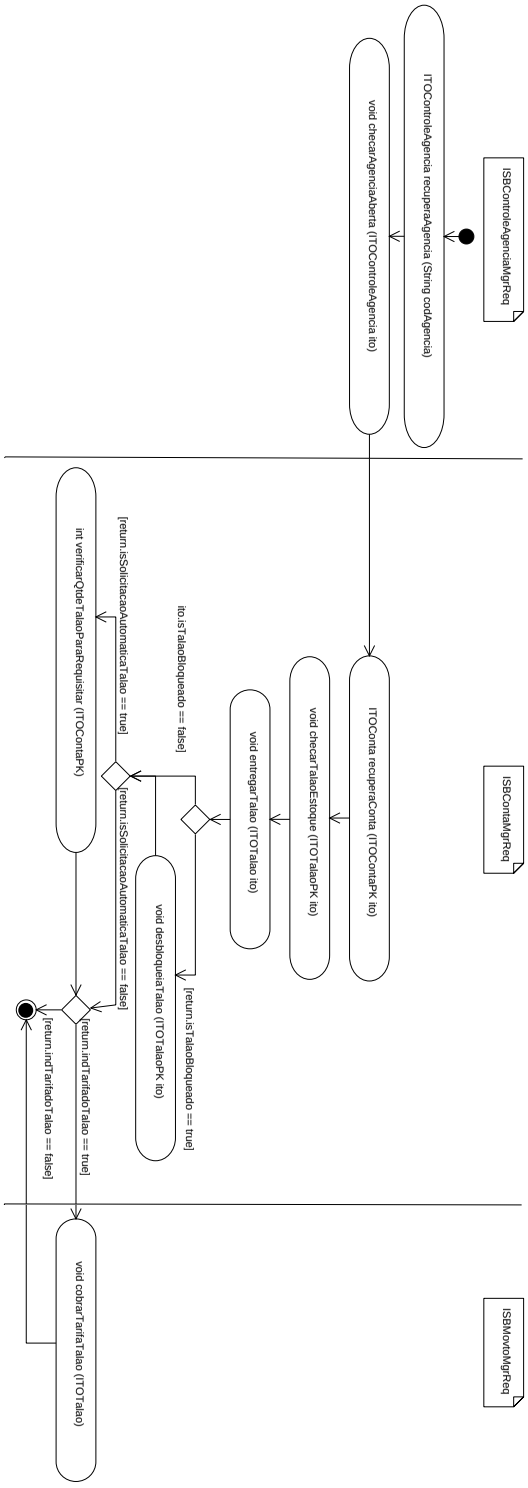

Figura 27: Diagrama para Identificação das Operações de Negócio: Caso de Uso Entrega de Talão de Cheque

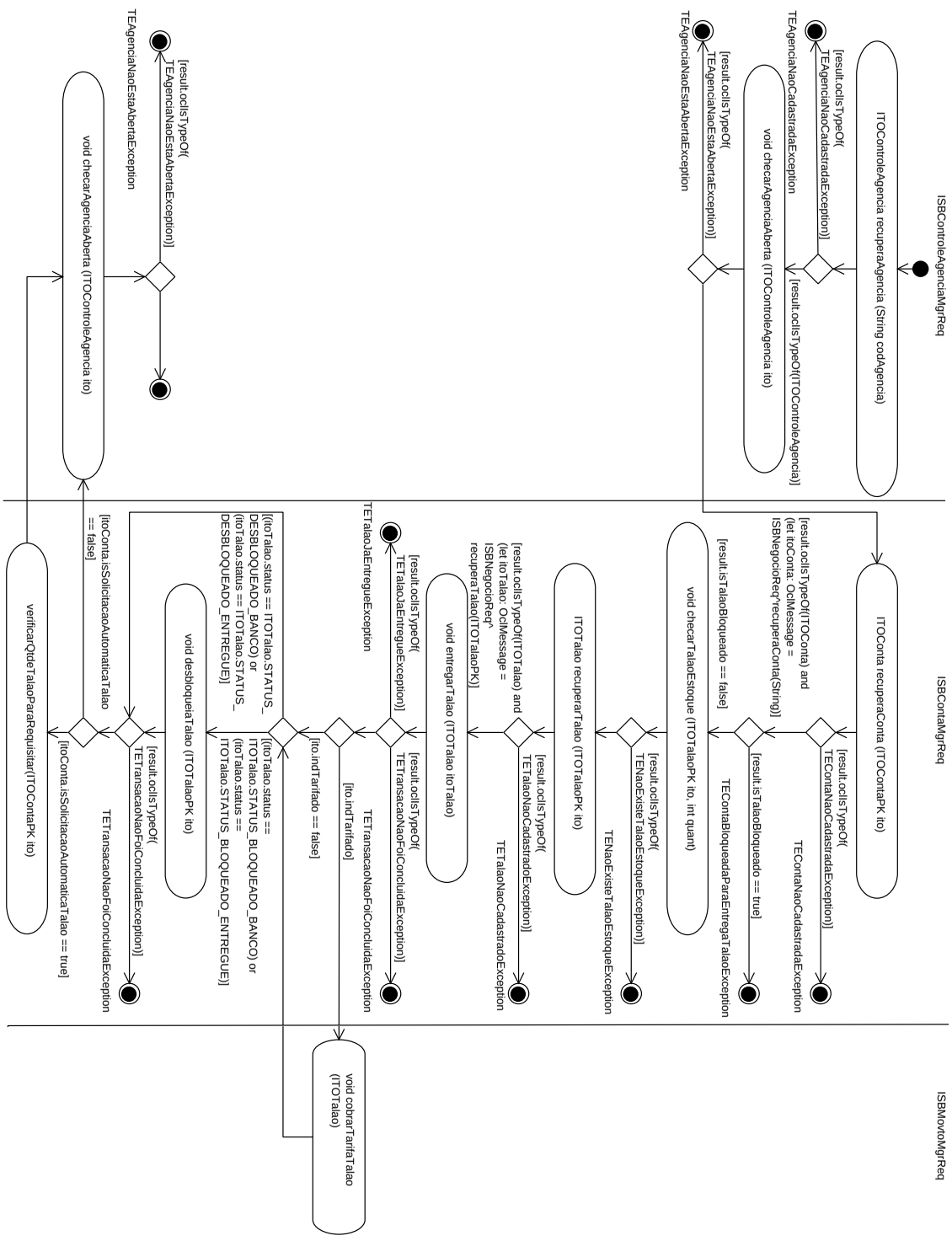

Figura 28: Diagrama para Definição Comportamento Excepcional: Caso de Uso Entrega de Talão de Cheque

6.2.3 Caso de Uso Sustação de Cheque

Identificação das Operações de Negócio (Figura 29)

Definição Comportamento Excepcional (Figura 6.2.3)

Revisão dos componentes a serem testados

A interface IUtil foi criada por representar uma funcionalidade comum a vários componentes, mas muito simples: a manipulação de datas. Por essa simplicidade, decidiu-se não criar stubs para substituir esta interface requerida, incorporando-a ao componente ideal a ser testado. Para que a interface seja desconsiderada na geração dos testes, o diagrama será editado antes da geração para ocultar a interação com esta interface. A situação retratada nesta parte do diagrama será incluída no diagrama criado durante a especificação final dos componentes, com o fornecimento de uma data inválida.

Definição de Restrições Dinâmicas

Não há restrições dinâmicas.

Definição Interfaces Requeridas

Componente: operacaoChequeSustado (Interface IOperacaoChequeSustadoNegReq)

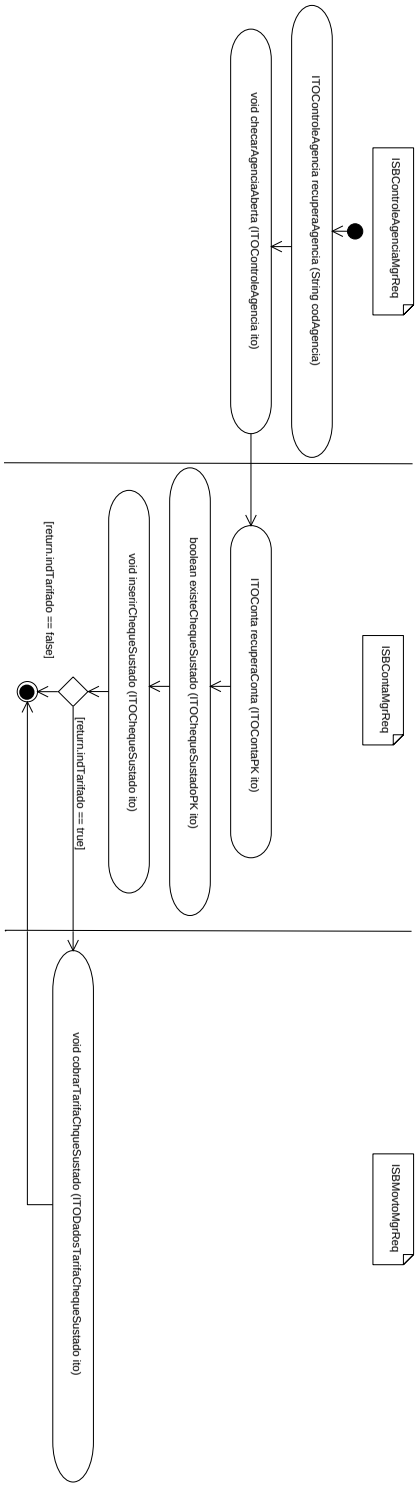

Figura 29: Diagrama para Identificação das Operações de Negócio: Caso de Uso Sustação de Cheque

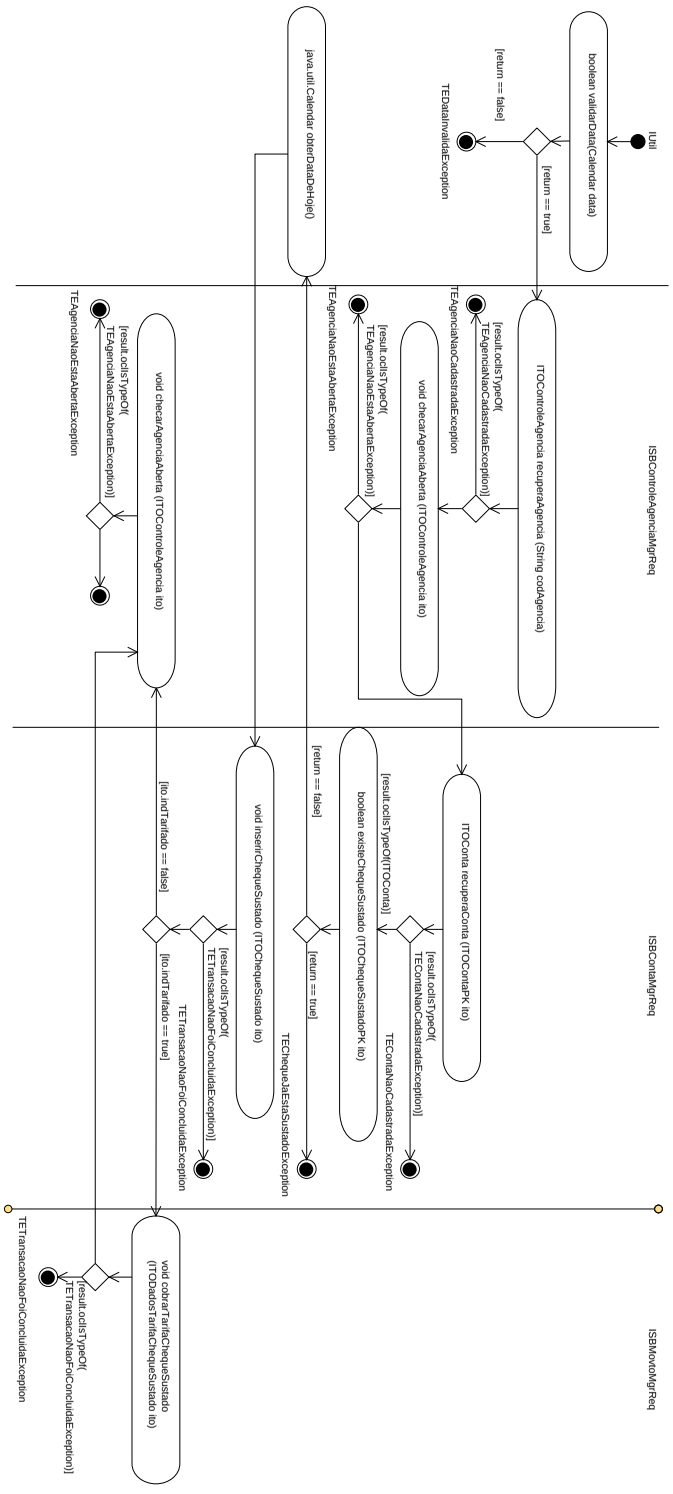

Figura 30: Diagrama para Definição Comportamento Excepcional: Caso de Uso Sustação de Cheque

6.2.4 Caso de Uso Captura de Cheque

Identificação das Operações de Negócio (Figura 31)

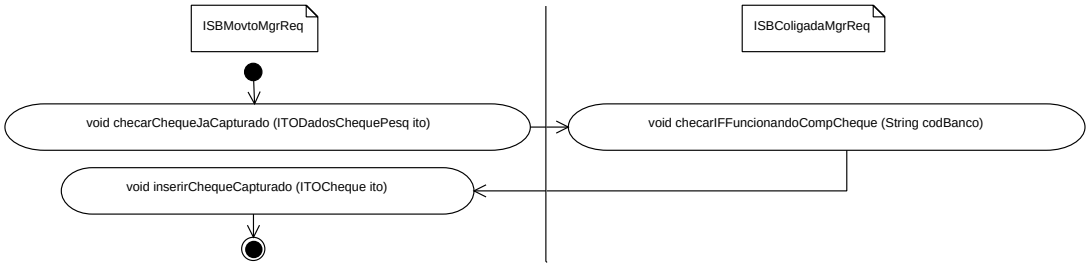

Figura 31: Diagrama para Identificação das Operações de Negócio: Caso de Uso Captura de Cheque

Definição Comportamento Excepcional (Figura 32)

Revisão dos componentes a serem testados

Assim como no componente operacoesChequeSustado, neste componente a interface IUtil não será substituída por um *stub*: antes da geração dos testes a raia do diagrama referente à interface IUtil será ocultada, e a situação excepcional será incluída no diagrama criado durante a especificação final dos componentes.

Definição de Restrições Dinâmicas

Não há restrições dinâmicas.

Definição Interfaces Requeridas

Componente: operacaoChequesCapturados (Interface: IOperacaoChequesCapturados-NegReq)

6.2.5 Caso de Uso Cancelar Contrato

Identificação das Operações de Negócio (Figura 33)

Definição Comportamento Excepcional (Figura 34)

Revisão dos componentes a serem testados

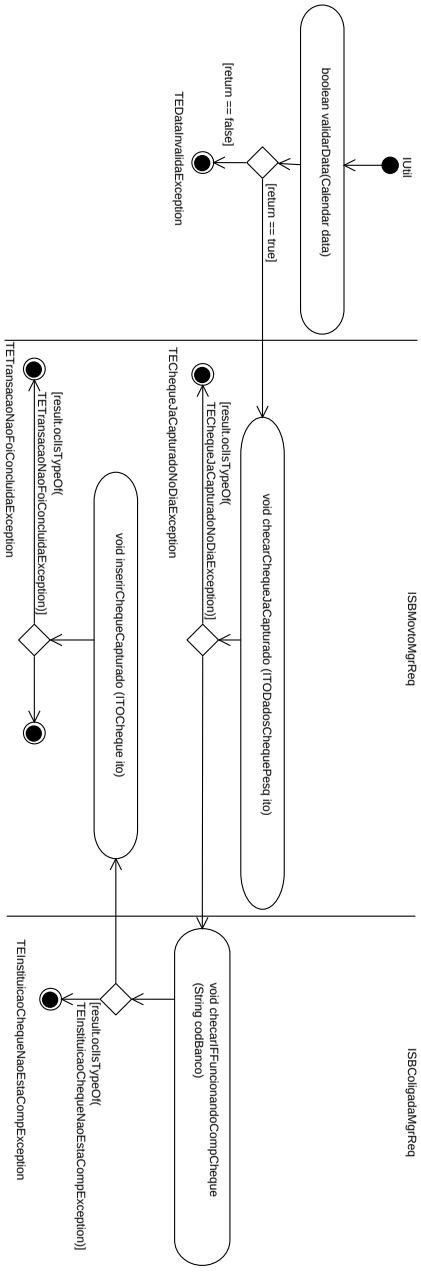

Figura 32: Diagrama para Definição do Comportamento Excepcional das Operações de Negócio: Caso de Uso Captura de Cheque

Assim como nos componentes operacoesChequeCapturado e operacoesChequeSustado, a interface IUtil não será substituída por um *stub* dada a sua simplicidade, sendo a sua raia oculta no diagrama durante a geração dos testes.

Definição de Restrições Dinâmicas

Não há restrições dinâmicas.

Definição Interfaces Requeridas

Componente: operacaoConta (interface requerida: IOperacaoContaNegReq)

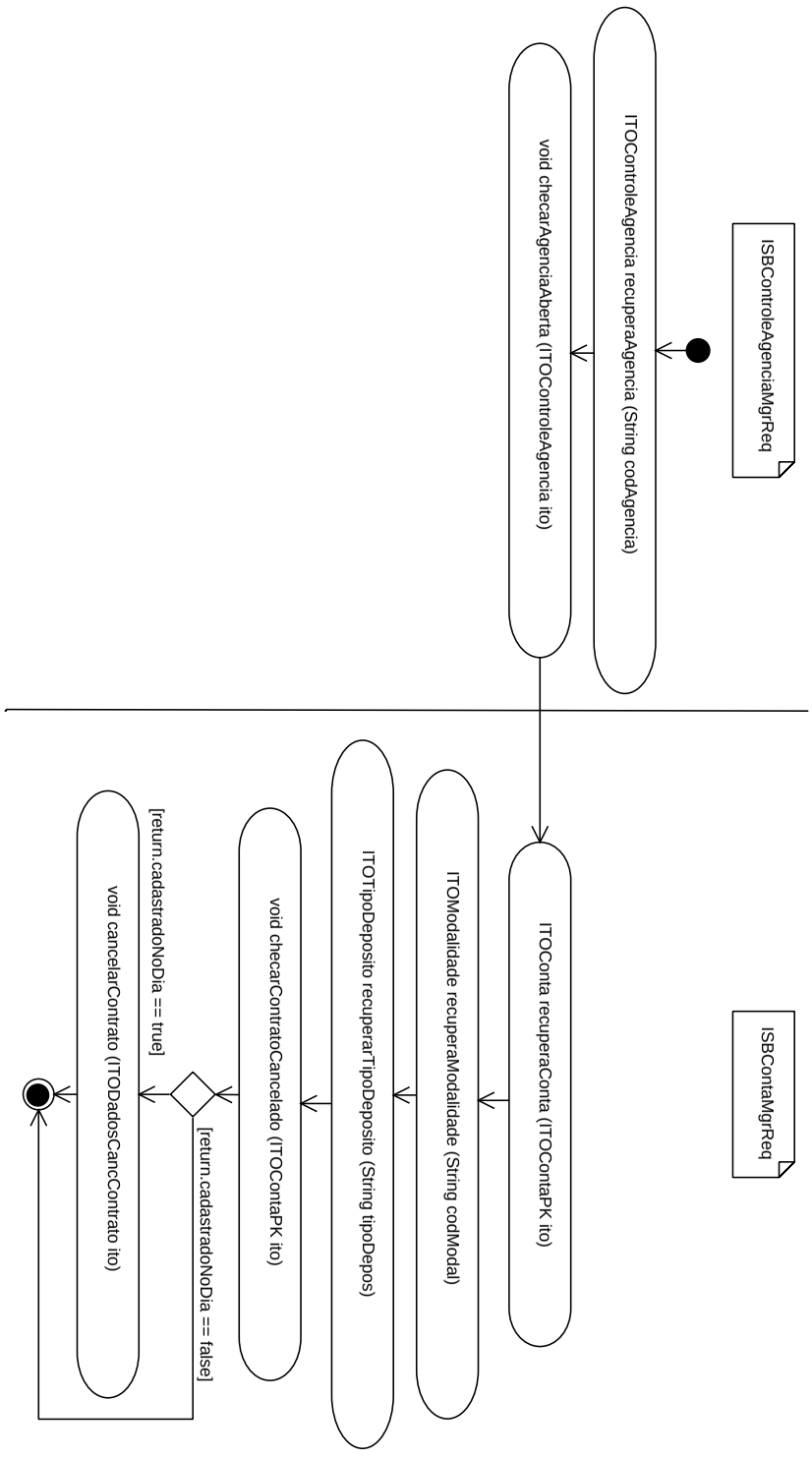

Figura 33: Diagrama para Identificação das Operações de Negócio: Caso de Uso Cancelamento de Contrato

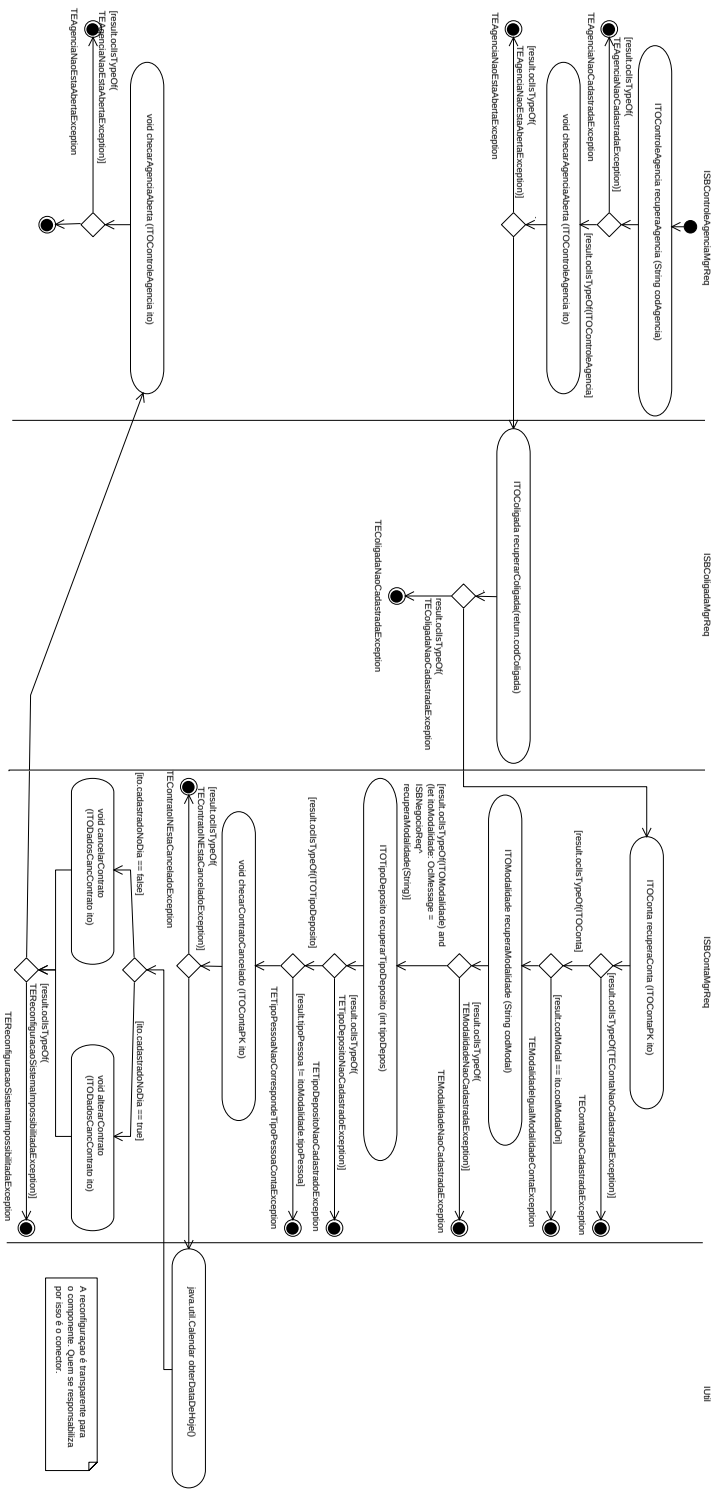

Figura 34: Diagrama para Definição Comportamento Excepcional: Caso de Uso Cancelamento de Contrato

6.2.6 Caso de Uso Cadastramento de Limite Adicional

Identificação das Operações de Negócio (Figura 35)

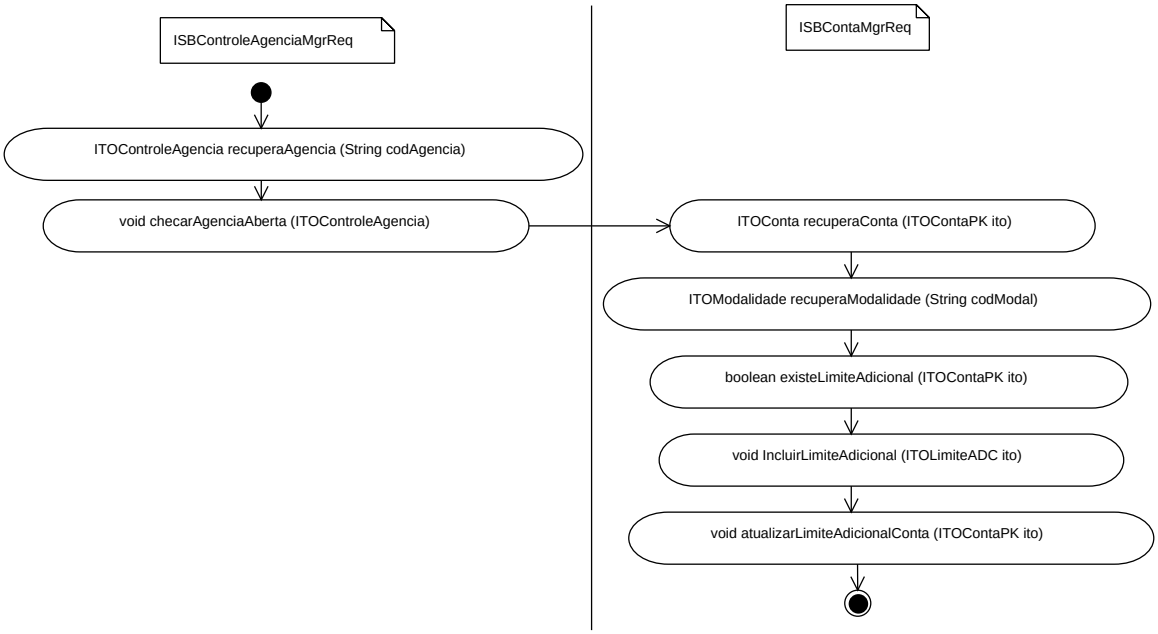

Figura 35: Diagrama para Identificação das Operações de Negócio: Caso de Uso Cadastramento de Limite Adicional

Definição Comportamento Excepcional (Figura 36)

Revisão dos componentes a serem testados

Nenhuma modificação realizada.

Definição de Restrições Dinâmicas

Não há restrições dinâmicas.

Definição Interfaces Requeridas

Componente: operacaoConta (interface requerida: IOperacaoContaNegReq)

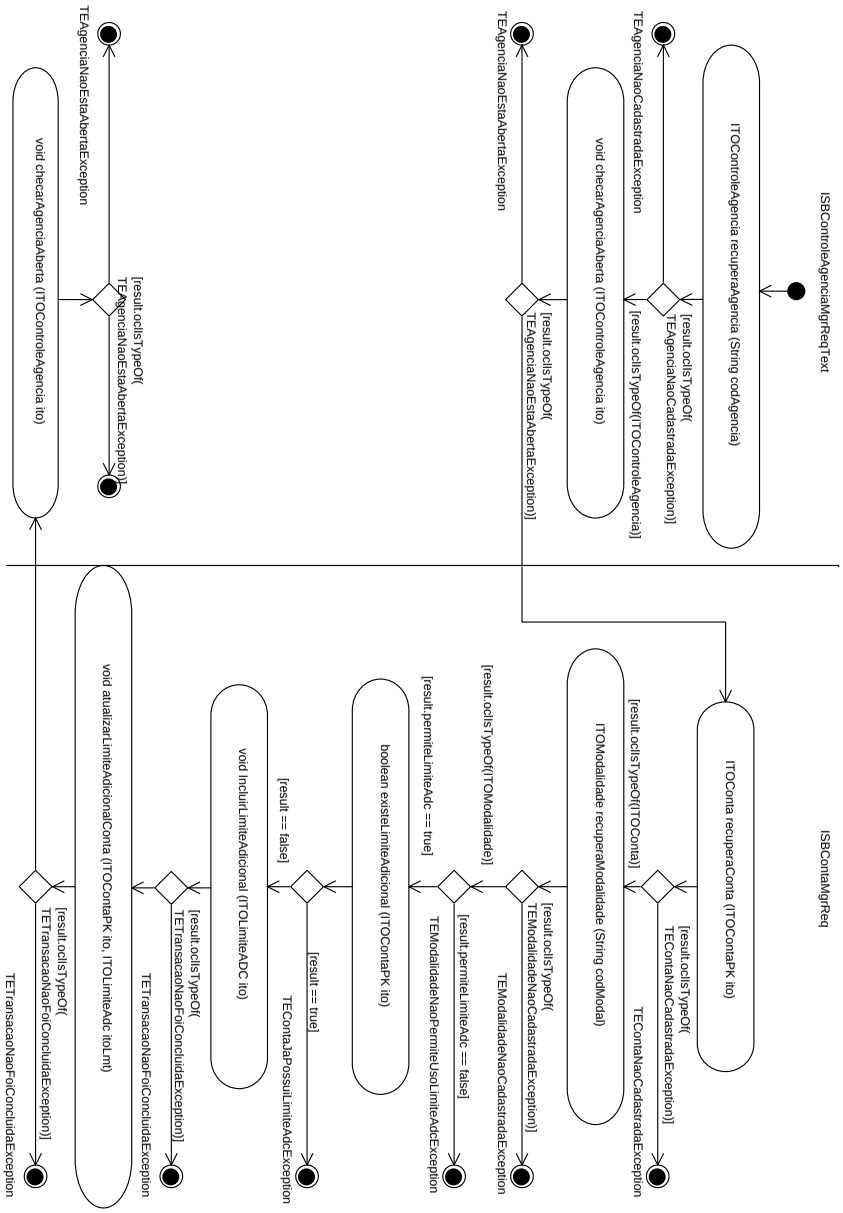

Figura 36: Diagrama para Definição Comportamento Excepcional: Caso de Uso Cadastramento de Limite Adicional

6.2.7 Estruturação dos Componentes Tolerantes a Falhas Ideais

Após a criação dos componentes do sistema e a análise das exceções propagadas por eles, pode-se realizar a estruturação segundo o modelo do componente tolerante a falhas ideal [AL90]. Para isso, para cada operação dos componentes normais, deve-se associar as possíveis exceções lançadas por ela.

Componentes de Sistema:

As associações dos componentes de sistema podem ser vistas nas Tabelas 21, 22, 23 e 24.

operacoesTalao.ISBReqTalaoMgr		
Operação:	Exceção:	Tratador:
? requisitarTalao(?)	E1(propagada)	IE1.lancar(E1 e) throws E1
	E2	IE2.lancar(E2 e) throws E2
	E3(propagada)	IE3.lancar(E3 e) throws E3
	E4	IE4.lancar(E4 e) throws E4
	E5(propagada)	IE5.lancar(E5 e) throws E5
	E6	IE6.lancar(E6 e) throws E6
	E7	IE7.lancar(E7 e) throws E7
	E8(propagada)	IE8.lancar(E8 e) throws E8
	E9	IE9.lancar(E9 e) throws E9
	E10	IE10.lancar(E10 e) throws E10
	E11	IE11.lancar(E11 e) throws E11
	E12	IE12.lancar(E12 e) throws E12
	E13	IE13.lancar(E13 e) throws E13
	E14	IE14.lancar(E14 e) throws E14
	E15	IE15.lancar(E15 e) throws E15
	E18(propagada)	IE18.lancar(E18 e) throws E18
	E19(propagada)	IE19.lancar(E19 e) throws E19
	E57(propagada, interna)	IE57.lancar(E57 e) throws E15
? identificarTipoTalao(?)	–	–
operacoesTalao.ISBEntregaTalaoMgr		
Operação:	Exceção:	Tratador:
? entregarTalao(?)	E1(propagada)	IE1.lancar(E1 e) throws E1
	E2	IE2.lancar(E2 e) throws E2
	E3(propagada)	IE3.lancar(E3 e) throws E3
	E4	IE4.lancar(E4 e) throws E4
	E5(propagada)	IE5.lancar(E5 e) throws E5
	E6	IE6.lancar(E6 e) throws E6
	E15(propagada)	IE15.lancar(E15 e) throws E15
	E20	IE20.lancar(E20 e) throws E20
	E21	IE21.lancar(E21 e) throws E21
	E22(propagada)	IE22.lancar(E22 e) throws E22
	E23(propagada)	IE23.lancar(E23 e) throws E23
	E24	IE24.lancar(E24 e) throws E24

Tabela 21: Componente de Sistema: *operacoesTalao*

Componentes de Negócio:

As associações dos componentes de negócio podem ser vistas na Tabela 25.

operacoesChequeSustado.ISBSustarChequeMgr		
Operação:	Exceção:	Tratador:
? sustarCheque(?)	E1(propagada)	IE1.lancar(E1 e) throws E1
	E2	IE2.lancar(E2 e) throws E2
	E3(propagada)	IE3.lancar(E3 e) throws E3
	E4	IE4.lancar(E4 e) throws E4
	E5(propagada)	IE5.lancar(E5 e) throws E5
	E15(propagada)	IE15.lancar(E15 e) throws E15
	E21	IE21.lancar(E21 e) throws E21
	E25	IE25.lancar(E25 e) throws E25
	E25	IE25.lancar(E25 e) throws E25
	E26	IE26.lancar(E26 e) throws E26
	E27	IE27.lancar(E27 e) throws E27
	E29(propagada)	IE29.lancar(E29 e) throws E29
? inserirCheque(?)	E15(propagada)	IE15.lancar(E15 e) throws E15

Tabela 22: Componente de Sistema: *operacoesChequeSustado*

operacoesChequeCapturado.ISBCapturaChequeMgr		
Operação:	Exceção:	Tratador:
? capturarCheque(?)	E15(propagada)	IE15.lancar(E15 e) throws E15
	E28	IE28.lancar(E28 e) throws E28
	E30	IE30.lancar(E30 e) throws E30
	E31	IE31.lancar(E31 e) throws E31
	E32	IE32.lancar(E32 e) throws E32
	E33	IE33.lancar(E33 e) throws E33
	E34	IE34.lancar(E34 e) throws E34
	E35	IE35.lancar(E35 e) throws E35
	E36	IE36.lancar(E36 e) throws E36
	E37	IE37.lancar(E37 e) throws E37
	E38	IE38.lancar(E38 e) throws E38
	E39(propagada)	IE39.lancar(E39 e) throws E39
	E40(propagada)	IE40.lancar(E40 e) throws E40
	E41	IE41.lancar(E41 e) throws E41
	E42	IE42.lancar(E42 e) throws E42
	E43	IE43.lancar(E43 e) throws E43
	E44	IE44.lancar(E44 e) throws E44
? calcularSequenciaCheque(?)	–	–
? checarConsistenciaDadosCheque(?)	?	?

Tabela 23: Componente de Sistema: *operacoesChequeCapturado*

operacoesConta.ISBCancelaContratoContaMgr		
Operação:	Exceção:	Tratador:
? cancelarContrato(?)	E1(propagada)	IE1.lancar(E1 e) throws E1
	E2	IE2.lancar(E2 e) throws E2
	E3(propagada)	IE3.lancar(E3 e) throws E3
	E4	IE4.lancar(E4 e) throws E4
	E5(propagada)	IE5.lancar(E5 e) throws E5
	E6	IE6.lancar(E6 e) throws E6
	E15(interna, propagada)	IE15.reconfigurar(E15 e) throws E56
	E19(propagada)	IE19.lancar(E19 e) throws E19
	E45	IE45.lancar(E45 e) throws E45
	E46	IE46.lancar(E46 e) throws E46
	E47	IE47.lancar(E47 e) throws E47
	E48(propagada)	IE48.lancar(E48 e) throws E48
	E50	IE50.lancar(E50 e) throws E50
	E53(propagada)	IE53.lancar(E53 e) throws E53
	E54	IE54.lancar(E54 e) throws E54
	E56	IE56.lancar(E56 e) throws E56
operacoesConta.ISBCadLimiteAdcMgr		
Operação:	Exceção:	Tratador:
? cadastrarLimiteAdc(?)	E1(propagada)	IE1.lancar(E1 e) throws E1
	E2	IE2.lancar(E2 e) throws E2
	E3(propagada)	IE3.lancar(E3 e) throws E3
	E4	IE4.lancar(E4 e) throws E4
	E5(propagada)	IE5.lancar(E5 e) throws E5
	E6	IE6.lancar(E6 e) throws E6
	E15(propagada)	IE15.lancar(E15 e) throws E15
	E19(propagada)	IE19.lancar(E19 e) throws E19
	E49	IE49.lancar(E49 e) throws E49
	E50	IE50.lancar(E50 e) throws E50
	E51	IE51.lancar(E51 e) throws E51
	E52(propagada)	IE52.lancar(E52 e) throws E52

Tabela 24: Componente de Sistema: *operacoesConta*

gerenciamentoConta.ISBContaMgrReq		
Operação:	Exceção:	Tratador:
ITOConta recuperarConta(ITOContaPK ito)	E5	IE5.lancar(E5 e) throws E5
void checarTitular(ITOContaPK ito, String codCliente)	E8	IE8.lancar(E8 e) throws E8
ITOModalidade recuperarModalidade(String codModal)	E19	IE19.lancar(E19 e) throws E19
void incluirTalao(ITOTalao ito)	E57	IE57.lancar(E57 e) throws E57
void checarTalaoEstoque(ITOTalaoPK ito, ?quant?)	E22	IE22.lancar(E22 e) throws E22
void entregarTalao(ITOTalao ito)	E15 E23	IE15.lancar(E15 e) throws E15 IE23.lancar(E23 e) throws E23
void desbloqueiaTalao(ITOTalaoPK ito)	E15	IE15.lancar(E15 e) throws E15
int verificarQtdTalaoParaRequisitar(ITOContaPK ito)	–	–
boolean existeChequeSustado(ITOTalao ito)	–	–
void existeChequeSustado(ITOTalao ito)	E15	IE15.lancar(E15 e) throws E15
ITOTipoDeposito recuperarTipoDeposito(String tipoDepos)	E53	IE53.lancar(E53 e) throws E53
void checarContratoCancelado(ITOContaPK ito)	E48	IE48.lancar(E48 e) throws E48
void cancelarContrato(ITODadosCancContrato ito)	E15	IE15.lancar(E15 e) throws E15
boolean existeChequeSustado(ITOTalao ito)	–	–
void atualizarLimiteAdicionalConta(ITOConta ito)	E15	IE15.lancar(E15 e) throws E15

gerenciamentoColigada.ISBColigadaMgrReq		
Operação:	Exceção:	Tratador:
ITOColigada recuperarColigada(String codColigada)	E18	IE18.lancar(E18 e) throws E18
void checarIFFuncionandoCompCheque(String codBanco)	E40	IE40.lancar(E40 e) throws E40

gerenciamentoMovimento.ISBMovimentoMgrReq		
Operação:	Exceção:	Tratador:
void cobrarTarifaTalao(ITOTalao ito)	E15	IE15.lancar(E15 e) throws E15
void cobrarTarifaChequeSustado(ITODadosTarifaChequeSustado ito)	E15	IE15.lancar(E15 e) throws E15
void checarChequeJaCapturado(ITODadosChequePesq ito)	E39	IE39.lancar(E39 e) throws E39
void inserirChequeCapturado(ITOCheque ito)	E15	IE15.lancar(E15 e) throws E15

gerenciamentoControleAgencia.ISBControleAgenciaMgrReq		
Operação:	Exceção:	Tratador:
ITOControleAgencia recuperarAgencia(String codAgencia)	E3	IE3.lancar(E3 e) throws E3
void checarAgenciaAberta(ITOControleAgencia ito)	E1	IE1.lancar(E1 e) throws E1

Tabela 25: Componentes de Negócio: *'gerenciamentoConta', 'gerenciamentoColigada', 'gerenciamentoMovimento' e 'gerenciamentoControleAgencia'*

6.3 Especificação Final dos Componentes

Suas subfases são:

1. Refinamento das Interfaces dos Componentes: *Atividade de Desenvolvimento*

   Entradas: interfaces de sistema e de negócio.

   Nesta fase, as interfaces dos componentes podem ser agrupadas. Esse agrupamento possibilita a redução do número de interfaces, sem comprometer demasiadamente a coesão entre as respectivas operações.

   Saídas: interfaces de sistema e de negócio refatoradas.

2. Diagramas das Interfaces Providas: *Atividade de Testes*

   Entradas: interfaces de sistema e casos de uso.

   São diagramas de atividades construídos para cada interface, que ilustram a seqüência lógica de chamada do métodos de uma interface, além de exceções lançadas de acordo com valores de parâmetros passsados ao método.

   As exceções são representadas por nós finais contendo como rótulo o tipo da exceção, e são ligadas a atividade que representa o método por uma aresta contendo a condição de guarda para o lançamento da exceção.

   Saídas: diagramas de atividades para cada interface escolhida representando o fluxo de execução da interface.

3. Modelo de Informação das Interfaces (Interface Information Model): *Atividade de Desenvolvimento*

   Entradas: interfaces de negócio e de sistema, modelo de tipos do negócio, tipos de dados.

   É definido um modelo de informações para cada interface normal existente. Este modelo consiste da identificação das entidades do modelo de tipos do negócio *(Business Type Model)* que sejam relevantes para a interface. Além dessas entidades, são destacados os tipos de dados *(data types)* utilizados.

   Saídas: lista de entidades relevantes para cada uma das interfaces

4. Formalização das Assertivas: *Atividade de Desenvolvimento e Testes*

   Entradas: casos de uso (assertivas), interfaces de sistema e negócio, tipos de dados, classes de exceção.

   Opcional para o desenvolvimento, esta atividade é fundamental para os testes, pois gera as assertivas na linguagem OCL para posterior implementação da arquitetura do componente testável. O objetivo da formalização é em OCL é a conversão automática para a implementação em Java usando programação orientada a aspectos.

   Apesar de serem baseadas nas assertivas levantadas durante a especificação de requisitos, as assertivas formais diferem das iniciais pois levam em conta entradas e saídas

do método, além de sua interação com as interfaces requeridas. Assim, muitas pré-condições levantadas na fase de requisitos não podem ser consideradas nessa fase, já que não é possível verificá-las apenas baseando-se em informações disponíveis para o componente de sistema. No caso da pré-condição "Conta não cadastrada", por exemplo, sua verficação depende da interação com o componente de negócio operacaoConta, aumentando a complexidade da assertiva, o que não é recomendado [1].

Por isso, decidiu-se formalizar as assertivas ao pares, como propõe o método UML Components [CD00]. Para cada resultado esperado do método, a pré-condição necessária para aquele resultado e a pós-condição esperada. O resultado normal é assinalado por norm.condition/norm.post, e os excepcionais por nomeExceção.condition/nomeExceção.post

Saídas: contratos das interfaces dos componentes em OCL.

6.3.1 Componente de Sistema: operacoesTalao

Refinamento das Interfaces dos Componentes

Houve agrupamento nas interfaces do componente *'operacoesTalao'*. As interfaces *'ISBReqTalaoMgr'* e *'ISBEntregaTalaoMgr'* foram agrupadas na interface *'ISBObtencaoTalaoMgr'*. A nova interface pode ser vista na Tabela 26; e o componente 'operaçõesTalao' pode ser visto na Figura 37.

Interface de Sistema: ISBObtencaoTalaoMgr
Operações:
void requisitarTalao(ITODadosReqTalao ito)
int identificarTipoTalao(ITODadosReqTalao ito)
void entregarTalao(ITODadosEntTalao ito)

Tabela 26: Interface e Operações *ISBObtencaoTalaoMgr = ISBReqTalaoMgr + ISBEntregaTalaoMgr*

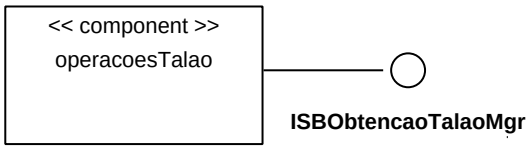

Figura 37: Componente de Sistema: *operacoesTalao*

Diagrama de Atividades das Interfaces Providas
(Figura 38)

Modelo de Informações das Interfaces (Interface Information Model)

As entidades do modelo de informações da Interface ISBObtencaoTalaoMgr podem ser vistas na Figura 39. Os tipos de dados utilizados estão listados na Tabela 27.

Tipos de Dados da Interface de Sistema: ISBObtencaoTalaoMgr
ITODadosReqTalao, ITODadosEntTalao.

Tabela 27: Tipos de Dados utilizados pela Interface de Negócio: ISBObtencaoTalaoMgr

Formalização das Assertivas

context operacaoTalao
inv: true
context operacaoTalao.ISBObtencaoTalaoMgr :: void requisitarTalao (ITO-DadosReqTalao ito)

pre: true
– *Normal*
norm.condition: (ito.codAgencia !=) and (ito.codAgencia != null) and
(ito.nroConta !=) and (ito.nroConta != null) and
(ito.nomeUsuario !=) and (ito.nomeUsuario != null) and
(ito.codCliente1 !=) and (ito.codCliente1 != null) and
(ito.qtdeTalao > 0) and
((ito.tipoTalao == ITOTalao.TIPO_TALAO_CONTINUO) or
(ito.tipoTalao == ITOTalao.TIPO_TALAO_NORMAL) or
(ito.tipoTalao == ITOTalao.TIPO_TALAO_SEM_CPMF)) and
((ito.tipoSolic == ITOTalao.TIPO_SOLICITACAO_AUTOMATICA) or
(ito.tipoSolic == ITOTalao.TIPO_SOLICITACAO_MANUAL)) and
(ito.codCliente1 != ito.codCliente2)

norm.post: true – *Método retorna void*

– *E1 - Agência não está aberta*
TEAgenciaNaoEstaAbertaException.condition:
(let message: OclMessage = ISBNegocioReq^checarAgenciaAberta(ITOControleAgencia)))
in
message.hasReturned() and
message.result().oclIsTypeOf(TEAgenciaNaoEstaAbertaException)
TEAgenciaNaoEstaAbertaException.post:
result.oclIsTypeOf(TEAgenciaNaoEstaAbertaException) and
(result.getCodAgencia() == ito.codAgencia@pre)

– *E2 - Agência inválida*
TEAgenciaInvalidaException.condition: (ito.codAgencia ==) or (ito.codAgencia == null)
TEAgenciaInvalidaException.post:
result.oclIsTypeOf(TEAgenciaInvalidaException) and
(result.getCodAgencia() == ito.codAgencia@pre)

– *E3 - Agência Não cadastrada*
TEAgenciaNaoCadastradaException.condition:
(let message: OclMessage = ISBNegocioReq^recuperaAgencia(String))) in
message.hasReturned() and
message.result().oclIsTypeOf(TEAgenciaNaoCadastradaException)
TEAgenciaNaoCadastradaException.post:
result.oclIsTypeOf(TEAgenciaNaoCadastradaException) and
(result.getCodAgencia() == ito.codAgencia@pre)

– *E4 - Conta inválida*
TEContaInvalidaException.condition: (ito.nroConta ==) or (ito.nroConta == null)
TEContaInvalidaException.post:
result.oclIsTypeOf(TEContaInvalidaException) and
(result.getNumConta() == ito.nroConta@pre)

– *E5 - Conta não cadastrada*
TEContaNaoCadastradaException.condition:
(let message: OclMessage = ISBNegocioReq^recuperaConta(String))) in
message.hasReturned() and
message.result().oclIsTypeOf(TEContaNaoCadastradaException)
TEContaNaoCadastradaException.post:
result.oclIsTypeOf(TEContaNaoCadastradaException) and
(result.getNumConta() == ito.nroConta@pre)

– *E6 - Usuário inválido*
TEUsuarioInvalidoException.condition: (ito.nomeUsuario ==) or (ito.nomeUsuario == null)
TEUsuarioInvalidoException.post:
result.oclIsTypeOf(TEUsuarioInvalidoException) and
(result.getNomeUsuario() == ito.nomeUsuario@pre)

– *E7 - Código do cliente inválido*
TECodClienteInvalidoException.condition: (ito.codCliente1 ==) or (ito.codCliente1 == null)
TECodClienteInvalidoException.post:
result.oclIsTypeOf(TECodClienteInvalidoException) and
(result.getCodCliente1() == ito.codCliente1@pre)

– *E8 - Cliente não é titular da conta*
TEClienteNaoETitularContaException.condition: Idem normal
TEClienteNaoETitularContaException.post: (((result.getCodAgencia() == ito.codAgencia@pre) and (result.getNumConta() == ito.numConta@pre) and ((result.getCodCliente1() == ito.codCliente1@pre) or ((ito.codCliente2 !=) or (ito.codCliente2 != null)) and (result.getCodCliente2() == ito.codCliente2@pre))

– *E9 - A quantidade de talão requisitada deve ser maior que zero*
TEQtdeTalaoDeveSerMaiorZeroException.condition: (ito.qtdeTalao < 0)
TEQtdeTalaoDeveSerMaiorZeroException.post:
result.oclIsTypeOf(TEQtdeTalaoDeveSerMaiorZeroException)

– *E10 - Tipo de talao invalido*
TETipoTalaoInvalidoException.condition:
(ito.tipoTalao != ITOTalao.TIPO_TALAO_CONTINUO) and
(ito.tipoTalao != ITOTalao.TIPO_TALAO_NORMAL) and
(ito.tipoTalao != ITOTalao.TIPO_TALAO_SEM_CPMF)
TETipoTalaoInvalidoException.post:
result.oclIsTypeOf(TETipoTalaoInvalidoException) and
(result.getTipoTalao() == ito.tipoTalao@pre)

– *E11 - Tipo de solicitação inválido*
TETipoSolicitacaoTalaoInvalidoException.condition:
(ito.tipoSolic != ITOTalao.TIPO_SOLICITACAO_AUTOMATICA) and
(ito.tipoSolic != ITOTalao.TIPO_SOLICITACAO_MANUAL)
TETipoSolicitacaoTalaoInvalidoException.post:
result.oclIsTypeOf(TETipoTalaoInvalidoException) and
(result.getTipoSolicit() == tipoSolicit@pre)

– *E12 - Clientes devem ser distintos*
TEClientesDevemSerDistintosException.condition:
(ito.codCliente1 == ito.codCliente2)
TEClientesDevemSerDistintosException.post:
result.oclIsTypeOf(TEClientesDevemSerDistintosException) and
(result.getCodCliente1() == ito.codCliente1@pre) and
(result.getCodCliente2() == ito.codCliente2@pre)

– *E13 - Requisição de talão bloqueada para esta conta*
TERequisicaoTalaoEstaBloqueadaContaException.condition:
(let message: OclMessage = ISBNegocioReq^recuperaConta(String))) in
message.hasReturned() and
message.result().oclIsTypeOf(ITOConta) and
(message.result().isTalaoBloqueado == true)

TERequisicaoTalaoEstaBloqueadaContaException.post:
result.oclIsTypeOf(TERequisicaoTalaoEstaBloqueadaContaException)

– *E14 - Requisição de talão impossibilitada para conta interna*
TERequisicaoTalaoImpossibilitadaContaInternaException.condition:
(let message: OclMessage = ISBNegocioReq^recuperaConta(String)) in
message.hasReturned() and
message.result().oclIsTypeOf(ITOConta) and
(message.result().tipoDepos == ITOConta.TIPO_DEPOSITO_INTERNO)
TERequisicaoTalaoImpossibilitadaContaInternaException.post:
result.oclIsTypeOf(TERequisicaoTalaoImpossibilitadaContaInterfaceException)

– *E18 - Coligada não cadastrada*
TEColigadaNaoCadastradaException.condition:
(let message: OclMessage = ISBNegocioReq^recuperaColigada(String)) in
message.hasReturned() and
message.result().oclIsTypeOf(TEColigadaNaoCadastradaException)
TEColigadaNaoCadastradaException.post:
result.oclIsTypeOf(TEColigadaNaoCadastradaException) and
(let message2: OclMessage = ISBNegocioReq^recuperaAgencia(String)) in
message2.hasReturned() and
message2.result().oclIsTypeOf(ITOControleAgencia) and
(result.getCodColigada() == message2.result().codColigada)

– *E19 - Modalidade não cadastrada*
TEModalidadeNaoCadastradaException.condition:
(let message: OclMessage = ISBNegocioReq^recuperaModalidade(String)) in
message.hasReturned() and
message.result().oclIsTypeOf(TEModalidadeNaoCadastradaException)
TEModalidadeNaoCadastradaException.post:
result.oclIsTypeOf(TEModalidadeNaoCadastradaException) and
(let message2: OclMessage = ISBNegocioReq^recuperaConta(String)) in
message2.hasReturned() and
message2.result().oclIsTypeOf(ITOConta) and
(result.getCodModalidade() = message2.result().codModal)

– *E57 - Talão já Cadastrado*
TERegistroJaExistenteException.condition:
(let message: OclMessage = ISBNegocioReq^incluiTalao(ITOTalao)) in
message.hasReturned() and
message.result().oclIsTypeOf(TERegistroJaExistenteException)
TERegistroJaExistenteException.post:
if (result.oclIsKindOf(java.lang.Exception)
result.oclIsTypeOf(TETransacaoNaoFoiConcluida)

context operacaoTalao.ISBObtencaoTalaoMgr ::
void entregarTalao (ITODadosEntTalao ito)

pre: true
– *Normal*
norm.condition: (ito.codAgencia !=) and (ito.codAgencia != null) and
(ito.codAgenciaLocal !=) and (ito.codAgenciaLocal != null) and
(ito.nroConta !=) and (ito.nroConta != null) and
(ito.especieTalao !=) and (ito.especieTalao != null) and
(ito.nroPriCheque !=) and (ito.nroPriCheque != null) and
(ito.nomeUsuario !=) and (ito.nomeUsuario != null) and
norm.post: true *Método retorna void*

– *E1 - Agência não está aberta*
TEAgenciaNaoEstaAbertaException.condition:
(let message: OclMessage = ISBNegocioReq^checarAgenciaAberta(ITOControleAgencia))
in
message.hasReturned() and
message.result().oclIsTypeOf(TEAgenciaNaoEstaAbertaException)
TEAgenciaNaoEstaAbertaException.post: (
result.oclIsTypeOf(TEAgenciaNaoEstaAbertaException) and
(result.getCodAgencia() == ito.codAgencia@pre)

– *E2 - Agência inválida*
TEAgenciaInvalidaException.condition:
(ito.codAgencia ==) or (ito.codAgencia == null) or (ito.codAgenciaLocal ==) or
(ito.codAgenciaLocal == null)
TEAgenciaInvalidaException.post:
result.oclIsTypeOf(TEAgenciaInvalidaException) and
(result.getCodAgencia() == ito.codAgencia@pre)

– *E3 - Agência Não cadastrada*
TEAgenciaNaoCadastradaException.condition:
(let message: OclMessage = ISBNegocioReq^recuperaAgencia(String)) in
message.hasReturned() and
message.result().oclIsTypeOf(TEAgenciaNaoCadastradaException)
TEAgenciaNaoCadastradaException.post:
result.oclIsTypeOf(TEAgenciaNaoCadastradaException) and
(result.getCodAgencia() == ito.codAgencia@pre)

– *E4 - Conta inválida*
TEContaInvalidaException.condition:
(ito.nroConta ==) or (ito.nroConta == null)

TEContaInvalidaException.post:
result.oclIsTypeOf(TEContaInvalidaException) and
(result.getNumConta() == ito.nroConta@pre)

– *E5 - Conta não cadastrada*
TEContaNaoCadastradaException.condition:
(let message: OclMessage = ISBNegocioReq^recuperaConta(String)) in
message.hasReturned() and
message.result().oclIsTypeOf(TEContaNaoCadastradaException)
TEContaNaoCadastradaException.post:
result.oclIsTypeOf(TEContaNaoCadastradaException) and
(result.getNumConta() == ito.nroConta@pre)

– *E6 - Usuário inválido*
TEUsuarioInvalidoException.condition:
(ito.nomeUsuario ==) or (ito.nomeUsuario == null)
TEUsuarioInvalidoException.post:
result.oclIsTypeOf(TEUsuarioInvalidoException) and
(result.getNomeUsuario() == ito.nomeUsuario@pre)

– *E15 - Transação não foi concluída*

– *E20 - Espécie Inválida*
TEEspecieTalaoInvalidaException.condition:
(ito.especieTalao ==) or (ito.especieTalao == null)
TEEspecieTalaoInvalidaException.post:
result.oclIsTypeOf(TEEspecieTalaoInvalidaException) and
(result.getEspecie() == ito.especieTalao@pre)

– *E21 - Número do primeiro cheque inválido*
TENroPriChequeInvalidoException.condition:
(ito.nroPriCheque ==) or (ito.nroPriCheque == null)
TENroPriChequeInvalidoException.post:
result.oclIsTypeOf(TENroPriChequeInvalidoException) and

– *E22 - Não existe talão em estoque*
TENaoExisteTalaoEstoqueException.condition:
(let message: OclMessage = ISBNegocioReq^checarTalaoEstoque(ITOTalaoPK ito, int quant)) in
message.hasReturned() and
message.result().oclIsTypeOf(TENaoExisteTalaoEstoqueException)
TENaoExisteTalaoEstoqueException.post:
result.oclIsTypeOf(TENaoExisteTalaoEstoqueException)

– *E23 - Talão já foi entregue*
TETalaoJaEntregueException.condition:
(let message: OclMessage = ISBNegocioReq^entregarTalao(ITOTalao)) in
message.hasReturned() and
message.result().oclIsTypeOf(TETalaoJaEntregueException)
TETalaoJaEntregueException.post:
result.oclIsTypeOf(TETalaoJaEntregueException)

– *E24 - Conta bloqueada para entrega de talão*
TEContaBloqueadaParaEntregaTalaoException.condition:
(let message: OclMessage = ISBNegocioReq^recuperaConta(ITOContaPK)) in
message.hasReturned() and
message.result().oclIsTypeOf(ITOConta) and
message.isTalaoBloqueado == true **TEContaBloqueadaParaEntregaTalaoException.post::**
result.oclIsTypeOf(TEContaBloqueadaParaEntregaTalaoException)

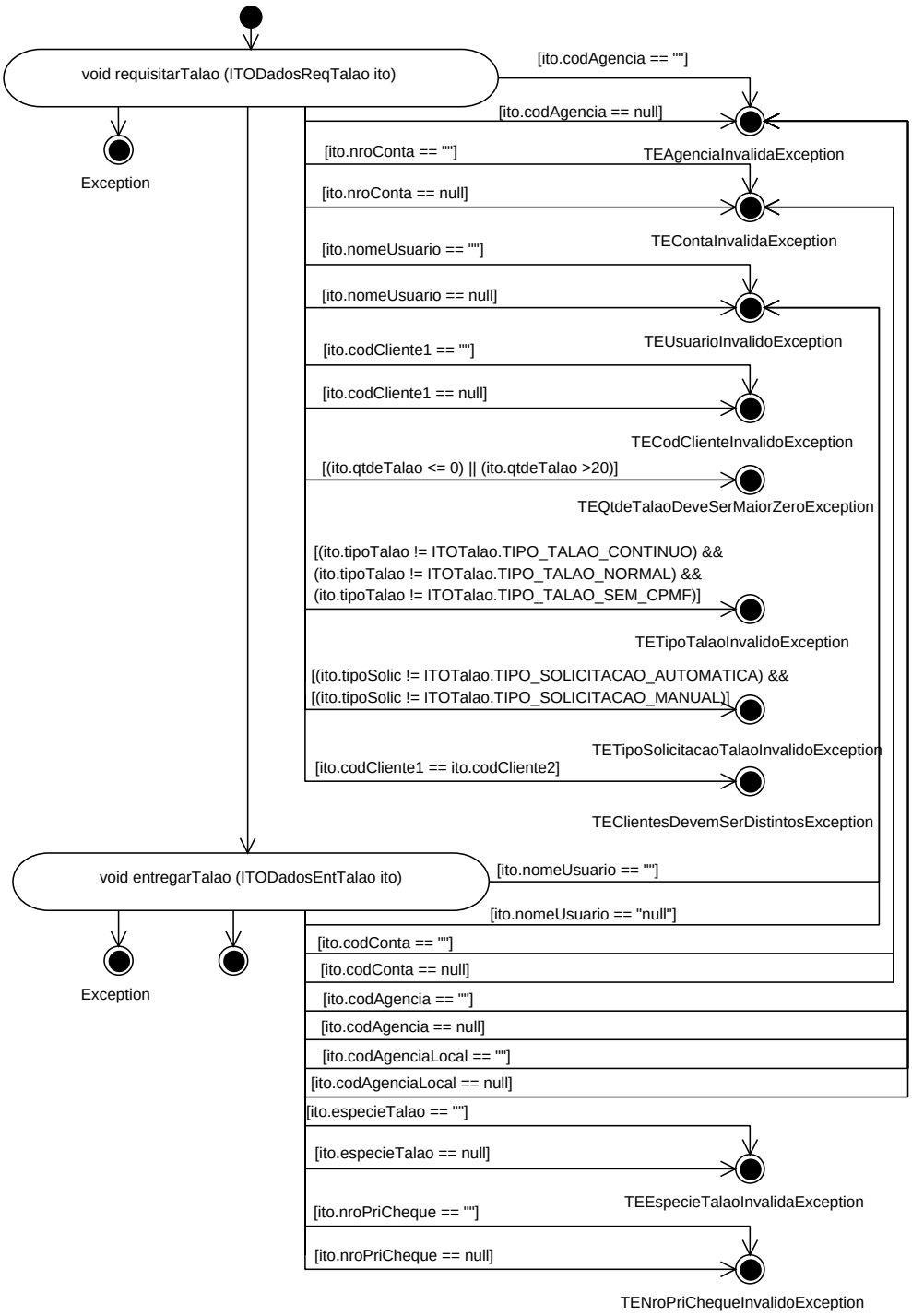

Figura 38: Diagrama da Interface Provida do Componente operacaoTalao: Interface IS-BObtencaoTalaoMgr

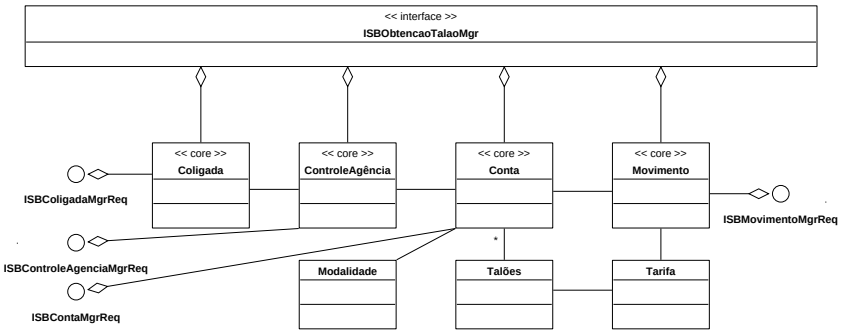

Figura 39: Interface Information Model da Interface: ISBObtencaoTalaoMgr

6.3.2 Componente de Sistema: operacoesChequesCapturados

Refinamento das Interfaces dos Componentes

Não houve agrupamento nas interfaces do componente *'operacoesChequesCapturados'*.

Diagrama de Atividades das Interfaces Providas
(Figura 40)

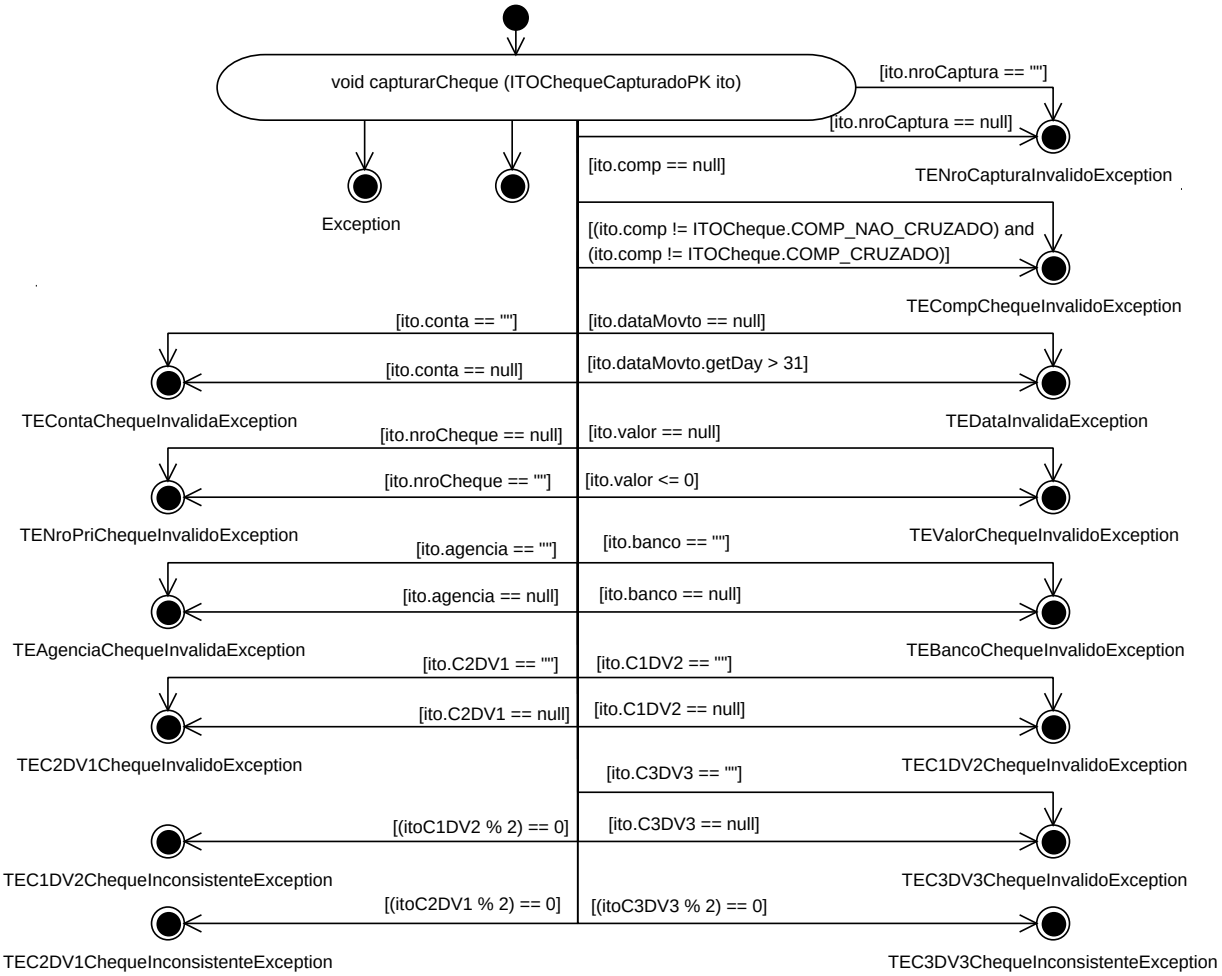

Figura 40: Diagrama da Interface Provida do Componente operacaoChequesCapturados: Interface ISBCapturaChequeMgr

Modelo de Informações das Interfaces (Interface Information Model)

As entidades do modelo de informações da Interface ISBCapturaChequeMgr podem ser vistas na Figura 41. Os tipos de dados utilizados estão listados na Tabela 28.

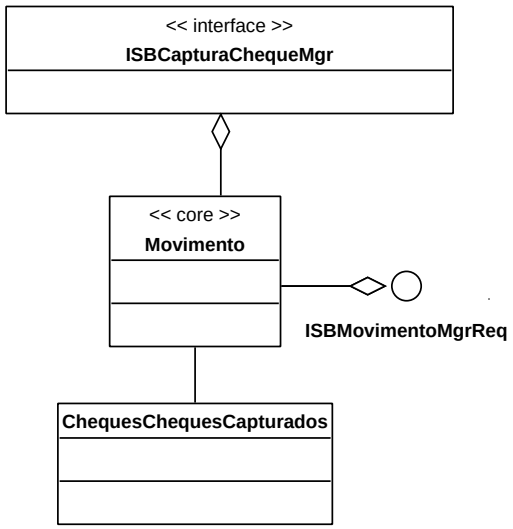

Figura 41: Interface Information Model da Interface: ISBCapturaChequeMgr

Tipos de Dados da Interface de Sistema: ISBCapturaChequeMgr
ITOChequeCapturadoPK, ITOCheque.

Tabela 28: Tipos de Dados utilizados pela Interface de Negócio: ISBCapturaChequeMgr

Formalização das Assertivas

context operacaoChequesCapturados
inv: Não tem
context operacaoChequesCapturados.ISBCapturaChequeMgr ::
void capturarCheque (ITOChequeCapturado ito)

pre: true
– *Normal*
norm.condition:
(ito.dataMovto != null) and (ito.dataMovto.getDay() > 31) and
(ito.nroCaptura !=) and (ito.nroCaptura != null) and
(ito.valor > 0) and
((ito.comp == ITOCheque.COMP_NAO_CRUZADO) or

(ito.comp == ITOCheque.COMP_CRUZADO)) and
(ito.banco !=) and (ito.banco != null) and
(ito.agencia !=) and (ito.agencia != null) and
(ito.C1DV2 !=) and (ito.C1DV2 != null) and ((ito.C1DV2 % 2) != 0) and
(ito.nroConta !=) and (ito.nroConta != null) and
(ito.C2DV1 !=) and (ito.C2DV1 != null) and ((ito.C2DV1 % 2) != 0) and
(ito.C3DV3 !=) and (ito.C3DV3 != null) and ((ito.C3DV3 % 2) != 0) and
(ito.nroCheque !=) and (ito.nroCheque != null)
norm.post: true

– *E15 - Transação não foi concluída*

– *E28 - Data inválida*
TEDataInvalidaException.condition:
(ito.dataSustacao == null) or (ito.dataSustacao.getDay() > 31)
TEDataInvalidaException.post:
result.oclIsTypeOf(TEDataInvalidaException) and
(result.getData() == ito.dataSustacao@pre)

– *E30 - Número da captura inválido*
TENroCapturaInvalidoException.condition:
(ito.nroCaptura ==) or (ito.nroCaptura ==null)
TENroCapturaInvalidoException.post:
result.oclIsTypeOf(TENroCapturaInvalidoException) and
(result.getNumCaptura() == ito.nroCaptura@pre)

– *E31 - Valor do cheque inválido*
TEValorChequeInvalidoException.condition:
(ito.valor != null) and (ito.valor <= 0)
TEValorChequeInvalidoException.post:
result.oclIsTypeOf(TEValorChequeInvalidoException) and
(result.getNumCheque() == ito.nroCheque@pre) and
(result.getValor() == ito.valor)

– *E32 - Comp do cheque inválido*
TECompChequeInvalidoException.condition:
(ito.comp != ITOCheque.COMP_NAO_CRUZADO) and
(ito.comp != ITOCheque.COMP_CRUZADO)
TECompChequeInvalidoException.post:
result.oclIsTypeOf(TECompChequeInvalidoException) and
(result.getNumCheque() == ito.nroCheque@pre) and
(result.getComp() == ito.comp)

– *E33 - Banco do cheque inválido*

TEBancoChequeInvalidoException.condition: (ito.banco ==) or (ito.banco ==null)
TEBancoChequeInvalidoException.post:
result.oclIsTypeOf(TEBancoChequeInvalidoException) and
(result.getNumCheque() == ito.nroCheque@pre) and
(result.getBanco() == ito.banco)

– *E34 - Agência do cheque inválida*
TEAgenciaChequeInvalidaException.condition: (ito.agencia ==) or (ito.agencia ==null)
TEAgenciaChequeInvalidaException.post:
result.oclIsTypeOf(TEAgenciaChequeInvalidaException) and
(result.getNumCheque() == ito.nroCheque@pre) and
(result.getAgencia() == ito.agencia)

– *E35 - C1DV2 do cheque inválido*
TEC1DV2ChequeInvalidoException.condition: (ito.C1DV2 ==) or (ito.C1DV2 ==null)
TEC1DV2ChequeInvalidoException.post:
result.oclIsTypeOf(TEC1DV2ChequeInvalidoException) and
(result.getNumCheque() == ito.nroCheque@pre) and
(result.getC1DV2() == ito.C1DV2)

– *E36 - Conta do cheque inválida*
TEContaChequeInvalidaException.condition: (ito.nroConta ==) or (ito.nroConta == null)
TEContaChequeInvalidaException.post:
result.oclIsTypeOf(TEContaChequeInvalidaException) and
(result.getNumCheque() == ito.nroCheque@pre) and
(result.getConta() == ito.nroConta)

– *E37 - C2DV1 do cheque inválido*
TEC2DV1ChequeInvalidoException.condition: (ito.C2DV1 ==) or (ito.C2DV1 ==null)
TEC2DV1ChequeInvalidoException.post:
result.oclIsTypeOf(TEC2DV1ChequeInvalidoException) and
(result.getNumCheque() == ito.nroCheque@pre) and
(result.getC2DV1() == ito.C2DV1)

– *E38 - Número do cheque inválido*
TENroChequeInvalidoException.condition: (ito.nroCheque ==) or (ito.nroCheque ==null)
TENroChequeInvalidoException.post:
result.oclIsTypeOf(TENroChequeInvalidoException) and
(result.getNumCheque() == ito.nroCheque@pre)

– *E39 - Cheque já capturado neste dia (data do movimento)*
TEChequeJaCapturadoNoDiaException.condition:
(let message: OclMessage = ISBNegocioReq^checarChequeJaCapturado(ITODadosChequePesq))
in
message.hasReturned() and
message.result().oclIsTypeOf(TEChequeJaCapturadoNoDiaException)
TEChequeJaCapturadoNoDiaException.post:
result.oclIsTypeOf(TEChequeJaCapturadoNoDiaException) and
(result.getNumCheque() == ito.nroCheque@pre) and
(result.getDataMovto() == ito.dataMovto)

– *E40 - Instituição financeira do cheque não está funcionando no sistema de compensação*
TEInstituicaoChequeNaoEstaCompException.condition:
(let message: OclMessage = ISBNegocioReq^checarIFFuncionandoCompCheque(String))
in
message.hasReturned() and
message.result().oclIsTypeOf(TEInstituicaoChequeNaoEstaCompException)
TEInstituicaoChequeNaoEstaCompException.post:
result.oclIsTypeOf(TEInstituicaoChequeNaoEstaCompException) and
(result.getCodBanco() == ito.banco@pre)

– *E41 - C3DV3 do cheque inválido*
TEC3DV3ChequeInvalidoException.condition: (ito.C3DV3 ==) and (ito.C3DV3 ==null)
TEC3DV3ChequeInvalidoException.post:
result.oclIsTypeOf(TEC3DV3ChequeInvalidoException) and
(result.getNumCheque() == ito.nroCheque@pre) and
(result.getC3DV3() == ito.C3DV3)

– *E42 - C1DV2 do cheque inconsistente*
TEC1DV2ChequeInconsistenteException.condition:
(ito.C1DV2 != null) and ((itoC1DV2 % 2) == 0)
TEC1DV2ChequeInconsistenteException.post:
result.oclIsTypeOf(TEC1DV2ChequeInconsistenteException) and
(result.getNumCheque() == ito.nroCheque@pre) and
(result.getC1DV2() == ito.C1DV2)

– *E43 - C2DV1 do cheque inconsistente*
TEC2DV1ChequeInconsistenteException.condition:
(ito.C2DV1 != null) and ((itoC2DV1 % 2) == 0)
TEC2DV1ChequeInconsistenteException.post:
result.oclIsTypeOf(TEC2DV1ChequeInconsistenteException) and

(result.getNumCheque() == ito.nroCheque@pre) and
(result.getC2DV1() == ito.C2DV1)

– *E44 - C3DV3 do cheque inconsistente*
TEC3DV3ChequeInconsistenteException.condition:
(ito.C3DV3 != null) and ((itoC3DV3 % 2) == 0)
TEC3DV3ChequeInconsistenteException.post:
result.oclIsTypeOf(TEC3DV3ChequeInconsistenteException) and
(result.getNumCheque() == ito.nroCheque@pre) and
(result.getC3DV3() == ito.C3DV3)

6.3.3 Componente de Sistema: operacoesChequeSustado

Refinamento das Interfaces dos Componentes

Não houve agrupamento nas interfaces do componente *'operacoesChequeSustado'*.

Diagrama de Atividades das Interfaces Providas
(Figura 42)

Modelo de Informações das Interfaces (Interface Information Model)

As entidades do modelo de informações da Interface ISBSustarChequeMgr podem ser vistas na Figura 43. Os tipos de dados utilizados estão listados na Tabela 29.

Tipos de Dados da Interface de Sistema: ISBSustarChequeMgr
ITODadosChequeSustado.

Tabela 29: Tipos de Dados utilizados pela Interface de Negócio: ISBSustarChequeMgr

Formalização das Assertivas

As exceções cuja assertiva foi apresentada no componente anterior não serão reapresentadas por questão de espaço.

context operacaoChequeSustado
inv: true

context operacaoChequeSustado.ISBSustarChequeMgr ::
void sustarCheque (ITODadosChequeSustado ito)
pre: true

– *Normal*
norm.condition:
(ito.codAgencia !=) and (ito.codAgencia != null) and
(ito.nroConta !=) and (ito.nroConta != null) and
(ito.nroPriCheque !=) and (ito.nroPriCheque != null) and
(ito.codAgenciaLocal !=) and (ito.codAgenciaLocal != null) and
(ito.qtdeCheques > 0) and (ito.qtdeCheques <= 20) and
((ito.motivoSustacao == ITOChequeSustado.MOTIVO_SUSTACAO_OUTRO) or
(ito.motivoSustacao == ITOChequeSustado.MOTIVO_SUSTACAO_PERDA) or
(ito.motivoSustacao == ITOChequeSustado.MOTIVO_SUSTACAO_ROUBO)) and
((ito.situacaoCheque == ITOChequeSustado.SITUACAO_CHEQUE_ASSINADO) or
(ito.situacaoCheque == ITOChequeSustado.SITUACAO_CHEQUE_BRANCO) or
(ito.situacaoCheque ==
ITOChequeSustado.SITUACAO_CHEQUE_TOTALMENTE_PREENCHIDO)) and

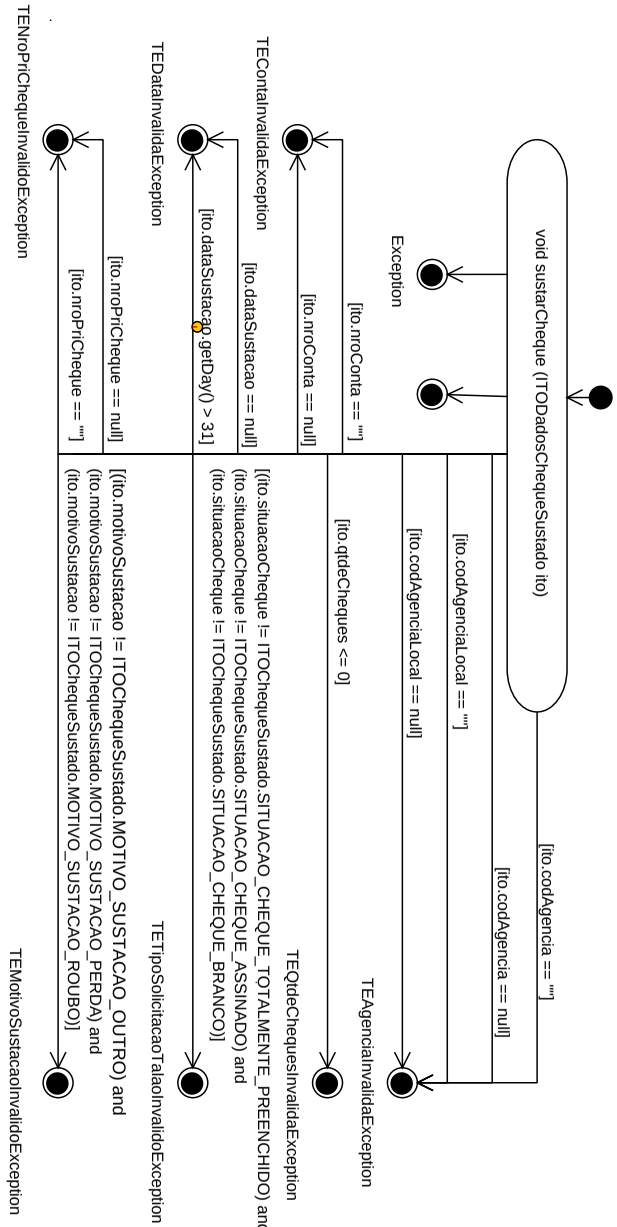

Figura 42: Diagrama da Interface Provida do Componente operacaoChequeSustado: Interface ISBSustarChequeMgr

(ito.dataSustacao != null) and (ito.dataSustacao.getDay() > 31)
norm.post: true

– *E1 - Agência não está aberta*

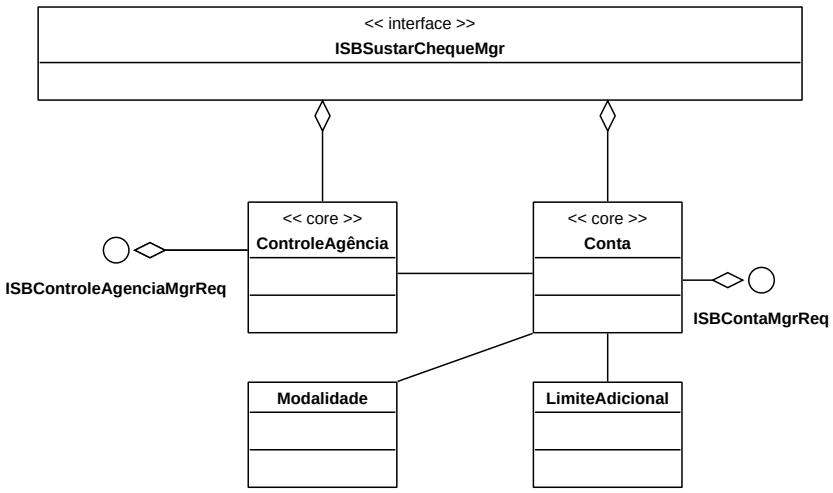

Figura 43: Interface Information Model da Interface: ISBSustarChequeMgr

– *E2 - Agência inválida*

– *E3 - Agência Não cadastrada*

– *E4 - Conta inválida*

– *E5 - Conta não cadastrada*

– *E15 - Transação não foi concluída*

– *E21 - Número do primeiro cheque inválido*
TENroPriChequeInvalidoException.condition:
(ito.nroPriCheque ==) or (ito.nroPriCheque == null)
TENroPriChequeInvalidoException.post:
result.oclIsTypeOf(TENroPriChequeInvalidoException) and
(result.getNroCheque() == ito.PriCheque@pre)

– *E25 - Quantidade de cheques inválida*
TEQtdeChequesInvalidaException.condition:
(ito.qtdeCheques <= 0) and (ito.qtdeCheques > 20)
TEQtdeChequesInvalidaException.post:
result.oclIsTypeOf(TEQtdeChequesInvalidaException) and
(result.getQuantCheques() == ito.qtdeCheques@pre)

– *E26 - Motivo de sustação inválido*

TEMotivoSustacaoInvalidoException.condition:
(ito.motivoSustacao != ITOChequeSustado.MOTIVO_SUSTACAO_OUTRO) and
(ito.motivoSustacao != ITOChequeSustado.MOTIVO_SUSTACAO_PERDA) and
(ito.motivoSustacao != ITOChequeSustado.MOTIVO_SUSTACAO_ROUBO)
TEMotivoSustacaoInvalidoException.post:
result.oclIsTypeOf(TEMotivoSustacaoInvalidoException) and
(result.getMotivoSustacao() == ito.motivoSustacao@pre)

– *E27 - Situação de sustação inválida*

TESituacaoSustacaoInvalidaException.condition:
(ito.situacaoCheque != ITOChequeSustado.SITUACAO_CHEQUE_ASSINADO) and
(ito.situacaoCheque != ITOChequeSustado.SITUACAO_CHEQUE_BRANCO) and
(ito.situacaoCheque !=
ITOChequeSustado.SITUACAO_CHEQUE_TOTALMENTE_PREENCHIDO)
TESituacaoSustacaoInvalidaException.post:
result.oclIsTypeOf(TESituacaoSustacaoInvalidaException) and
(result.getSituacaoCheque() == ito.situacaoCheque@pre)

– *E28 - Data inválida*

TEDataInvalidaException.condition:
(ito.dataSustacao == null) or (ito.dataSustacao.getDay() > 31)
TEDataInvalidaException.post:
result.oclIsTypeOf(TEDataInvalidaException) and
(result.getData() == ito.dataSustacao@pre)

– *E29 - Cheque já está sustado*

TEChequeJaEstaSustadoException.condition:
(let message: OclMessage = ISBNegocioReq^existeChequeSustado(ITOChequeSustadoPK))
in
message.hasReturned() and
message.result().oclIsTypeOf(TEChequeJaEstaSustadoException)
TEChequeJaEstaSustadoException.post:
result.oclIsTypeOf(TEChequeJaEstaSustadoException) and
(result.getNumCheque() == ito.nroPriCheque@pre)

6.3.4 Componente de Sistema: operacaoConta

Refinamento das Interfaces dos Componentes

Não houve agrupamento nas interfaces do componente *'operacaoConta'*.

Diagrama de Atividades das Interfaces Providas

Esse diagrama pode ser visto nas Figuras 44 e 45.

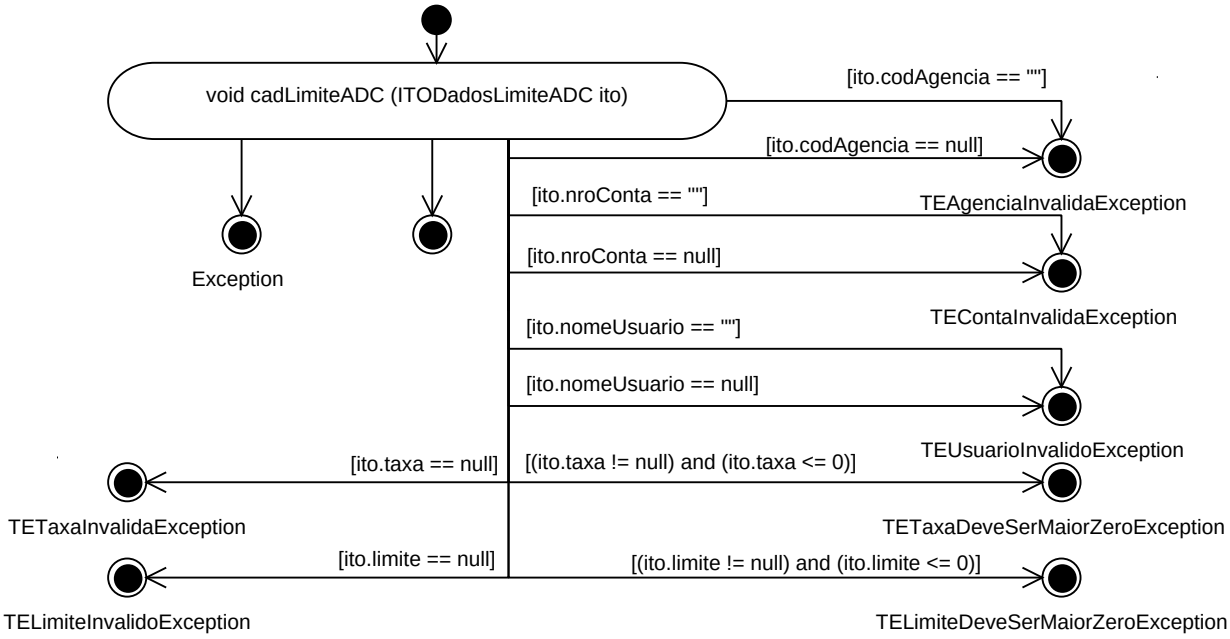

Figura 44: Diagrama da Interface Provida do Componente operacaoConta: Interface ISBCadLimiteADCMgr

Modelo de Informações das Interfaces (Interface Information Model)

As entidades do modelo de informações da Interface ISBCadLimiteADCMgr podem ser vistas na Figura 46. Os tipos de dados utilizados estão listados na Tabela 30.

As entidades do modelo de informações da Interface ISBCancelaContratoContaMgr podem ser vistas na Figura 47. Os tipos de dados utilizados estão listados na Tabela 31.

Formalização das Assertivas

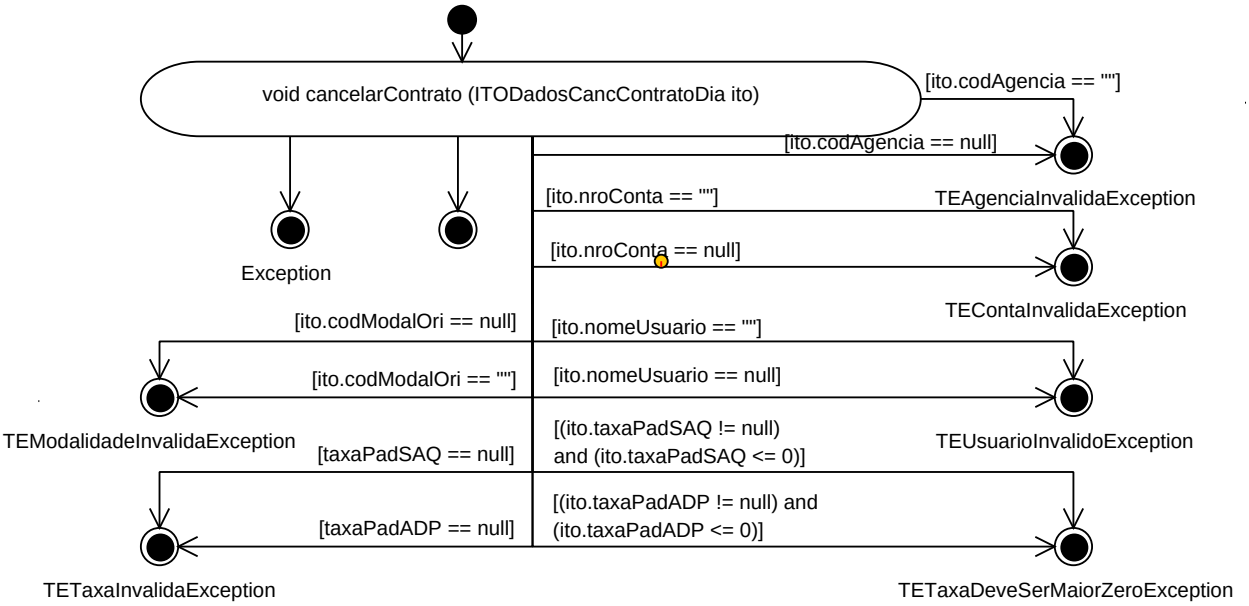

Figura 45: Diagrama da Interface Provida do Componente operacaoConta: Interface ISBCancelaContratoMgr

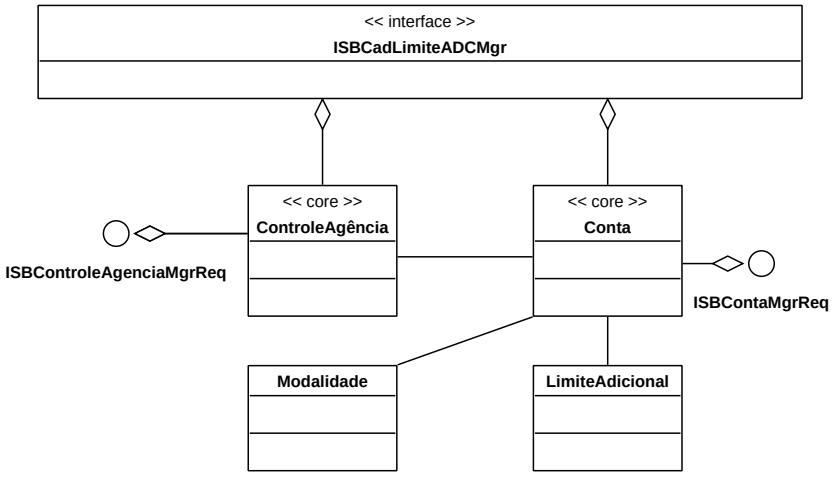

Figura 46: Interface Information Model da Interface: ISBCadLimiteADCMgr

As exceções cuja assertiva foi apresentada no componente anterior não serão reapresentadas por questão de espaço.

context operacaoConta

Tipos de Dados da Interface de Sistema: ISBCadLimiteADCMgr
ITOLimiteADC.

Tabela 30: Tipos de Dados utilizados pela Interface de Negócio: ISBCadLimiteADCMgr

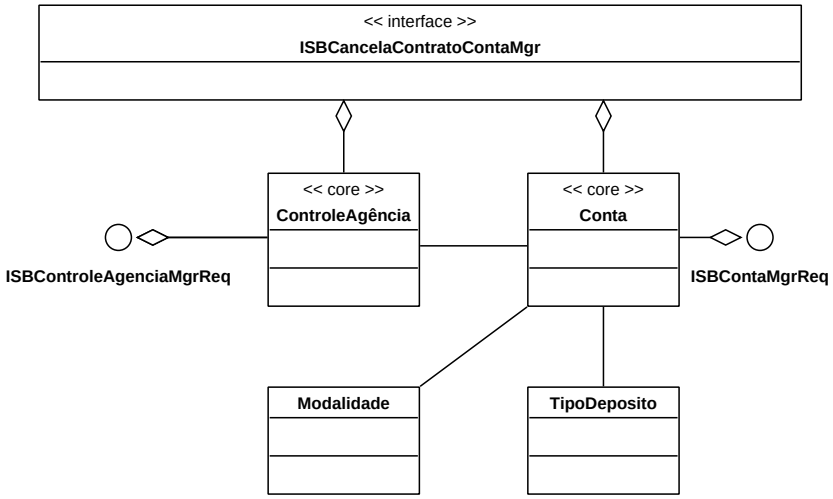

Figura 47: Interface Information Model da Interface: ISBCancelaContratoContaMgr

Tipos de Dados da Interface de Sistema: ISBCancelaContratoContaMgr
ITOCancContratoDia.

Tabela 31: Tipos de Dados utilizados pela Interface de Negócio: ISBCancelaContratoContaMgr

inv: true

context operacaoConta.ISBCancelaContratoMgr ::
void cancelarContrato (ITODadosCancContratoDia ito)
pre: true

– *Normal*

norm.condition:
(ito.codAgencia !=) and (ito.codAgencia != null) and
(ito.nroConta !=) and (ito.nroConta != null) and
(ito.nomeUsuario !=) and (ito.nomeUsuario != null) and
(ito.codModalOri !=) and (ito.codModalOri != null) and
(taxaADP != null) and (taxaADP > 0) and
(taxaSAQ != null) and (taxaSAQ > 0)

norm.post: true

– *E1 - Agência não está aberta*

– *E2 - Agência inválida*

– *E3 - Agência Não cadastrada*

– *E4 - Conta inválida*

– *E5 - Conta não cadastrada*

– *E6 - Usuário inválido*

– *E15 - Transação não foi concluída*

– *E18 - Coligada não cadastrada*

– *E19 - Modalidade não cadastrada*

– *E45 - Modalidade inválida*

TEModalidadeInvalidaException.condition:
(ito.codModalOri ==) or (ito.codModalOri == null)
TEModalidadeInvalidaException.post:
result.oclIsTypeOf(TEModalidadeInvalidaException) and
(result.getModalidade() == ito.codModalOri

– *E46 - Modalidade igual a modalidade da conta*

TEModalidadeIgualModalidadeContaException.condition:
(let message: OclMessage = ISBNegocioReq^recuperaConta(String)) in
message.hasReturned() and
message.result().oclIsTypeOf(ITOConta) and
message.result().codModal == ito.codModalOri
TEModalidadeIgualModalidadeContaException.post:
result.oclIsTypeOf(TEModalidadeInvalidaException)

– *E47 - Tipo de pessoa da modalidade não corresponde ao tipo de pessoa da conta*

TETipoPessoaNaoCorrespondeTipoPessoaContaException.condition:
(let messageM: OclMessage = ISBNegocioReq^recuperaModalidade(String) and
messageTD: OclMessage = ISBNegocioReq^recuperaTipoDeposito(String)) in
messageM.hasReturned() and messageTD.hasReturned() and
messageM.result().oclIsTypeOf(ITOModalidade) and
messageTD.result().oclIsTypeOf(ITOTipoDeposito) and
messageM.result().tipoPessoa == messageTD.result().tipoPessoa
TETipoPessoaNaoCorrespondeTipoPessoaContaException.post:
result.oclIsTypeOf(TETipoPessoaNaoCorrespondeTipoPessoaContaException)

– *E48 - Contrato já está cancelado*
TEContratoINEstaCanceladoException.condition:
(let message: OclMessage = ISBNegocioReq^checarContratoCancelado(ITOContaPK)) in
message.hasReturned() and
message.result().oclIsTypeOf(TEContratoINEstaCanceladoException)
TEContratoINEstaCanceladoException.post:
result.oclIsTypeOf(TEContratoINEstaCanceladoException)

– *E53 - Tipo de depósito não cadastrado*
TETipoDepositoNaoCadastradoException.condition:
(let message: OclMessage = ISBNegocioReq^recuperarTipoDeposito(int)) in
message.hasReturned() and
message.result().oclIsTypeOf(TETipoDepositoNaoCadastradoException)
TETipoDepositoNaoCadastradoException.post:
result.oclIsTypeOf(TETipoDepositoNaoCadastradoException) and
(let messageC: OclMessage = ISBNegocioReq^recuperarConta(String)) in
messageC.hasReturned() and
messageC.result().oclIsTypeOf(ITOConta) and
(result.getTipoDeposito() == messageC.result().tipoDepos)

– *E54 - Taxa inválida*
TETaxaInvalidaException.condition:
(let message: OclMessage = ISBNegocioReq^recuperarConta(String)) in
message.hasReturned() and
message.result().oclIsTypeOf(ITOConta) and
((ito.taxaPadADP == null) or (ito.taxaPadSAQ == null) or
((ito.taxaPadADP != null) and (ito.taxaPadADP.floatValue() > message.result().maxTaxaADP)) or
((ito.taxaPadSAQ != null) (ito.taxaPadSAQ.floatValue() > message.result().maxTaxaSAQ)))
TETaxaInvalidaException.post:
result.oclIsTypeOf(TETaxaInvalidaException) and
(((result.getIdTaxa() == "ADP") and (result.getValorTaxa() == ito.taxaADP)) or
(result.getIdTaxa() == "SAQ") and (result.getValorTaxa() == ito.taxaSAQ))
– *E56 - Reconfiguração Impossibilitada*
TEReconfiguracaoSistemaImpossibilitadaException.condition: Idem normal
TEReconfiguracaoSistemaImpossibilitadaException.post:

context operacaoConta.ISBCadLimiteADCMgr :: void cadLimiteADC (ITODadosLimiteADC ito)
pre: true

– *Normal*

norm.condition: (ito.codAgencia !=) and (ito.codAgencia != null) and
(ito.nroConta !=) and (ito.nroConta != null) and
(ito.limite != null) and (ito.limite > 0) and
(ito.taxa != null) and (ito.taxa > 0)

norm.post: true

- *E1 - Agência não está aberta*
- *E2 - Agência inválida*
- *E3 - Agência Não cadastrada*
- *E4 - Conta inválida*
- *E5 - Conta não cadastrada*
- *E6 - Usuário inválido*
- *E15 - Transação não foi concluída*
- *E19 - Modalidade não cadastrada*
- *E49 - O limite deve ser maior que zero*

TELimiteDeveSerMaiorZeroException.condition:
(ito.limite != null) and (ito.limite > 0)
TELimiteDeveSerMaiorZeroException.post:
result.oclIsTypeOf(TELimiteDeveSerMaiorZeroException) and
(result.getLimite() == ito.limite)

- *E50 - A taxa deve ser maior que zero*

TETaxaDeveSerMaiorZeroException.condition:
(ito.taxa != null) and (ito.taxa > 0)
TETaxaDeveSerMaiorZeroException.post:
result.oclIsTypeOf(TETaxaDeveSerMaiorZeroException) and
((ito.taxaPadADP == null) or (ito.taxaPadSAQ == null) or
((ito.taxaPadADP != null) and (ito.taxaPadADP.floatValue() > message.result().maxTaxaADP)) or
((ito.taxaPadSAQ != null) and (ito.taxaPadSAQ.floatValue() > <= message.result().maxTaxaSAQ))) and
(result.getTaxa() == ito.taxa)

- *E51 - A modalidade não permite uso de limite adicional*

TEModalidadeNaoPermiteUsoLimiteAdcException.condition:

(let message: OclMessage = ISBNegocioReq^recuperaModalidade(String)) in
message.hasReturned() and
message.result().oclIsTypeOf(ITOModalidade) and
(message.result().permiteLimiteAdc == false)
TEModalidadeNaoPermiteUsoLimiteAdcException.post:
result.oclIsTypeOf(TEModalidadeNaoPermiteUsoLimiteAdcException) and
(let messageM: OclMessage = ISBNegocioReq^recuperarModalidade(String)) in
messageM.hasReturned() and
messageM.result().oclIsTypeOf(ITOModalidade) and
(result.getTipoDeposito() == messageC.result().codModal)

– *E52 - A conta já possui um limite adicional*
TEContaJaPossuiLimiteAdcException.condition:
(let message: OclMessage = ISBNegocioReq^existeLimiteAdicional(ITOContaPK)) in
message.hasReturned() and
(message.result() == true)
TEContaJaPossuiLimiteAdcException.post:
result.oclIsTypeOf(TEContaJaPossuiLimiteAdcException)

– *E54 - Taxa inválida*
TETaxaInvalidaException.condition: (ito.taxa == null)
TETaxaInvalidaException.post:
result.oclIsTypeOf(TETaxaInvalidaException) and
(result.getValorTaxa() == null)

– *E55 - Limite inválido*
TELimiteInvalidoException.condition: ito.limite == null
TETaxaInvalidaException.post:
result.oclIsTypeOf(TETaxaInvalidaException) and
(result.getValorTaxa() == null)

6.3.5 Componente de Negócio: gerenciamentoConta

Refinamento das Interfaces dos Componentes

Não houve agrupamento nas interfaces do componente *'gerenciamentoConta'*.

Modelo de Informações das Interfaces (Interface Information Model)

As entidades do modelo de informações da Interface ISBContaMgrReq podem ser vistas na Figura 48. Os tipos de dados utilizados estão listados na Tabela 32.

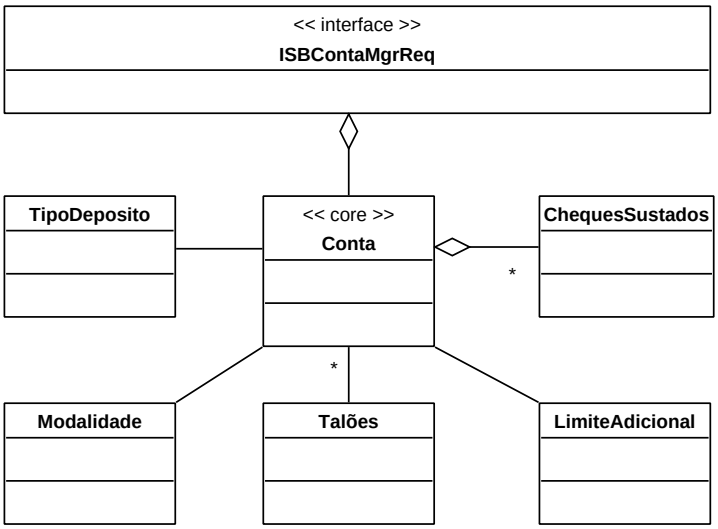

Figura 48: Interface Information Model da Interface: ISBContaMgrReq

Tipos de Dados da Interface de Negócio: ISBContaMgrReq
ITOConta, ITOContaPK, ITOModalidade, ITOTalao, ITOTalaoPK, ITOTipoDeposito, ITODadosCancContrato.

Tabela 32: Tipos de Dados utilizados pela Interface de Negócio: ISBContaMgrReq

6.3.6 Componente de Negócio: gerenciamentoColigada

Refinamento das Interfaces dos Componentes

Não houve agrupamento nas interfaces do componente *'gerenciamentoColigada'*.

Modelo de Informações das Interfaces (Interface Information Model)

As entidades do modelo de informações da Interface ISBColigadaMgrReq podem ser vistas na Figura 49. Os tipos de dados utilizados estão listados na Tabela 33.

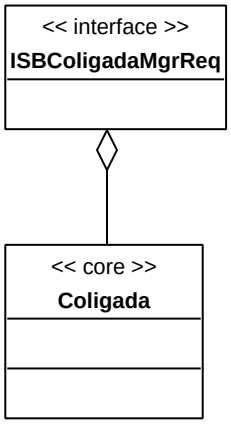

Figura 49: Interface Information Model da Interface: ISBColigadaMgrReq

Tipos de Dados da Interface de Negócio: ISBColigadaMgrReq
ITOColigada.

Tabela 33: Tipos de Dados utilizados pela Interface de Negócio: ISBColigadaMgrReq

6.3.7 Componente de Negócio: gerenciamentoMovimento

Refinamento das Interfaces dos Componentes

Não houve agrupamento nas interfaces do componente *'gerenciamentoMovimento'*.

Modelo de Informações das Interfaces (Interface Information Model)

As entidades do modelo de informações da Interface ISBMovimentoMgrReq podem ser vistas na Figura 50. Os tipos de dados utilizados estão listados na Tabela 34.

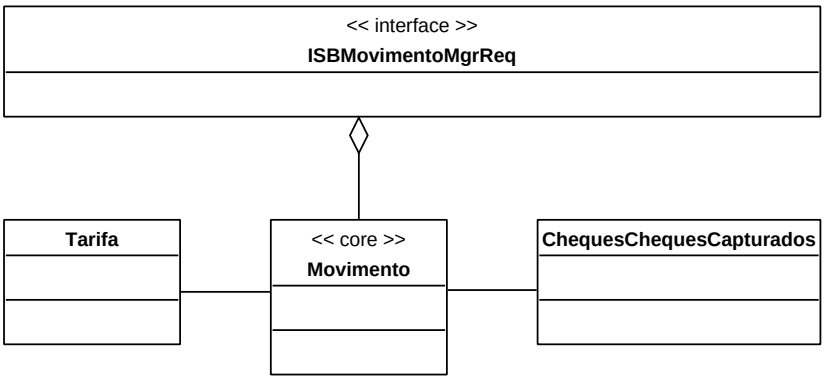

Figura 50: Interface Information Model da Interface: ISBMovimentoMgrReq

Tipos de Dados da Interface de Negócio: ISBMovimentoMgrReq
ITOTalao, ITODadosTarifaChequeSustado, ITODadosChequePesq, ITOCheque.

Tabela 34: Tipos de Dados utilizados pela Interface de Negócio: ISBMovimentoMgrReq

6.3.8 Componente de Negócio: gerenciamentoControleAgencia

Refinamento das Interfaces dos Componentes

Não houve agrupamento nas interfaces do componente *'gerenciamentoControleAgencia'*.

Modelo de Informações das Interfaces (Interface Information Model)

As entidades do modelo de informações da Interface ISBMovimentoMgrReq podem ser vistas na Figura 51. Os tipos de dados utilizados estão listados na Tabela 35.

Tipos de Dados da Interface de Negócio: ISBControleAgenciaMgrReq
ITOControleAgencia.

Tabela 35: Tipos de Dados utilizados pela Interface de Negócio: ISBControleAgenciaMgr-Req

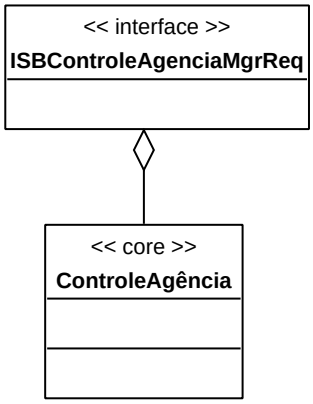

Figura 51: Interface Information Model da Interface: ISBControleAgenciaMgrReq

6.3.9 Componente Excepcional: tratadorAgencia

Refinamento das Interfaces dos Componentes

Houve agrupamento nas interfaces do componente *'tratadorAgencia'*. Por efetuar um tratamento simplificado (apenas propagação excepcional), todas as interfaces do componente foram agrupadas na interface *'ITratadorAgencia'*. A nova interface pode ser vista na Tabela 36; e o componente 'tratadorAgencia' pode ser visto na Figura 52.

Interface Excepcional: ITratadorAgencia
Operações:
void lancar(E1 e) throws E1
void lancar(E2 e) throws E2
void lancar(E3 e) throws E3

Tabela 36: Interface e Operações *ITratadorAgencia*

6.3.10 Componente Excepcional: tratadorConta

Refinamento das Interfaces dos Componentes

Houve agrupamento nas interfaces do componente *'tratadorConta'*. Por efetuar um tratamento simplificado (apenas propagação excepcional), todas as interfaces do componente foram agrupadas na interface *'ITratadorConta'*. A nova interface pode ser vista na Tabela 37; e o componente 'tratadorConta' pode ser visto na Figura 53.

6.3.11 Componente Excepcional: tratadorUtil

Refinamento das Interfaces dos Componentes

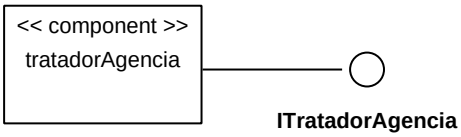

Figura 52: Componente Excepcional: *tratadorAgencia*

Interface Excepcional: ITratadorConta
Operações:
void lancar(E4 e) throws E4
void lancar(E5 e) throws E5
void lancar(E46 e) throws E46
void lancar(E47 e) throws E47
void lancar(E48 e) throws E48
void lancar(E49 e) throws E49
void lancar(E51 e) throws E51
void lancar(E52 e) throws E52
void lancar(E53 e) throws E53

Tabela 37: Interface e Operações *ITratadorConta*

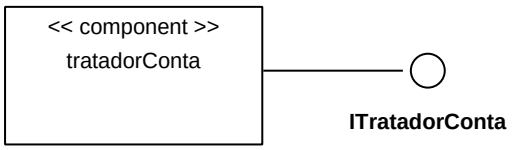

Figura 53: Componente Excepcional: *tratadorConta*

Houve agrupamento nas interfaces do componente *'tratadorUtil'*. Por efetuar um tratamento simplificado (apenas propagação excepcional), todas as interfaces do componente foram agrupadas na interface *'ITratadorUtil'*. A nova interface pode ser vista na Tabela 38; e o componente 'tratadorUtil' pode ser visto na Figura 54.

6.3.12 Componente Excepcional: tratadorCliente

Refinamento das Interfaces dos Componentes

Interface Excepcional: ITratadorUtil
Operações:
void lancar(E6 e) throws E6
void lancar(E28 e) throws E28
void lancar(E50 e) throws E50
void lancar(E54 e) throws E54

Tabela 38: Interface e Operações *ITratadorUtil*

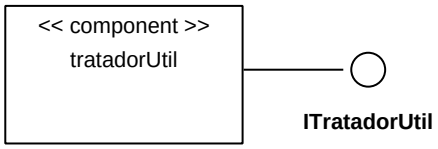

Figura 54: Componente Excepcional: *tratadorUtil*

Houve agrupamento nas interfaces do componente *'tratadorCliente'*. Por efetuar um tratamento simplificado (apenas propagação excepcional), todas as interfaces do componente foram agrupadas na interface *'ITratadorCliente'*. A nova interface pode ser vista na Tabela 39; e o componente 'tratadorCliente' pode ser visto na Figura 55.

Interface Excepcional: ITratadorCliente
Operações:
void lancar(E7 e) throws E7
void lancar(E8 e) throws E8
void lancar(E12 e) throws E12

Tabela 39: Interface e Operações *ITratadorCliente*

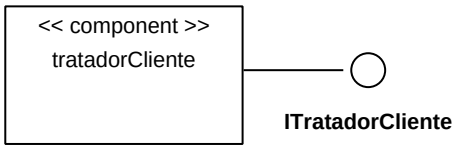

Figura 55: Componente Excepcional: *tratadorCliente*

6.3.13 Componente Excepcional: tratadorTalao

Refinamento das Interfaces dos Componentes

Houve agrupamento nas interfaces do componente *'tratadorTalao'*. Devido a maioria das interfaces efetuarem um tratamento simplificado (apenas propagação excepcional), essas interfaces foram agrupadas na interface *'ITratadorPropagacaoTalao'*. Essa interface pode ser vista na Tabela 40; e o componente 'tratadorTalao' pode ser visto na Figura 56.

Interface Excepcional: ITratadorPropagacaoTalao
Operações:
void lancar(E9 e) throws E9
void lancar(E10 e) throws E10
void lancar(E11 e) throws E11
void lancar(E13 e) throws E13
void lancar(E14 e) throws E14
void lancar(E20 e) throws E20
void lancar(E21 e) throws E21
void lancar(E22 e) throws E22
void lancar(E23 e) throws E23
void lancar(E24 e) throws E24

Tabela 40: Interface e Operações *ITratadorPropagacaoTalao*

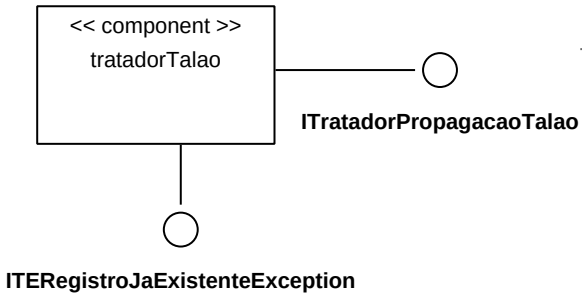

Figura 56: Componente Excepcional: *tratadorTalao*

6.3.14 Componente Excepcional: tratadorTransacao

Refinamento das Interfaces dos Componentes

Não houve agrupamento nas interfaces do componente *'tratadorTransacao'*.

6.3.15 Componente Excepcional: tratadorColigada

Refinamento das Interfaces dos Componentes

Como o componente *'tratadorColigada'* só tem uma interface, não houve necessidade de agrupamento.

6.3.16 Componente Excepcional: tratadorModalidade

Refinamento das Interfaces dos Componentes

Houve agrupamento nas interfaces do componente *'tratadorModalidade'*. Por efetuar um tratamento simplificado (apenas propagação excepcional), todas as interfaces do componente foram agrupadas na interface *'ITratadorModalidade'*. A nova interface pode ser vista na Tabela 41; e o componente 'tratadorModalidade' pode ser visto na Figura 57.

Interface Excepcional: ITratadorModalidade
Operações:
void lancar(E19 e) throws E19
void lancar(E45 e) throws E45

Tabela 41: Interface e Operações *ITratadorModalidade*

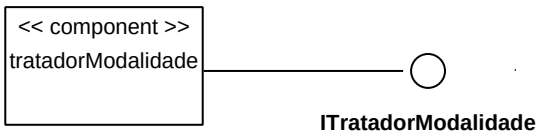

Figura 57: Componente Excepcional: *tratadorModalidade*

6.3.17 Componente Excepcional: tratadorChequeSustado

Refinamento das Interfaces dos Componentes

Houve agrupamento nas interfaces do componente *'tratadorChequeSustado'*. Por efetuar um tratamento simplificado (apenas propagação excepcional), todas as interfaces do componente foram agrupadas na interface *'ITratadorChequeSustado'*. A nova interface pode ser vista na Tabela 42; e o componente 'tratadorChequeSustado' pode ser visto na Figura 58.

Interface Excepcional: ITratadorChequeSustado
Operações:
void lancar(E25 e) throws E25
void lancar(E26 e) throws E26
void lancar(E27 e) throws E27
void lancar(E29 e) throws E29

Tabela 42: Interface e Operações *ITratadorChequeSustado*

6.3.18 Componente Excepcional: tratadorChequesCapturados

Refinamento das Interfaces dos Componentes

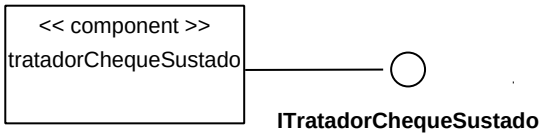

Figura 58: Componente Excepcional: *tratadorChequeSustado*

Houve agrupamento nas interfaces do componente *'tratadorChequesCapturados'*. Por efetuar um tratamento simplificado (apenas propagação excepcional), todas as interfaces do componente foram agrupadas na interface *'ITratadorChequesCapturados'*. A nova interface pode ser vista na Tabela 43; e o componente 'tratadorChequesCapturados' pode ser visto na Figura 59.

Interface Excepcional: ITratadorChequesCapturados
Operações:
void lancar(E30 e) throws E30
void lancar(E31 e) throws E31
void lancar(E32 e) throws E32
void lancar(E33 e) throws E33
void lancar(E34 e) throws E34
void lancar(E35 e) throws E35
void lancar(E36 e) throws E36
void lancar(E37 e) throws E37
void lancar(E38 e) throws E38
void lancar(E39 e) throws E39
void lancar(E40 e) throws E40
void lancar(E41 e) throws E41
void lancar(E42 e) throws E42
void lancar(E43 e) throws E43
void lancar(E44 e) throws E44

Tabela 43: Interface e Operações *ITratadorChequesCapturados*

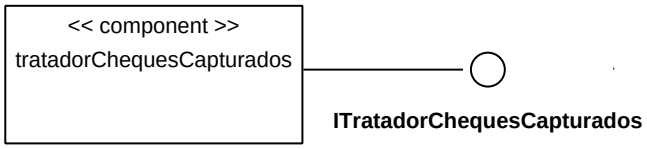

Figura 59: Componente Excepcional: *tratadorChequesCapturados*

7 Provisionamento dos Componentes

A fase de provisionamento dos componentes deve garantir a materialização dos componentes especificados. Isso pode ocorrer de três formas: (1) Reutilização de componentes já existentes; (2) Aquisição de componentes de prateleira; (3) Implementação de componentes novos. Em todos os casos, o componente deve ser testado no novo ambiente de utilização, especialmente no caso de sistemas confiáveis.

Os componentes são implementados de forma cíclica: cada componente é especificado, implementado e testado separadamente. É interessante o estabelecimento de uma ordem de implementação dos componentes para que a integração do sistema comece o mais cedo possível.

No caso do sistema especificado, ocorreram as situações (1) e (3): os componentes que implementam as funcionalidades especificadas nos casos de uso (componentes de sistema) foram implementados, e todas as operações de infra-estrutura requeridas pelo sistema (componentes de negócio) foram reutilizadas de componentes existentes.

Para cada uma das situações há diferentes atividades a serem realizadas. As atividade serão rapidamente descritas nesta introdução, e detalhadas durante a apresentação dos exemplos, para facilitar o entendimento. Nesta fase serão descritas apenas as atividades relativas ao componente operacoesConta, já que são similares para os outros componentes.

Para o caso de reutilização de componentes, as atividades são:

1. Mapeamento entre as suas operações e as operações requeridas especificadas
2. Implementação de *wrappers*
3. Avaliação da qualidade do componente
4. Implementação da arquitetura do componente testável

Para os componentes a serem implementados, as atividades a serem realizadas para cada componente são listadas a seguir. É importante observar que nem todas as atividades de teste serão realizadas para todos os componentes, apenas para os selecionados na fase Identificação dos Componentes.

1. Especificação da Estrutura Interna do Componente: *Atividade de Desenvolvimento*

   Entradas: Interfaces do componente, casos de uso

   No caso dos componentes implementados, é necessário especificar a estrutura interna dos mesmos de acordo com alguma tecnologia de desenvolvimento disponível. O modelo de materialização utilizado para os componentes foi o modelo COSMOS [JGR03], que explicita tanto a especificação interna dos componentes, quanto a especificação das interações entre eles. As principais características do COSMOS são: (1) Materialização dos elementos arquiteturais; (2) Separação entre os requisitos funcionais e não-funcionais; (3) Separação explícita entre a especificação e a implementação; (4) Declaração explícita das dependências dos componentes; (5) Separação entre a herança de código e a herança de tipos; (5) Baixo acoplamento entre as classes de

implementação. Todas essas características proporcionam uma maior facilidade para atividades de evolução do sistema.

Saídas: Diagrama de classes da estrutura interna do componente

2. Geração dos componentes testáveis (componentes Tracker e Tester): *Atividade de Testes*

   Os componentes Tester e Tracker 2 possuem estruturas internas semelhantes [RM04]. A estrutura interna do componente Tester é ilustrada na Figura 60, e é a mesma para quaisquer componentes sob teste. A interface ITesting e as classes Testing e Setup são utilizadas para configuração da instrumentação do componente. ILogAssert e LogAssert são responsáveis pelo registro das violações em um arquivo de *log*. Apenas as bibliotecas e especificações são customizadas de acordo com o componente sob teste.

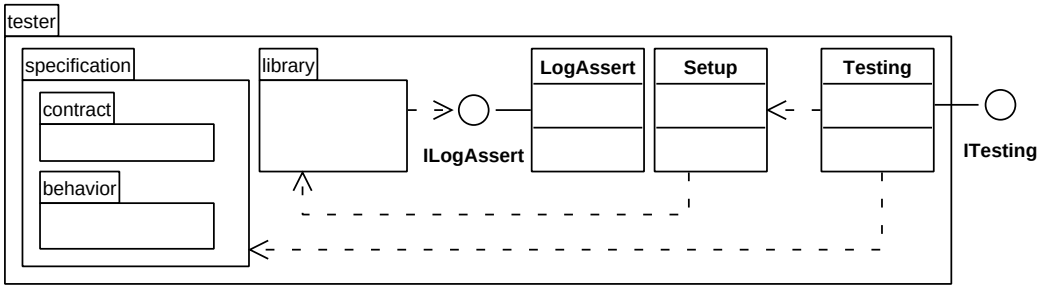

Figura 60: Arquitetura do componente Tester.

O pacote *specification* (presente apenas no componente Tester) contém as especificações contratual e comportamental do componente, respectivamente nos pacotes *contract* e *behavior*. O pacote *library* contém os aspectos para verificação de assertivas, no caso do componentes Tester, e para os mecanismos de monitoração, para o componente Tracker.

As especificações foram geradas em fases anteriores. Nesta fase são geradas as bibliotecas dos componentes Tracker e Tester, como descrevem as subatividades a seguir.

Geração das bibliotecas do componente Tracker

Entradas: interfaces do componente.

Para a geração da biblioteca de monitoração, são necessárias apenas as assinaturas das operações, obtidas a partir das interfaces dos componentes. As assinaturas servem de base para a construção dos aspectos que interceptam a execução dos métodos e possibilitam a monitoração dos métodos.

Saídas: biblioteca do componente Tracker.

Geração das bibliotecas do componente Tester

Entradas: especificação contratual.

As bibliotecas do componente Tester são geradas a partir das especificações contratuais do componente, contidas no pacote *specification*. Para cada interface, são criados aspectos para verificação da invariante e das pré e pós condições dos métodos, implementando as regras contidas na especificação contratual.

Os algoritmos para criação dos aspectos são apresentados no Apêndice **??**.

Saídas: biblioteca do componente Tester.

3. Geração dos casos de teste (XML): *Atividade de Testes*

   Entradas: especificação comportamental.

   Os casos de teste são primeiramente gerados em XML, independente de linguagem de programação, generalizando a ferramenta. Os casos de teste são gerados a partir dos diagramas de atividades: os diagramas são percorridos por um algoritmo de busca em profundidade, e cada caminho obtido é convertido em um caso de teste.

   Todos os diagramas produzidos para um componente (nas fases de especificação final dos componente e interação dos componentes) são integrados para a geração dos testes. Os diagramas das interações passam a ser subatividades dos métodos representados no diagrama da especificação final: este diagrama ilustra o comportamento entre os métodos da interface provida, e os de interação representam o comportamento interno de cada método da interface provida.

   A busca em profundidade começa no início dos diagramas da especificação final e chega a cada nó final, cobrindo todas as arestas, tanto dos diagramas principais quanto das subatividades. Os diagramas produzidos na interação dos componentes são percorridos de forma recursiva: ao se deparar com uma atividade representando um método que se comunica com as interfaces requeridas, o algoritmo considera o diagrama do método como uma subatividade, e o inclui no percurso.

   Os caminhos obtidos no diagrama da interação dos componentes representam os valores a serem retornados pelos stubs para aquela chamada específica, enquanto os nós finais representam os valores a serem retornados pelo método da interface provida. Vale ressaltar que os casos de teste gerados ainda não possuem todos os dados de teste, pois geralmente apenas o tipo do objeto é coletado pelo percurso no diagrama.

   Saídas: casos de teste em XML incompletos.

4. Geração dos dados de teste (XML): *Atividade de Testes*

   Entradas: casos de teste em XML incompletos, casos de uso.

   Como o algoritmo não é capaz de gerar todos os dados necessários para os testes, nesta fase é realizada a verificação dos dados gerados e complemento dos faltantes. É importante a consulta aos casos de uso para geração de dados significativos para o negócio.

   Saídas: casos de teste em XML completos.

5. Geração dos *stubs*: *Atividade de Testes*

Entradas: interfaces requeridas.

As classes que servem como stubs implementam as interfaces requeridas do componente sob teste, substituindo os componentes requeridos. Para a geração automática é necessária apenas a assinatura dos métodos a serem implementados. A estrutura de uma classe stub é uniforme, contendo vetores para armazenamento dos valores a serem retornados pelos stubs, métodos para atribuição desses valores e a limpeza dos vetores.

Saídas: código fonte das classes dos stubs.

6. Geração dos casos de teste (linguagem de programação): *Atividade de Testes*

   Entradas: casos de teste em XML, componente sob teste, classes de stubs.

   Nesta fase são gerados os casos de teste executáveis, na linguagem do sistema que está sendo testado. Neste estudo de caso foi utilizada a linguagem Java para implementação dos componentes com o framework JUnit, seguindo algumas convenções, já que o algoritmo de tradução do XML para JUnit ainda não foi criado.

   Saídas: código fonte dos casos de teste.

7. Aplicação dos Testes: *Atividade de Testes*

   Entradas: código fonte dos casos de teste, componente sob teste, classes de stubs.

   Durante o provisionamento, deve ser testados os componentes isolados, que já foram implementados. Os casos de teste são implementados na linguagem do sistema, os testes são executados e os seguintes documentos são preenchidos:

   - Diário de teste, com os registros cronológicos da execução;
   - Relatório resumo dos testes, com a descrição resumida das atividades de teste e uma avaliação dos resultados;
   - Relatório de incidência de testes, registrando eventos ocorridos durante os testes que mereçam análise posterior;
   - Relatório de encaminhamento do item de teste, encaminhando as correções para as equipes responsáveis.

   Saídas: diário de testes, relatório resumo de testes, relatório de incidência dos testes, relatório de encaminhamento do item de teste.

O restante da seção está dividida entre os componentes reutilizados e os implementados. São descritas as atividades realizadas para a reutilização dos componentes, e os passos na implementação e geração dos casos de teste do componente operacoesConta.

7.1 Componentes Reutilizados

O mapeamento entre as operações requeridas especificadas e as operações reutilizadas pode ser visto nas Tabelas 44, 45, 46 e 47.

gerenciamentoConta.ISBContaMgrReq			
Componente Especificado		Componente Reutilizado	
Operação:	Exceção:	Operação:	Exceção:
ITOConta recuperarConta(ITOContaPK ito)	E5	*eb.IEBContaMgr*.DT25 getContaGeneric(DT1 ito)	E5
void checarTitular(ITOContaPK ito, String codCliente)	E8	*eb.IEBContaMgr*.boolean isTitularConta(DT1 ito, String codCliente)	–
ITOModalidade recuperarModalidade(String codModal)	E19	*eb.IEBTabelasBasicasMgr*.DT27 getModalidade(String codModal)	E19
void incluirTalao(ITOTalao ito)	E57	*eb.IEBOperacaoMgr*.void cadReqTalaoIns(DT5 ito)	E57
void checarTalaoEstoque(ITOTalaoPK ito, ?quant?)	E22	*eb.ISBOperacaoMgr*.boolean verificarTalaoEstoque(DT4 ito, int quant)	–
void entregarTalao(ITOTalao ito)	E15 E23	*eb.ISBOperacaoMgr*.void cadEntTalaoIns(Dt5 ito)	E15 E23
void desbloqueiaTalao(ITOTalaoPK ito)	E15	*eb.IEBContaMgr*.void desbloquearTalao(DT4 ito)	E15
int verificarQtdTalaoParaRequisitar(ITOContaPK ito)	–	*eb.ISBContaMgr*.int verificarQtdTalaoContaRequisitar(DT1 ito)	–
boolean existeChequeSustado(ITOTalao ito)	–	*eb.ISBOperacaoMgr*.boolean existeChequeSustado(DT13 ito)	–
void inserirChequeSustado (ITOChequeSustado ito)	E15	*eb.ISBOperacaoMgr*.void cadChqSustadoIns(DT13 ito)	E15
ITOTipoDeposito recuperarTipoDeposito(String tipoDepos)	E53	*eb.ISBOperacaoMgr*.DT9 getTipoDeposito(String tipDepos)	E53
void checarContratoCancelado(ITOContaPK ito)	E48	*eb.ISBContaMgr*.boolean checarSeContratoJaCancelado(ITOContaPK ito)	–
void cancelarContrato(ITODadosCancContrato ito)	E15	*eb.ISBContaMgr*.void cancelarContrato(D28 ito)	E15
void atualizarLimiteAdicionalConta(ITOConta ito)	E15	*eb.ISBContaMgr*.void atuLimiteAdc(DT1 ito, BigDecimal limite)	E15
void incluirLimiteAdicional (ITOLimiteADC ito)	E15	*eb.ISBOperacaoMgr*.void cadLimAdcIns(DT12 ito)	E15
boolean existeLimiteAdicional (ITOContaPK ito)	–	*eb.ISBOperacaoMgr*.boolean existeLimAdc(DT1 ito)	–

Tabela 44: Reutilizações para Compor o Componente de Negócio: *'gerenciamentoConta'*

7.1.1 Avaliação da Qualidade dos Componentes

Por não terem sido selecionados para serem testados (fase Identificação dos Componentes), não foram realizadas as etapas "Avaliação da Qualidade dos Componentes"e "Realização dos Testes"nos componentes reutilizados.

7.2 Componentes Implementados

Como a estrutura de todos os componentes implementados é similar, apenas a implementação do componente operacoesConta será detalhada neste relatório. O código fonte dos outros componentes, incluindo seus mecanismos de monitoração, assertivas, stubs e casos de teste, podem ser consultados no CD em anexo.

Especificação da Estrutura Interna do Componente
A Figura 61 mostra o componente operaçõesConta estruturado segundo o modelo COSMOS.

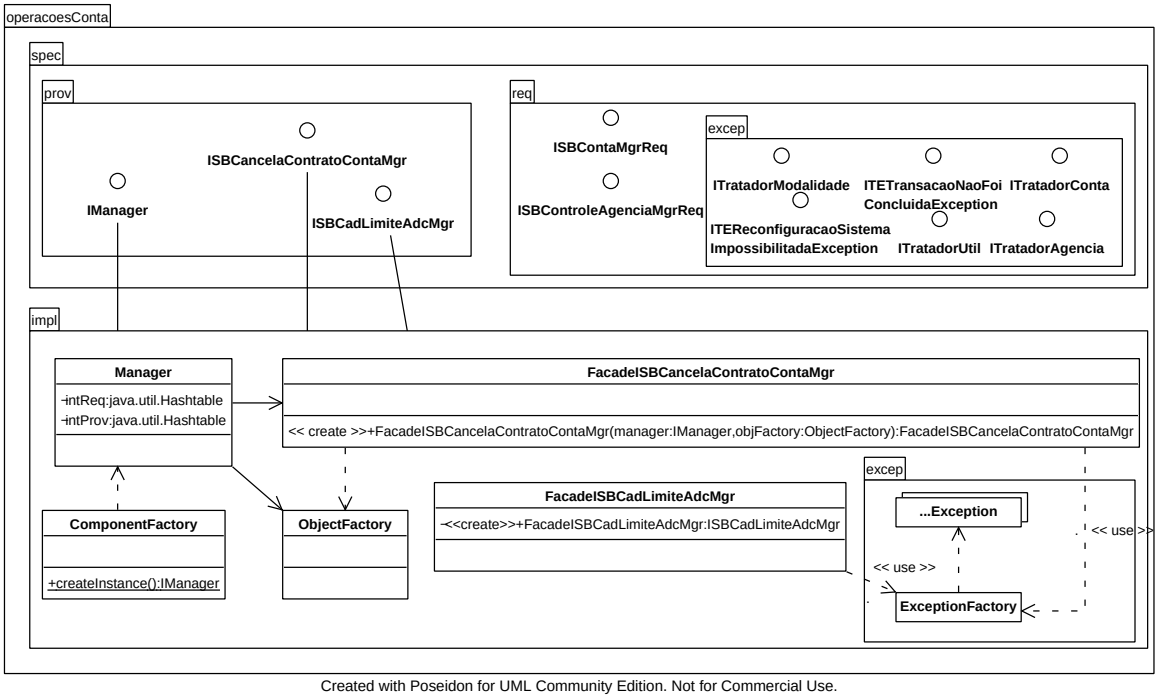

Figura 61: Estruturação Interna do Componente operacoesConta

Geração das bibliotecas

Para a inclusão de mecanismos de monitoração e verificação de assertivas no componente sob teste, foi utilizada a programação orientada a aspectos, que permite a instrumentação do componente mesmo que seu código fonte não esteja disponível.

Como a programação orientada a aspectos ainda não é um paradigma largamente conhecido, nesta seção será apresentada uma breve introdução, seguida dos passos para geração das bibliotecas dos componentes Tracker e Tester.

Programação Orientada a Aspectos - Conceitos

A programação orientada a aspectos é uma nova metodologia que possibilita a separação de interesses transversais[1] (*crosscuting concerns*) pela introdução de uma nova unidade de modularização - o aspecto. Esses interesses transversais são implementados como aspectos, e a composição do sistema é feita pelo combinador (*weaver*), que combina o componente de negócio e os aspectos contendo os interesses transversais.

Assertivas e mecanismos de monitoração podem ser considerados interesses transversais, já que estão presentes em todos os módulos mas não fazem parte diretamente da funcionalidade do sistema. A implementação desses conceitos como aspectos, além de contribuir com a manutenibilidade, permite que eles sejam incluídos e retirados do sistema automaticamente, mesmo sem a presença do código fonte (em tempo de carga). A ferramenta utilizada foi AspectJ, uma extensão orientada a aspectos para a linguagem Java. É a ferramenta mais completa para orientação a aspectos em Java, sendo desenvolvida como código aberto desde 1997 e integrada ao projeto Eclipse [Eclipse].

A extensão para suporte à orientação a aspectos introduziu alguns novos conceitos à linguagem Java:

- Ponto de junção (*join point*): é um conceito abstrato, e representa um determinado ponto no fluxo de execução, como, por exemplo, a chamada de um método.
- Conjunto de junção (*pointcut*): representa determinados pontos de junção do fluxo de execução, para os quais podem ser especificadas regras de combinação como, por exemplo, a chamada do método `setCoalFeedRate()`.
- Adendo (*advice*): contém o código a ser executado quando um determinado ponto de junção (capturado por um conjunto de junção) é atingido. Existem três tipos: adendo anterior (*before*), que define ações a serem realizadas antes do ponto de junção; adendo posterior (*after*), que define ações a serem realizadas após o ponto de junção; e adendo de contorno (*around*), que define ações que englobam a execução do ponto de junção. Um exemplo de adendo anterior seria a verificação de uma pré-condição a ser realizada antes da execução do método `setCoalFeedRate()`.
- Declarações intertipos (*inter-type declarations*): possibilitam a modificação da estrutura do programa estaticamente, como a criação de novos atributos e métodos, por exemplo.
- Aspectos (*aspects*): estruturas similares à classes Java que encapsulam as regras de combinação construídas com a utilização dos conceitos anteriores.

[1] a tradução para português segue a terminologia definida durante o Primeiro Workshop Brasileiro de Desenvolvimento de *Software* Orientado a Aspectos (WASP04), e está disponível em http://twiki.im.ufba.br/bin/view/AOSDbr/TermosEmPortugues

Neste trabalho o conceito de declaração intertipos não foi utilizado, pois não foi realizada modificação estática. Amostras do código produzido contendo conjuntos de junção, adendos anterior e posterior, e aspectos são exibidos nas Figuras 62, 63 e 65.

Geração das bibliotecas do componente Tracker

As bibliotecas são geradas automaticamente a partir das interfaces requeridas e providas do componente. É criado um aspecto para cada uma destas interfaces. No caso do componente operacoesConta, foram criados os aspectos `ISBCadLimiteAdcMgrPO`, `ISBCancelaContratoContaMgrRO` e `ISBNegocioMgrReqRO`.

Os códigos dos aspectos ISBCadLimiteAdcMgrPO (completo) e ISBNegocioMgrReqRO (parcial) são exibidos nas Figuras 62 e 63. Ambos são construídos de forma similar: cada método da interface é interceptado antes e depois de sua execução, seja seu retorno normal ou excepcional, e os dados são registrados no *log* pelo método de LogTrace (similar à LogAssert, Figura 60).

A interceptação é realizada pelo conjunto de junção (Figura 62, linhas 6 a 8), que intercepta o método e recolhe o valor de seus parâmetros de entrada. O adendo anterior (linhas 11 a 14) registra o início da execução do método. O adendo posterior normal (linhas 17 a 20) registra o retorno do método, caso este termine normalmente, enquanto o adendo posterior excepcional (linhas 23 a 26) registram a exceção lançada pelo método, caso este termine de forma excepcional.

Para cada interface, os passos para a criação do aspecto são:

1. Para cada método da interface, criar:

   (a) Um conjunto de junção para interceptar o método. Caso seja da interface provida (Figura 62, linhas 5 a 8), o conjunto de junção deve interceptar a execução do método (comando `execution` + assinatura); caso seja da interface requerida (Figura 63, linhas 5 a 9), deve interceptar a chamada do método (comando `call` + assinatura) durante a execução dos métodos públicos do componente (comando `cflow` + public * *.*(..)). O conjunto de junção deve ainda coletar os argumentos do método (comando `args`).

   (b) Um adendo anterior relativo ao conjunto de junção, registrando a chamada do método e o valor dos parâmetros com `LogTrace.writeOperationalTraceEntry()` (Figura 62, linhas 10 a 14, e Figura 63, linhas 11 a 15).

   (c) Um adendo posterior para captura de retorno normal (*returning()*), registrando o término e o valor de retorno com `LogTrace.writeOperationalTraceReturn()` (Figura 62, linhas 16 a 20, e Figura 63, linhas 17 a 21).

   (d) Um adendo posterior para captura de retorno excepcional (*throwing()*), registrando o término e a exceção com `LogTrace.writeOperationalTraceReturn()` (Figura 62, linhas 22 a 26, e Figura 63, linhas 23 a 27).

```
1  public privileged aspect ISBCadLimiteAdcMgrPO {
2
3   private LogTrace log = LogTrace.singleton();
4
5   //Conjunto de junção para monitoramento do metodo cadastrarLimiteAdc
6   private pointcut cadastrarLimiteAdcMethod (ITOLimiteAdc in):
7     execution(public void ISBCadLimiteAdcMgr.cadastrarLimiteAdc (ITOLimiteAdc))
8     && args (in); //in: argumento do método cadastrarLimiteAdc
9
10  //Adendo anterior para monitoramento do metodo cadastrarLimiteAdc
11  before (ITOLimiteAdc in) : cadastrarLimiteAdcMethod(in) {
12    log.writeOperationalTraceEntry("operacaoConta.ISBCadLimiteAdcMgr",
13                                   "cadastrarLimiteAdc (ITOLimiteAdc)" , in.toString(),"");
14  }
15
16  //Adendo posterior para monitoramento do metodo cadastrarLimiteAdc: retorno normal
17  after (ITOLimiteAdc in) returning(): cadastrarLimiteAdcMethod(in) {
18    log.writeOperationalTraceReturn("operacaoConta.ISBCadLimiteAdcMgr",
19                                   "cadastrarLimiteAdc (ITOLimiteAdc)", "");
20  }
21
22  //Adendo posterior para monitoramento do metodo cadastrarLimiteAdc: retorno excepcional
23  after (ITOLimiteAdc in) throwing (Exception e): cadastrarLimiteAdcMethod(in) {
24    log.writeOperationalTraceReturn("operacaoConta.ISBCadLimiteAdcMgr",
25                                    "cadastrarLimiteAdc (ITOLimiteAdc)",     e.getClass().getName());
26  }
27 }
```

Figura 62: Código do aspecto de monitoração ISBCadLimiteAdcMgrPO.

Geração das bibliotecas do componente Tester

A biblioteca do componente Tester é estruturada de acordo com as interfaces do componente. Para cada interface é criado um pacote, contendo seus aspectos: um para verificação das invariantes, e dois para cada método (um para as pré-condições e outro para as pós).

Para as invariantes, todos os métodos da interface devem ser interceptados, sendo a invariante verificada antes e depois de cada um. Para dos aspectos referentes às invariantes, os passos a serem seguidos são (caso o valor da invariante seja diferente de *true*):

1. Criar um conjunto de junção que intercepte a execução de todos os métodos públicos da interface (comando `execution` + public * interface.*(..)).

2. Criar um adendo anterior e um posterior relativo ao conjunto de junção para a verificação da invariante. Caso a invariante seja violada, registrar a violação com `LogAssert.writeInvViolation()` (a mensagem detalhada tem que ser definida manualmente).

Para as pré e pós-condições, os métodos devem ser individualmente interceptados, já

```
1  public privileged aspect ISBNegocioMgrReqRO {
2
3   private LogTrace log = LogTrace.singleton();
4
5   //Conjunto de junção para monitoramento do metodo recuperaAgencia
6   private pointcut recuperaAgenciaMethod(String in):
7     call(public ITOControleAgencia ISBNegocioReq.recuperaAgencia (String))
8     && args(in)
9     && cflow(execution(public * *.*(..)));
10
11  //Adendo anterior para monitoramento do metodo recuperaAgencia
12  before (String in) : recuperaAgenciaMethod(in) {
13    log.writeOperationalTraceEntry("ISBControleAgenciaMgrReq",
14                                   "recuperaAgencia", "codAgencia = " + in, "");
15  }
16
17  //Adendo posterior para monitoramento do metodo recuperaAgencia: retorno normal
18  after (String in) returning(ITOControleAgencia out): recuperaAgenciaMethod(in) {
19    log.writeOperationalTraceReturn("ISBControleAgenciaMgrReq",
20                                    "recuperaAgencia", out.toString());
21  }
22
23  //Adendo posterior para monitoramento do metodo recuperaAgencia: retorno excepcional
24  after (String in) throwing (Exception e): recuperaAgenciaMethod(in) {
25    log.writeOperationalTraceReturn("ISBControleAgenciaMgrReq",
26                                    "recuperaAgencia", e.getClass().getName());
27  } ...
28 }
```

Figura 63: Código do aspecto de monitoração ISBNegocioMgrReqRO.

que cada um possui diferentes contratos. Para as pré-condições, os passos para criação do aspecto referente a um método são (caso o valor da pré-condição seja diferente de *true*):

1. Criar um conjunto de junção que intercepte a execução do método ao qual a pré-condição se refere (comando `execution` + assinatura).

2. Criar um adendo anterior relativo ao conjunto de junção para a verificação da pré-condição. Caso a pré-condição seja violada, registrar a violação com `LogAssert.writePreViolation()` (a mensagem detalhada tem que ser definida manualmente)

Para o componente operacoesConta foram criados o pacotes ISBCancelaContratoContaMgr e ISBCadLimiteAdcMgr. Como as invariantes e pré-condições valem *true* (Seção 6.3), foram criados os aspectos ISBCancelaContratoContaMgr.CancelarContratoContaPost e ISBCadLimiteAdcMgr.CadLimiteAdcPost, para verificação das pós-condições. O código do aspecto CadLimiteAdcPost é apresentado nas Figuras 65 e 66 (Parte 1 e 2), que contém apenas a verificação do retorno normal e das exceções E4 (Conta Inválida) e E5 (Conta não cadastrada - Figura **??**) devido à extensão do código. Os passos para a criação dos aspectos referentes às pós-condições são:

– E5 - Conta não cadastrada
TEContaNaoCadastradaException.condition:
(let message: OclMessage = ISBNegocioReq^recuperaConta(String))) in
message.hasReturned() and
message.result().oclIsTypeOf(TEContaNaoCadastradaException)
TEContaNaoCadastradaException.post:
result.oclIsTypeOf(TEContaNaoCadastradaException) and
(result.getNumConta() == ito.nroConta@pre)

Figura 64: Parte da pós-condição do método CancelarContratoConta.

Como o algoritmo ainda não foi implementado, os aspectos foram criados manualmente, seguindo os passos do algoritmo. Este, porém, não cobre a tradução dos aspectos para OCL, então foram adotadas algumas diretrizes para implementação.

Pós-condição normal:

- (linhas X aY): Transformar o conteúdo de norm.condition em uma expressão, substituindo os `and` por &&, e verificar se a negativa da expressão é válida (`if (!expression)`). Em caso afirmativo, registrar que uma exceção deveria ter sido lançada.

- Se `norm.post != true`, transformar o conteúdo de norm.condition em uma expressão, substituindo os `and` por &&, e verificar se a expressão é válida. Em caso afirmativo, verificar se norm.post é válido. Caso negativo, registrar que a pós-condição normal foi violada.

Pós-condição excepcional:

- Se o método se relaciona com outros componentes, criar uma variável do tipo Exception (linha U), um conjunto de junção para interceptar a execução de quaisquer métodos requeridos (linhas X a I) e um adendo posterior excepcional para captura da exceção lançada (linhas L a X).

- Se todas as exceções forem propagadas, criar uma verificação de propagação genérica no começo do adendo excepcional (linhas X a T).

- Se exceção.condition faz referência a chamada de um método, criar uma variável, um conjunto de junção e um adendo posterior (normal ou excepcional, dependendo do que será verificado) específicos para a captura do método chamado (linhas L a X). No caso, foi um adendo posterior excepcional, já que o retorno verificado era de um tipo descendente de `Exception`.

- Caso a exceção seja relativa a um erro de contexto da exceção, exibir na mensagem o contexto propagado e o certo.

```
1  public privileged aspect CadLimiteAdcPost {
2
3   private LogAssert log = new LogAssert();
4   private Exception exception = null;
5   private ITOConta itoconta = null;
6
7   //acesso ao retorno da operacao requerida recuperaConta
8   private pointcut interface_requerida_recuperaConta():
9     (call(public ITOConta ISBNegocioReq.recuperaConta (ITOContaPK)))
10    && cflow(assertion(ITOLimiteAdc));
11
12  //coleta retorno da operacao requerida recuperaConta
13  after () returning (ITOConta ito): interface_requerida_recuperaConta() {
14    itoconta = ito;
15  }
16  //acesso aos retornos das interfaces requeridas
17  private pointcut interface_requerida():
18    (call(public *  ISBNegocioReq.* (..)))
19    && cflow(assertion(ITOLimiteAdc));
20
21  //coleta retornos das interfaces requeridas
22  after () throwing (Exception e): interface_requerida() {
23    exception = e;
24  }
25  //pointcut pra assertivas
26  private pointcut assertion(ITOLimiteAdc ito):
27    execution(public void ISBCadLimiteAdcMgr.cadastrarLimiteAdc (ITOLimiteAdc))
28    && args (ito);
29
30  //pos condicao: retorno normal
31  after(ITOLimiteAdc ito) returning (): assertion(ito) {
32   if (!((ito.codAgencia != "") && (ito.codAgencia != null) && (ito.nroConta != "") &&
33         (ito.nroConta != null) && (ito.limite.intValue()> 0) && (ito.taxa.intValue() > 0))
34         && (exception == null))
35     log.registrarViolacaoPos("ISBCadLimiteAdcMgr", "cadastrarLimiteAdc(ITOLimiteAdc)",
36                              "Excecao nao foi lancada");
37  }
38  //pos condicao: retorno excepcional
39  after(ITOLimiteAdc ito) throwing (Exception e): assertion(ito) {
40   DeclaredException ed = (DeclaredException)e;
41   //busca pela primeira exceção lançada, retirando o encapsulamento ocorrido durante a propagação
42   e = (Exception)ed.getInitialCause();
43   if (exception != null)
44     if (exception.getClass().getName() != e.getClass().getName())
45       log.registrarViolacaoPos("ISBCadLimiteAdcMgr", "cadastrarLimiteAdc(ITOLimiteAdc)",
46                                "Excecao " + exception.getClass().getName() +
47                                "nao foi propagada.");
```

Figura 65: Código do aspecto de verificação de contrato CadLimiteAdcPost (Parte 1).

```
48    // E4 - Conta inválida - exceção de interface
49    if ((ito.nroConta == "") || (ito.nroConta == null))
50      //Excecao nao foi lancada
51      if (!e.getClass().getName().equals("TEContaInvalidaException"))
52        log.registrarViolacaoPos("ISBCadLimiteAdcMgr", "cadastrarLimiteAdc(ITOLimiteAdc)",
53                                 "Excecao TEContaInvalidaException nao foi lancada." +
54                                 "\nValor do parametro nroConta: " + ito.nroConta);
55       else { //Erro no contexto da excecao
56         TEContaInvalidaException ee = (TEContaInvalidaException)e;
57         if (!(ee.getNumConta() == ito.nroConta))
58           log.registrarViolacaoPos("ISBCadLimiteAdcMgr", "cadastrarLimiteAdc(ITOLimiteAdc)",
59                                    "Erro no contexto da excecao TEContaInvalidaException" +
60                                    "\nValor atual: " + ee.getNumConta() +
61                                    "\nValor do parametro nroConta: " + ito.nroConta);
62       }
63    // E5 - Conta nao cadastrada - exceção propagada - verificação de contexto
64    if (e.getClass().getName().equals("TEContaNaoCadastradaException")) {
65      TEContaNaoCadastradaException ee = (TEContaNaoCadastradaException)e;
66      if (!(ee.getNumConta() == ito.nroConta))
67        log.registrarViolacaoPos("ISBCadLimiteAdcMgr", "cadastrarLimiteAdc(ITOLimiteAdc)",
68                                 "Erro no contexto da excecao TEContaNaoCadastradaException" +
69                                 "\nValor atual: " + ee.getNumConta()+
70                                 "\nValor do parametro nroConta: " + ito.nroConta);
71    }
72  }
```

Figura 66: Código do aspecto de verificação de contrato CadLimiteAdcPost (Parte 2).

Geração dos casos de teste (XML)

Como ainda não há ferramenta para percorrer o diagrama, os caminhos foram extraídos manualmente. Para agilizar a execução do algoritmo, inicialmente foram extraídos apenas os caminhos dos diagramas das interfaces requeridas, como exibidos na Figura 67.

.eps Error

Figura 67: Exemplos de caminhos extraídos no diagrama de interação dos componentes.

Cada caminho é armazenado em através da tag <`Interaction`>, formada por vários <`StubCall`>. A tag <`StubCall`> armazena os vértices do diagrama: o nome da interface e o nome do método (os argumentos não são armazenados pois não são necessários para os casos de teste). O retorno é obtido das condições de guarda das arestas e armazenado na tag <`Exp`>, com sua classificação (exceptional ou normal) e o tipo de dado retornado (datatype). Caso o retorno tenha que ser armazenado (indicado pela expressao let .. in), o atributo *Exp.keep* recebe o valor true.

Após a geração dos caminhos dos diagramas de interação, o diagrama da interface provida é percorrido. A Figura 69 ilustra alguns dos caminhos. Os caminhos acinzentados são cenários de exceções de interface, por isso o diagrama relativo ao método nem é acionado.

```
<Interaction>
 <StubCall interface="ISBNegocioMgrReq" name="recuperaAgencia" keep="false">
  <Exp type="normal" datatype="class ITOControleAgencia"></Exp>
 </StubCall>
 <StubCall interface="ISBNegocioMgrReq" name="checarAgenciaAberta" keep="false">
  <Exp type="normal" datatype=""></Exp> <!-- retorno void -->
 </StubCall>
 <StubCall interface="ISBNegocioMgrReq" name="recuperaColigada" keep="false">
  <Exp type="normal" datatype="ITOColigada"></Exp> <!-- retorno void -->
 </StubCall>
 <StubCall interface="ISBNegocioMgrReq" name="recuperaConta" keep="false">
  <Exp type="exceptional" datatype="class TEContaNaoCadastradaException"></Exp>
 </StubCall>
<Interaction>
```

Figura 68: XML gerado a partir do diagrama de interação dos componentes do método CancelarContratoConta.

O caminho em destaque, porém, tem parte do percurso no diagrama da subatividade: o caminho destacado na Figura 67 complementa o o caminho em destaque na Figura 69, inclusive substituindo seu resultado esperado. O XML é ilustrado na Figura 70, contendo o o xml apresentado na Figura 68.

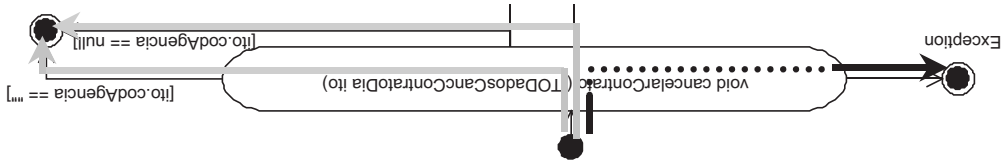

Figura 69: Exemplos de caminhos extraídos no diagrama de especificação final de componentes.

```
<TestCase>
 <MethodCall component="operacoesConta" interface= "ISBCancelaContratoContaMgr"
             name= "cancelarContrato">
  <Input>
   <Arg name="ito" datatype="class ITOCancContratoDia"></Arg>
  </Input>
  <Interaction><!-- Figura anterior--></Interaction>
  <!-- Exp contido na Interaction -->
  <Exp type="exceptional" datatype="class TEContaNaoCadastradaException"></Exp>
 <MethodCall>
<TestCase>
```

Figura 70: XML gerado a partir do diagrama de interação dos componentes do método CancelarContratoConta.

Durante o percurso do diagrama da especificação final, a cada método que pos-

sui uma subatividade associada, é buscado no XML do diagrama da subatividade uma <Interaction> cujo valor de retorno esperado seja o mesmo, ou de uma mesma classe, ou de uma subclasse do valor de retorno esperado pelo método da interface provida (valor contido no nó final). Para que todas as <Interaction>s sejam consumidas, as já usadas são marcadas através do atributo `used`. Após o completar o percurso no diagrama principal, as <Interaction>s não utilizadas são aproveitadas em casos de teste duplicados.

Cada diagrama principal gera um objeto <TestSuite>, que representa um conjunto de casos de teste e é composto por vários <TestCase>. Um objeto <TestCase> representa um caso de teste, correspondente a um caminho no diagrama principal. É formado por vários objetos <MethodCall>, que correspondem aos vértices do diagrama da especificação final, e contêm as informações necessárias para a invocação de um método: seus argumentos (<Arg>), valores a serem retornados pelos *stubs* (<Interaction>) e valores esperados de retorno (<Exp>). A *tag* (<Result>) é preenchida durante a execução dos testes, como retorno real do método. Chamadas a construtores são representadas por <MethodCall>s sem valor de retorno, isto é, sem a tag <Exp>.

Geração dos dados de teste (XML)

Conforme se pode notar nas Figuras 68 e 70, os dados de teste gerados a partir do diagrama são restritos ao tipo do objeto (tags <Exp> e <Arg>). Assim, dados extras devem ser fornecidos manualmente.

Neste projeto, foi decidido por acrescentar ao XML apenas os valores de argumentos, conforme Figura 71, já que os retornos e seus contextos já são verificados pelos contratos. Como o argumento do método cancelarContrato é um objeto, internamente à tag <Argument> referente ao argumento ito é inserida a tag <Object>, que faz referência à criação de um objeto, e contém o valor de seus atributos (tag <Attribute>). Caso os valores fossem passados através de chamadas de métodos, seria utilizada a tag <Method>, contendo o atributos name, para o nome do método, e a tag interna <Parameter>, cuja estrutura é igual à da tag <Attribute>, e representa os parâmetros do método.

Geração dos *stubs*

A geração dos *stubs* segue as decisões tomadas nas fases de identificação e interação dos componentes sobre o escopo dos testes. As interfaces requeridas com baixa controlabilidade e/ou alta complexidade são substituídas por stubs, permitindo a manipulação de suas saídas durante a execução dos testes.

Para a geração dos stubs é necessário apenas a interface requerida a ser implementada. A Figura 72 ilustra parte do código fonte do stub substituto da interface de negócio ISBNegocioMgrReq, requerida pelo componente operacoesConta. Apenas as construções relativas à implementação do método `recuperaConta()` são exibidas. Na verdade, a classe implementa todos os 12 métodos da interface, e possui 333 linhas de código.

Os passos para a construção de um *stub* são executados para cada método a ser implementado:

```
<TestCase>
 <MethodCall component="operacoesConta" interface= "ISBCancelaContratoContaMgr"
             name= "cancelarContrato">
  <Input>
   <Arg name="ito" datatype="class ITOCancContratoDia">
    <Object>
     <Attribute name="indIsentoTxADP" datatype="boolean"><value>FAZER</value></Attribute>
     <Attribute name="indIsentoTxSAQ" datatype="boolean"><value>FAZER</value></Attribute>
     <Attribute name="indTxPadADP" datatype="boolean"><value>FAZER</value></Attribute>
     <Attribute name="indTxPadSAQ" datatype="boolean"><value>FAZER</value></Attribute>
     <Attribute name="taxaPadADP" datatype="java.math.BigDecimal">
      <value>FAZER</value>
     </Attribute>
     <Attribute name="taxaPadSAQ" datatype="java.math.BigDecimal">
      <value>FAZER</value>
     </Attribute>
     <Attribute name="cadastradaNoDia" datatype="boolean"><value>FAZER</value></Attribute>
    </Object>
   </Arg>
  </Input>
  <Interaction><!-- Figura anterior--></Interaction>
  <!-- Exp contido na Interaction -->
  <Exp type="exceptional" datatype="class TEContaNaoCadastradaException"></Exp>
 <MethodCall>
<TestCase>
```

Figura 71: XML acrescido manualmente dos valores dos dados.

1. Construir um vetor de 10 posições do tipo `Object`, que funcionará como uma fila. Os valores são armazenados em uma fila pois um mesmo método pode ser chamado mais de uma vez durante um caso de teste (linha 4). Os valores devem ser armazenados na ordem de chamada do método.

2. Construir um método para inclusão de um valor do tipo `Object` no fim da fila (linhas 7 a 9). O parâmetro é do tipo geral para que o método possa retornar tanto seu valor normal quanto exceções.

3. Construir um método para remoção de todos os elementos da fila do método (linhas 12 a 14).

4. Implementar o método, retornando o primeiro elemento da fila (e removendo-o). Caso o elemento seja uma exceção prevista pelo método, esta deverá ser lançada (linhas 17 a 28).

Geração dos casos de teste (linguagem de programação)

De acordo com a decisão tomada na fase de interação dos componentes que os testes seriam realizados nos componentes ideais, os casos de teste executáveis foram gerados apenas

```
1  public class StubISBNegocioReq implements ISBNegocioReq {
2
3   //Filas para armazenamento dos valores a serem retornados por cada um dos métodos
4   private Vector recuperaContaVector = new Vector(10);
5
6   //Método para inclusão de valores na respectiva fila
7   public void setRecuperaConta(Object ito) {
8     recuperaContaVector.add(ito);
9   }
10
11  //Método para esvaziar uma determinada fila
12  public void setRecuperaContaEmpty() {
13    recuperaContaVector.removeAllElements();
14  }
15
16  //Método da interface ISBNegocioMgrReq implementado, retira um valor da fila
17  public ITOConta recuperaConta(ITOContaPK ito) throws TEContaNaoCadastradaException {
18    Object o = recuperaContaVector.firstElement();
19    recuperaContaVector.removeElementAt(0);
20    if (o instanceof ITOConta) {
21      ITOConta it = (ITOConta)o;
22      return it;
23    } else if (o instanceof TEContaNaoCadastradaException) {
24      TEContaNaoCadastradaException e = (TEContaNaoCadastradaException)o;
25      throw e;
26    } else throw (new Error("Objeto desconhecido atribuido a stub"));
27  }
28 }
```

Figura 72: Código da classe stub que implementa a interface ISBNegocioMgrReq, requerida por operacoresConta.

após a implementação completa dos componentes ideais, que ocorre na fase de montagem.

Aplicação dos Testes

Assim como na geração dos casoDe acordo com a decisão tomada na fase de interação dos componentes que os testes seriam realizados nos componentes ideais, os casos de teste executáveis foram gerados apenas após a implementação completa dos componentes ideais, que ocorre na fase de montagem.

gerenciamentoColigada.ISBColigadaMgrReq			
Operação:	Exceção:	Operação:	Exceção:
ITOColigada recuperarColigada(String codColigada)	E18	*ib.ISBIntegracaoMgr*.DT23 getColigada(String codColigada)	E18
void checarIFFuncionandoCompCheque(String codBanco)	E40	*eb.ISBConfiguracaoMgr*.void checarIFForaCompCheque(String codBanco)	E40

Tabela 45: Reutilizações para Compor o Componente de Negócio: gerenciamentoColigada

gerenciamentoMovimento.ISBMovimentoMgrReq			
Operação:	Exceção:	Operação:	Exceção:
void cobrarTarifaTalao(ITOTalao ito)	E15	*eb.ISBOperacaoMgr*.void cobrarTarifaTalao(DT5 ito)	E15
void cobrarTarifaChequeSustado(ITODadosTarifaChequeSustado ito)	E15	*eb.ISBOperacaoMgr*.void cobrarTarifaSustacaoCheque(DT1 ito)	E15
void checarChequeJaCapturado(ITODadosChequePesq ito)	E39	*eb.ISBMovtoMgr*.void checarChequeJaCapturado(DT29 ito)	E39
void inserirChequeCapturado(ITOCheque ito)	E15	*eb.ISBMovtoMgr*.void inserirChequeCapturado(DT11 ito)	E15

Tabela 46: Reutilizações para Compor o Componente de Negócio: *gerenciamentoMovimento*

gerenciamentoControleAgencia.ISBControleAgenciaMgrReq			
Operação:	Exceção:	Operação:	Exceção:
ITOControleAgencia recuperarAgencia(String codAgencia)	E3	*eb.ISBConfiguracaoMgr*.DT25 recuperaAgencia(DT24 ito)	E3
void checarAgenciaAberta(ITOControleAgencia ito)	E1	*eb.ISBConfiguracaoMgr*.boolean checarAgenciaAberta(DT07 ito)	–

Tabela 47: Reutilizações para Compor o Componente de Negócio: *gerenciamentoControleAgencia*

8 Montagem do Sistema

A fase de montagem do sistema consiste na integração dos componentes para construção da aplicação como um todo. Basicamente, a atividade desempenhada durante esta fase é a construção dos conectores, que podem ser internos aos componentes ideais ou arquiteturais.

Assim, a montagem possui duas sub-fases: a montagem dos componentes ideais e a montagem do sistema. A fase de montagem dos componentes ideais possui as seguintes fases:

1. Implementação do componente ideal

   Entradas: especificação do conector, código fonte do componente normal e tratadores.

   Nesta atividade o conector do componente ideal é implementado de acordo com sua especificação, produzindo o componente ideal já integrado.

   Saídas: componente ideal implementado.

2. Geração dos casos de teste (linguagem de programação)

   Entradas: casos de teste em XML (já com os dados de teste preenchidos), componente sob teste, classes de stubs.

   Podendo ser realizada aqui ou na fase de provisionamento, nesta atividade os casos de teste são implementados numa linguagem específica, neste caso, a linguagem Java. Foi utilizado o framework JUnit, como será detalhado na descrição do exemplo.

   Saídas: código fonte dos casos de teste.

3. Aplicação dos Testes

   Entradas: código fonte dos casos de teste, componente sob teste, classes de stubs.

   Após a aplicação dos testes nos componentes ideais, os seguintes documentos são preenchidos:

   - Diário de teste, com os registros cronológicos da execução;
   - Relatório resumo dos testes, com a descrição resumida das atividades de teste e uma avaliação dos resultados;
   - Relatório de incidência de testes, registrando eventos ocorridos durante os testes que mereçam análise posterior;
   - Relatório de encaminhamento do item de teste, encaminhando as correções para as equipes responsáveis.

   Saídas: diário de testes, relatório resumo de testes, relatório de incidência dos testes, relatório de encaminhamento do item de teste.

Para a montagem do sistema, há apenas atividades de desenvolvimento, já que o método não cobre a realização de testes de integração. As atividades são:

1. Implementação dos conectores do sistema

   Entradas: componentes arquiteturais implementados (componentes ideais)

   Saídas: configuração da arquitetura instanciável

2. Construção do programa principal

   Entradas: configuração da arquitetura instanciável

   Saídas: programa para a instanciação da arquitetura do sistema

Como na Seção anterior, também nesta Seção o componente operacoesConta será utilizado como exemplo, cuja geração será detalhada durante a descrição dos produtos de cada atividade.

8.1 Montagem dos Componentes Ideais

Implementação do componente ideal

A Figura 73 mostra o componente arquitetural operacoesContaArq. Os conectores internos e arquiteturais foram implementados utilizando o COSMOS. Na Figura 73, o conector interno liga o componente normal operacoesConta e os respectivos componentes excepcionais, materializando o componente arquitetural, que segue as diretrizes de um componente ideal.

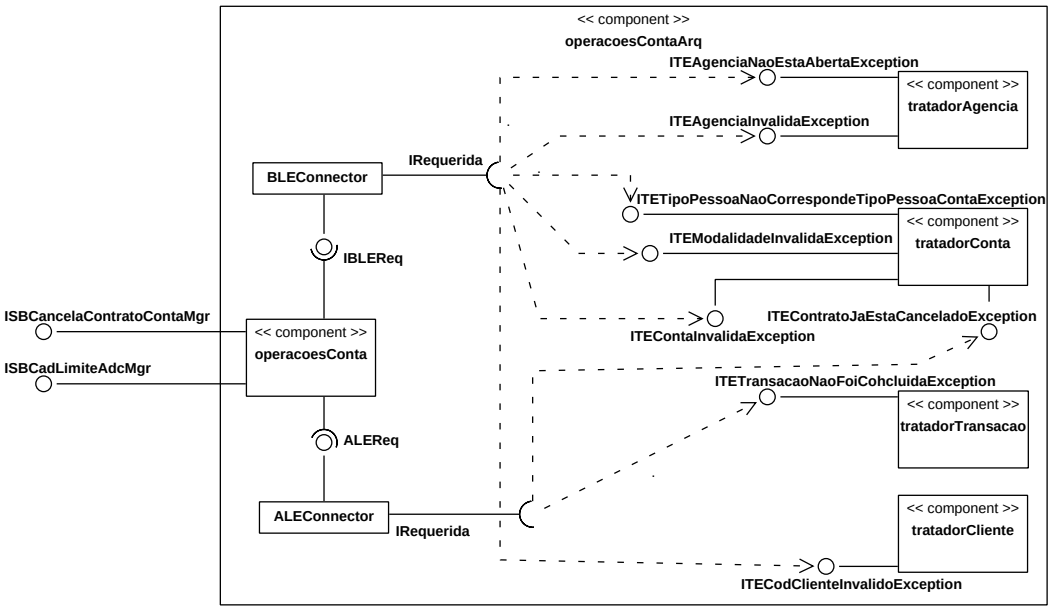

Figura 73: Estruturação do Componente Arquitetural operacoesContaArq

Após a implementação dos conectores internos, deve-se proceder a criação dos componentes tolerantes a falhas ideais. Para isso, é necessário implementar o método da classe inicial do sistema que será responsável pela montagem dos componentes arquietutais. Em

resumo, esse método deve instanciar o componente normal, os excepcionais e os conectores internos. Em seguida, deve realizar a ligação desses componentes. Por se tratar de apenas uma classe, a ligação entre os componentes excepcionais e os conectores internos é feita através da passagem das referências ao construtor da própia classe. Em relação aos componentes do sistema, as dependências são supridas através da execução da operação `setRequiredInterface(String nome, Object facade)`, da interface IManager do COSMOS. A Figura 74 mostra um exemplo desse procedimento de ligação entre o componente normal `CN` e o componente excepcional `CE`, através do conector interno `ConInt`.

```
1     ...
2 CN.spec.prov.IManager cn = CN.impl.ComponentFactory.createInstance();
3 CE.spec.prov.IManager ce = CE.impl.ComponentFactory.createInstance();
4 ConInt conInt = new ConInt(ce.getProvidedInterface("IPCE"));
5 cn.setRequiredInterface(conInt);
```

Figura 74: Exemplo em Java da Ligação entre dois Componentes

Por apresentar uma visão integrada dos componentes normal e excepcional, a partir desse momento é possível realizar testes de unidade relativos aos componentes arquiteturais do sistema, isto é, testar os componentes tolerantes a falhas ideais como unidades de implementação.

Geração dos casos de teste (linguagem de programação)

Para implementação dos *drivers* de teste foi utilizado o framework JUnit, internamente ao ambiente Eclipse. O JUnit é um framework voltado para testes de regressão, utilizado especialmente por metodologias ágeis de desenvolvimento para a construção de testes de unidade pelos próprios desenvolvedores.

Apesar do foco em testes de unidade, o JUnit pode ser utilizado em outras fases, como o teste de componentes. O framework define como estruturar os *drivers* de teste, e fornece ferramentas para executá-los. Os casos de teste são armazenados em classes descendentes da classe *TestCase* do framework, em métodos cujos nomes se iniciam em "test". Os resultados recebidos são comparados aos resultados esperados através de métodos da classe `Assert`. Há também os métodos auxiliares `setUp()`, executado antes de cada caso de teste e utilizado para a preparação do ambiente de testes; e `tearDown`, executado após cada caso de teste, para possível liberação de recursos.

Neste trabalho os *drivers* são gerados a partir dos <`TestSuite`>s contidos nos arquivos XML. Cada <`TestSuite`>, que contém os testes para uma determinada interface, é implementado em uma classe descendente de `TestCase`. Cada <`TestCase`> é transformado um método de teste.

Os drivers são responsáveis pela atribuição dos valores a serem retornados pelos stubs, pelas chamadas dos métodos a serem testados e pela avaliação dos resultados obtidos. Cada caso de teste é responsável por cada uma dessas fases, possuindo a estrutura apresentada na Figura 75, que representa um caso de teste que espera retorno excepcional. Caso o retorno

não seja uma exceção, a linha 8 deve ser removida.

```
1  public void testIP1 () {
2  //Atribuição dos valores a serem retornados por cada método do stub
3  //contido no caminho do caso de teste
4  //Criação dos possíveis objetos a serem passados como parâmetro
5    try {
6      //Chamadas dos métodos da interface provida
7      //Caso de teste falha se uma exceção não é lançada:
8      assertFalse("Excecao nao foi lancada", true);
9    } catch (Exception e) {
10     //o metodo DeclaredException.getInitialCause() busca a exceção mais interna
11     //(a original) dos diversos encapsulamentos:
12     DeclaredException ed = (DeclaredException)e;
13     //Verificação se o resultado é o esperado (neste caso, a exceção lançada):
14     assertEquals(ed.getInitialCause().getClass().getName(),"TipoEsperado");
15   }
16 }
```

Figura 75: Estrutura de um caso de teste com resultado excepcional.

Toda a informação necessária para preenchimento do caso de teste está contida no XML, que deve ser consultado da seguinte maneira:

1. Atribuição dos valores a serem retornados por cada método do stub: os valores estão contidos nas tags <`Interaction`> . <`StubCall`> do <`TestCase`>. Podem ser obtidos o nome do método (<`StubCall`>`.name`), se o retorno é normal ou excepcional (<`StubCall`> . <`Exp`>`.type`), o tipo <`StubCall`> . <`Exp`>`.datatype` e o valor do retorno (interior tag <`StubCall`> . <`Exp`>).

2. Criação dos possíveis objetos a serem passados como parâmetro: os valores estão contidos nas tags <`Input`> . <`Arg`> do <`TestCase`>.

3. Chamadas dos métodos da interface provida: as assinaturas dos métodos estão contidas nas tags <`MethodCall`> do <`TestCase`>. É importante que a ordem de chamada seja preservada, representando o fluxo de execução dos métodos.

4. Verificação se o resultado é o esperado: o valor esperado está contidos na tag <`Exp`> do <`TestCase`>.

Durante a tradução do XML para JUnit foi observado que muitos métodos eram executados em todos os casos de teste: montagem do componente ideal, atribuição de valores aos métodos do stub chamados durante a execução do caso de teste, criação dos objetos passados como parâmetro ao método da interface requerida. Por isso, foi estabelecida a seguinte estrutura para o conjunto de casos de testes:

1. Criação e instanciação dos atributos referentes às partes do componente ideal: componente normal e tratadores.

2. Declaração dos atributos referentes aos stubs (não são instanciados ainda por serem destruídos após cada caso de teste).

3. Instanciação dos datatypes que serão utilizados nos casos de teste (tanto como parâmetros para os métodos da interface provida quanto como retornos de stubs).

4. Método privado `void montaComponente()`: responsável pela montagem do componente ideal, o método é dividido em ligação do componente e interfaces requeridas de negócio, e ligação do componente com os tratadores.

5. Método privado `void setStubReturn()`: responsável pela atribuição de valores aos stubs. Neste método, os valores atribuídos seguem o caminho para término normal do caso de teste. Os retornos que levam caminhos excepcionais são atribuídos internamente ao caso de teste específico.

6. Método do JUnit `void setUp()`: realiza a criação dos objetos parâmetros dos métodos das interfaces providas (seguindo o cenário normal), e invoca os métodos `setStubReturn()` e `montaComponente()`.

7. Casos de Teste: implementação dos <`TestCase`>s.

Utilizando essa estrutura, as atividades de atribuição dos valores dos stubs e dos parâmetros de entrada foram suprimidas dos casos de teste, ficando restritas às mudanças necessárias para que o caminho seja percorrido.

A Figura 76 o código de três casos de teste referentes à interface ISBCancelaContratoContaMgr: o `testIP1` ilustra uma exceção causada por um parâmetro inválido, presente no diagrama da especificação final (primeira seta cinza na Figura 69); o `testIPR4` uma exceção propagada a partir de um stub, presente no diagrama de interação dos componentes (caminho em destaque na Figura 67); e o `testIP15`, o caminho que representa o cenário normal de execução. O código completo desta e de outras classes de teste pode ser consultado no CD anexo.

Aplicação dos Testes

Os testes foram aplicados utilizando o ambiente integrado do JUnit ao Eclipse. As Tabelas 48 e 49 trazem a primeira e segunda partes do relatório de encaminhamento do item de teste, que foi repassado para a equipe de desenvolvimento para correção dos defeitos encontrados.

8.2 Montagem do Sistema

Implementação dos conectores do sistema

A Figura 77 mostra a integração do componente operacoesContaArq com o componente da camada superior, que utiliza os seus serviços. Perceba que essa ligação é intermediada por um conector arquitetural. Os tratadores implementados nesse conector desempenham

```
1  public class TestISBCancelaContratoContaMgr extends TestCase {
2
3   public void testIP1 () {
4     ito.codAgencia = ""; //parametro inválido
5     try {
6       isb.cancelarContratoConta(ito);
7       assertFalse("Excecao nao foi lancada", true);
8     } catch (Exception e) {
9     //o metodo DeclaredException.getInitialCause() busca a exceção mais interna
10    //(a original) dos diversos encapsulamentos
11      DeclaredException ed = (DeclaredException)e;
12      assertEquals(ed.getInitialCause().getClass().getName(),"TEAgenciaInvalidaException");
13    }
14  }
15  public void testIPR4 () {
16    //retirada dos valores normais contidos na fila do stub
17    isbnegocio.setRecuperaContaEmpty();
18    //valor excepcional a ser retornado pelo método do stub
19    isbnegocio.setRecuperaConta(new TEContaNaoCadastradaException());
20    try {
21      isb.cancelarContratoConta(ito);
22      assertFalse("Excecao nao foi lancada", true);
23    } catch (Exception e) {
24      DeclaredException ed = (DeclaredException)e;
25      assertEquals(ed.getInitialCause().getClass().getName(),"TEContaNaoCadastradaException");
26    }
27  }
28  //caso de teste cenário normal
29  public void testIP15() {
30  //não é necessário atribuir valores, já que os valores normais são
31  //atribuídos no início de todos os casos de teste.
32    try {
33      isb.cancelarContratoConta(ito);
34    } catch (Exception e) {
35      assertFalse(true);
36    }
37  }
38 }
```

Figura 76: Código fonte de três casos de teste da interface ISBCancelaContratoContaMgr.

funções de reconfiguração do sistema, uma vez que caso a operação `cancelarContrato(...)` não seja executada com sucesso, o conector tenta executar o serviço por um componente redundante. A Figura 78 mostra o diagrama de seqüência que simula a execução dessa reconfiguração. Nesse caso, optou-se por adotar uma configuração temporária.

Construção do programa principal

Após a finalização da especificação dos conectores do sistema, deve-se prosseguir com a sua implementação, conforme sugerido pelo COSMOS no seu modelo de conectores (Seção **??**). De forma análoga à montagem dos componentes arquiteturais do sistema, após a imple-

Relatório de encaminhamento do item de teste			
Contexto	Relatório dos testes dos componentes ideais		
Participantes	Camila Rocha		
Abrangência	Componentes de Sistema operacoesConta, operacoesTalao, operacoesChequeSustado e operacoesChequeCapturado, integrados aos tratadores e ao componenteUtilitário		
Componente	operacoesChequeCapturado		
Interface	ISBCapturaChequeMgr		
Total Casos de Teste	27		
Sumário dos Defeitos	**Quantidade**	**Casos de Teste**	
JUnit	2	IP5 e IP6	
Contrato	2	IP4 e IP5	
Descrição dos Defeitos			
Caso de Teste	**Falha Introduzida**	**Defeito Ocorrido**	**Observações**
IP4	Parâmetro comp = 0	O atributo nroCheque da exceção TECompChequeInvalidoException vale null, diferentemente do valor passado como parâmetro.	
IP5	Parâmetro dataMovimento = null	Exceção lançada: NullPointerException. Exceção esperada: TEDataInvalidaException.	
IP6	Campo dia do parâmetro dataMovimento = 40	Não foi lançada a exceção TEDataInvalidaException, o caso de teste terminou normalmente.	Como a data é armazenada pela linguagem como um valor inteiro que representa a soma dos campos, a soma resultou em uma data válida, porém diferente da passada como parâmetro.
Componente	operacoesTalao		
Interface	ISBObtencaoTalaoMgr		
Total Casos de Teste	60		
Sumário dos Defeitos	**Quantidade**	**Casos de Teste**	
JUnit	2	IP20 e IP21	
Contrato	2	IP20 e IP21	
Descrição dos Defeitos			
Caso de Teste	**Falha Introduzida**	**Defeito Ocorrido**	**Observações**
IP20	Parâmetro codAgenciaLocal = null	Exceção TEAgenciaInvalidaException não foi lançada	
IP21	Parâmetro codAgenciaLocal =	Exceção TEAgenciaInvalidaException não foi lançada	

Tabela 48: Relatório de encaminhamento do item de teste (Parte 1).

mentação dos conectores arquiteturais, é necessário implementar o método responsável pela integração do sistema propriamente dita, a fim de "plugar" os componentes nos conectores, de acordo com as dependências de cada um. Isso também é feito através da execução das operações `setRequiredInterface(String nome, Object facade)`, das interfaces IManager de cada um dos componentes COSMOS.

Do ponto de vista da dependência entre os passos que o programa principal deve executar, primeiro devem ser montados os componentes arquiteturais do sistema, que foram especificados na Seção **??**. Só então esses componentes podem ser integrados para constituir

Componente	operacoesChequeSustado		
Interface	ISBSustarChequeMgr		
Total Casos de Teste	60		
Sumário dos Defeitos	**Quantidade**	**Casos de Teste**	
JUnit	4	IP3, IP4, IP11 e IP12	
Contrato	3	IP3, IP4 e IP11	
Descrição dos Defeitos			
Caso de Teste	**Falha Introduzida**	**Defeito Ocorrido**	**Observações**
IP3	Parâmetro codAgenciaLocal = null	Exceção TEAgenciaInvalidaException não foi lançada	
IP4	Parâmetro codAgenciaLocal =	Exceção TEAgenciaInvalidaException não foi lançada	
IP11	Parâmetro dataSustacao = = null	Exceção lançada: NullPointerException. Exceção esperada: TEDataInvalidaException.	
IP12	Campo dia do parâmetro dataSustacao = 32	Não foi lançada a exceção TEDataInvalidaException, o caso de teste terminou normalmente.	Idem IP6 operacoesChequeCapturado.

Componente	operacoesConta		
Interface	ISBCadLimiteAdcMgr		
Total Casos de Teste	20		
Sumário dos Defeitos	**Quantidade**	**Casos de Teste**	
JUnit	2	IP9, IP10	
Contrato	2	IP9, IP10	
Descrição dos Defeitos			
Caso de Teste	**Falha Introduzida**	**Defeito Ocorrido**	**Observações**
IP9	Parâmetro taxa = null	Exceção lançada: TETaxaDeveSerMaiorZeroException. Exceção esperada: TETaxaInvalidaException.	
IP10	Parâmetro limite = null	Exceção lançada: TELimiteDeveSerMaiorZeroException. Exceção esperada: TELimiteInvalidoException.	

Interface	ISBCancelaContratoContaMgr		
Total Casos de Teste	27		
Sumário dos Defeitos	**Quantidade**	**Casos de Teste**	
JUnit	0		
Contrato	0		

Tabela 49: Relatório de encaminhamento do item de teste (Parte 2).

	Exceção:	Interface do Tratador
E56	TEReconfiguracaoSistemaImpossibilitadaException Atributos: nomeOperacao:String nomeCompFalhou:String nomeCompAlternativo:String causaErro:String	ITEReconfiguracaoSistemaImpossibilitadaException Operações: void lancar(E56 e) throws E56

Tabela 50: Exceção arquitetural tratada no conector conNegOperacoesConta

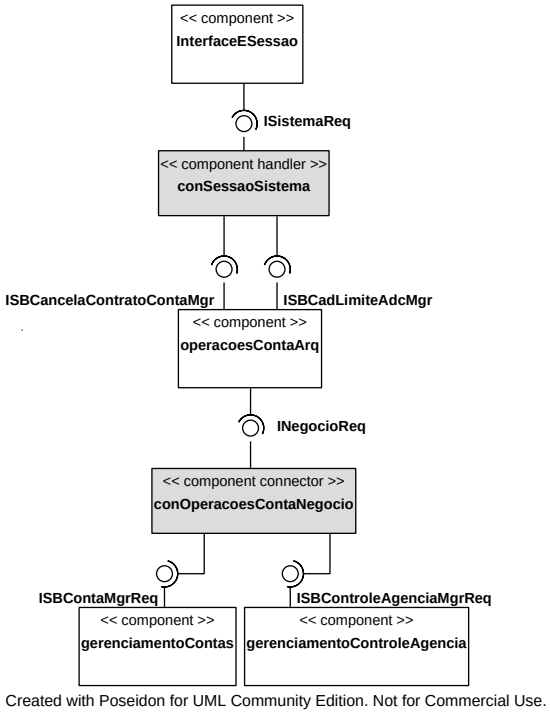

Figura 77: Parte da Arquitetura do Sistema

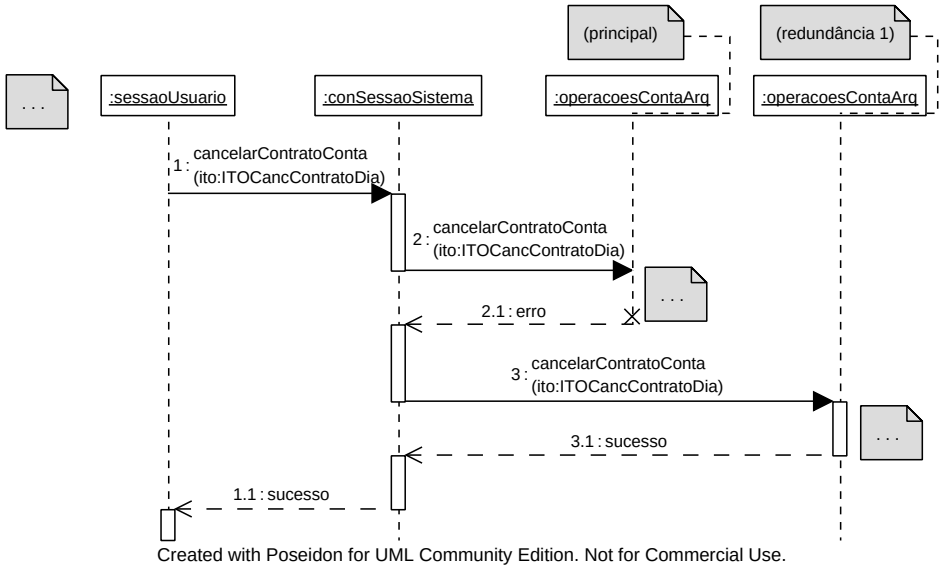

Figura 78: Simulação da Reconfiguração do Serviço cancelarContrato(...)

o sistema propriamente dito.

9 Avaliação dos Resultados

Como explicado na Seção 4, o estudo de caso consistiu da aplicação do processo UML Components adaptado com a especificação do comportamento excepcional no desenvolvimento de um sistema financeiro. Basicamente, o objetivo desse sistema é registrar e controlar o envio de talões de cheques e os limites de crédito. Como mostrado nesse relatório, ele foi especificado utilizando as atividades e diretrizes do processo adaptado ao longo do desenvolvimento.

O estudo de caso foi planejado pelo autor e executado por outras duas pessoas, sendo uma delas especialista no domínio do negócio. O tempo gasto na fase de especificação dos casos de uso referentes às seis funcionalidades foi de 64 pessoas-hora. E como até a data da execução do estudo não tínhamos disponível uma ferramenta de geração automática de código voltada ao COSMOS, o tempo gasto na fase de provisionamento dos componentes também foi de 64 pessoas-hora. A fase de provisionamento consistiu da implementação dos componentes da camada de sistema e da construção dos *wrappers* para os componentes reutilizados. Sendo assim, o tempo útil total gasto na execução do estudo de caso foi de 128 pessoas-hora, ou aproximadamente 0,75 pessoas-mês, distribuídas conforme mostrado na Figura 79.

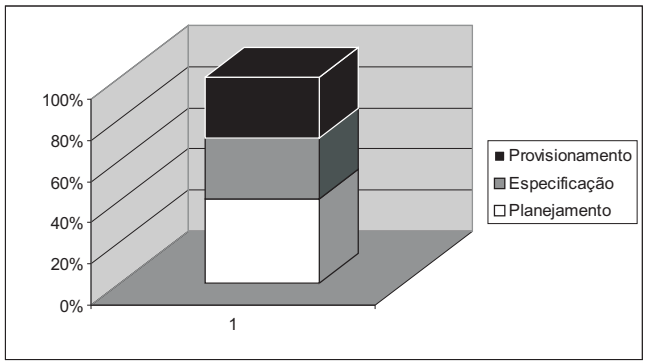

Figura 79: Distribuição do Custo Total Durante as Fases do Desenvolvimento

Devido à falta de conhecimento do método MDCE+, antes de cada fase do desenvolvimento, foi necessário uma seção de treinamento com os executores do estudo de caso. Este treinamento consumiu aproximadamente 10% do tempo gasto na fase de especificação de requisitos e 7% do tempo gasto na fase de especificação. Em um segundo estudo de caso com a mesma equipe, esse tempo não seria mais necessário.

A Figura 80 mostra a proporção do tempo gasto na fase de especificação (32% do total), dentro dos seus três estágios.

Após a execução do estudo de caso, o processo adaptado foi avaliado de acordo com alguns critérios de qualidade. Essa análise foi feita a partir da coleta da opinião dos executores através de questionários anônimos. Os questionários apresentam questões objetivas e algumas sugestões e comentários abertos. Inicialmente, foi feita uma avaliação de cada fase do desenvolvimento separadamente; em seguida, o processo adaprado foi avaliado de uma

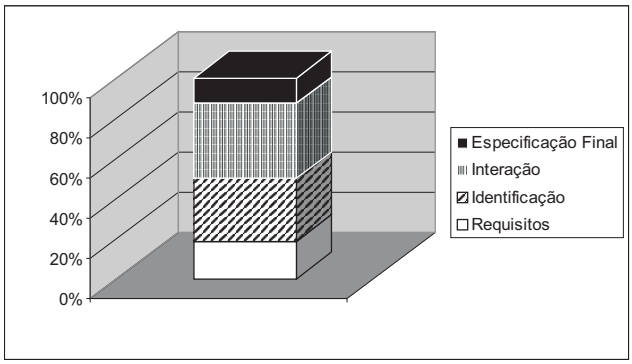

Figura 80: Distribuição do Custo da Fase nos Estágios da Especificação

maneira geral. As respostas das questões mostram as opiniões em uma escala de 1 (pior) a 5 (melhor). O resultado final dessa avaliação pode ser visto nas Tabelas 51 e 52.

Analisando as Tabelas 51 e 52, os principais benefícios da adoção do método MDCE+ foram as seguintes: (i) o método foi considerado satisfatoriamente fácil de usar. Apesar dele ser considerado intuitivo, a fase de especificação dos componentes apresentou um grande número de atividades, o que aumentou a sua complexidade e reduziu a média do processo. (ii) A qualidade das exceções foi considerada melhor. As exceções descobertas utilisando o processo adaptado apresentou uma alta qualidade. Essa classificação se baseou na relação entre as exceções e a lógica do negócio, o que proporciona uma maior independência entre o sistema e a linguagem de programação adotada. (iii) Refinamento do tipo do erro. O aumento da qualidade das exceções, aliado ao maior número de exceções detectadas (proximadamente 20%), aumentou o detalhamento dos erros e segundo a opinião dos especialistas, deve facilitar a detecção dos erros nas atividades de manutenção. (iv) Uma boa relação custo x benefício. Apesar do tempo de aprendizagem ser considerado alto, foi considerado que os benefícios apresentados anteriormente foram compensatórios. No contexto de sistemas críticos, julgou-se que essa relação seria ainda mais satisfatória.

Critério Avaliado	Especificação de Requisitos	Especificação do Componente
- Entendimento	Fácil (4)	Médio/Fácil (3,7)
- Seqüência lógica	Intuitivo (4)	Intuitivo (4)
- Possibilidade de mudança de requisitos durante a especificação	Quase sempre (3,5)	Quase sempre (3,7)
- Qualidade das exceções	Diretamente relacionada ao negócio (4)	Diretamente relacionada ao negócio (4)
- Número de exceções descobertas	Alto (4,5)	Alto (4,7)
- Documentação do sistema	Bom (4)	Bom (4)
- Nível de conhecimento necessário	Principalmente do negócio (4)	Principalmente do negócio (4)

Tabela 51: Análise das Fases da Especificação (valores de 1 a 5)

Critério Avaliado	Avaliação Geral
- Entendimento	Médio/Fácil (3,8)
- Seqüência lógica	Intuitivo (4)
- Possibilidade de mudança de requisitos durante a especificação	Quase sempre (3,6)
- Qualidade das exceções	Diretamente relacionada ao negócio (4)
- Número de exceções descobertas	Alto (4,6)
- Documentação do sistema	Bom (4)
- Nível de conhecimento necessário	Principalmente do negócio (4)
- Número de documentos produzidos (artefatos obrigatórios e opcionais)	Médio (3)
- Número de documentos produzidos (apenas artefatos obrigatórios)	Médio/Baixo (3,5)
- *Overhead* de tempo	Médio/Pouco (3,5)
- Custo x benefício (sistemas não críticos)	Um pouco caro / Barato (3,5)
- Custo x benefício (sistemas críticos)	Barato, vale à pena prevenir (4)

Tabela 52: Análise Geral do Processo Adaptado (valores de 1 a 5)

9.1 Avaliação do Produto

Além da dessa análise feita ao processo adaptado, no decorrer do estudo de caso, também foi avaliada a qualidade do produto final. Esta análise se baseou principalmente nos seguintes critérios:

- Opiniões de especialistas em relação ao custo adicional e ao benefício em relação à manutenibilidade[2];
- A qualidade das exceções, isto é, quão próximo do domínio elas estão, e conseqüentemente, quão independente da linguagem de programação adotada;
- O número de exceções relevantes (que representam erros graves) que foram identificadas;
- Especificação das atividades de tratamento e mascaramento de falhas (possibilidade de recuperar de estados errôneos ou de serviços indisponíveis).

Apesar de utilizar mais de 20% do tempo com atividades de especificação do sistema, o *overhead* total do desenvolvimento foi cosiderado baixo. De acordo com os executores do estudo de caso, isso se deve à melhoria da documentação do sistema referente aos comportamentos normal e excepcional. Esse detalhamento na documentação facilita a implementação do sistema.

Com a utilização da abordagem sistemática do processo adaptado, foram descobertas aproximadamente 20% a mais de exceções. Esse resultado foi obtido ao comparar com outros sistemas similares da empresa. Esse acréscimo no número de exceções representa um maior refinamento dos tipos de falhas. Aliado ao crescimento da qualidade das exceções e à maior qualidade das informações contextuais, esse acréscimo deve facilitar as atividades de manutenção relacionadas à identificação e correção de *bugs*. Apesar de esperado, esse

[2]do inglês *maintainability*

benefício não foi quantificado efetivamente. Para isso, deve-se fazer experimentos maiores com abrangência em todo o ciclo de desenvolvimento, incluindo atividades de manutenções corretivas e perfectivas.

Outra característica observada no decorrer da especificação foi o aumento da qualidade das exceções descobertas. Essa melhoria foi decorrente da separação das exceções de acordo com a proximidade da lógica do negócio. Conseqüentemente, com a utilização de uma hierarquia de classificação de exceções própria, a maioria das exceções são independentes da linguagem de programação adotada. Essa melhoria facilita, por exemplo, a mudança de plataforma de desenvolvimento ou até mesmo o desenvolvimento em linhas de produção[3]. A preocupação constante com a contextualização das informações também contribuiu para a melhoria da qualidade do tratamento, uma vez que foi possível disponibilizar informações mais detalhadas para a elaboração de relatórios de falhas.

A facilidade para a especificação de exceções arquiteturais (nesse caso, que envolvem reconfiguração) foi outro benefício percebido. As exceções de outros sistemas equivalentes apenas sinaliza a impossibilidade de execução. Dessa forma, não existia uma distinção no tratamento entre os serviços críticos e não críticos.

Devido ao aspecto não crítico de alguns casos de uso, como por exemplo Requisitar Talão de Cheques, o estudo de caso também foi importante para avaliar como o método pode se adequar ao desenvolvimento de sistemas de software sem requisitos críticos de disponibilidade. Essa adequação interferiu principalmente na sub-fase de interação entre os componentes, durante o refinamento do modelo de falhas. Form especificadas exceções genéricas, que foram utilizadas para substituir as mais específicas, satisfazendo as decisões de projeto.

9.2 Avaliação do Método

Com a utilização do método em apenas um estudo de caso, só foi possível fazer uma análise preliminar do método MDCE+, de acordo com os seis critérios de qualidade de processo sugeritos por Sommerville [Som01]. Essa análise foi feita em conjunto, tanto pelos planejadores, quanto pelos executors do estudo de caso, onde cada um dos critérios foi classificado numa escala de *Baixo*, *Médio*, ou *Alto*, seguido de uma justificativa breve. As características analizadas foram as seguintes:

Facilidade de Entendimento. *Média.* Mesmo com o grande número de atividades, a separação clara entre as naturezas de cada uma dessas atividades facilitou o entendimento.

Visibilidade. *Alta.* Os resultados produzidos em todas as fases são definidos claramente através dos documentos produzidos.

Suporte Ferramental. *Alta.* Todos os modelos necessários podem ser construídos em ferramentas CASE atuais, porém, mais facilidades poderiam ser adicionadas a essas ferramentas, tal como a geração automática de código para os componentes COSMOS. Atualmente, está sendo desenvolvido o ambiente Bellatrix [TFGR04], que irá abranger todas as fases do método, dando suporte tanto à produção de modelos, quanto à orientação das atividades do método e à geração automática de código-fonte.

[3]do inglês *product line production*

Confiabilidade. *Média.* O progresso das fases do método é documentado pelo desenvolvimento de artefatos de entrada/saída. O fato desses artefatos serem usados na fase seguinte facilita a antecipação da identificação de erros, reduzindo os custos das correções. Porém, alguns tipos de falhas de especificação não são identificadas facilmente, como por exemplo, inconsistências entre os requisitos e a especificação de casos de uso.

Robustêz. *Baixa.* Como não existem muitas diretrizes para gerenciamento do projeto, problemas inexperados podem comprometer a viabilidade do sistema.

Manutenibilidade. *Média.* Com a separação clara das atividades de acordo com as respectivas fases de desenvolvimento e papéis desempenhados, a inclusão de novas fases ou a modificação de fases anteriores fica facilitada. A remoção, porém, não é tão simples, uma vez que isso poderia provocar "efeitos colaterais´´ decorrentes da dependência que uma atividade tem dos artefatos produzidos pelas suas antecessoras.

Agilidade. *Baixa.* A prioridade do método é a confiabilidade do produto final e o detalhamento da documentação para auxiliar os testes. Por isso, sem o auxílio de ferramentas CASE apropriadas, os documentos produzidos reduzem a produtividade da fase de desenvolvimento.

10 Conclusões

Neste relatório foi apresentado um estudo de caso com o método MDCE+, um método para modelagem e teste do comportamento excepcional de sistemas baseados em componentes. O método MDCE+ melhora a confiabilidade dos sistemas desenvolvidos, uma vez que oferece uma maneira sistemática de modelar e testar as exceções e os seus tratadores, distribuindo essas atividades em todo o ciclo de desenvolvimento do software. Essa maneira estruturada e rigorosa de detectar e tratar as exceções no contexto de ocorrência de falhas é particularmente relevante para os sistemas que possuem requisitos críticos de confiabilidade.

A principal característica do método MDCE+ é a execução em paralelo das atividades de desenvolvimento e testes, seguindo a abordagem de testes baseados em modelos[4]. Esse paralelismo reduz o *overhead* decorrente da sua adoção, favorecento a aplicação do método em sistemas reais, que possuem requisitos de tempo cada vez mais críticos [Som01]. Outra característica importante do método é a interação constante entre os desenvolvedores e os testadores através dos artefatos especificados. Muitos desses artefatos são compartilhados pelos dois times. Essa interação contribui para a melhoria da qualidade desses artefatos. A sincronização é garantida através de atividades de atualização conjunta desses artefatos comuns.

Além do fato de conduzir as atividades de desenvolvimento e testes em paralelo, o método MDCE+ enfatiza o aspecto arquitetural do sistema, contribuindo assim para a definição do fluxo de exceções na arquitetura. Em relação aos aspectos de manutenibilidade, o método MDCE+ proporciona uma separação clara entre as especificações dos componentes que implementam os comportamentos normal e excepcional. Além disso, o métopo contém atividades para compor esses dois comportamentos segundo a estrutura do componente tolerante a falhas ideal [AL90]. Esta qualidade simplifica a representação dos componentes

[4]do inglês *model based testing*

e conseqüentemente, proporciona um melhor entendimento, uma maior confiabilidade, e uma maior manutenibilidade do sistema.

A principal limitação observada no MDCE+ foi a obrigatoriedade de atualização dos artefatos especificados. Dessa forma, a falta de comprometimento da equipe de desenvolvimento pode comprometer a geração automática dos casos de teste, implicando em re-trabalho e atraso na entrega do produto. Para lidar com essa limitação, as fases de atualização conjunta dos artefatos, que são propostas pelo método MDCE+, devem ser executadas impreterivelmente, envolvendo as equipes de desenvolvimento e testes.

Nosso trabalho futuro mais imediato é a construção de um conjunto de ferramentas CASE que sejam orientadas ao método MDCE+. Essas ferramentas devem auxiliar a construção dos diversos artefatos produzidos no método e a geração automática de casos de teste. Além disso, atualmente só foram especificadas as atividades dos testes unitários de componentes, assim, outro trabalho futuro importante é complementar o método para conter atividades de inspeções e de testes de integração e de sistema, onde possam ser aplicadas técnicas de teste de robustês[5].

Finalmente, nós pretendemos fazer uma separação entre o método MDCE+ e as atividades do processo UML Components. Essa separação tornará possível a inserção do mesmo método em outros processos de desenvolvimento de software.

Como a execução do estudo de caso consistiu da aplicação de um processo com fases semelhantes ao UML Components, a validação do método para reuso foi parcial. Em outras palavras, foi testada a atividade de reutilização de componentes após a especificação (na fase de provisionamento dos componentes). Em relação à maximização do reuso, como trabalho futuro, deverão ser avaliadas as demais atividades de reutilização de componentes: (i) negociação dos requisitos a partir dos componentes já existentes; e (ii) reuso de *framework* e de componentes arquiteturais.

Referências

[AL90] T. Anderson and P. A. Lee. *Fault Tolerance: Principles and Practice*. Springer-Verlag, 2nd edition, 1990.

[1] Robert V. Binder. *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[BRJ99] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language user guide*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1999.

[CD00] John Chessman and John Daniels. *UML Components*. Addison-Wesley, 2000.

[dLR99] Rogério de Lemos and A. Romanovsky. Exception handling in a cooperative object-oriented approach. In *Proc. of the 2nd IEEE ISORC'99*, May 1999.

[5] do inglês *robustness testing*

[BFR04] P.H. da S. Brito, F. Castor Filho, , and C. M. F. Rubira. Um Método para Modelagem de Exceções em Desenvolvimento Baseado em Componentes. In *IV WDBC*, 2004.

[DW99] Desmond D'Souza and Alam Cameron Wills. *Objects, Components, and Frameworks with UML The Catalysis Approach.* Addison-Wesley, 2nd edition, 1999.

[Fer01] Gisele R. M. Ferreira. Tratamento de exceções no desenvolvimento de sistemas confiáveis baseados em componentes. Master's thesis, IC, Unicamp, December 2001.

[Goo75] John B. Goodenough. Exceptional handling: Issues and a proposed notation. *CACM*, 18(12), 1975.

[JGR03] Moacir C. Silva Jr., Paulo A. C. Guerra, and Cecília Rubira. A java model for evolving software systems. *IEEE International Conference on Automated Software Engineering(ASE'03)*, October 2003.

[K+97] Gregor Kiczales et al. Aspect-oriented programming. In *Proc. of ECOOP'97.* LNCS 1241, 1997.

[Eclipse] Eclipse Project. Eclipse Foundation. http://www.eclipse.org, 2005.

[RdLeFCF04] C. Rubira, R. de Lemos, and G. Ferreira e F. Castor Filho. Exception handling in the development of dependable component-based systems. In *Software Practice and Experience*, 2004.

[RM04] Camila Ribeiro Rocha and Eliane Martins. A strategy to improve component testability without source code. In Sami Beydeda, Volker Gruhn, Johannes Mayer, Ralf Reussner, and Franz Schweiggert, editors, *SOQUA/TECOS*, volume 58 of *LNI*, pages 47–62. GI, 2004.

[RS03] Darrel Reimer and Harini Srinivasan. Analysing exception usage in large java applications. In *Proc. of ECOOP Workshop on Exception Handling in Object-Oriented Systems(EHOOS'2003)*, 2003.

[Som01] Ian Sommerville. *Software Engineering.* Addison-Wesley, 6th edition, 2001.

[Szy00] Clemens Szyperski. Component software and the way ahead. In Gary T. Leavens and Murali Sitaraman, editors, *Foundations of Component-Based Systems*, chapter 1, pages 1–20. Cambridge University Press, 2000.

[TFGR04] R. Tomita, F. Castor Filho, P. Guerra, and C. Rubira. Bellatrix: Um ambiente para suporte arquitetural ao desenvolvimento baseado em componentes. In *IV WDBC*, 2004.