Uma Abordagem Arquitetural com Tratamento de Exceções para Sistemas de Software Baseados em Componentes

Vinicius Asta Pagano

Trabalho Final de Mestrado Profissional

Instituto de Computação
Universidade Estadual de Campinas

Uma Abordagem Arquitetural com Tratamento de Exceções para Sistemas de Software Baseados em Componentes

Vinícius Asta Pagano

Campinas, 14 de Agosto de 2004

Banca Examinadora:

Profa. Dra. Cecília Mary Fischer Rubira (Orientadora)
Instituto de Computação – Universidade de Campinas

Profa. Dra. Ana Cristina Vieira de Melo
Instituto de Matemática e Estatística – Universidade de São Paulo

Profa. Dra. Eliane Martins
Instituto de Computação – Universidade de Campinas

Profa. Dra. Ariadne Maria Brito Rizzoni de Carvalho (Suplente)
Instituto de Computação – Universidade de Campinas

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Pagano, Vinícius Asta

P14u Uma abordagem arquitetural com tratamento de exceções para sistemas de software baseados em componentes / Vinícius Asta Pagano -- Campinas, [S.P. :s.n.], 2004.

Orientadora : Cecília Mary Fischer Rubira

Trabalho final (mestrado profissional) - Universidade Estadual de Campinas, Instituto de Computação.

1. Software - Arquitetura. 2. Software (Sistemas). 3. Tolerância a falhas (Computação). 4. Software – Confiabilidade. I. Rubira, Cecília Mary Fischer. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Uma Abordagem Arquitetural com Tratamento de Exceções para Sistemas de Software Baseados em Componentes

Este exemplar corresponde a redação do Trabalho Final devidamente corrigida e defendida por Vinícius Asta Pagano e aprovada pela Banca Examinadora.

Campinas, 14 de Agosto de 2004

Cecília Mary Fischer Rubira

Profa. Dra. Cecília Mary Fischer Rubira
(Orientadora)

Trabalho Final apresentado ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Resumo

A especificação da arquitetura de software é fundamental para a construção de um sistema de software que atenda a requisitos de confiabilidade e que seja baseado em componentes de software reutilizáveis. Componentes reutilizáveis podem ser implementados sem o conhecimento de um sistema onde serão reutilizados. A arquitetura de software deste sistema deve fornecer o contexto de utilização de componentes de modo que eles possam ser devidamente identificados e integrados a este sistema, atendendo aos requisitos de confiabilidade. Para isto, a arquitetura deve ser bem organizada e apresentar tanto a especificação normal quanto a especificação excepcional de seus componentes, além dos conectores que interligam esses componentes.

Este trabalho propõe uma solução centrada na arquitetura baseada em um método para a especificação de componentes da arquitetura de um sistema e de seu comportamento excepcional (MECE). O método MECE promove a definição de uma arquitetura com a especificação normal e excepcional de seus componentes e a identificação de conectores. Através da especificação excepcional identifica-se quais exceções cada componente deve lançar em suas interfaces providas, e quais exceções associadas as suas interfaces requeridas o componente deve tratar. Mesmo com uma arquitetura bem especificada pode ser que não se encontre um componente reutilizável que atenda à sua especificação excepcional, sendo necessário um trabalho de adaptação no momento da integração deste componente na arquitetura definida. A solução apresentada neste trabalho propõe o uso de estratégias de estruturação arquitetural para tratamento de exceções, que auxiliam este trabalho de adaptação e a integração de componentes a um sistema.

A solução proposta foi implementada num estudo de caso de um sistema real, onde inicialmente foi especificado e implementado o comportamento normal do sistema e depois o seu comportamento excepcional, usando o método MECE e as estratégias de estruturação arquitetural para tratamento de exceções.

Abstract

The software architecture specification is the base for a component-based software construction that must be compliant with dependability requirements. A component-based software is constructed with reusable components. Reusable components can be implemented without knowing the context of the systems where these components would be reused. The software architecture of a system must provide the reuse context in order to allow the identification of reusable components and the integration of these components to this system. The software architecture must be well organized and present the normal and exceptional specification of its components and the connectors that must be used to connect these components.

This work proposes one architecture solution based on a method for normal and exceptional specification of system components (MECE). The method MECE provides the definition of architecture with the normal and exceptional specification of its components and the definition of the connectors. The component exceptional specification identifies the exceptions that must be thrown by a component in its provided interfaces, and the exceptions that must be caught in its required interfaces. Even with a well-specified architecture, maybe it is not possible to identify a reusable component that is totally compliant with the exceptional specification, which requires adaptation activities to integrate this reusable component to the defined architecture. The solution presented in this work proposes the use of architectural strategies for exception handling that guides the adaptation activities and the components integration to a system.

The proposed solution was implemented in a study case of a real system, where firstly the normal behavior of the system was implemented, and secondly the exceptional behavior was implemented, applying the method MECE and the architectural strategies for exception handling.

Agradecimentos

Gostaria de agradecer a Deus pelas oportunidades e pelas pessoas que tem colocado em minha vida.

Agradeço aos meus pais e irmão pela educação disciplinada e exemplos de dedicação, comprometimento, iniciativa, respeito, honestidade e humildade.

Agradeço a minha esposa Andréa, pela paciência e tolerância nos momentos de minha ausência devido aos estudos e trabalho, e por sua companhia.

Aos meus orientadores do meio acadêmico, Cecília Rubira e Paulo Astério, e do meio profissional, Álvaro, Cristiane, Rosana, Lisiane, Edilson, Paulo Tavares, Marcionilia, Alexandre, Victor e Milton Maia, que fizeram com que o Vinícius amadurecesse muito.

Agradeço ao Moacir Caetano, que muito me ajudou nos estudos e implementação do modelo COSMOS.

Agradeço aos professores e funcionários do IC.

Aos meus familiares e amigos de infância, colégio, faculdade e trabalho.

Acredito muito no estilo de aprendizado "Mestre/Aprendiz". Espero que eu esteja aprendendo devidamente com todas as pessoas que conheço, como um bom aprendiz, e que eu esteja passando os ensinamentos adiante para outras pessoas.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Quadros

Capítulo 1

Introdução

A demanda por sistemas de software confiáveis tem aumentado em muitas áreas de nossa sociedade. Com a finalidade de atender a essa demanda, e diminuir o custo de desenvolvimento destes sistemas, a construção de novos sistemas de software reutilizando componentes de software já existentes é extremamente adequada. **Componentes** são unidades de software especificadas, desenvolvidas e testadas separadamente e que podem ser integradas para se construir algo com maior funcionalidade [Szyperski97]. Um **sistema de software baseado em componentes** é caracterizado pela composição de componentes reutilizáveis que devem ser identificados e integrados ao sistema, tendo como base a especificação da arquitetura deste sistema. A **arquitetura de um sistema** define seus componentes, suas interações, e suas funcionalidades [Sommerville01]. Dois componentes podem interagir diretamente um com o outro, ou podem interagir através de um conector. Um **conector** é um componente que interliga componentes. Um conjunto de componentes e conectores interligados definem uma **configuração de componentes**.

Um sistema de software confiável é caracterizado por atender requisitos de confiabilidade. **Confiabilidade** é uma propriedade de um sistema que possibilita aos seus usuários confiarem, justificadamente, nos serviços que ele oferece [Lee+90]. Quando um sistema de software deixa de oferecer as funcionalidades previstas em seus serviços, ou o faz em desacordo com suas especificações, diz-se que ocorreu um **defeito** no sistema. A ocorrência de um defeito de software é decorrente de um estado interno errôneo do sistema, definido como **erro**. Para que ocorra um erro deve haver uma causa, a qual se chama de **falha** (*fault*) [Lee+90].

Pode-se aplicar no projeto de um sistema, mecanismos de tolerância a falhas para reagir de modo ordenado e previsível quando uma falha ocorrer. Um mecanismo de tolerância a falhas é iniciado quando se detecta um erro no sistema. Quando um erro é detectado, o sistema deve ser recuperado e levado para um estado livre de erros. Existem duas abordagens de recuperação: por retrocesso (*backward error recovery*) e por avanço (*forward error recovery*). A recuperação por retrocesso conduz o sistema para um estado correto, anterior a falha. A recuperação por avanço conduz o sistema para um novo estado supostamente correto.

Uma implementação de mecanismo de tolerância a falhas, com abordagem de recuperação por avanço, é o **tratamento de exceções**. Quando um erro é detectado, uma **condição excepcional** é caracterizada e é sinalizada através do lançamento de uma **exceção**. Este lançamento leva a um desvio do fluxo normal de instruções para que a condição excepcional seja tratada [SunWeb]. Durante o tratamento da condição excepcional, o tratador pode levar o sistema a um estado supostamente livre de erros e fazer com que o sistema volte a seu fluxo normal de instruções. Caso o tratador não consiga levar o sistema para um estado livre de erros, deve sinalizar para um tratador de um contexto mais abrangente que o sistema ainda está em um estado errôneo.

Condições excepcionais podem ser previstas na especificação excepcional de um sistema ou de seus componentes, sendo assim chamadas de **condições excepcionais antecipadas** [Guerra04]. Estas condições excepcionais antecipadas são sinalizadas num sistema, por meio de exceções definidas com base em tipos de **exceções declaradas** na especificação do sistema. Por outro lado, **condições excepcionais não antecipadas** [Guerra04] também podem ocorrer em um sistema, mas estas condições geralmente são sinalizadas por **exceções não declaradas** na sua especificação.

Para que um sistema atenda a requisitos de confiabilidade e seja baseado em componentes reutilizáveis, ele deve se basear numa arquitetura bem organizada que contenha a especificação normal e excepcional dos componentes. A **especificação normal** de um componente deve descrever seu comportamento normal definindo suas interfaces providas e requeridas, bem como as operações de cada interface. A **especificação excepcional** de um componente deve descrever seu comportamento excepcional definindo quais exceções devem ser lançadas pelas operações de um componente e quais as exceções, associadas a suas interfaces requeridas, o componente deve tratar. Além disso, a arquitetura deve especificar a interação entre os componentes e conectores em termos de suas interfaces providas e requeridas.

A especificação dos componentes em uma arquitetura é a base para que um desenvolvedor de componentes possa implementar componentes, ou para que um integrador possa identificar componentes reutilizáveis adequados para serem integrados ao sistema. Desta forma, a especificação da arquitetura tem um papel especialmente importante para a construção de novos sistemas de software confiáveis e baseados em componentes.

1.1 Motivação e Problema

A arquitetura de um sistema baseado em componentes fornece o contexto de utilização de componentes neste sistema. Os componentes a serem integrados no sistema podem ter sido implementados sem o conhecimento deste sistema.

Considere um exemplo simples onde a arquitetura de um sistema de software para reservas de livros de uma biblioteca especifica um componente com a funcionalidade de efetuar a reserva de um livro. A especificação deste componente deveria descrever suas interfaces providas e requeridas e as operações que o componente deve implementar, entre elas, a operação de efetuar uma reserva de um livro. Além disso, deveria especificar quais exceções podem ocorrer durante a operação de reserva e como o componente deve tratar estas exceções, e deveria especificar também quais exceções o componente deveria lançar para componentes clientes dele, que requisitam a reserva de um livro. Com base na especificação da arquitetura e do componente, um integrador poderia escolher um componente já existente para ser reutilizado no sistema, ou um desenvolvedor poderia implementar um novo componente com a funcionalidade de reserva de livro.

Alguns obstáculos poderiam surgir num cenário como este:

(1) a especificação excepcional do componente poderia não apresentar quais exceções ele deve lançar caso ocorra um erro na execução da operação de reserva de livro;

(2) a especificação excepcional poderia não apresentar quais exceções o componente deve tratar quando ocorrerem erros em operações de interfaces requeridas, necessárias para a reserva de um livro;

(3) a arquitetura poderia estar bem especificada e organizada, porém um componente pronto poderia não atender a especificação excepcional, como por exemplo: (i) não estando preparado para lançar as exceções especificadas nas operações de interfaces providas, (ii) não contendo os tratadores para as tipos de exceções especificadas em operações de interfaces requeridas, (iii) possuindo uma falha de projeto que levasse a um erro e gerasse uma condição excepcional não antecipada, o que levaria ao lançamento de uma exceção não declarada na especificação do sistema.

Os obstáculos apresentados acima demonstram dois tipos de problemas. Um deles está associado à má especificação excepcional de componentes em uma arquitetura. Outro problema está associado ao fato de que componentes desenvolvidos em um contexto podem ser reaproveitados em outros contextos com especificações diferentes. Os dois tipos de problemas podem comprometer a construção de um novo sistema de software confiável baseado em componentes. O primeiro tipo de problema dificulta que desenvolvedores implementem componentes novos, e que integradores identifiquem componentes reutilizáveis já existentes que atendam aos requisitos do sistema, incluindo os requisitos de confiabilidade. O segundo tipo de problema dificulta a integração de um componente já existente que atenda à especificação normal e à especificação excepcional.

1.2 Solução Proposta

Em um processo de desenvolvimento de sistemas de software baseados em componentes [Chessman+01], tem-se as fases de: (1) definição de requisitos, (2) especificação de componentes da arquitetura, (3) provisionamento de componentes, onde componentes já existentes podem ser identificados para reutilização ou novos componentes podem ser implementados, (4) integração dos componentes, onde os componentes implementados ou identificados são integrados para prover uma funcionalidade do sistema, (5) testes das funcionalidades do sistema e (6) instalação do sistema.

Observando os tipos de problemas identificados na subseção 1.1, percebe-se que existe a necessidade de uma solução centrada na arquitetura que oriente a construção de sistemas de software confiáveis e baseados em componentes. A solução deve ser centrada na arquitetura pois a arquitetura contém a especificação dos componentes e de suas interações que servirão de base para as fases de provisionamento e integração de componentes. Além disso, percebe-se que esta solução também deve apresentar orientações para as fases de provisionamento e integração para permitir que componentes já existentes possam ser integrados atendendo à especificação normal e excepcional.

O objetivo deste trabalho é propor uma solução para os problemas definindo um conjunto de atividades que, com base em requisitos de um sistema, orientem arquitetos na definição de uma arquitetura de software contendo a especificação normal e a especificação excepcional de seus

componentes. Este conjunto de atividades é definido em um **Método para Especificação de Componentes da arquitetura de um sistema e de seu comportamento Excepcional (MECE)**. Além disso, a solução proposta neste trabalho apresenta orientações para as fases de provisionamento e integração de componentes através da aplicação de **estratégias de estruturação arquitetural para tratamento de exceções** que promovem a adaptação e integração de um componente a um sistema atendendo à especificação excepcional.

Durante a fase de provisionamento pode-se optar por implementar um novo componente. É importante que esta implementação esteja em conformidade com a especificação do componente apresentada na arquitetura do sistema. Neste sentido, este trabalho orienta o uso do modelo de estruturação de componentes COSMOS [Silva03], que será apresentado na seção 2.7 (Fundamentos Teóricos) e utilizado no estudo de caso desta monografia.

1.2.1 Introdução ao Método MECE

O método MECE define atividades de definição de requisitos e especificação de uma arquitetura em camadas, contendo a especificação normal e excepcional de cada componente. Caso se opte por implementar um componente novo durante a fase de provisionamento, o método MECE apresenta orientações para implementação.

O método MECE é caracterizado pela integração entre a Metodologia de Definição de Comportamento Excepcional (MDCE) [Ferreira01] e o processo de desenvolvimento de software baseado em componentes UML Components [Cheesman+01]. Para garantir que uma arquitetura de um sistema baseado em componentes seja devidamente organizada e especifique o comportamento normal de componentes, o método MECE utiliza conceitos do processo UML Components. O processo UML Components é focado nas fases de definição de requisitos e especificação de componentes de uma arquitetura, além de apresentar orientações para provisionamento de componentes e integração dos mesmos para a implementação do sistema. Para garantir que o comportamento excepcional seja devidamente definido, o método MECE utiliza conceitos da metodologia MDCE, que orienta a definição do comportamento excepcional durante as fases de especificação de requisitos, projeto e implementação de sistemas de software baseado em componentes.

A integração acontece com a aplicação de conceitos de especificação de requisitos da metodologia MDCE durante a fase de definição de requisitos do processo UML Components, e

com a aplicação de conceitos de projeto arquitetural, propostos pela metodologia MDCE, durante a fase de especificação de componentes e arquitetura do processo UML Components. Caso se opte por implementar componentes durante a fase de provisionamento, o método MECE orienta que a fase de implementação proposta pela metodologia MDCE seja aplicada. A integração entre a metodologia MDCE e o processo UML Components é possível pois MDCE é uma metodologia genérica [Ferreira01] e UML Components é um processo de desenvolvimento de software. Um outro estudo de integração da metodologia MDCE com outro processo, o *Catalysis* [Desmond98], foi realizado em [Ferreira01].

Além de possuir atividades resultantes da integração entre a metodologia MDCE e o processo UML Components, o método MECE adicionalmente apresenta atividades associadas a especificação de componentes e arquitetura que orientam: (1) a identificação de exceções que os componentes devem lançar e tratar; (2) o uso de conectores para facilitar a integração a um sistema de componentes já existentes. Como exemplo de aplicação destas orientações, considere que um integrador precise integrar um componente C1 que atende à especificação normal, mas que não é capaz de tratar uma exceção especificada e que pode ser lançada por outro componente C2. O integrador poderia criar um conector entre C1 e C2 para traduzir esta exceção para uma exceção que fosse entendida por C1, sem ter que alterar o componente C1 nem o componente C2.

Com a aplicação do método MECE tem-se como artefato uma arquitetura com a especificação normal e excepcional de seus componentes. A especificação normal de um componente apresenta suas interfaces providas com suas operações, suas interfaces requeridas, e sua interação com outros componentes e conectores. A especificação excepcional apresenta quais exceções devem ser lançadas pelo componente nas operações de suas interfaces providas, e quais exceções devem ser tratadas devido a erros em operações de interfaces requeridas.

1.2.2 Estratégias de Estruturação Arquitetural para Tratamento de Exceções

Por melhor que uma arquitetura esteja especificada e organizada com seus componentes e conectores, pode ser que componentes reutilizáveis não atendam à especificações excepcionais, como discutido na subseção 1.1. Para lidar com este problema, a solução apresentada neste trabalho utiliza a proposta de estratégias de estruturação arquitetural para tratamento de exceções

definidas inicialmente em [Guerra+03] e evoluídas em [Guerra04], que são aplicadas durante a fase de provisionamento e integração.

Estas estratégias propõem responsabilidades para componentes reutilizáveis e conectores de modo que: (1) exceções não declaradas, associadas a condições excepcionais não antecipadas, não se propaguem livremente pelas camadas do sistema, devendo ser transformadas em exceções que estejam definidas em uma hierarquia de tipos abstratos de exceções conhecida pelo sistema, e (2) que se trate a incompatibilidade entre tipos de exceções de dois componentes diferentes, ou seja, caso um componente servidor lance uma exceção que um componente cliente não saiba tratar, deve-se transformar esta exceção em uma exceção conhecida pelo componente cliente.

Para que exceções não declaradas possam ser transformadas em exceções definidas na hierarquia de tipos de exceções, são necessárias adaptações de componentes reutilizáveis para que possam transformar exceções não declaradas em exceções definidas na hierarquia de tipos de exceções. Por outro lado, é necessária a definição de tratadores em componentes e conectores para que estes possam lidar com estas exceções definidas na hierarquia de tipos de exceções. Para tratar a incompatibilidade entre tipos de exceções, é necessário o uso de tratadores em componentes e conectores para transformar uma exceção lançada por um componente em uma nova exceção que outro componente saiba tratar. As estratégias intra-componente e inter-componente, definidas em [Guerra04], apresentam orientações para a realização destas adaptações e para definições de tratadores para os componentes e conectores.

1.3 Trabalhos Relacionados

Issarny e Banâtre [Issarny+01] apresentam uma abordagem de tratamento de exceções no nível de arquitetura especificando componentes, conectores, exceções de configuração e tratadores para estas exceções os quais atuam na reconfiguração da arquitetura do sistema que está sendo executado. Além disso, este trabalho apresenta um mapeamento da especificação da arquitetura de componentes para sua implementação, com uso de um ambiente chamado Aster. A reconfiguração do sistema durante sua execução pode levar à parada de determinados componentes envolvidos na reconfiguração. Observa-se que a reconfiguração de arquitetura não é tratada nesta monografia, mas pode ser estudada no futuro levando-se em consideração reuso e confiabilidade em sistemas de software baseados em componentes.

O trabalho de Dellarocas [Dellarocas97] apresenta a idéia de que um sistema deve conter um módulo de tratamento de exceções que presta serviço para os componentes do sistema, e que os componentes do sistema não devem conter meios sofisticados para tratamento de exceções. Segundo a idéia, o módulo de tratamento de exceções monitora os componentes e caso uma exceção seja lançada, o módulo a trata. O módulo de tratamento deve se basear em padrões de comportamentos normais de componentes para saber quando deve agir e tratar comportamentos excepcionais, ou seja, caso um componente não esteja respeitando um padrão de comportamento normal, o módulo tratador entra em ação e verifica como lidar com exceções que podem ter ocorrido. O módulo tratador possui um conjunto de regras para saber como lidar com as exceções ocorridas em componentes e para fazer com que os componentes retornem para estados adequados. Observa-se que se um componente que possui comportamento normal muito específico, e que pode gerar exceções ainda não aprendidas pelo conjunto de regras do módulo tratador, é integrado a um sistema proposto por [Dellarocas97], deve-se alterar o módulo tratador de exceção correndo-se o risco de interferir em tratamentos de outras condições excepcionais. De modo diferente, com o método MECE e o uso de estratégias de estruturação arquitetural para tratamento de exceções, esta monografia apresenta orientações para que os componentes sejam especificados com atividades normais e excepcionais.

O trabalho de Romanovsky [Romanovsky01] apresenta a idéia de envolver um componente com um *Wrapper*[1] que lida com as exceções que podem alcançar o componente, ou que podem se originar no próprio componente. Além disso, propõe o uso de *Dynamic Actions*[2] que são unidades de ações atômicas que envolvem uma cadeia de componentes. Em uma *Dynamic Action* um componente C1 chama uma operação de uma interface provida de um componente C2, que por sua vez chama uma operação de uma interface provida de C3, e assim por diante, para realizar uma operação do sistema. Caso ocorra uma exceção em uma *Dynamic Action* todos os componentes envolvidos atuam na recuperação do estado do sistema. Observa-se que com a proposta do *Wrapper* e de *Dynamic Actions,* Romanovsky está propondo estratégias de tratamento de exceções intra e inter-componente. A proposta de estruturação de tratamento de exceções no nível de arquitetura usada neste monografia apresenta estratégias para tratamento de exceções intra e inter-componente, mas em um contexto mais abrangente de reuso de componentes e uso de conectores.

---

[1] *Wrapper* é um invólucro criado para envolver um componente de software. Com este invólucro, pode-se adaptar as funcionalidades providas pelo componente envolvido.

[2] *Dynamic Actions* são ações dinâmicas realizadas por um cadeia de componentes de software.

O trabalho de Guerra [Guerra04] apresenta uma abordagem arquitetural com tolerância a falhas em sistemas de software baseados em componentes, com estudos sobre ambientes para desenvolvimento de software baseado em componentes, com referências a pesquisas realizadas sobre confiabilidade, estilos arquiteturais, componentes e conectores, e com as estratégias de estruturação arquitetural para tratamento de exceções, originalmente definida em [Guerra+03]. Esta monografia tem o foco específico em arquitetura com tratamento de exceções, enquanto o trabalho de [Guerra04] é mais abrangente.

A Tabela 1 apresenta os principais tópicos do escopo de cada trabalho relacionado a esta monografia. O 'X' indica que o trabalho relacionado, ou esta monografia, aplica ou lida com o item descrito em uma linha.

	[Issarny +01]	[Dellarocas 97]	[Romanovsky 01]	[Guerra 04]	Esta Monografia
Lida com Exceções no Nível Arquitetural.	X	X	X	X	X
Lida com Especificação de Componentes e Arquitetura visando Reuso e Confiabilidade.					X (MECE)
Apresenta Estratégias de Estruturação Arquitetural para Tratamento de Exceções.				X	
Utiliza Estratégias de Estruturação Arquitetural para Tratamento de Exceções.				X	X
Utiliza Tratamento de Exceções Inter e Intra-componente.			X	X	X
Apresenta Solução de Tratamento de Exceções de Configuração de Arquitetura.	X				
Utiliza Solução que Promove Conformidade entre Arquitetura e Implementação.	X			X	X
Define um Único Módulo Tratador de Exceção para Todo o Sistema.		X			
Lida com Ambiente de Desenvolvimento de Software Baseados em Componentes.				X	

Tabela 1 – Escopo de Trabalhos Relacionados a esta Monografia

1.4 Organização deste Documento

Esta monografia está dividida em capítulos organizados da seguinte forma:

Capítulo 2 – Fundamentos Teóricos: Como base para este trabalho, este capítulo apresenta os fundamentos teóricos sobre: desenvolvimento baseado em componentes, confiabilidade em sistemas de software, arquitetura de software, processo de desenvolvimento de software baseado em componentes como o UML Components, a Metodologia de Definição de Comportamento Excepcional (MDCE), estratégias de estruturação arquitetural para tratamento de exceções, e um modelo para estruturação de componentes (COSMOS).

Capítulo 3 – O Método MECE: Este capítulo apresenta um método para a especificação de componentes da arquitetura de um sistema e de seu comportamento excepcional (MECE) baseado na integração da metodologia MDCE com o processo UML Components. O capítulo mostra como acontece a aplicação de conceitos de especificação de requisitos da metodologia MDCE durante a fase de definição de requisitos do UML Components, e a aplicação de conceitos de projeto arquitetural, propostos por MDCE, durante a fase de especificação de componentes e arquitetura do UML Components. Além disso, o capítulo apresenta orientações do método MECE para definição de exceções que componentes devem lançar e tratar, e orientações de uso de conectores.

Capítulo 4 – Estudo de Caso com Sistema Real: Este capítulo apresenta um estudo de caso da solução proposta neste trabalho, através da aplicação do método MECE e das estratégias de estruturação arquitetural para tratamento de exceções em um sistema de software real, chamado Telestrada. Mais especificamente no subsistema de gerência de reclamações via *Web*, deste sistema. Este estudo envolveu o uso de tecnologias amplamente utilizadas em mercado como: IBM WebSphere Server [IBMWeb], IBM WebSphere Studio, banco de dados Oracle, UML Plug In do Eclipse [OMondoWeb], J2SE [J2SEWeb], J2EE [J2EEWeb] e Struts [JakartaWeb]. Por fim, este capítulo apresenta os resultados do estudo de caso.

Capítulo 5 – Contribuições, Trabalhos Futuros e Conclusões: Este capítulo apresenta as contribuições do trabalho realizado, possíveis trabalhos futuros e as conclusões.

Referências Bibliográficas: Apresenta referências utilizadas para estudos.

Apêndice A – Casos de Uso do Telestrada: Apresenta casos de uso do estudo de caso do Telestrada.

Apêndice B – Diagramas da Arquitetura do Telestrada: Apresenta diagramas de componentes, e suas interfaces, do estudo de caso do Telestrada.

Apêndice C – Telas do Telestrada: Apresenta as páginas da aplicação *Web* do Telestrada.

Apêndice D - *Scripts* de Criação de Tabelas de Banco de Dados: Para o estudo de caso, teve-se a preparação de um ambiente de desenvolvimento de software, que incluiu a configuração de um servidor de banco de dados e a criação da base de dados. Este apêndice apresenta o *script* para criação das tabelas de base de dados.

Capítulo 2
Fundamentos Teóricos

Este capítulo apresenta fundamentos teóricos sobre: práticas de desenvolvimento de software baseado em componentes (subseção 2.1), confiabilidade em sistemas de software (subseção 2.2), arquitetura de software (subseção 2.3), o processo de desenvolvimento de sistemas de software baseados em componentes chamado UML Components (subseção 2.4), e sobre a Metodologia de Definição do Comportamento Excepcional (MDCE) para sistemas de software baseados em componentes (subseção 2.5). Estes fundamentos formam a base conceitual para a definição do método MECE.

Além do método MECE, a solução arquitetural apresentada nesta monografia orienta o uso de estratégias de estruturação arquitetural para tratamento de exceções, definidas em [Guerra04]. Este capítulo também apresenta uma introdução a estas estratégias (subseção 2.6).

No estudo de caso que envolveu a aplicação da solução proposta neste trabalho, utilizou-se o modelo de estruturação de componente COSMOS para a implementação de um sistema real. Os fundamentos sobre o COSMOS são apresentados neste capítulo (subseção 2.7) para promover o entendimento do estudo de caso.

2.1 Desenvolvimento Baseado em Componentes

Desenvolvimento baseado em componentes é uma prática de desenvolvimento de sistemas de software realizada através de integração de componentes de software já existentes. Vários autores adotam diferentes definições de componentes.

"Um componente é um pacote desenvolvido e testado separadamente, distribuído como uma unidade que pode ser integrada com outros componentes para construir algo com maior funcionalidade" [Szyperski97]. Esta é a definição adotada nesta monografia.

"Componentes são módulos ou estruturas de dados, desenvolvidos e compilados separadamente, e unidos para formar aplicações que interagem através de interfaces" [Chambers96].

"Um componente é uma parte física e substituível de um sistema, que empacota implementação e oferece a realização de um conjunto de interfaces" [Uml99].

Um componente deve ter claramente a divisão entre suas interfaces e sua implementação. Um componente apresenta suas interfaces providas, com a descrição de suas operações, e suas interfaces requeridas. Uma **interface provida** identifica um ponto de acesso aos serviços que o componente provê para o ambiente onde é utilizado. Uma **interface requerida** identifica um ponto de acesso aos serviços que o componente requer do ambiente. Em sua implementação, um componente deve conter as classes ou funções capazes de executar as operações providas em suas interfaces. O propósito da separação de um componente em duas partes (interfaces e implementação) é propiciar a flexibilidade para que o componente possa ser conectado a outros componentes e substituído por outros.

A especificação de um componente é importante para que integradores possam identificar componentes já implementados para que sejam reutilizados em sistemas, ou para que desenvolvedores possam implementar um componente que seja reutilizável. Ao reutilizar um componente, um integrador pode ou não ter acesso ao código fonte deste componente.

A especificação de um componente ocorre durante a definição de uma arquitetura de um sistema, ao qual o componente implementado será integrado.

2.2 Confiabilidade em Sistemas de Software

Quando um sistema de software deixa de oferecer as funcionalidades previstas em seus serviços, ou o faz em desacordo com suas especificações, diz-se que ocorreu um **defeito** no sistema. A ocorrência de um defeito de software compromete a confiabilidade de um sistema.

Confiabilidade é uma propriedade de um sistema que possibilita aos seus usuários confiarem, justificadamente, nos serviços que ele oferece [Lee+90]. A confiabilidade é expressa por atributos não-funcionais. Um destes atributos é a disponibilidade (*availability*) de um sistema, que expressa a garantia de que o sistema estará funcionando e apto a fornecer seus serviços no momento em que um cliente precisar. Outro atributo é a garantia de que o sistema estará funcionando pelo tempo exigido por uma operação (*reliability*). Um terceiro atributo é a segurança no uso (*security*), que expressa a garantia de que o sistema é capaz de preservar a integridade de informações mesmo na presença de um defeito. Como um quarto exemplo de atributo, tem-se a

segurança no funcionamento (*safety*) que expressa a garantia de que o sistema é capaz de evitar conseqüências graves na presença de defeito [Rubira+03].

2.2.1 Conceitos de Erro e Falha

O comportamento de um sistema é função de seu estado interno e das entradas que o sistema recebe. A ocorrência de um defeito pode ser associada a um estado interno errôneo do sistema, sob determinadas entradas. Este estado errôneo é chamado de **erro**. Considere um exemplo onde uma pessoa utiliza um servidor da Internet para comprar ações de empresas. Caso esta pessoa efetue compra de ações de uma empresa A, e receba uma confirmação de compra de ações de uma empresa B, tem-se um defeito. Este defeito pode ter sido provocado por um erro no sistema, quando o mesmo realizou uma consulta ao banco de dados para encontrar as informações de empresa. Para que um erro ocorra é necessária uma causa, a qual é chamada de **falha** [Lee+90]. Considerando o exemplo de compra de ações pela Internet, pode ter ocorrido uma falha associada a uma queda na conexão com o servidor de banco de dados durante a consulta dos dados da empresa.

Normalmente um sistema de software funciona interagindo com diversos agentes externos como usuários, dispositivos de entrada e saída, redes de comunicação, diferentes sistemas, entre outros. Durante o projeto e desenvolvimento de um sistema de software, cria-se modelos destes agentes e padrões de interações do sistema com os agentes. Existe uma distância entre estes modelos e o ambiente real onde o sistema de software será utilizado. Quanto maior esta distância, maior a probabilidade da existência de eventos que levem a falhas no sistema [Guerra04]. Um fator que pode fazer esta distância aumentar, é a complexidade de um sistema. A complexidade pode fazer com que projetistas e desenvolvedores introduzam falhas no sistema que levem o sistema para um estado errôneo. Quando se trata de sistemas de software baseados em componentes, pode-se ter uma situação de reutilização de componentes já existentes que contenham falhas de projeto. Uma falha em um componente de software pode gerar um erro que pode ser propagado para outros componentes de um sistema, causando novas falhas.

Durante o projeto e o desenvolvimento de sistemas confiáveis pode-se estabelecer hipóteses a respeito da ocorrência de falhas as quais o sistema estará sujeito. Durante o projeto e desenvolvimento de um sistema, projetistas e desenvolvedores criam hipóteses sobre falhas que podem ocorrer e causar erros no futuro sistema. Com base em uma hipótese de falha, uma pessoa

pode definir uma forma de prevenir esta falha ou de se tolerar esta falha quando a mesma ocorrer no sistema.

2.2.2 Conceitos de Prevenção e Tolerância a Falhas

Dado que falhas são as causas de ocorrências de defeitos, a abordagem mais intuitiva para lidar com falhas é se prevenir para que elas não ocorram. Para tal, deve-se identificar os tipos de falhas as quais um sistema está exposto e adotar medidas preventivas contra seus agentes causadores. Falhas de projeto e de implementação, por exemplo, podem ser prevenidas se pessoas qualificadas projetarem um sistema, se desenvolvedores experientes implementarem o sistema e se testes forem adequadamente planejados e executados também durante interações iniciais de um projeto, com testadores experientes.

Quando se trata de sistemas de software baseados em componentes, pode-se prevenir falhas com estudos de componentes a serem integrados no sistema, através de verificações se os componentes atendem às especificações do sistema que se está construindo.

Embora a prevenção de falhas seja essencial para a confiabilidade de um sistema, é muito difícil que se elimine a possibilidade de que uma falha ocorra em um sistema. Desta forma, é necessário que exista uma abordagem para **tolerância a falhas**. Nesta abordagem, mecanismos de tolerância a falhas são introduzidos em um sistema para lidar com a ocorrência de erros. Um mecanismo de tolerância a falhas é iniciado quando se detecta um erro no sistema. Quando um erro é detectado, o sistema deve ser recuperado e levado para um estado livre de erros. Existem duas abordagens de recuperação: por retrocesso (*backward error recovery*) e por avanço (*forward error recovery*). A recuperação por retrocesso procura conduzir o sistema para um estado correto, anterior a falha. A recuperação por avanço procura conduzir o sistema para um novo estado correto.

Ao levar o sistema para um estado livre de erros, um mecanismo de tolerância a falhas pode tentar descobrir qual foi a falha que ocasionou o erro, e proteger o sistema contra uma nova falha do mesmo tipo. Em um sistema de software baseado em componentes por exemplo, caso uma versão recente de um componente apresente uma falha, o sistema pode deixar de usar esta versão do componente para usar uma versão mais antiga. Além disso, o mecanismo de tolerância a falhas

pode enviar uma mensagem para o operador do sistema informando que é necessária a substituição do componente que apresentou a falha.

Uma implementação de mecanismo de tolerância a falhas, com abordagem de recuperação por avanço, é o **tratamento de exceções**. Quando um erro é detectado, uma **condição excepcional** é caracterizada e é sinalizada através do lançamento de uma **exceção**. Este lançamento leva a um desvio do fluxo normal de instruções para que a condição excepcional seja tratada [SunWeb]. Durante o tratamento da condição excepcional o tratador pode levar o sistema a um estado livre de erros e fazer com que o sistema volte a seu fluxo normal de instruções.

Condições excepcionais podem ser previstas na especificação excepcional de um sistema, sendo assim chamadas de **condições excepcionais antecipadas**. Estas condições excepcionais antecipadas são sinalizadas num sistema, por meio de exceções definidas com base em tipos de **exceções declaradas** na especificação do sistema. Nem todas as condições excepcionais podem ser previstas e antecipadas em uma especificação de um sistema. Normalmente as **condições excepcionais não antecipadas** estão associadas à ausência de hipóteses de falhas para um sistema. As condições excepcionais não antecipadas são geralmente sinalizadas por **exceções não declaradas** na especificação de um sistema.

2.3 Arquitetura de Software

Como comentado na introdução desta monografia, a **arquitetura de um sistema de software** define o mesmo em função de seus componentes, suas interações e suas funcionalidades [Sommerville01], além de apresentar os conectores e as interações entre componentes e conectores. Uma arquitetura de software apresenta diferentes visões [Kruchten95] de um sistema, assim como a arquitetura de uma casa possui a planta do encanamento, da parte elétrica e da divisão dos cômodos. Diferentes visões são necessárias para especificar diferentes estruturas de sistemas de software. Uma visão pode, por exemplo, ser necessária para especificar a comunicação entre processos e *threads* [Silberschatz+01] de um sistema, outra visão pode especificar fluxos de dados.

Uma arquitetura bem organizada, com suas diferentes visões, procura promover o comum entendimento de um sistema por parte de arquitetos, desenvolvedores, integradores, patrocinadores, testadores e usuários finais. Além disso, a arquitetura pode influenciar na divisão

de uma organização de software para a construção de um sistema, onde cada divisão da organização fica responsável pela implementação ou integração de um componente, ou de uma funcionalidade que envolve mais do que um componente. Uma arquitetura bem organizada promove uma melhor comunicação entre as divisões de uma organização de software.

Uma arquitetura de um sistema de software possui propriedades arquiteturais, derivadas dos requisitos de software, como: modificabilidade, reusabilidade, desempenho, escalabilidade, confiabilidade, entre outros. Uma propriedade arquitetural deve persistir por todo o desenvolvimento do sistema. Para garantir que uma determinada propriedade possa persistir, utiliza-se um estilo arquitetural adequado para descrever a propriedade [Monroe+97]. Um **estilo arquitetural** descreve características de uma classe de arquiteturas de software, através de tipos de componentes, conectores e regras de formação da topologia da arquitetura [Bass+02]. Um exemplo de estilo arquitetural é o estilo em camadas, que pode promover a modificabilidade e reusabilidade. A camada de mais alto nível é a que faz a interface com usuário do sistema, e a de mais baixo nível é responsável pela persistência de informações do sistema e comunicação com sistema operacional. Entre estas duas camadas podem existir outras intermediárias.

Para construir um sistema de software confiável e baseado em componentes que promova o reuso de componentes, deve-se definir uma arquitetura de software bem organizada e baseada em um estilo arquitetural que promova a reusabilidade e confiabilidade. Esta arquitetura deve conter a especificação de componentes, conectores e suas interações. A especificação de um componente deve conter suas interfaces providas e requeridas, as operações que deve implementar, quais condições excepcionais pode gerar e quais condições excepcionais pode tratar, sendo que estas últimas condições excepcionais podem ser geradas no próprio componente ou em outros componentes do sistema. A especificação de um conector deve definir que componentes ele interliga. A interação entre componentes e conectores define a **configuração de componentes** de uma arquitetura de software. A configuração é a base para que o sistema estabeleça as ligações entre componentes e conectores, ao ser inicializado.

Ao se definir uma arquitetura, tem-se uma base para iniciar o desenvolvimento de componentes ou para se integrar componentes prontos. A definição da arquitetura de um sistema depende dos requisitos do sistema a ser construído. Com base nos requisitos, pode-se iniciar um

processo de desenvolvimento de sistemas de software baseado em componentes, onde uma de suas fases é definição da arquitetura com especificação de componentes.

2.4 Processo de Desenvolvimento de Sistemas de Software Baseados em Componentes

Como dito na subseção 1.2, um **processo de desenvolvimento de sistemas de software baseados em componentes** geralmente possui fases de: definição de requisitos, definição arquitetural com especificação de componentes, provisionamento de componentes, integração de componentes, testes do sistema e instalação do sistema. No provisionamento ocorre o desenvolvimento dos componentes ou a reutilização de componentes já existentes, com base nas especificações normal e excepcional dos componentes. Na fase de integração, os componentes desenvolvidos e reutilizados são integrados. Antes da integração, pode-se testar cada componente separadamente e depois da integração, testa-se as funcionalidades do sistema com os componentes integrados.

Um exemplo de processo é o processo UML Components [Chessman+01] que define fases para de definição de requisitos, especificação de componentes e arquitetura, e orientações para provisionamento dos componentes e integração dos mesmos, como pode ser visto na Figura 1. O principal foco do processo UML Components está nas fases de definição de requisitos e especificação de componentes.

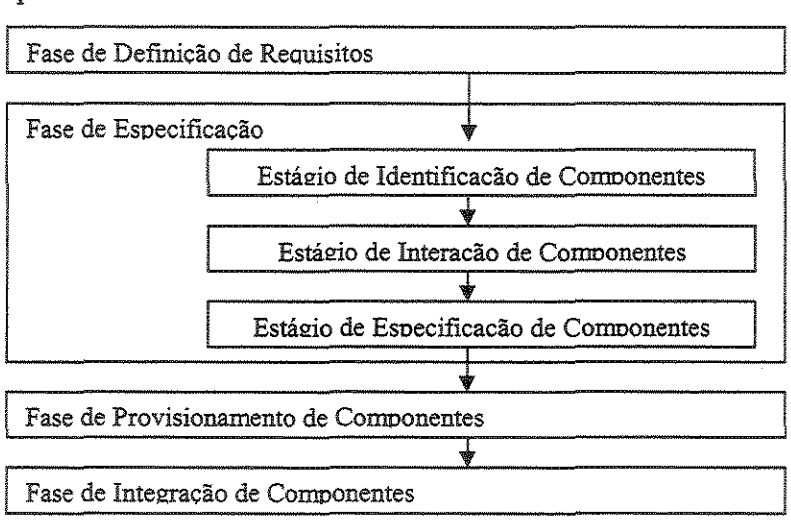

Figura 1 – Fases do Processo UML Components

2.4.1 Fase de Definição de Requisitos

A fase de definição de requisitos tem como início a definição do chamado **processo de negócio** do sistema a ser implementado. Num exemplo de um sistema para reserva de livros de uma biblioteca, um processo de negócio descreveria as atividades que devem ser realizadas pelo bibliotecário e pelo sistema para que uma reserva seja feita. O processo UML Components, propõe que um diagrama de atividades seja utilizado como base para a descrição do processo. A Figura 2 apresenta parte do processo de negócio para um sistema de reserva de livros de uma biblioteca, com atividades desempenhadas pelo sistema.

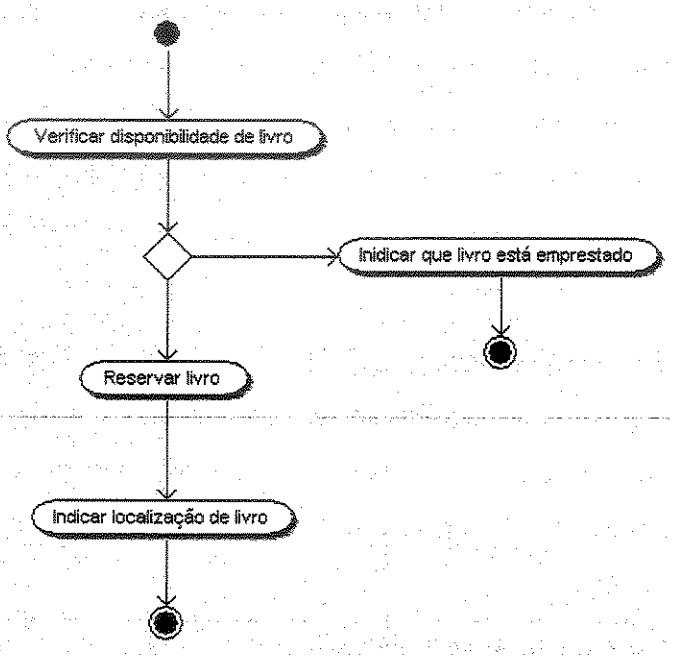

Figura 2 – Exemplo de Atividades de um Processo de Negócio

Com base na descrição do processo de negócio, define-se o modelo conceitual de negócio e os casos de uso do sistema. O **modelo conceitual de negócio** apresenta as relações entre as entidades pertencentes ao domínio do negócio do sistema a ser construído. A Figura 3 apresenta um exemplo de parte de um modelo conceitual de negócio para um sistema de reserva de livros de uma biblioteca, onde: uma biblioteca possui diversos usuários e livros, um usuário pode reservar vários livros, um livro possui uma localização e uma rede de bibliotecas possui várias bibliotecas.

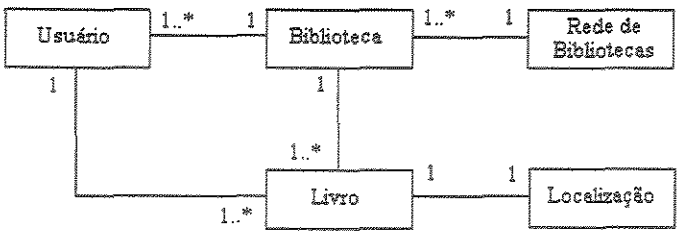

Figura 3 – Exemplo de Modelo Conceitual de Negócio

Um **caso de uso** [CockburnWeb] especifica requisitos de sistema descrevendo interações entre o usuário e o sistema. Ele deve ser apresentado com um identificador, com a definição de um ator, de um objetivo e com a seqüência de passos que descrevem o cenário principal (também chamado de cenário primário, ou de sucesso). Além disso, um caso de uso pode ter extensões (também chamadas de cenários alternativos), cujos passos devem ser apresentados logo após a lista de passos do cenário principal. Neste trabalho, um cenário alternativo não é considerado um cenário excepcional. No caso do exemplo do sistema de reserva de livros de uma biblioteca, pode-se ter o seguinte caso de uso:

Nome: Reserva de Livro.

Ator que o inicia: Bibliotecário.

Objetivo: Reservar um livro.

Cenário Principal:

1. Usuário de biblioteca pede para que um livro seja reservado.
2. Bibliotecário acessa sistema.
3. Sistema valida usuário.
4. Sistema realiza a reserva.
5. Bibliotecário mostra para usuário o local onde livro se encontra.

Extensão (caso o livro esteja reservado para outro usuário, os passos 4 e 5 são diferentes):

4. Sistema indica que livro não está disponível.
5. Bibliotecário avisa usuário que livro não está disponível.

Após definição de casos de uso em função da descrição do processo de negócio do sistema, deve-se verificar se existe a necessidade de se criar novos casos de uso para registro e remoção de entidades pertencentes ao modelo conceitual de negócio. No exemplo do sistema de reserva de livros, percebe-se que deve-se criar um caso de uso para registro de livros e outro caso de uso para remoção de livros.

A Figura 4 apresenta as atividades da fase de definição de requisitos do processo UML Components. Inicia-se assim a definição do comportamento normal do sistema.

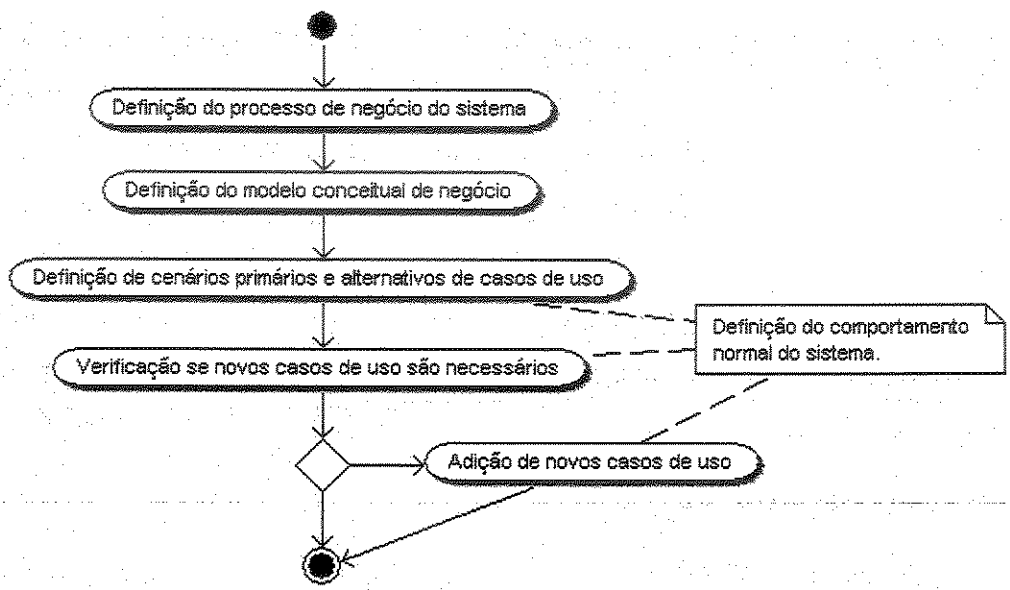

Figura 4 – Atividades de Definição de Requisitos do Processo UML Components

2.4.2 Fase de Especificação

A fase de especificação de componentes do processo UML Components tem como entrada os casos de uso e o modelo conceitual de negócio, elaborados durante a fase de definição de requisitos. A saída da fase de especificação é uma arquitetura em camadas com a especificação de seus componentes. A arquitetura de camadas, referenciada pelo processo UML Components, é formada por uma camada de interface com usuário que é responsável pela apresentação e coleta de informações, seguida por uma camada de diálogo com o usuário que é responsável pelo controle de sessões de usuários que acessam o sistema, que por sua vez é seguida de uma camada de serviços de sistema e por uma camada de serviços de negócios. A camada de serviços de negócios pode utilizar uma camada de mais baixo nível que gerencia a persistência de dados do sistema e se

comunica com sistema operacional. A Figura 5 apresenta estas camadas. O processo UML Components promove a especificação de componentes apenas para as camadas de sistema e negócios, destacadas na figura.

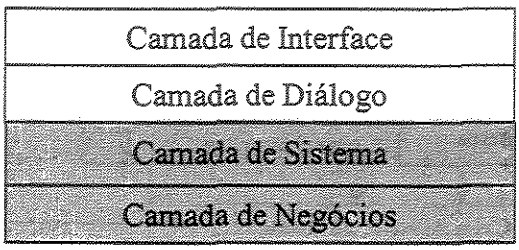

Figura 5 – Camadas de uma Arquitetura Considerada pelo Processo UML Components

A fase de especificação do processo UML Components possui três estágios: **identificação de componentes**, **interação de componentes** e **especificação propriamente dita de componentes**, como pode ser visto na Figura 6 [Chessman+01]. As atividades de cada estágio, suas entradas e saídas são apresentados nas subseções seguintes. A especificação de arquitetura com componentes é utilizada para o provisionamento de componentes do sistema. Após o provisionamento, inicia-se a fase de integração dos componentes [Chessman+01].

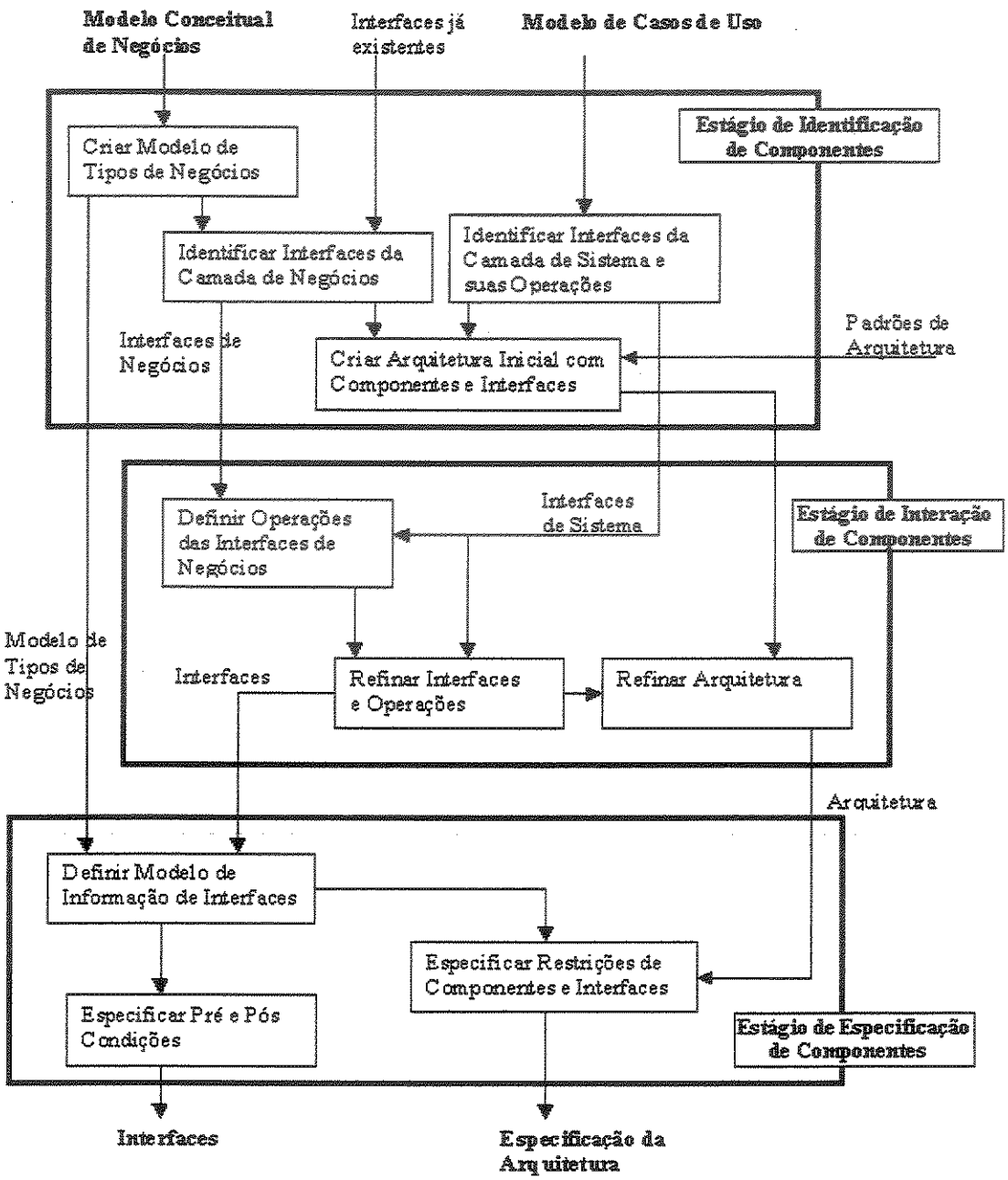

Figura 6 – *Workflow* de Especificação de Componentes do Processo UML Components

Estágio de Identificação de Componentes

A Figura 7 apresenta um diagrama de atividades deste estágio que tem como entrada os casos de uso e o modelo conceitual de negócio.

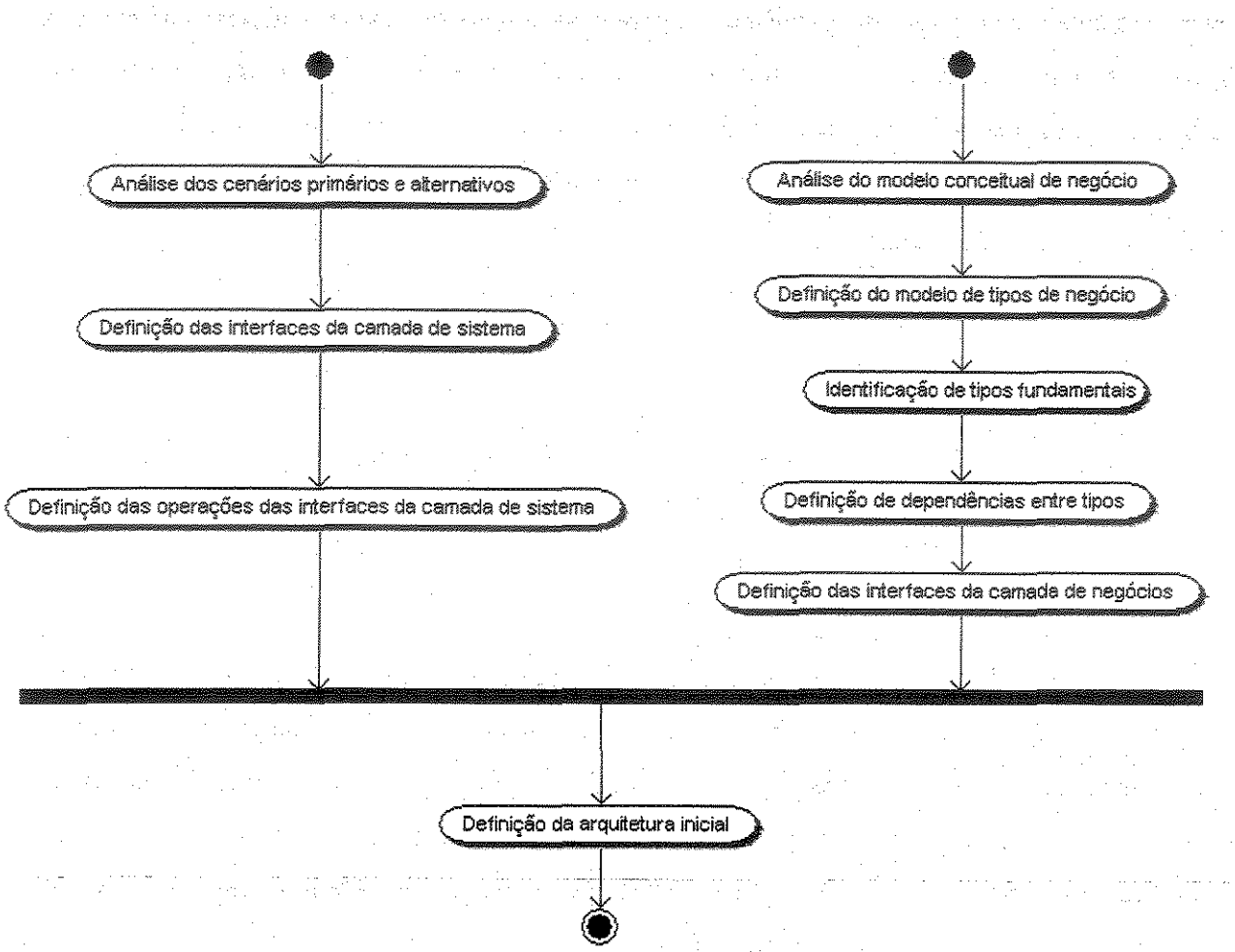

Figura 7 – Atividades do Estágio de Identificação de Componentes

Cada caso de uso é mapeado para um **tipo de interface** (*interface type*) da camada de sistema. Um tipo de interface será realizado por uma interface de um componente quando o mesmo for implementado. Para cada passo de um caso de uso, de seu cenário primário e alternativos, verifica-se se a ação do passo representa uma responsabilidade do sistema. Se sim, o passo é mapeado para uma ou mais operações do tipo de interface definida para o caso de uso.

Como exemplo, no caso do sistema de reserva de livro de uma biblioteca, teria-se a definição de uma interface IReservaLivro, para o caso de uso "Reserva de Livro", com um método reservaLivro(identificadorDoUsuário, identificadorDoLivro), para o passo 4 do cenário primário.

Além da identificação dos tipos de interfaces e operações da camada de sistema, neste estágio também se identifica os tipos de interfaces da camada de negócios. Para tal, analisa-se e evolui-se o modelo conceitual de negócio para o modelo de tipos de negócio. Esta evolução se dá

pela remoção de entidades que não precisam ser gerenciadas pelo sistema de software. O exemplo da Figura 8 apresenta parte do modelo de tipos de negócio para o sistema de reserva de livros de uma biblioteca. Considera-se que o sistema que roda em uma biblioteca não fará comunicação com outro sistema de outra biblioteca, portanto a entidade "Rede de Bibliotecas" é removida.

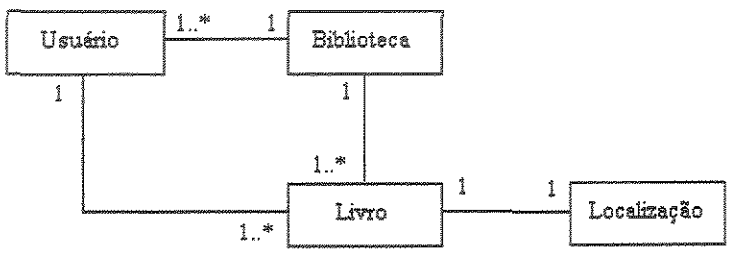

Figura 8 – Exemplo de Modelo de Tipos de Negócio

Tendo o modelo de tipos de negócio, identificam-se os tipos fundamentais do negócio (*core types*). Os **tipos fundamentais** são as principais informações gerenciadas pelo sistema, geralmente não dependem de outros tipos para existir no sistema. Para cada tipo fundamental identificado, define-se um tipo de interface da camada de negócios para se responsabilizar pela gerência deste tipo fundamental. O processo UML Components convenciona que um tipo de interface da camada de negócios deve ter o sufixo *Mgt*.

Tipos que não são considerados fundamentais, dependem de um tipo fundamental para existir. Na atividade de identificação de dependências de tipos, define-se qual tipo é dependente de qual tipo. Tipos dependentes são gerenciados pelas interfaces, definidas para o tipo fundamental do qual eles dependem. A Figura 9 apresenta a relação de dependência e responsabilidade através do uso de um losango preenchido.

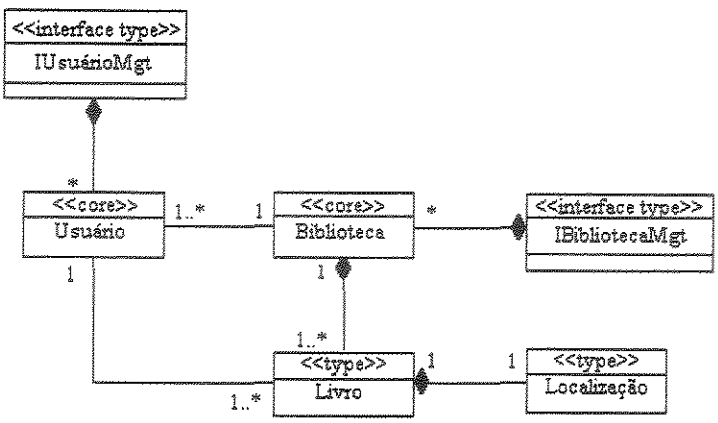

Figura 9 – Exemplo de Diagrama de Responsabilidades

Nesta figura, IUsuárioMgt e IBibliotecaMgt são tipos de interface que deverão ser realizados por interfaces de componentes da camada de negócios. Estas interfaces serão responsáveis pelo gerenciamento de informações de usuários e bibliotecas, respectivamente. Os tipos Usuário e Biblioteca são fundamentais, enquanto Livro e Localização são dependentes e serão gerenciados pela interface IBibliotecaMgt. O processo UML Components define estereótipos para tipo de interface <<interface type>>, tipo fundamental <<core>> e tipo dependente <<type>>. Além disso, os atributos dos tipos devem ser mostrados em diagramas. Isto não foi feito aqui para efeito de simplificação.

Tendo definido os tipos de interfaces das camadas de sistema e negócios, inicia-se a atividade de construção da arquitetura. Para tal, define-se a identificação inicial dos componentes que provêm estas interfaces tanto na camada de sistema quanto na camada de negócios. Além disso, define-se a primeira versão da arquitetura descrevendo qual interface da camada de negócios é requerida por qual componente da camada de sistema. Para cada interface da camada de negócios, geralmente, define-se um componente que a oferece. Convenciona-se que este componente tem o sufixo *Mgr*. Para um conjunto de interfaces da camada de sistema, define-se um ou mais componentes para oferecê-las, dependendo das características das interfaces e do quão próximas elas estão em termos de funcionalidade. A Figura 10 apresenta uma arquitetura inicial para a reserva de livro de uma biblioteca.

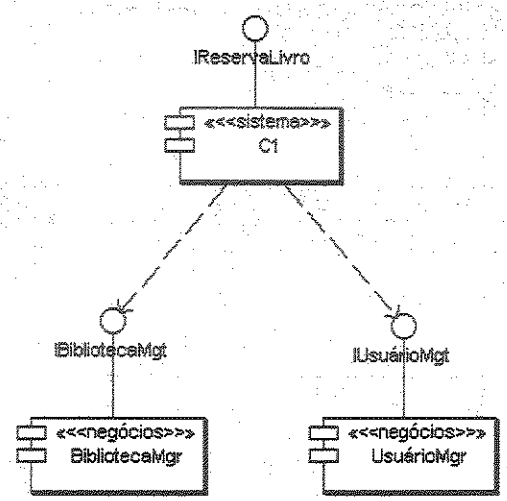

Figura 10 – Exemplo de Arquitetura Inicial com Base em UML Components

Estágio de Interação de Componentes

A Figura 11 apresenta um diagrama de atividades deste estágio.

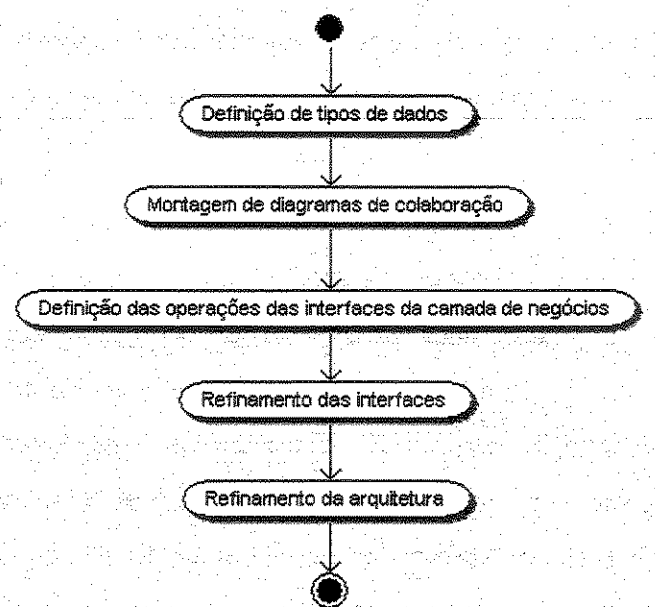

Figura 11 – Atividades do Estágio de Interação de Componentes

A primeira atividade deste estágio é identificar os **tipos de dados** do sistema, que são estruturas de informação usadas para definição de parâmetros e retornos de operações de tipos de

interfaces de componentes. Por exemplo, em uma operação de reserva de livro poderia-se retornar um tipo Livro contendo seus atributos e incluindo sua localização.

Neste estágio de interação de componentes, define-se as operações dos tipos de interfaces da camada de negócios, incluindo seus parâmetros e retornos. Para tal, inicia-se um diagrama de colaboração para cada operação de um tipo de interface da camada de sistema. Como um componente da camada de sistema requer interfaces da camada de negócios, cada diagrama de colaboração que começa com uma chamada de operação da camada de sistema, deve mostrar a chamada para uma operação da camada de negócios.

A Figura 12 apresenta um exemplo onde o componente C1 oferece a interface IReservaLivro. Este componente requer duas operações de interfaces da camada de negócios. Quando o componente C1 recebe uma requisição para reservar um livro para um usuário, ele utiliza a operação validaUsuário(identificadorDoUsuário) da interface IUsuárioMgt, para validar o usuário. Caso o usuário seja válido, o componente C1 reserva o livro utilizando a operação reservaLivro(identificadorDoUsuário, identificadorDoLivro) da interface IBibliotecaMgt. Identifica-se assim que operações da camada de negócios seriam necessárias para que uma operação da camada de sistema fosse executada.

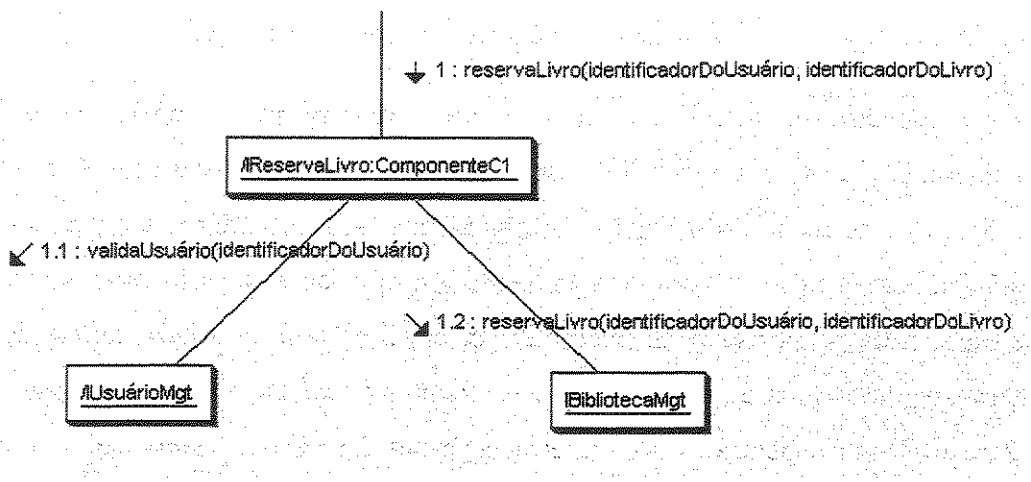

Figura 12 – Diagrama de Colaboração para Identificação de Operações da Camada de Negócios

Após a identificação das operações das interfaces de negócios, refina-se as interfaces e operações de modo a verificar se as devidas responsabilidades estão com as devidas interfaces, ou seja, se uma operação que está definida em uma interface não deveria ser definida em outra interface. Caso haja refinamento de interface pode haver refinamento de arquitetura, definida no

estágio anterior. Um exemplo seria a criação de uma nova interface que define operações comuns a vários componentes, o que poderia levar a definição de um componente novo para oferecer apenas esta interface.

Estágio de Especificação de Componentes

A Figura 13 apresenta as atividades deste estágio de especificação de componentes.

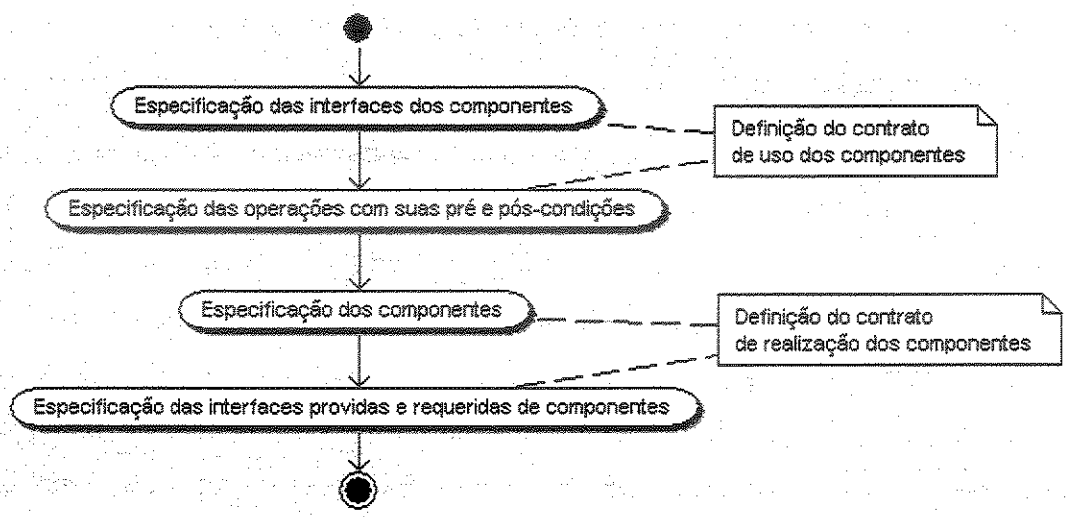

Figura 13 – Atividades do Estágio de Especificação de Componentes

Neste estágio especifica-se interfaces e componentes com base nas informações definidas nos estágios anteriores. Define-se os contratos de uso e de realização de componentes. O **contrato de uso** de um componente está associado à especificação de interfaces providas desse componente. É importante para que clientes de um componente conheçam devidamente as interfaces providas por ele. Deve-se especificar as operações de cada interface em função de seus parâmetros, de seus retornos, das alterações de estados do componente em função de suas chamadas, e de suas pré e pós-condições. Para auxiliar a especificação de interfaces, o processo UML Components propõe o uso de modelos de informação de interface, que contém os tipos de dados necessários para se definir os parâmetros, retornos, pré e pós-condições das operações.

O **contrato de realização** está associado à especificação de um componente para que instâncias do mesmo o realizem devidamente. É a base para que desenvolvedores implementem instâncias dos componentes ou para que integradores possam identificar componentes já existentes

2.5.1 Fase de Especificação de Requisitos

A entrada para esta fase é a descrição de requisitos do sistema e o artefato resultante é um conjunto de cenários de casos de uso e diagramas que detalham os cenários. A metodologia MDCE propõe a descrição completa de cada caso de uso com as especificações separadas de seus comportamentos normal e excepcional.

A especificação do comportamento normal é composto por um cenário de sucesso, ou primário, e vários cenários alternativos, todos com suas pré e pós-condições. A especificação do comportamento excepcional é composto por cenários excepcionais, que podem ser cenário de falha ou cenário recuperável [Ferreira01]. Neste trabalho, o cenário de falha definido pela metodologia MDCE será referenciado como cenário irrecuperável.

Um **cenário irrecuperável** é caracterizado por uma situação inaceitável para o sistema continuar funcionando. Um **cenário recuperável** é um desvio de um curso de execução para tratamento de atividades mal sucedidas, visando trazer o sistema a um estado normal para continuar em suas tarefas.

Considere um exemplo baseado na Figura 15, de uma aplicação *Web* para realização de transações financeiras na bolsa de valores. Esta aplicação é caracterizada por três servidores: SA, S1 e S2. O servidor SA implementa funcionalidades da camada de interface e diálogo com usuário (Ver definições de camadas da Figura 5), sendo responsável por receber requisições de transações financeiras de usuários e passar estas requisições para um servidor de aplicação S1 ou para um servidor de aplicação S2, de modo balanceado. Os servidores S1 e S2 seriam responsáveis pelas camadas de sistema, negócios (Ver definições de camadas da Figura 5) e de acesso a banco de dados. Caso o servidor S1 deixe de funcionar, o servidor SA passa requisições apenas para S2, até que o servidor S1 retorne ao seu funcionamento (e vice-versa).

para serem reutilizados. A especificação de um componente está baseada nas descrições das interfaces providas e requeridas por este componente, além das definições das restrições das operações de suas interfaces. Deve-se apresentar na especificação o que ocorre quando uma operação de um componente é requisitada, ou seja, que outras operações de interfaces requeridas são utilizadas para o componente realizar a operação requisitada.

2.4.3 Provisionamento e Integração

Tem-se como saída do *workflow* da Figura 6 as interfaces providas pelos componentes das camadas de sistema e negócios, além da definição da arquitetura. A partir deste momento, pode-se implementar estes componentes ou identificar componentes reutilizáveis que satisfaçam as especificações, isto caracteriza a fase de provisionamento do processo. Uma vez definidos os componentes que farão parte de um sistema, pode-se integrá-los para que se possa utilizar o sistema, definindo assim a fase de Integração.

2.5 Metodologia de Definição do Comportamento Excepcional (MDCE)

MDCE é uma metodologia de definição do comportamento excepcional de um sistema aplicada as fases de especificação de requisitos, projeto e implementação de sistemas de software baseado em componentes [Ferreira01], como pode ser visto na Figura 14. A fase de especificação de requisitos é composta de três etapas que são descritas na subseção seguinte.

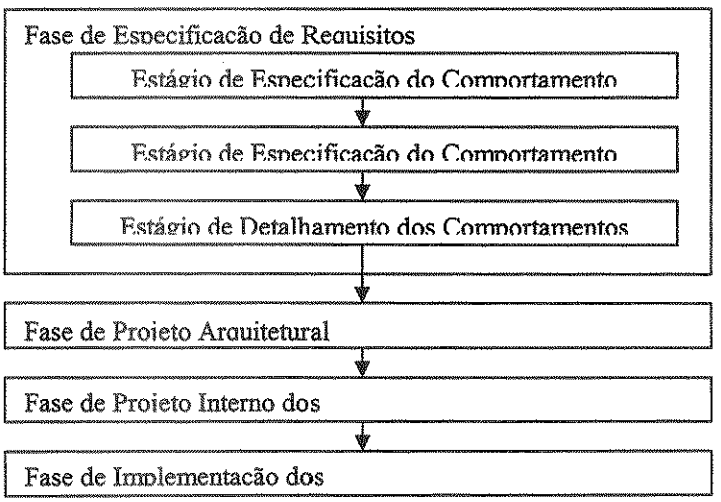

Figura 14 – Fases da Metodologia MDCE

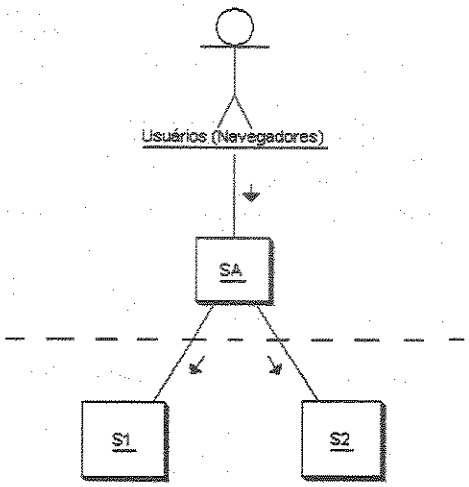

Figura 15 – Exemplo de Servidores Balanceados

Para o exemplo acima, considere que durante uma transação financeira sendo atendida pelo servidor S1, ocorra um problema de falta de energia que faça com que o servidor S1 deixe de funcionar. Esta situação define um cenário excepcional recuperável, onde o tratamento seria realizar o *rollback* desta transação e avisar o usuário para tentar novamente. Além disso, deve-se desviar as novas requisições apenas para o servidor S2. Considere que antes que servidor S1 volte a funcionar, o servidor S2 também pare de funcionar. Isto define um cenário excepcional irrecuperável, pois tem-se uma situação inaceitável para que o sistema continue funcionando. Um tratamento possível seria o servidor SA executar as seguintes operações: enviar uma mensagem para o administrador do sistema descrevendo o problema e não passar novas requisições de usuários para os servidores S1 e S2, além de informar aos usuários que o sistema está fora do ar e que ações de recuperação estão sendo realizadas.

Como comentado, esta fase é dividida em três estágios: definição do comportamento normal, definição do comportamento excepcional e detalhamento dos comportamentos. Segundo a metodologia MDCE, após se definir o comportamento normal, pode-se definir em paralelo o comportamento excepcional e se detalhar o comportamento normal. Após estas atividades, pode-se detalhar o comportamento excepcional. Estas atividades estão ilustradas no diagrama da Figura 16. Os detalhamentos dos comportamentos devem seguir orientações definidas em [Ferreira01].

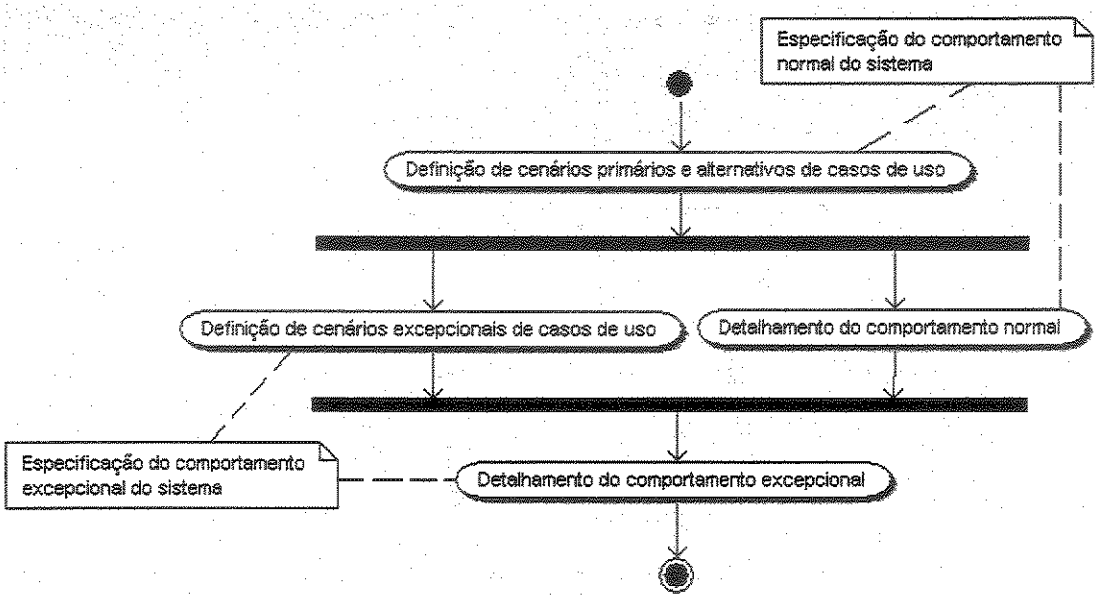

Figura 16 – Atividades da Fase de Especificação de Requisitos da Metodologia MDCE

Para se identificar cenários excepcionais, deve-se analisar cada passo de um cenário primário e cada passo de um cenário alternativo, onde o sistema é o responsável pela ação. Caso uma falha possa ocorrer em alguns destes passos, um cenário recuperável ou irrecuperável deve ser definido. Diferentes hipóteses de falhas podem ser definidas para os passos de casos de uso.

Um cenário excepcional é caracterizado:

(1) Pela descrição de uma situação excepcional associada a um erro que uma falha pode causar;

(2) Pela descrição do erro ocorrido. A ocorrência deste erro é chamada de **Sinal** do cenário excepcional;

(3) Por uma seqüência de atividades que implementam as ações de recuperação do erro. Esta seqüência de atividades é definida pelo **Tratador** do cenário excepcional;

(4) Por pós-condições que definem o estado do sistema após execução do cenário excepcional. Normalmente uma falha de um sistema associada a um passo de um caso de uso leva a violações de pré ou pós-condições deste caso de uso.

Os cenários excepcionais formam a base para a definição das condições excepcionais, que serão sinalizadas através de exceções em sistemas, e para os tratadores destas condições.

2.5.2 Fase de Projeto Arquitetural

A metodologia MDCE propõe o uso do conceito do **Componente Ideal Tolerante a Falhas** (*Idealized Fault-Tolerant Component*) [Randell+95], como base para a definição da arquitetura em camadas e sugere que as interações excepcionais entre os componentes sejam explicitamente definidas no projeto arquitetural. Desta forma, o estilo arquitetural considerado para aplicação da metodologia MDCE é o estilo baseado no Componente Ideal Tolerante a Falhas, o que permite que a propriedade arquitetural 'confiabilidade' persista durante o projeto arquitetural.

De acordo com a abordagem de uso do Componente Ideal Tolerante a Falhas, o sistema é dividido em vários componentes, cada um com uma parcela da responsabilidade de tolerância a falhas. Cada componente é dividido em duas partes, uma parte para a atividade normal e outra para a atividade excepcional. O Componente Ideal Tolerante a Falhas é apresentado na Figura 17.

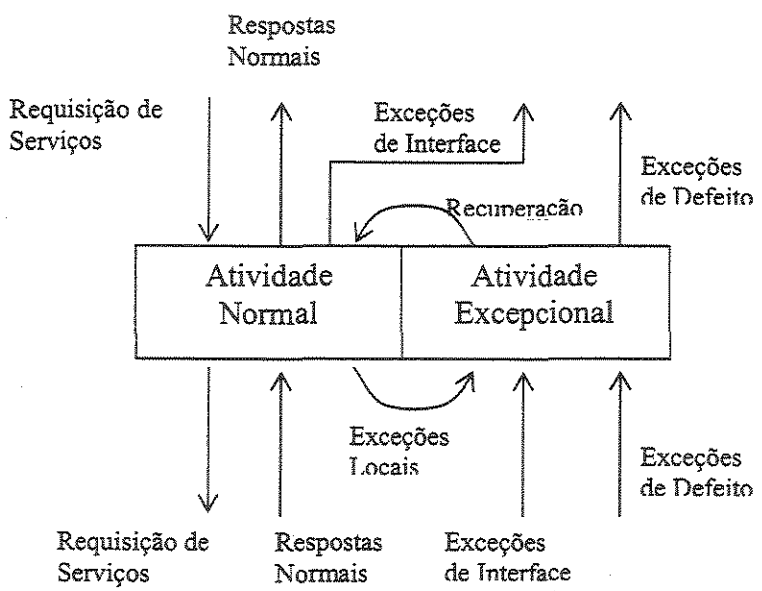

Figura 17 – Componente Ideal Tolerante a Falhas

Uma **exceção de interface** está associada a uma violação de uma pré-condição e deve ser tratada pelo componente que fez a requisição. A **exceção local** está associada a uma violação de uma pós-condição e sinaliza um erro a ser corrigido internamente pelo próprio componente, que após as ações de recuperação pode retornar uma resposta normal. A **exceção de defeito** sinaliza um erro que não pode ser tratado localmente pelo componente e provoca um lançamento de exceção para o componente que requisitou o serviço.

Considere um sistema onde um componente cliente pede serviços de um componente servidor que poderá, durante a execução do serviço, lidar com uma condição excepcional, como apresentado na Figura 18. Esta condição excepcional pode ser tratada internamente no componente servidor, caracterizando assim uma exceção local, como a exceção E3. Por outro lado, esta condição excepcional pode ser sinalizada pelo lançamento de uma exceção de volta para o cliente, que deve ser capaz de tratá-la, como a exceção E2. Além disso, exceções locais ao cliente podem ocorrer, como por exemplo a exceção E1. No projeto arquitetural deste sistema, um componente ideal tolerante a falhas deve ser definido para o componente cliente e outro para o componente servidor. Além disso, as interações excepcionais devem estar explicitadas na arquitetura.

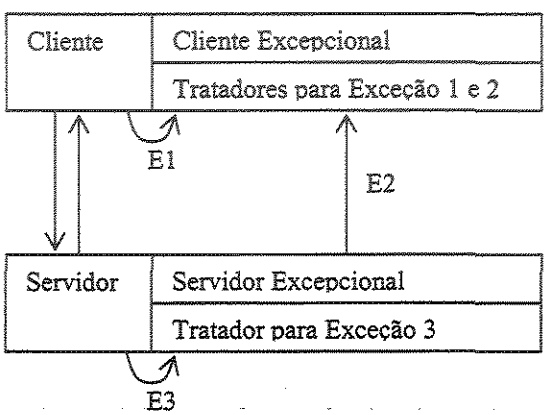

Figura 18 – Componentes Ideais – MDCE

É importante ressaltar que outros complicadores podem existir em um sistema de software baseado em componentes. Pode-se citar: o paralelismo de requisições a um componente e a requisição assíncrona. Por paralelismo entende-se a existência de mais do que uma *thread* fazendo requisições para um mesmo componente. Neste caso uma requisição que tenha uma falha pode afetar outra requisição que estava sendo executada normalmente. Por requisição assíncrona entende-se que uma requisição a um componente inicia uma nova *thread* para a execução da operação em um componente. Neste caso, se acontecer uma falha que leve a um erro, o componente deve notificar quem o requisitou.

2.5.3 Fase de Projeto Interno dos Componentes

Nesta fase define-se o modelo de classes internas de um componente. A metodologia MDCE propõe que sejam definidas duas hierarquias de classes, a hierarquia normal e a

excepcional. A **hierarquia normal** implementa a funcionalidade do componente e deixa explicito quais exceções pode lançar. A **hierarquia excepcional** explicita que tratadores tratam que exceções. Propõe ainda que exceções lançadas pela hierarquia normal sejam consideradas internas, pois são tratadas pelo hierarquia excepcional, e exceções lançadas pela hierarquia excepcional sejam externas pois são tratadas por componentes clientes.

A Figura 19 apresenta um projeto considerando as hierarquias comentadas [Ferreira01]. Nesta figura um componente Cliente tem um método m1 que faz uma requisição para outro componente executar o método m3. No método m3, exceções internas I_E2 ou I_E3 podem ocorrer e devem ser tratadas por um objeto da classe Servidor_Excepcional. Caso o método tratadorE2() deste objeto não consiga tratar a exceção I_E2 ou a exceção I_E3, ele lança uma exceção externa E_E2 que deve ser tratada por um objeto da classe Cliente_Excepcional. Como I_E3 herda de I_E2 pode-se reaproveitar o tratadorE2 da classe Servidor_Excepcional para lidar com ambas. Por sua vez, a classe Cliente tem um Cliente_Excpecional associado que também trata a exceção interna I_E1.

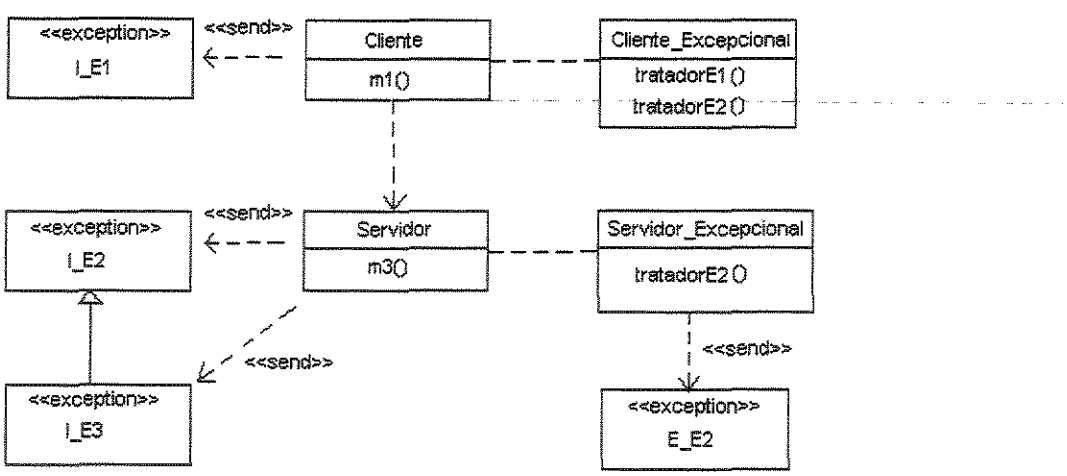

Figura 19 – Exemplo de Hierarquia de Exceções de Projeto Interno de Componentes da Metodologia MDCE

2.5.4 Fase de Implementação

Nesta fase, codificam-se os modelos de classes internas dos componentes. Uma das propostas de MDCE para esta fase, é colocar um bloco *try/catch*, como no caso de Java, na hierarquia normal, em alguma classe onde uma exceção possa ocorrer. Caso uma exceção seja capturada, esta classe chama um método da classe excepcional, correspondente, da hierarquia

excepcional. Esta classe excepcional é considerada o tratador da situação excepcional. A Figura 20 apresenta um exemplo.

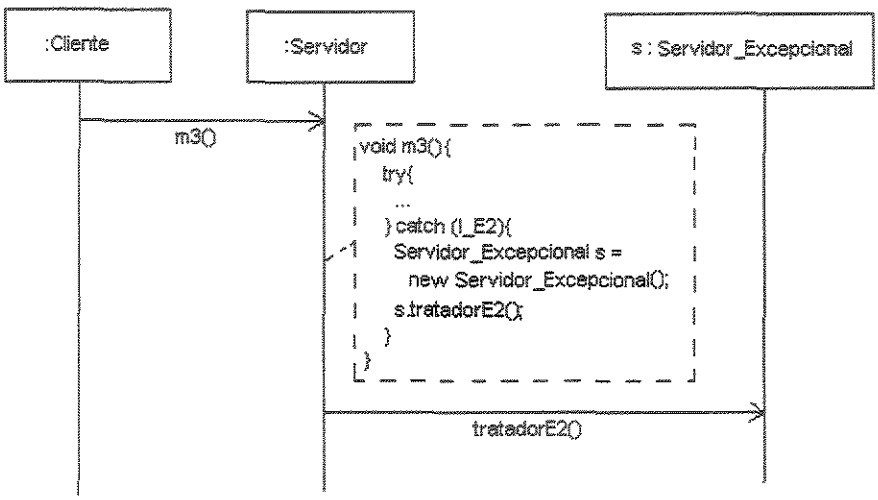

Figura 20 – Exemplo de Implementação Proposto da Metodologia MDCE

2.6 Proposta de Estratégias de Estruturação Arquitetural para Tratamento de Exceções

Como comentado, componentes podem ser desenvolvidos em contextos diferentes daqueles onde serão reutilizados. Ao ser integrado a um novo contexto, um componente deve atender as especificações de comportamento normal e excepcional definidas neste novo contexto.

Por melhor que sejam as especificações pode ser que este componente, ao ser integrado, apresente uma falha de projeto que levaria a geração de uma condição excepcional não antecipada, que seria sinalizada por uma exceção não definida na especificação do sistema. Esta exceção poderia ser propagada por diferentes componentes do sistema levando os mesmos a estados de erro. Além disso, componentes reutilizáveis podem não estar preparados para lançar ou tratar exceções definidas em especificações excepcionais apresentadas em uma arquitetura. Desta forma, poderia haver incompatibilidade entre tipos de exceções que são lançadas por um componente servidor e tipos de exceções esperadas por um componente cliente deste componente servidor. A proposta de estratégias de estruturação arquitetural para tratamento de exceções [Guerra04] lida

com estas situações através de duas estratégias complementares: uma estratégia intra-componente e uma estratégia inter-componente.

A **estratégia intra-componente** é baseada na orientação de que exceções não declaradas, que sinalizam condições excepcionais não antecipadas, não devem ser propagadas livremente pelas camadas da arquitetura de um sistema. Estas exceções devem ser devidamente tratadas e transformadas em exceções bem conhecidas pelo sistema, ou seja, em exceções que são definidas com base em tipos abstratos de exceções declarados na especificação do comportamento excepcional do sistema. A proposta de estruturação arquitetural para tratamento de exceções propõe uma hierarquia de tipos abstratos de exceções que deve ser conhecida pelos componentes e utilizada durante o tratamento das exceções não declaradas. Para que exceções não declaradas possam ser transformadas em exceções bem conhecidas pelo sistema, são necessárias adaptações em componentes, a serem integrados, para que os mesmos lancem exceções conhecidas. Além disso, os tratadores associados a componentes do sistema devem estar preparados para lidarem com estas exceções bem conhecidas. A estratégia intra-componente auxilia na realização destas adaptações e tratamentos.

A **estratégia inter-componente** cuida do tratamento da incompatibilidade entre tipos de exceções que são lançadas por um componente servidor e tratadas por um componente cliente, através da definição de tratadores de conectores que são responsáveis por traduzir exceções entre componentes. Ou seja, caso um componente servidor lance uma exceção que um componente cliente não entenda, o conector deve fazer a tradução desta exceção para uma nova exceção que possa ser tratada pelo componente cliente.

Esta subseção apresenta a hierarquia de tipos de exceção, e as estratégias intra e inter-componente.

2.6.1 Hierarquia de Tipos de Exceções

A Figura 21 [Guerra04] mostra a **hierarquia de tipos abstratos de exceções**, que tem no seu topo o supertipo Exception. Abaixo deste supertipo mais genérico, são definidos dois tipos: DeclaredException e UndeclaredException. DeclaredException é a raiz da hierarquia de tipos de exceções declarados nas especificações dos componentes. O supertipo UndeclaredException é a raiz de uma hierarquia utilizada pela implementação de um componente para definir as exceções

cujos tipos não estariam declarados nas especificações, ou seja, associadas a condições excepcionais não antecipadas durante especificações.

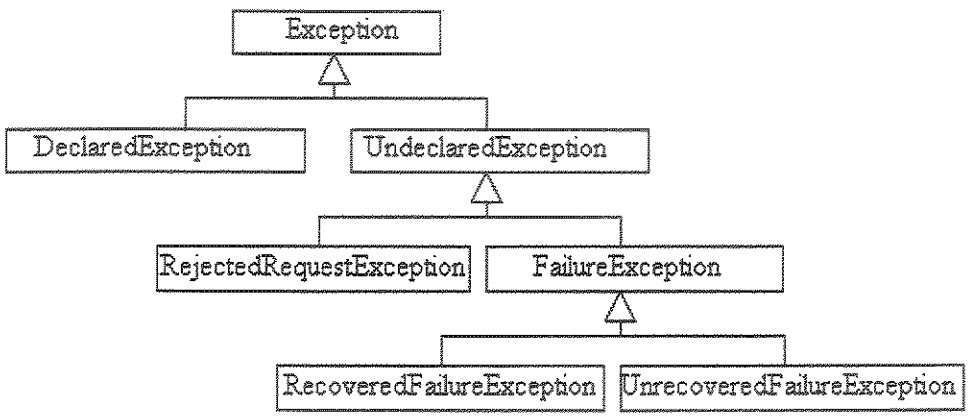

Figura 21 – Hierarquia de Tipos Abstratos de Exceções

A hierarquia de UndeclaredException se divide em dois outros supertipos: RejectedRequestException e FailureException. O tipo RejectedRequestException é utilizado para sinalizar violações de pré-condições que resultem na rejeição das requisição de serviços de um componente. Exceções do tipo FailureRequestException são utilizadas para sinalizar violações de pós-condições que resultem na rejeição das requisições de serviços correspondentes.

O supertipo FailureException se divide em dois outros tipos abstratos: RecoveredFailureException e UnrecoveredFailureException. Exceções do tipo RecoveredFailureException são utilizadas quando a implementação do componente garante que nenhum efeito foi produzido sobre o estado do componente ou no seu ambiente, apesar de um erro ter ocorrido. Exceções do tipo UnrecoveredFailureException são utilizadas quando a implementação do componente fracassa na execução da operação e não pode garantir a ausência de efeitos colaterais, produzidos pela execução da operação, sobre o estado interno do componente ou no sistema [Guerra04].

2.6.2 Estratégia Intra-Componente

A estratégia intra-componente é aplicada no estágio de provisionamento de um processo de desenvolvimento baseado em componentes. Nesse estágio, um componente especificado na arquitetura de um sistema é implementado por desenvolvedores, ou um componente pronto é identificado para reuso. No caso de reuso, pode ser necessário algum tipo de adaptação deste

componente ao novo sistema. No caso de um novo componente a ser desenvolvido, o mesmo deve também poder ser reutilizado em outros sistemas. A estratégia intra-componente se aplica tanto ao desenvolvimento de um novo componente a ser reusado quanto na adaptação de um componente já existente para sua reutilização num novo sistema [Guerra04].

A estratégia intra-componente visa elevar a reusabilidade de componentes de software através das seguintes táticas para interfaces requeridas:

(1) declarar explicitamente na especificação de um componente todos os tipos de exceções, associados a condições excepcionais antecipadas, que podem ocorrer nas operações das interfaces requeridas de um componente. Um componente que implemente estas interfaces deve conter tratadores para estas exceções;

(2) Um componente que implemente estas interfaces pode prover tratadores para exceções filhas de UndeclaredException e um tratador mais genérico para exceções não associadas a hierarquia de tipos de exceções e para exceções não declaradas nas interfaces.

Esta estratégia também propõe táticas para interfaces providas:

(1) declarar explicitamente na especificação de um componente todas as exceções que podem ocorrer nas operações destas interfaces e possam ser lançadas pelo componente;

(2) qualquer condição excepcional não antecipada nas especificações deve ser sinalizada por uma exceção que herde de RejectedRequestException, RecoveredFailureException ou UnrecoveredFailureException.

Além destas táticas, a estratégia intra-componente define responsabilidades de comportamento excepcional que devem ser assumidas por componentes que estão sendo desenvolvidos ou por componentes prontos que serão integrados a um sistema. Estas responsabilidades são descritas nas próximas subseções que apresentam a aplicação da estratégia para componentes novos e componentes prontos.

Aplicação da Estratégia para um Novo Componente

Para a aplicação da estratégia no desenvolvimento de um novo componente, considere a representação de um componente na Figura 22 [Guerra04]. Nela, um componente provê uma interface que é implementada por uma fachada, e requer uma interface oferecida por uma fachada.

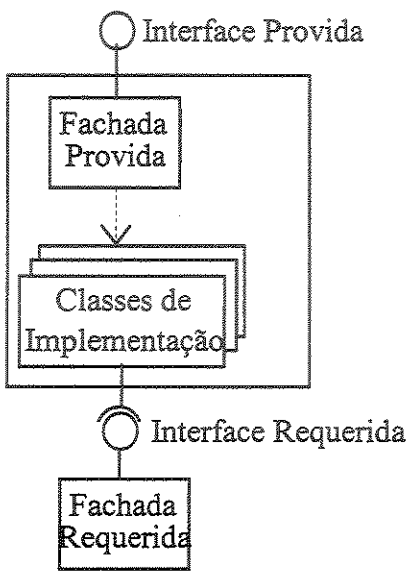

Figura 22 – Especificação de um Componente para Aplicação de Estratégia Intra-Componente

A estratégia propõe a criação de dois tipos de tratadores, um associado a fachada provida, chamado de **Tratador de Fronteira**[3] (BLE), e outro associado a classes de implementação, chamado de **Tratador de Aplicação**[4] (ALE). A Figura 23 apresenta os tratadores.

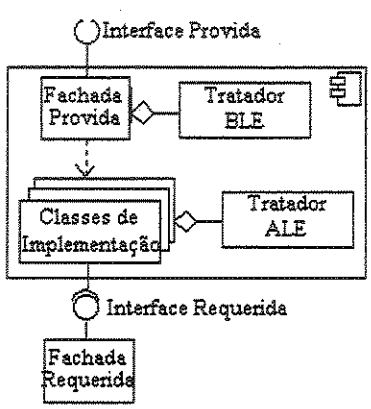

Figura 23 – Especificação de um Componente com Tratadores

[3] Em inglês: *Boundary Level Exception Handler (BLE)*
[4] Em inglês: *Application Level Exception Handler (ALE)*

A responsabilidade pelo comportamento excepcional do componente é dividida entre esses elementos da seguinte forma [Guerra04]:

(1) as classes de implementação são responsáveis por detectar todas as condições excepcionais antecipadas na especificação do componente;

(2) as fachadas do componente podem, a critério da implementação, detectar condições excepcionais não antecipadas que impliquem na violação das pré-condições e pós-condições especificadas para as operações;

(3) as classes de implementação podem, também a critério da implementação, detectar qualquer tipo de condição excepcional não antecipada que possa interferir no comportamento normal do componente;

(4) as condições excepcionais que não possam ser tratadas internamente são sinalizadas para os clientes das operações pela própria classe de implementação ou fachada que detecta a condição;

(5) as condições excepcionais que podem ser tratadas internamente são sinalizadas para um tratador ALE ou BLE, do próprio componente, através do lançamento de uma exceção interna, que é uma exceção de um tipo definido internamente ao componente;

(6) os tratadores ALE e BLE são responsáveis pelo tratamento das exceções internas lançadas, respectivamente, pelas classes de implementação e fachadas do componente;

(7) os tratadores ALE são responsáveis também pelo tratamento das exceções externas, que são as exceções lançadas pelas fachadas externas do componente.

Aplicação da Estratégia para Reuso de um Componente

Nesse caso, pode ser necessária a adaptação do componente. Essa adaptação é feita através de um *wrapper*, com os seguintes novos elementos: interceptadores das fachadas providas do componente, e tratadores de exceções associados a esses interceptadores, que são os tratadores BLE. A Figura 24 apresenta esta adaptação.

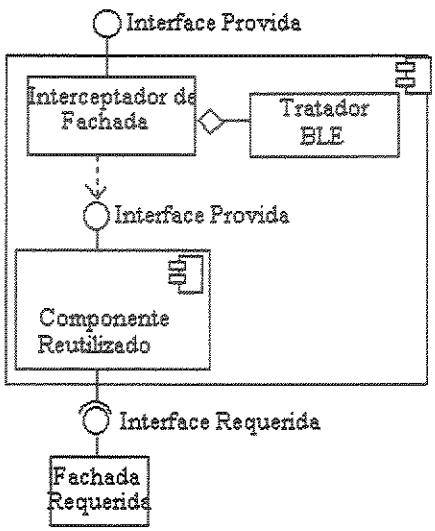

Figura 24 – Adaptação de um Componente Pronto à Estratégia Intra-Componente

As responsabilidades atribuídas aos elementos de um novo componente são distribuídas da seguinte forma [Guerra04]:

(1) a implementação do componente reutilizado assume as responsabilidades atribuídas às classes de implementação e aos tratadores ALE;

(2) os interceptadores de fachada e os tratadores BLE correspondentes são responsáveis por adaptar os tipos das exceções lançadas pela implementação do componente à hierarquia comum de tipos abstratos de exceções. Nesse processo de adaptação, as exceções lançadas que não sejam dos tipos declarados na especificação das suas interfaces são convertidas para um subtipo de UndeclaredException;

(3) os interceptadores de fachada e os tratadores BLE assumem, adicionalmente, as demais responsabilidades atribuídas, no caso anterior, às fachadas do componente e aos tratadores BLE, respectivamente.

2.6.3 Estratégia Inter-Componente

Esta estratégia define **tratadores para o nível de conectores**[5] (CLE) que podem adaptar interfaces de componentes convertendo exceções lançadas por um componente em exceções que

[5] Em inglês: Connector Level Exception Handler

outro entenda. Um conector deve prover tratadores apropriados para exceções de qualquer tipo que possam ser lançadas através das interfaces providas dos componentes. Isso inclui exceções para:

(1) os tipos de exceções declarados nas especificações dessas interfaces, que são subtipos de DeclaredException;

(2) os tipos de exceções não declarados nessas especificações, mas que podem ser previstos pela implementação, que são os tipos abstratos RejectedRequestException, RecoveredFailureException, e UnrecoveredFailureException;

(3) outros tipos de exceções que possam ser lançadas por uma implementação, em condições não previstas pela mesma.

Em [Guerra04] é apresentada a Tabela 2 com regras para definir quais exceções podem ser lançadas por um CLE e em que condições, considerando que um conector liga um componente cliente a um componente servidor.

Tipo da exceção lançada pelo servidor	Tipo da exceção propagada para o cliente
Um tipo E1 declarado nas especificações de ambas interfaces, a provida pelo servidor e a requerida pelo cliente.	E1 (a exceção pode ser propagada automaticamente)
Um tipo E1, declarado na especificação da interface provida pelo servidor, para o qual existe um tipo E2 correspondente declarado na especificação da interface requerida pelo cliente (com a mesma semântica de defeito). Por exemplo: E2 é um supertipo de E1.	E2
Um tipo E1, declarado na especificação da interface provida pelo servidor, para o qual não existe nenhum tipo correspondente declarado na especificação da interface requerida pelo cliente (com a mesma semântica de defeito).	Um subtipo de um dos tipos abstratos RejectedRequestException, RecoveredFailureException ou UnrecoveredFailureException, a ser escolhido de acordo com a semântica de defeito especificada para o tipo E1.
Um tipo E1 que é um subtipo de UndeclaredException	E1 (a exceção pode ser propagada automaticamente)
Qualquer outro tipo não declarado na especificação da interface provida pelo servidor e que não é um subtipo de UndeclaredException	Um subtipo de UnrecoveredFailureException

Tabela 2 – Exceções que um CLE pode Lançar

2.7 COSMOS

Como comentado este trabalho utiliza o COSMOS [Silva03] para orientar a conformidade entre arquitetura de um sistema e sua implementação. Além disso, o COSMOS é utilizado na implementação dos estudos de caso.

O **COSMOS** é um modelo de estruturação de componentes composto por três sub modelos: (1) O modelo de especificação de componente, que define as interfaces requeridas e providas de um componente, (2) o modelo de implementação de componente, que define como os serviços providos pelas interfaces são implementados internamente no componente, e (3) o modelo de conector que define a conexão entre componentes.

2.7.1 Modelo de Especificação de Componente

O **modelo de especificação de componente** define que cada componente de uma arquitetura deve ser dividido em dois pacotes: de especificação (*spec*) e de implementação (*impl*). O pacote de especificação deve ser dividido em dois sub pacotes, um para as interfaces providas (*prov*) e outro para as interfaces requeridas pelo componente (*req*). Estas interfaces têm a visibilidade *public*. O pacote *impl* contém as classes que implementam as operações definidas nas interfaces. Estas classes possuem visibilidade *package* e ficam escondidas dos clientes de um componente. A única exceção é a classe ComponentFactory, que instancia o componente e que pode ser acessada diretamente por um componente cliente. A Figura 25 [Silva03] apresenta os pacotes de um componente de um componente A.

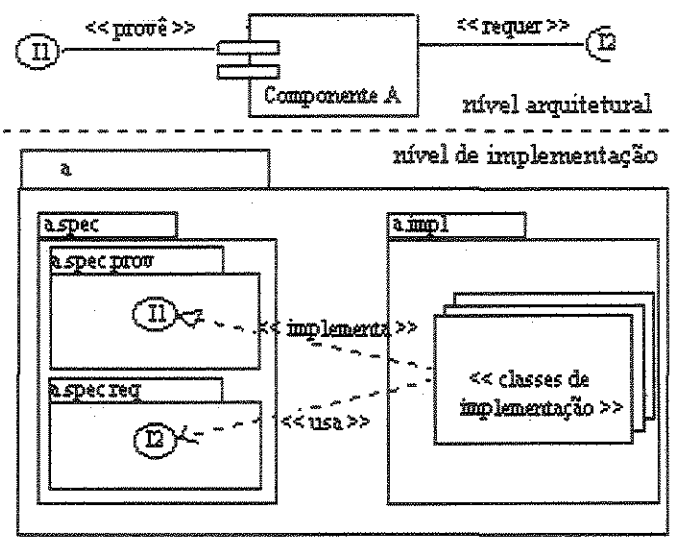

Figura 25 – Pacotes de um Componente Proposto por COSMOS

O modelo de especificação determina que a requisição do acesso aos serviços prestados por um componente seja realizado através da interface IManager deste componente. Um componente deve possuir uma classe Manager que materializa a interface IManager. Esta interface provê

acesso aos objetos do componente que são responsáveis pelas operações das interfaces providas deste componente. Estes objetos são fachadas[6] [Gamma+95] que materializam as interfaces providas pelo componente.

Desta forma, como pode ser visto na Figura 26, um cliente requisita o acesso a serviços prestados por um componente servidor através da interface IManager, obtendo em tempo de execução um objeto da classe Manager. O cliente requisita uma interface provida para o Manager, que em tempo de execução retorna uma fachada para o cliente. Um objeto de fachada deve possuir referências para objetos de classes do componente que efetivamente implementam as operações das interfaces providas. Quando uma fachada recebe uma requisição de um cliente para execução de uma operação, a fachada delega a execução para o devido objeto. No exemplo da figura, a fachada F materializa a interface I1, e possui uma referência para objeto O, para quem delega a execução de uma operação requisita pelo cliente. A fachada F possui um método método1(), que é chamado pelo Cliente. O objeto O possui um método método2(), que é chamado pela fachada F.

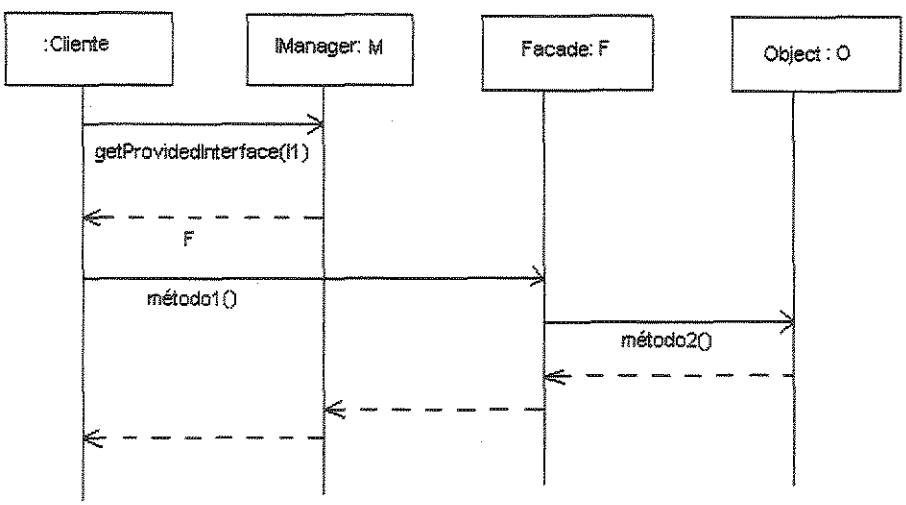

Figura 26 – Exemplo de Requisições em um Componente Especificado por COSMOS

[6] Em inglês o termo associado é Facade. Nas figuras desta seção será usado o termo Facade. A fachada provê acesso as operações providas por um componente.

2.7.2 Modelo de Implementação de Componente

O **modelo de implementação de um componente** especifica como as interfaces providas pelo componente são implementadas. A Figura 27 a seguir [Silva03], apresenta o pacote de implementação.

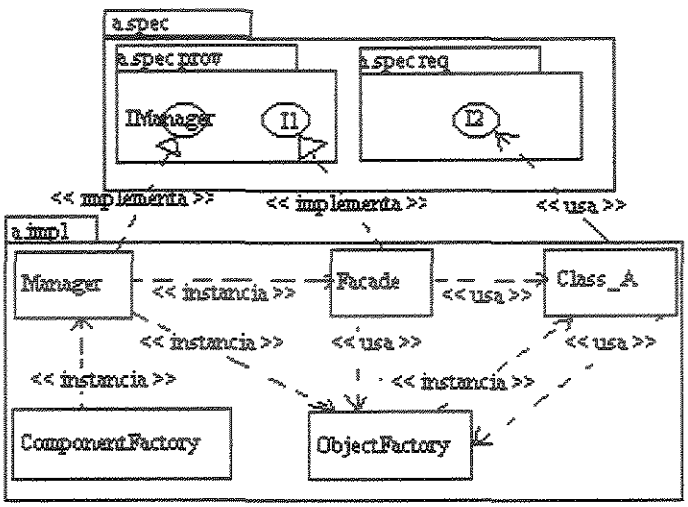

Figura 27 – Pacote de Implementação e suas Classes de Acordo com COSMOS

O pacote *impl* possui quatro classes, a Manager, o ComponentFactory, pelo menos uma Fachada (Facade) e um ObjectFactory. Como comentado, a Fachada materializa uma interface provida pelo componente e delega a execução de uma operação desta interface para um objeto de uma classe definida no componente. Para uma Fachada obter uma referência para este objeto, ela faz uma requisição para o ObjetcFactory, que é um *singleton* (uma classe que só pode ter uma instância em um dado momento) [Gamma+95]. Pode-se observar na figura que um objeto delegado por uma fachada pode requerer uma interface provida por outro componente.

2.7.3 Modelo de Conector

O **modelo de conector** especifica a conexão entre interfaces requeridas e interfaces providas, permitindo assim que dois ou mais componentes possam ser conectados em uma configuração de componentes.

De acordo com o modelo COSMOS, um conector deve ser implementado através de um pacote que por sua vez deve conter um pacote chamado *impl*. Um conector não define interfaces providas e requeridas. O pacote *impl* deve conter um ComponentFactory, Manager, uma Fachada

e a interface IManager. A Figura 28 apresenta um conector "ab", que conecta os componentes "a" e "b".

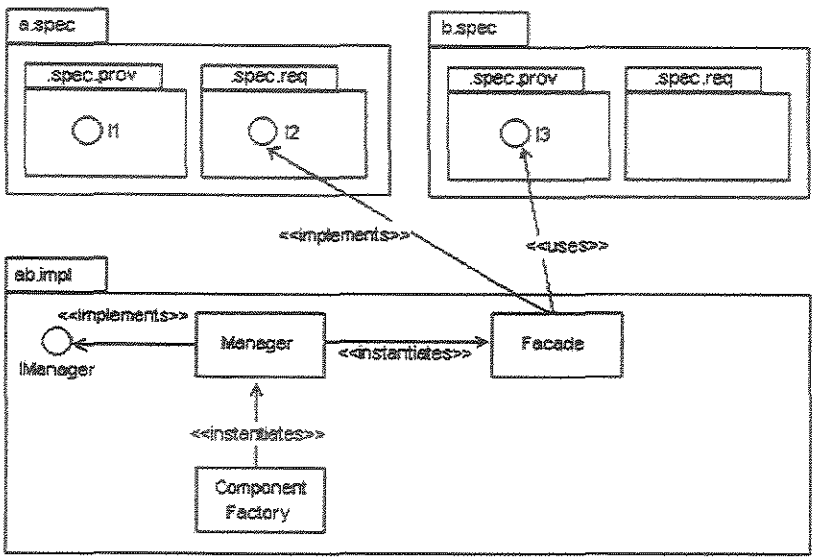

Figura 28 – Modelo de Conector do COSMOS

A Figura 29 apresenta um diagrama de seqüência com uma requisição de um componente cliente sendo atendida pelo o componente "a" que por sua vez requer interfaces do componente "b". Nesta figura pode-se observar o papel do conector.

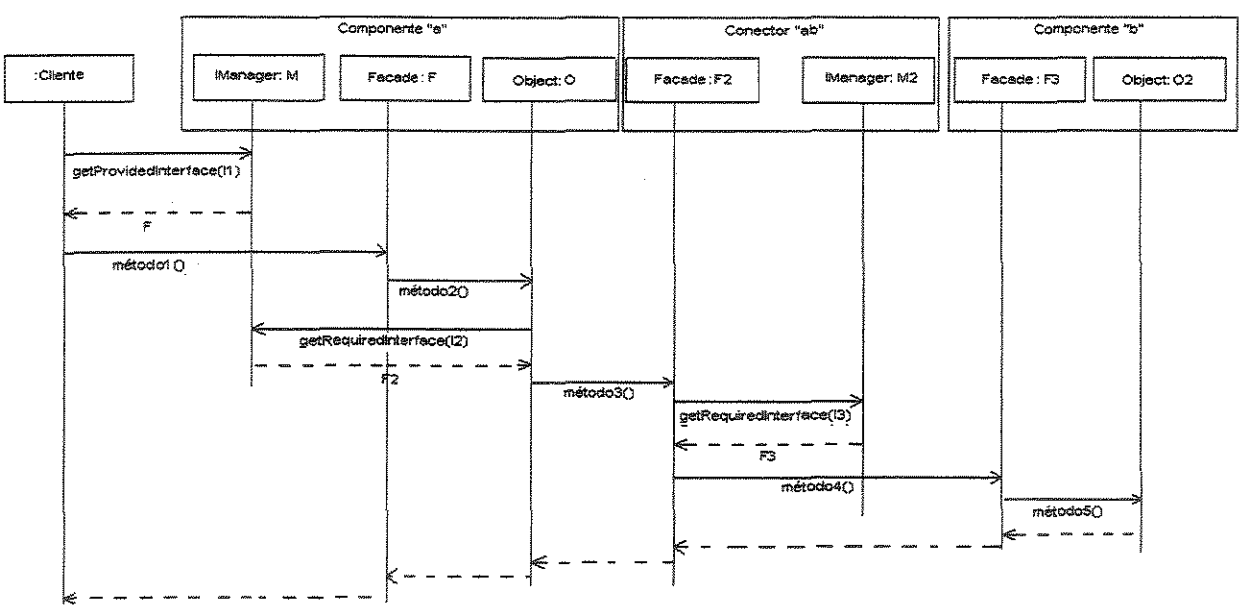

Figura 29 – Exemplo de Fluxo de Execução em Componentes

Em tempo de execução um cliente do componente "a" pergunta ao Manager M do componente "a" quem é o responsável pela interface provida I1. O Manager de "a" retorna a fachada F do componente "a". O cliente chama o método m1 da fachada F, que por sua vez delega a execução da operação para um objeto O.

O objeto O faz uma requisição para o Manager M de "a" para saber quem é responsável pela interface requerida I2 e recebe como retorno a fachada F2 do conector, para quem faz a requisição da operação. Esta fachada pergunta ao Manager M2 do conector quem é o responsável pela interface "I3" e recebe como retorno uma fachada F3 do componente "b", e chama uma operação da fachada F3. Por sua vez, a fachada F3 delega a execução para um objeto O2 do componente "b".

As atribuições de responsabilidades sobre interfaces são realizadas no momento em que o sistema é inicializado. No caso da Figura 29, no momento em que o sistema é inicializado, define-se que a fachada F2 será responsável por oferecer a interface I2, e que a fachada F3 será responsável por oferecer a interface I3. Desta forma, conecta-se o conector "ab" ao componente "a", e ao componente "b", respectivamente. As definições de responsabilidades estão baseadas na configuração de componentes e conectores.

2.8 Considerações do Capítulo

Este capítulo apresentou fundamentos teóricos para lidar com definição arquitetural para sistemas de software confiáveis e baseados em componentes.

Com conhecimento dos conceitos de arquitetura de software, de componentes e do processo UML Components, arquitetos têm uma base para definir uma arquitetura de software bem organizada com especificação normal de componentes, permitindo que um integrador possa identificar um componente reutilizável que atenda à especificação normal.

Conhecendo os conceitos de confiabilidade e da metodologia MDCE, tem-se uma base para definir comportamento excepcional já nas fases iniciais de desenvolvimento de software, o que permite que se projete uma arquitetura levando-se em consideração o comportamento excepcional do sistema, que por sua vez servirá de base para a implementação do sistema.

Através das estratégias de estruturação arquitetural para tratamento de exceções tem-se orientações para as fases de provisionamento e integração de modo que novos componentes e componentes reutilizáveis atendam às especificações excepcionais definidas na arquitetura.

Com base nos conceitos de arquitetura, componentes de software, do processo UML Components, confiabilidade e da metodologia MDCE, este trabalho propõe uma solução para se definir uma arquitetura de um sistema, com especificação normal e excepcional de seus componentes. Esta solução é baseada no método MECE, que é apresentado no próximo capítulo. Além disso, a solução apresentada neste trabalho orienta o uso das estratégias de estruturação arquitetural para tratamento de exceções, para que componentes reutilizáveis possam ser adaptados atendendo às especificações excepcionais definidas na arquitetura.

Capítulo 3

O Método MECE

Este capítulo apresenta um método para especificação de componentes da arquitetura de um sistema e seu comportamento excepcional, o **método MECE**. Este método é baseado na integração entre a metodologia MDCE e o processo UML Components. A metodologia MDCE é genérica [Ferreira01], o que permite sua aplicação em diferentes processos de desenvolvimento de software baseados em componentes, inclusive no processo UML Components. Esta integração promove a definição de uma arquitetura com especificação normal e excepcional de componentes de um sistema, propiciando o reuso de componentes.

A integração entre a metodologia MDCE e o processo UML Components ocorre com a aplicação de atividades da fase de especificação de requisitos da metodologia MDCE na fase de definição de requisitos do processo UML Components, e com a aplicação de atividades da fase de projeto arquitetural da metodologia MDCE na fase de especificação de componentes do UML Components. Caso se deseje implementar componentes em vez de reutilizar componentes prontos, o método MECE orienta que se aplique a fase de projeto interno de componentes e a fase de implementação definidas pela metodologia MDCE.

Além da aplicação de atividades da metodologia MDCE nas fases de UML Components, o método MECE propõe a definição de novas atividades para a fase de especificação de componentes. Uma delas tem o objetivo de auxiliar a identificação de quais exceções podem ser lançadas por quais operações de componentes e quais exceções devem ser tratadas por componentes. A outra atividade propõe o refinamento da arquitetura com o uso de conectores o que auxilia o trabalho de um integrador, como será comentado adiante.

Os diagramas de atividades apresentados neste capítulo estão baseados nos diagramas de atividades do processo UML Components apresentados na subseção 2.4, e em suas subseções. Ao apresentar os diagramas neste capítulo, adiciona-se as atividades definidas pela metodologia MDCE e pelo método MECE, destacando-as através da cor cinza.

O processo UML Components propõe que se considere a definição de quais exceções cada operação de um componente pode lançar, mas não orienta um arquiteto em como identificar e especificar estas exceções com base em cenários excepcionais.

3.1 As Fases do Método MECE

A solução proposta nesta monografia é baseada na aplicação do método MECE e na aplicação das estratégias de estruturação arquitetural para tratamento de exceções proposta em [Guerra04]. O principal foco do método MECE está nas fases de especificação de requisitos e especificação de componentes, embora o método MECE apresente orientações para desenvolvimento de componentes com base na fase de implementação da metodologia MDCE. O foco das estratégias de estruturação arquitetural para tratamento de exceções está nas fases de provisionamento e integração.

A fase de definição de requisitos do método MECE envolve a especificação do comportamento normal e excepcional do sistema através da definição de cenários primários, alternativos e excepcionais dos casos de uso. A fase de especificação de componentes envolve a definição de uma arquitetura com a especificação normal e excepcional de seus componentes, e com a identificação de conectores. A Figura 30 apresenta as fases de definição de requisitos e especificação de componentes do método MECE.

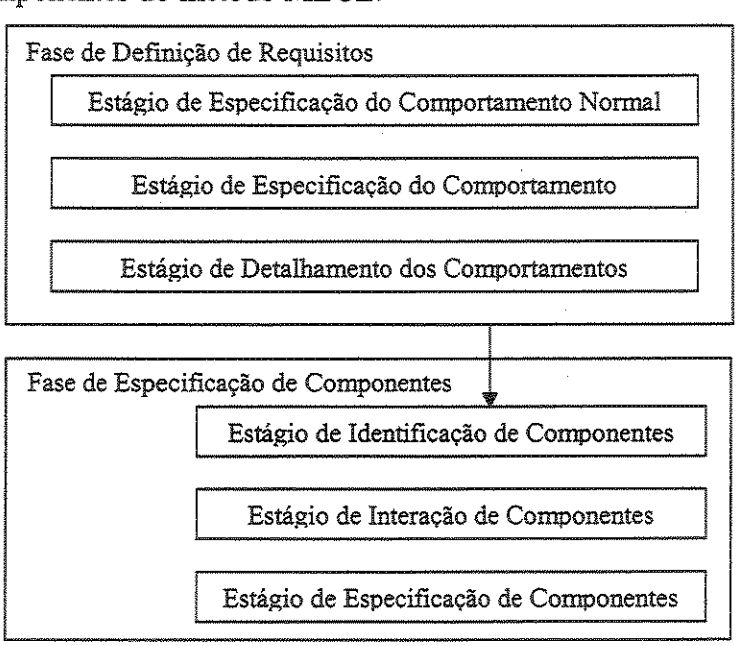

Figura 30 – Fases do Método MECE

3.1.1 Fase de Definição de Requisitos

A **fase de definição de requisitos** tem como início, a definição do processo de negócio de um sistema a ser construído. Com base neste processo de negócio, define-se o modelo conceitual de negócio e os casos de uso do sistema, assim como é feito pelo processo UML Components e apresentado na subseção 2.4.1. Os casos de uso são definidos com cenários primários e alternativos. Cada passo de um cenário deve ser composto de um sujeito e de um predicado, onde o sujeito da oração é o usuário do sistema, ou o próprio sistema, ou outro sistema que interage com o sistema a ser construído.

Neste momento aplicam-se conceitos da metodologia MDCE para a definição do comportamento excepcional do sistema, como apresentado na subseção 2.5.1. Inicia-se com a identificação das pré e pós-condições dos casos de uso definidos, e com a elaboração dos cenários excepcionais recuperáveis e irrecuperáveis. Para cada passo de um cenário primário ou alternativo, onde o sujeito da oração é o sistema a ser construído, deve-se verificar se uma condição excepcional pode acontecer com base em violações de pré ou pós-condições e em hipóteses de falhas. Em caso positivo, deve-se definir um cenário excepcional e associá-lo ao passo do caso de uso.

A Figura 31 apresenta as fases de definição de requisitos do método MECE. Ela é praticamente a soma das atividades da metodologia MDCE com atividades do processo UML Components para a fase de definição de requisitos. Esta figura contém as atividades de "Definição de Requisitos", da Figura 30. As atividades definidas pela metodologia MDCE são destacadas com a cor cinza.

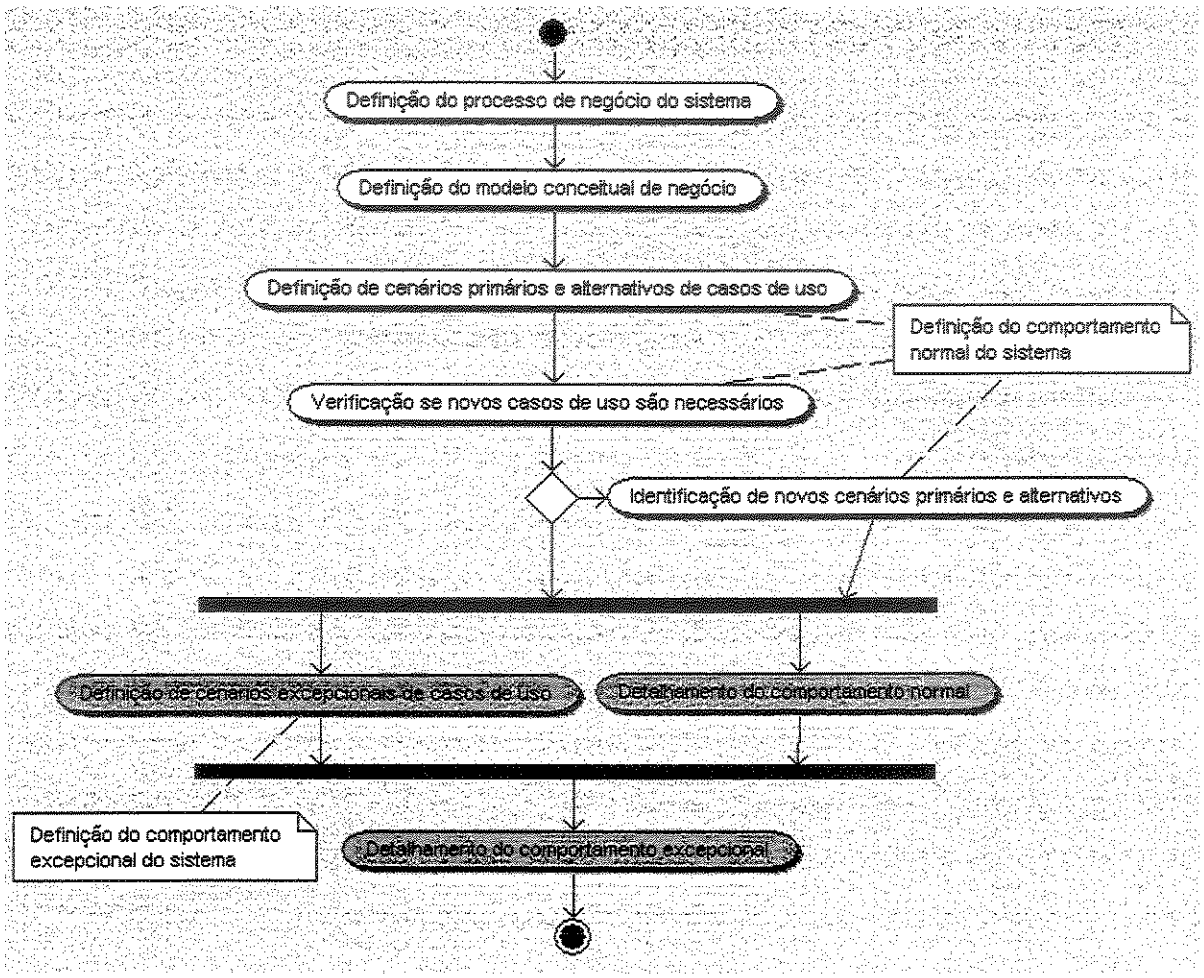

Figura 31 – Atividades de Definição de Requisitos do Método MECE

Em paralelo à definição dos cenários excepcionais, pode-se detalhar o comportamento normal, como proposto pela metodologia MDCE. Por fim, detalha-se o comportamento excepcional. Desta forma tem-se a definição do comportamento normal e excepcional do sistema, e parte-se para a fase de especificação dos componentes.

3.1.2 Fase de Especificação de Componentes

A **fase especificação de componentes** tem como entrada os cenários primários, alternativos e excepcionais, além do modelo conceitual de negócio. É dividida em três estágios baseados na proposta do processo UML Components: identificação de componentes, interação de componentes e especificação propriamente dita dos componentes. Aplica-se conceitos da metodologia MDCE nos estágios de identificação e interação de componentes, e propõe-se

atividades complementares para todos os estágios para: orientar a identificação de quais exceções devem ser tratadas e lançadas por componentes, e para orientar uso de conectores.

Estágio de Identificação de Componentes

Com base nos cenários primários e alternativos, este estágio aplica as atividades propostas pelo processo UML Components para identificação de interfaces da camada de sistema da arquitetura, e de suas operações. Lembrando que para um caso de uso define-se uma interface da camada de sistema da arquitetura, e para cada passo de um caso de uso onde o sujeito da oração é o sistema, pode-se definir uma operação desta interface.

A partir deste momento tem-se uma atividade para identificar quais condições excepcionais podem ocorrer durante a execução das operações das interfaces providas da camada de sistema. Considerando que um passo de um caso de uso pode ter uma pré e uma pós-condição associada a ele, e que um passo de um caso de uso leva à definição de uma operação, pode-se associar estas pré e pós-condições do passo do caso de uso, a esta operação.

Como a violação de uma pré ou pós-condição em um passo de um caso de uso leva à definição de um cenário excepcional para lidar com uma condição excepcional, pode-se definir uma exceção que poderá ser lançada pela operação associada ao passo do caso de uso, caso haja a violação de pré ou pós-condição. Esta exceção deve ser definida com base no Sinal do cenário excepcional e deverá ser tratada de acordo com a especificação do Tratador deste cenário excepcional.

Considere o simples exemplo de um sistema de reserva de livros em uma biblioteca e novamente o caso de uso de reserva de livro:

Nome: Reserva de Livro.

Ator que o inicia: Bibliotecário.

Objetivo: Reservar um livro.

Cenário Principal:

1. Usuário de biblioteca pede para que um livro seja reservado.
2. Bibliotecário acessa sistema.
3. Sistema valida usuário.

4. Sistema realiza a reserva.

5. Bibliotecário mostra para o usuário o local onde livro se encontra.

Extensão (caso o livro esteja reservado para outro usuário, os passos 4 e 5 são diferentes):

4. Sistema indica que livro não está disponível.

5. Bibliotecário avisa usuário que livro não está disponível.

Como já citado na subseção 2.4.2, pode-se definir a interface IReservaLivro da camada de sistema, com uma operação reservaLivro(identificadorDoUsuário , identificadorDoLivro), associada ao passo 4 do cenário primário. Considere que existe uma pré-condição associada ao passo 4 do cenário primário, que diz que o identificadorDoUsuário deve ser válido, e que existe uma pós-condição que diz que o livro que passa a estar reservado no sistema deve ter o identificador informado pelo usuário.

Durante a identificação de cenários excepcionais pode-se definir um cenário excepcional caracterizado pela violação da pré-condição mencionada. Neste cenário excepcional, o Sinal seria o evento da percepção de um valor inválido para o parâmetro identificadorDoLivro, e o Tratador descreveria a ação a ser tomada pelo sistema, como por exemplo não efetuar a reserva e avisar para o bibliotecário que o identificador não é válido. Pode-se também definir um cenário excepcional, caracterizado pela violação da pós-condição mencionada. Neste cenário, o Sinal seria um evento indicando que houve uma falha durante a operação da reserva do livro no sistema e que não é possível saber se a reserva foi realmente efetivada ou não. O Tratador apresentaria as ações que devem ser tomadas, como por exemplo: avisar o administrador do sistema que houve uma falha e que uma intervenção manual pode ser necessária para remover a reserva do livro do sistema, e apresentar uma mensagem para o bibliotecário.

Tendo como base os cenários excepcionais associados ao passo 4 do caso de uso, e considerando que o passo 4 foi mapeado para uma operação da interface IReservaLivro, pode-se definir que esta operação pode lançar as exceções: IdentificadorInválidoException e FalhaNaReservaException, sendo a primeira associada à violação da pré-condição citada acima, e a segunda associada a violação da pós-condição citada acima. Além disso, sabe-se que os tratadores destas exceções deverão funcionar de acordo com as descrições de seus Tratadores nos cenários excepcionais.

Além do mapeamento entre o Sinal do cenário excepcional para uma exceção do sistema, e do Tratador do cenário excepcional para a descrição de um tratador desta exceção, deve-se também mapear a pós-condição excepcional descrita no caso de uso para uma pós-condição excepcional da operação. Esta pós-condição deve definir as garantias oferecidas pelo componente ao término de uma execução mal sucedida da operação, quando é lançada uma exceção declarada na sua especificação.

O diagrama de atividades da Figura 32 apresenta a atividade de identificação de exceções, juntamente com as demais atividades do estágio de identificação de componentes. Esta nova atividade é destacada com a cor cinza.

Através desta figura também percebe-se que, com base no modelo conceitual de negócio, identificam-se as interfaces da camada de negócios exatamente como o processo UML Components propõe.

Como última atividade deste estágio, deve-se refinar a arquitetura. Durante o refinamento, já se conhece exceções que podem ocorrer em operações de componentes da camada de sistema. Estas exceções podem estar associadas a violações de pré ou pós-condições de operações dos componentes. Considerando o conceito do Componente Ideal Tolerante a Falhas, isto permite a identificação de exceções locais, caso se tenha violações de pós-condições, e de exceções de interface, caso se tenha violações de pré-condições. Também é possível definir quais exceções de defeito seriam lançadas caso um componente não consiga tratar adequadamente uma exceção local. Desta forma, durante a atividade de refinamento da arquitetura, os componentes da camada de sistema começam a ser especificados como componentes ideais tolerantes a falhas, seguindo assim orientações da metodologia MDCE para a fase de projeto arquitetural. Porém, ainda não se define as exceções que devem ser tratadas por um componente devido a erros em operações de interfaces requeridas por este componente. Para tal, é necessário iniciar o estágio de interação de componentes.

Ainda não é o momento adequado para que componentes da camada de negócios sejam especificados como componentes ideais tolerantes a falhas, pois não se tem definidas ainda quais são as exceções que podem ocorrer em suas operações. Estas definições ocorrem no estágio de interação de componentes.

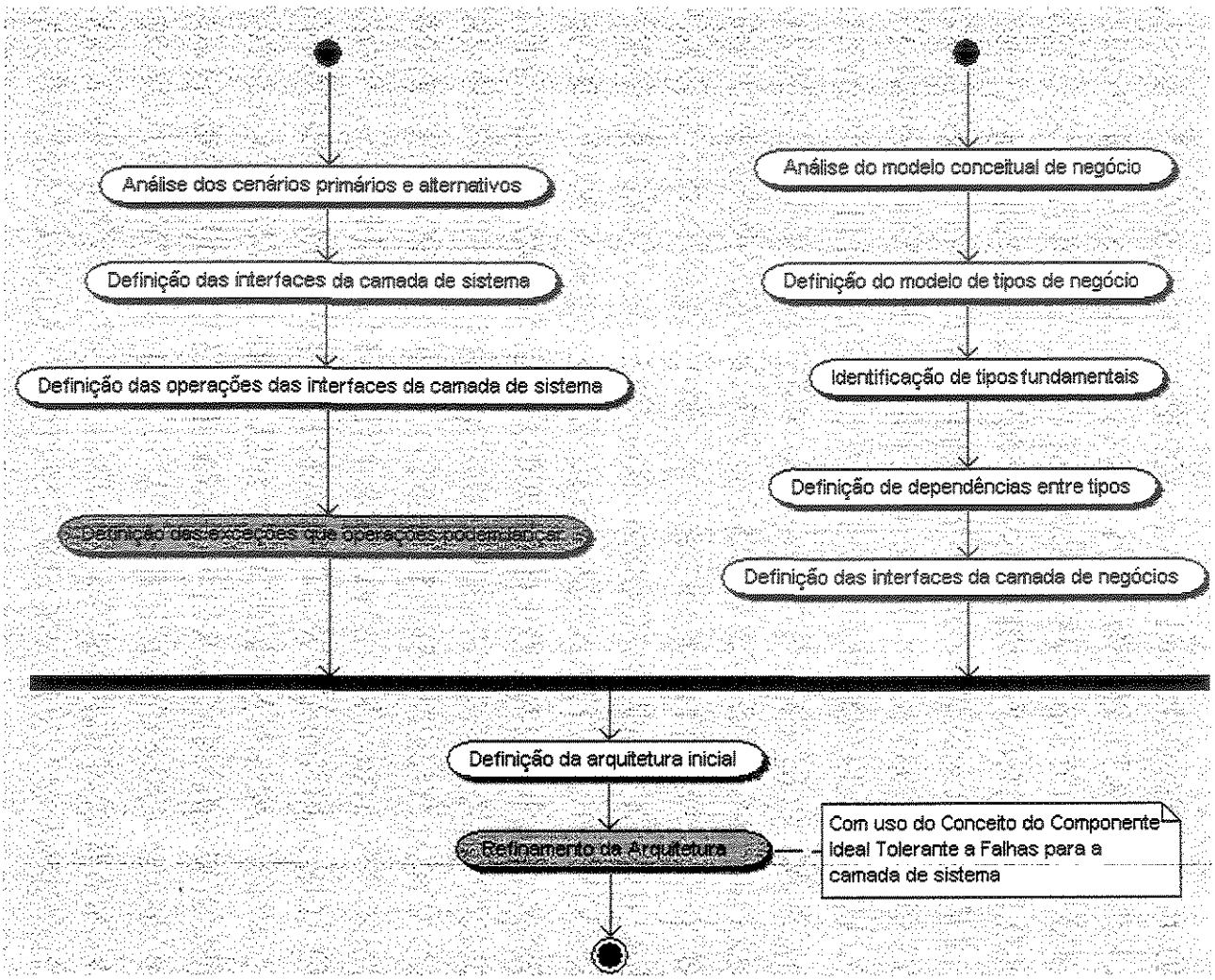

Figura 32 – Atividades de Identificação de Componentes do Método MECE

Estágio de Interação de Componentes

A Figura 33 apresenta as atividades deste estágio caracterizado pela definição das operações das interfaces da camada de negócios e das exceções que podem ocorrer durante a execução destas operações. Este estágio se inicia da forma proposta pelo processo UML Components: com a identificação dos tipos de dados e com a elaboração de diagramas de colaboração para a identificação das operações das interfaces da camada de negócios, como apresentado na subseção 2.4.2. A partir deste momento propõe-se uma atividade de evolução das descrições dos diagramas de colaboração para apresentar quais as exceções que podem ocorrer nas operações da camada de negócios.

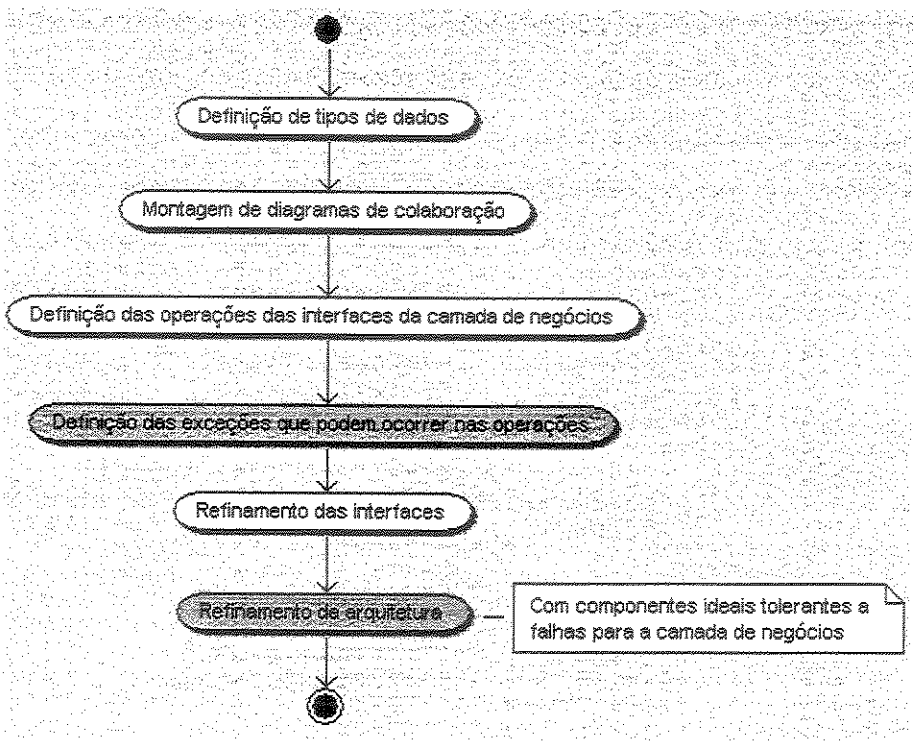

Figura 33 – Atividades do Estágio de Interação de Componentes do MECE

Considerando o exemplo de reserva de livro em uma biblioteca e a operação reservaLivro(identificadorDoUsuário, identificadorDoLivro) da interface IReservaLivro, elaborou-se o diagrama de colaboração já apresentado na Figura 12, do capítulo anterior. Neste exemplo, exceções podem ocorrer durante a operação validaUsuário(identificadorDoUsuário), da interface IUsuárioMgt, e durante a operação reservaLivro(identificadorDoUsuário, identificadorDoLivro), da interface IBibliotecaMgt. Estas exceções podem estar associadas a violações de pré ou pós-condições.

Considerando o conceito de Componente Ideal Tolerante a Falhas, pode-se classificar as exceções que podem acontecer na camada de negócios em: exceções locais, de interface, ou de defeito, assim como é feito para a camada de sistemas. Uma exceção de interface ou uma exceção de defeito, que possa ser lançada por um componente da camada de negócios, deve ser capturada por um componente da camada de sistema e tratada pela sua parte que implementa as atividades excepcionais. Por outro lado, uma exceção local que ocorra em um componente da camada de negócios, deve ser tratada pelo próprio componente da camada de negócios.

Considere um exemplo onde um componente C1, que oferece a interface IReservaLivro, chama a operação reservaLivro(identificadorDoUsuário, identificadorDoLivro) de um componente C2 que oferece a interface IBibliotecaMgt. Caso esta operação lance uma exceção de interface EI2, como mostrado na Figura 34, esta exceção deve ser tratada pelo componente C1, em sua parte responsável pela atividade excepcional. Caso ocorra uma exceção local no componente C2 durante esta operação, como a EL2, a mesma deve ser tratada no próprio componente C2. Caso o componente C2 não consiga tratar esta exceção adequadamente, uma exceção de defeito ED2 deve ser lançada para que o componente C1 a trate.

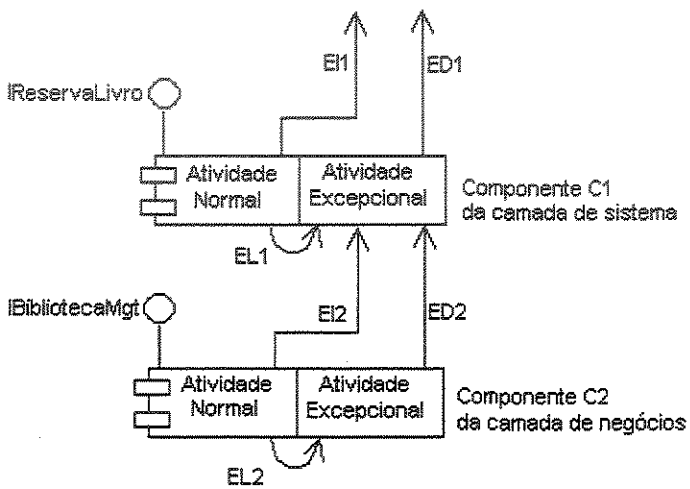

Figura 34 – Componentes Ideais Tolerantes a Falhas para Camada de Sistema e Camada de Negócios

Após a identificação das exceções das operações da camada de negócios, pode-se refinar as interfaces e a arquitetura como proposto pelo processo UML Components. Além disso, deve-se considerar o uso de conectores entre componentes de diferentes camadas, como será discutido na subseção 3.2.

Tem-se ao final desta fase uma arquitetura refinada com componentes ideais tolerantes a falhas, tanto na camada de sistema quanto na camada de negócios da arquitetura. Além disso, tem-se diagramas de colaboração indicando as chamadas das operações das camadas de sistema e negócios, e descrevendo que exceções podem ocorrer durante estas operações, e quais componentes tratam estas exceções.

Estágio de Especificação de Componentes

Neste estágio de especificação, deve-se seguir as atividades propostas pelo processo UML Components juntamente com uma nova atividade definida pelo método MECE. Com base nestas atividades define-se os contratos de uso e de realização de cada componente de uma arquitetura, contendo a especificação normal e excepcional de cada componente. Todas as informações definidas durante os estágios de identificação e interação de componentes são utilizadas nas especificações dos contratos.

A definição do contrato de uso de um componente contém a especificação de interfaces providas deste componente, que deve apresentar suas operações e indicar quais exceções podem ser lançadas por quais operações. Na definição do contrato de realização de um componente, além da especificação das interfaces providas, especifica-se também as interfaces requeridas do componente. A especificação das interfaces requeridas devem indicar quais operações são requeridas e quais exceções este componente deverá tratar devido a erros nestas operações.

A Figura 35 apresenta o diagrama de atividades para este estágio com as novas atividades propostas pelo método MECE destacadas com a cor cinza. Inicialmente define-se a especificação normal para os contratos de uso e de realização, como pode ser visto nas quatro primeira atividades. Posteriormente define-se a especificação excepcional que também passa a fazer parte dos contratos de uso e realização.

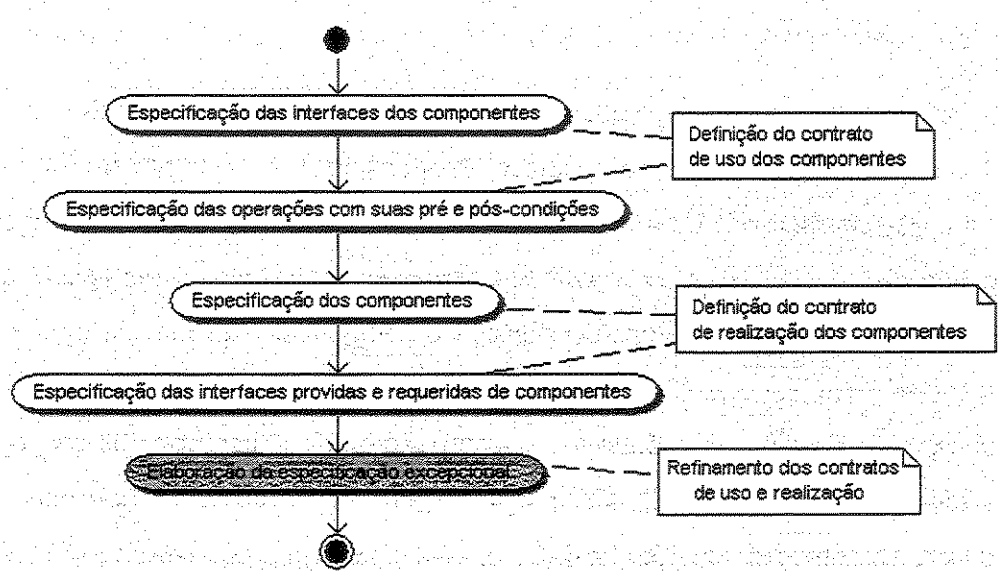

Figura 35 – Fases de Especificação de Componentes do Método MECE

Para representar a especificação excepcional em contratos de uso e de realização de componentes, o método MECE propõe o uso de **tabelas de especificação de comportamento excepcional** (TECE) de componentes. Para cada componente, deve-se criar uma **tabela TECE para interfaces providas** (TECEp) e uma **tabela TECE para interfaces requeridas** (TECEr).

A tabela TECEp é utilizada para representar especificação excepcional no contrato de uso de um componente. Uma tabela TECEp de um componente deve apresentar as interfaces providas, suas operações, a pré e pós-condição de cada operação e as exceções que cada operação deve lançar. Uma tabela TECEp apresenta também uma coluna que indica qual é a pós-condição excepcional após o componente lançar uma exceção associada a uma violação de pré-condição, e uma coluna que indica qual é a pós-condição excepcional após o componente lançar uma exceção associada a uma violação de pós-condição.

A Tabela 3 apresenta uma tabela TECEp para um componente C1 da camada de sistema de uma arquitetura. De acordo com esta tabela, o componente C1 provê as interfaces Interface1 e Interface2. Em sua Interface1, por exemplo, o componente C1 possui as operações operação1 e operação2. A operação1 possui a pré-condição1 e a pós-condição1. Caso haja uma violação da pré-condição1, o componente deve lançar uma exceção cujo tipo é exceçãoPré1. Caso haja uma violação da pós-condição1, o componente deve lançar uma exceção cujo tipo é exceçãoPós1. Caso seja lançada uma exceção do tipo exceçãoPré1, o componente deve garantir que sua pós-condição excepcional seja a pós-condiçãoPréE1. Caso seja lançada uma exceção do tipo exceçãoPós1, o componente deve garantir que sua pós-condição excepcional seja a pós-condiçãoPósE1.

Uma pré-condição pode ser violada de diferentes formas, podendo levar a diferentes exceções. O mesmo se aplica a uma pós-condição. A operação11 do componente C1, possui a pré-condição11 e a pós-condição11. Caso haja uma violação da pré-condição11 o componente pode lançar uma exceção cujo tipo é exceçãoPré11 ou exceçãoPré11B, dependendo da violação ocorrida. Caso haja uma violação da pós-condição11 o componente deve lançar uma exceção cujo tipo é exceçãoPós11 ou exceçãoPós11B, também dependendo da violação ocorrida. Caso seja lançada uma exceção do tipo exceçãoPré11, o componente deve garantir que sua pós-condição excepcional seja a pós-condiçãoPréE2. Caso seja lançada uma exceção do tipo exceçãoPós11, o componente deve garantir que sua pós-condição excepcional seja a pós-condiçãoPósE2. Caso seja lançada uma exceção do tipo exceçãoPré11B, o componente deve garantir que sua pós-condição

excepcional seja a pós-condiçãoPréE3. Caso seja lançada uma exceção do tipo exceçãoPós11B, o componente deve garantir que sua pós-condição excepcional seja a pós-condiçãoPósE3.

TECEp - Componente C1 / Camada de Sistema							
Interface Provida	**Operação**	**Pré-Condição**	**Pós-Condição**	**Exceção Associada a Pré-Condição (que pode ser lançada)**	**Exceção Associada a Pós-Condição (que pode ser lançada)**	**Pós-Condição Excepcional Associada a Violação da Pré-Condição**	**Pós-Condição Excepcional Associada a Violação da Pós-Condição**
Interface1	operação1	pré-condição1	pós-condição1	exceçãoPré1	exceçãoPós1	pós-condiçãoPréE1	pós-condiçãoPósE1
	operação11	pré-condição11	pós-condição11	exceçãoPré11	exceçãoPós11	pós-condiçãoPréE2	pós-condiçãoPósE2
				exceçãoPré11B	exceçãoPós11B	pós-condiçãoPréE3	pós-condiçãoPósE3
Interface2	operação2	pré-condição2	pós-condição2	exceçãoPré2	exceçãoPós2	pós-condiçãoPréE4	pós-condiçãoPósE4

Tabela 3 – TECE para Interfaces Providas

As tabelas TECEp e TECEr são utilizadas em conjunto para representar a especificação excepcional no contrato de realização de um componente. A tabela TECEr de um componente deve apresentar as interfaces requeridas com as operações disponibilizadas por elas, a pré e pós-condição de cada operação assim como as exceções que cada operação deve lançar em função de violações de pré ou pós-condição.

Como pode ser visto na Tabela 4, o componente C1 requer a interface InterfaceX que possui a operação operaçãoX. A operaçãoX possui uma pré-condiçãoX e uma pós-condiçãoX. Ao chamar a operaçãoX, o componente C1 deverá estar preparado para tratar uma exceção do tipo exceçãoPréX, caso ocorra uma violação da pré-condiçãoX. Ao chamar a operaçãoX, o componente C1 deverá estar preparado para tratar uma exceção do tipo exceçãoPósX, caso ocorra uma violação da pós-condiçãoX.

TECEr – Componente C1 / Camada de Sistema					
Interface Requerida	**Operação**	**Pré-Condição**	**Pós-Condição**	**Exceção Associada a Pré-Condição (que deve ser tratada)**	**Exceção Associada a Pós-Condição (que deve ser tratada)**
InterfaceX	operaçãoX	pré-condiçãoX	pós-condiçãoX	exceçãoPréX	exceçãoPósX
InterfaceY	operaçãoY	pré-condiçãoY	pós-condiçãoY	exceçãoPréY	exceçãoPósY

Tabela 4 – TECE para Interfaces Requeridas

3.2 Refinamento da Arquitetura com Uso de Conectores

Como visto, o método MECE propicia a definição de uma arquitetura com especificação normal e excepcional de seus componentes. Desta forma, o método MECE auxilia desenvolvedores a implementarem novos componentes e auxilia integradores a identificarem componentes já existentes para serem reutilizados.

Porém, no caso de integradores que devem buscar componentes já existentes, nada garante que estes integradores encontrem componentes prontos que atendam adequadamente às especificações de componentes definidas na arquitetura, principalmente as especificações que definem o comportamento excepcional destes componentes. Quando isto ocorre, integradores podem ser obrigados a alterar um componente pronto ou a criar um *wrapper*. Neste sentido é interessante existir um refinamento da arquitetura que promova uma alternativa para integradores lidarem com este problema. Este refinamento acontece com o uso de conectores, que são componentes usados para ligar componentes, como definido na subseção 2.3. Esta proposta de refinamento com conectores surgiu durante o estudo de caso que será apresentado no capítulo 4 desta monografia.

Propõe-se a criação de conectores para interligar componentes de camadas diferentes de uma arquitetura. Por exemplo, um conector simples pode ser usado para ligar um componente da camada de sistema com um componente da camada de negócios, como pode ser visto na Figura 36.

Um conector mais complexo poderia ser usado para ligar um componente da camada de sistema a vários componentes da camada de negócios.

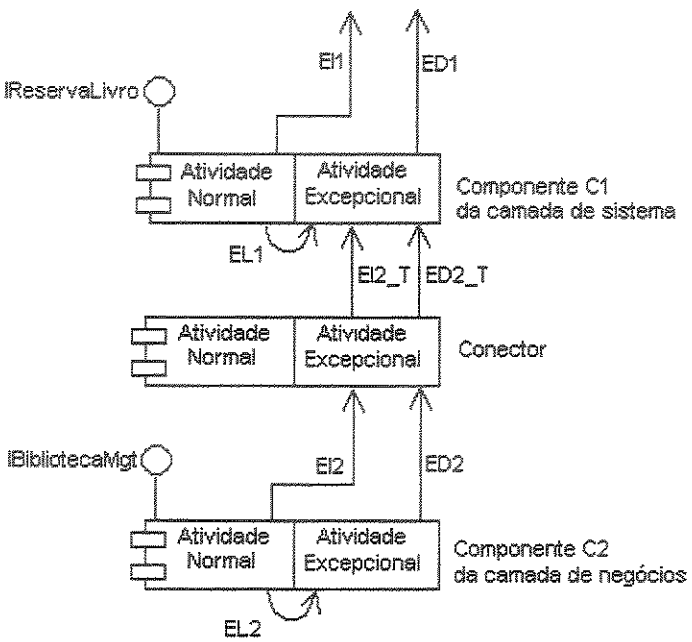

Figura 36 – Componentes Ideais Tolerantes a Falhas para Camada de Sistema e Negócios com Conector

Considerando que um componente da camada de sistema seja um componente cliente e um componente da camada de negócios seja um componente servidor, a vantagem do uso de um conector entre eles é que o mesmo pode traduzir exceção lançadas por um componente servidor em exceções que o componente cliente saiba tratar. Isto é importante, pois caso um integrador encontre um componente que possa ser usado na camada de sistema, mas que não esteja preparado para tratar as exceções lançadas pelo componente servidor da camada de negócios, o integrador não precisaria mudar nem o componente pronto nem o componente da camada de negócios. Ele criaria ou mudaria um conector, fazendo com que o mesmo traduzisse a exceção em algo que o componente já existente entendesse. No caso da Figura 36, considera-se que o componente C1 não sabe lidar com exceções EI2 e ED2, apenas com EI2_T e ED2_T. Por outro lado, caso um integrador encontre um componente da camada de negócios que lance uma exceção que não seja entendida por um componente da camada de sistema, ele poderia criar ou mudar um conector para realizar a tradução.

Também pode-se pensar em um conector contendo uma parte que implementa sua atividade normal e uma parte que implementa sua atividade excepcional. Desta forma, uma exceção de interface ou de defeito que seja lançada por um componente da camada de negócios deve ser tratada pela parte que cuida da atividade excepcional do conector, como pode ser visto na Figura 36. A parte de atividade normal apenas conecta um componente a outro direcionando as chamadas de operações entre eles.

É importante observar que a criação ou alteração dos conectores só é possível caso se conheça as especificações dos componentes e que as mesmas descrevam quais exceções devem ser lançadas e tratadas pelos componentes. Com a aplicação do método MECE pode-se obter estas especificações.

Por mais que um integrador crie ou altere conectores para traduzir exceções, podem existir situações onde componentes prontos contenham falhas de projeto. Para lidar com esta situação e para definir melhor os papéis de tratadores que existem em componentes e conectores, deve-se aplicar as estratégias de estruturação arquitetural para tratamento de exceções, descrita na subseção 2.6.

3.3 Considerações do Capítulo

Este capítulo apresentou o método MECE que é caracterizado pela integração entre a metodologia MDCE e o processo UML Components. Além de atividades definidas na metodologia MDCE e no processo UML Components, o método MECE apresenta novas atividades para a fase de especificação de componentes de uma arquitetura.

O método MECE define a especificação normal e a especificação excepcional de cada componente de uma arquitetura. A especificação excepcional de um componente promove: (1) a declaração explicita dos tipos de exceções, associados a condições excepcionais antecipadas, que podem ocorrer nas operações das interfaces requeridas de um componente; e (2) a declaração explicita na especificação de um componente de todas as exceções que podem ocorrer nas operações das interfaces providas e que possam ser lançadas pelo componente. Estes dois pontos são pré-requisitos para se aplicar a estratégia intra-componente apresentada na seção 2.6.2.

Além disso, o método MECE propõe o uso de conector entre componentes de camadas diferentes, sendo que um conector deve ter uma parte para sua atividade normal e outra para

atividade excepcional. A proposta de uso de conectores caracteriza um pré-requisito para se aplicar a estratégia inter-componente apresentada na seção 2.6.3. Através do uso do método MECE juntamente com o uso das estratégias da seção 2.6, tem-se uma proposta de solução centrada na arquitetura, com tratamento de exceções, para desenvolvimento de sistemas de software confiáveis e baseados em componentes.

Observa-se que a especificação excepcional definida pelo método MECE faz com que os contratos de uso e de realização de componentes apresentem tipos de exceções que podem ser lançadas e tratadas por componentes. Estas informações poderiam ser consideradas detalhadas para o nível de contrato, mas por outro lado isto permite que integradores tomem conhecimento destas exceções. Caso um integrador encontre um componente que atenda a especificação do comportamento normal, mas que não seja capaz de lidar com exceções do sistema, ele poderá alterar um conector que ligue este componente a outro componente do sistema, sem ter que alterar os componentes.

Com relação a limitações do método MECE pode-se observar que pelo fato dele ser baseado no processo UML Components, os componentes especificados pertencem as camadas de sistema e negócios de uma arquitetura. Mais estudos devem ser realizados sobre especificação de componentes das outras camadas superiores e inferiores, e de uso de conectores entre componentes destas outras camadas. Os conectores propostos pelo método MECE são simples e interligam apenas um componente a outro. Mais estudos podem ser realizados com conectores mais complexos que interliguem vários componentes. Além disso, o método MECE se aplica a sistemas com arquiteturas em camadas, e outros estudos devem ser realizados para outros estilos arquiteturais.

Capítulo 4

Estudo de Caso com o Sistema Telestrada

Este capítulo apresenta a aplicação da solução arquitetural proposta nesta monografia em um sistema real, chamado Telestrada. O estudo de caso foi realizado em dois incrementos. No primeiro incremento, aplicou-se o processo UML Components e implementou-se os componentes de acordo com o modelo COSMOS. No segundo incremento aplicou-se o método MECE para as fases de definição de requisitos e especificação dos componentes. Durante a fase de provisionamento e integração do segundo incremento, reutilizou-se componentes desenvolvidos no primeiro incremento, aplicando diretrizes das estratégias de estruturação arquitetural para tratamento de exceções.

As subseções deste capítulo apresentam uma visão geral do sistema Telestrada (subseção 4.1), a descrição das fases e dos objetivos dos incrementos do estudo de caso (subseção 4.2), a descrição das atividades realizadas no primeiro incremento (subseção 4.3), a descrição das atividades realizadas no segundo incremento com a aplicação da solução arquitetural proposta nesta monografia (subseção 4.4), a análise de resultados obtidos no estudo de caso (subseção 4.5), e considerações finais do capítulo (subseção 4.6).

4.1 Visão Geral do Sistema Telestrada

Paises têm construído rodovias por suas extensões territoriais e demandam um sistema capaz de monitorar e gerenciar as condições físicas e os serviços prestados nas rodovias. Telestrada é um sistema que permite monitorar e gerenciar rodovias. É composto por cinco subsistemas:

- Banco de Dados Central: Responsável pelo armazenamento de todas as informações do sistema.
- Gerenciador de Mapas de Rodovias: Responsável pela gerência da representação gráfica das rodovias, em forma de mapas, e de seu monitoramento.
- *Call Center*: Responsável por atender chamadas telefônicas de usuários de rodovias que precisam de informações ou desejam reclamar de algum aspecto.

- Terminais em Rodovias: Cada posto de polícia rodoviária tem um sistema capaz de consultar o banco de dados central para obter informações sobre as rodovias.
- Gerência de Reclamações: Aplicação *Web* responsável por gerenciar as reclamações a respeito das rodovias.

O estudo de caso está baseado na especificação da arquitetura do subsistema de gerência de reclamações e da implementação parcial do mesmo. Através deste subsistema um usuário de rodovia pode registrar uma reclamação sobre um trecho de um rodovia. Um usuário pode consultar sua reclamação e alterá-la se desejar.

As rodovias e trechos são previamente cadastradas no sistema, pelo subsistema de gerência de mapas[7]. Uma reclamação deve estar associada a um tipo específico de reclamação, sendo que tipos de reclamação podem ser cadastrados a qualquer momento pelo supervisor do sistema. Exemplo de tipos de reclamações: obras sem sinalização, buracos na pista, deslizamento de terra e acidente sem socorro. Cada trecho de rodovia possui um responsável por lidar com as reclamações e tomar providências. Um conjunto de reclamações de um trecho, associadas a um tipo de reclamação, podem caracterizar um processo de reclamações que é atribuído ao responsável do trecho, pelo supervisor do sistema.

O supervisor do sistema pode mover reclamações de um processo para outro, remover um processo do sistema e alterar o trecho de uma reclamação caso o mesmo esteja errado. Caso muitos usuários reclamem sobre um mesmo tipo de reclamação de um mesmo trecho, o supervisor pode associar uma resposta padrão a este trecho e tipo de reclamação. Caso um novo usuário deseje reclamar sobre este trecho e tipo de reclamação, ele observará a resposta padrão, podendo então prosseguir com o registro de reclamação ou se dar por satisfeito e não registrar uma nova reclamação.

O subsistema de reclamações foi escolhido por se tratar de uma aplicação *Web*, tendo N camadas, e por ter requisitos de confiabilidade como por exemplo: no registro de uma reclamação sobre um acidente sem socorro ou deslizamento de terra, um responsável pelo trecho tem que tomar as providências o mais rápido possível. Além disso, o sistema estaria disponível na Internet

[7] No momento dos experimentos, o módulo de gerência de mapas não estava implementado. Optou-se por popular o banco de dados com *scripts*.

"24hs / 7 dias" e usuários de rodovias, no futuro, poderão utilizar a aplicação com uso de *handsets*, como um *Palm* ou um telefone celular.

4.2 Os Incrementos do Estudo de Caso

Para estudar os benefícios da solução arquitetural proposta neste trabalho foi necessário preparar um ambiente de desenvolvimento de software e dividir o estudo de caso em dois incrementos.

O ambiente de desenvolvimento deveria ser o mais próximo possível de ambientes utilizados no mercado e a implementação deveria ser em uma linguagem amplamente utilizada. Optou-se pelo desenvolvimento em Java e por configurar um ambiente de desenvolvimento com o servidor de aplicações WebSphere da IBM [IBMWeb], com as ferramentas para desenvolvimento WebSphere Studio e Eclipse [OMondoWeb], e com o banco de dados Oracle. Os dois incrementos utilizaram este ambiente de desenvolvimento de software.

O primeiro incremento foi realizado para se obter componentes a serem reutilizados no segundo incremento, e para exercitar conceitos do processo UML Components e do modelo COSMOS. O segundo incremento foi realizado para se estudar o método MECE e as estratégias de estruturação arquitetural para tratamento de exceções.

No primeiro incremento todas as fases de um processo de desenvolvimento baseado em componentes foram realizadas, como pode ser visto na coluna da esquerda da Figura 37. Aplicou-se o processo UML Components sobre a descrição do subsistema de gerência de reclamação do Telestrada e obteve-se: (1) a definição dos casos de uso do subsistema de gerência de reclamações, com seus cenários primários e alternativos, (2) a definição da arquitetura do subsistema com a especificação normal de seus componentes.

Uma vez especificados os componentes, realizou-se a fase de provisionamento dos componentes, os quais foram implementados em Java e seguindo o modelo COSMOS, introduzido na subseção 2.7. Após o provisionamento teve-se a integração dos componentes com o desenvolvimento de conectores simples, onde cada conector interligava apenas dois componentes de camadas diferentes.

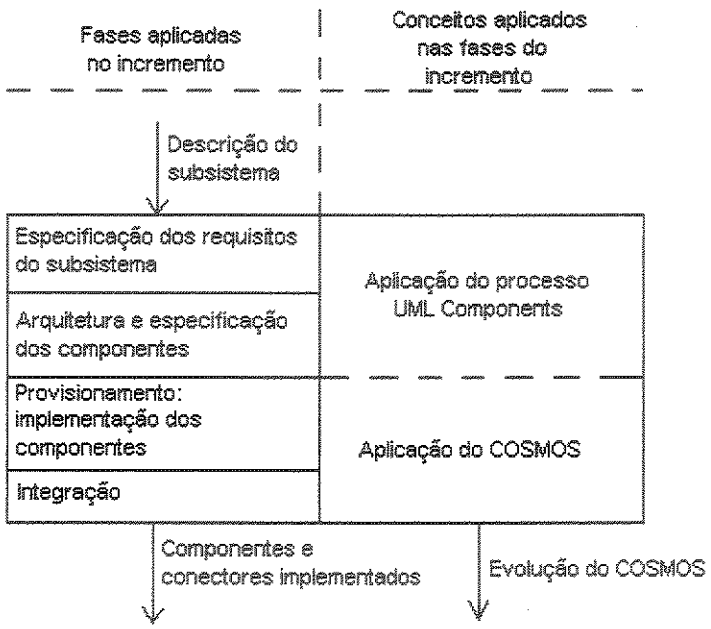

Figura 37 – Fases do Incremento 1 do Estudo de Caso

Ao se terminar este incremento obteve-se os componentes e conectores implementados e interligados, caracterizando uma primeira versão do subsistema de gerência de reclamações. Como comentado, a implementação do subsistema foi parcial, ou seja, nem todos os casos de uso especificados através do processo UML Components, foram implementados. Além disso, foi possível estudar e propor uma evolução para o COSMOS para lidar com tipos de dados comuns utilizados por diferentes componentes, como pode ser visto em [Silva03].

No segundo incremento todas as fases de um processo de desenvolvimento baseado em componentes foram realizadas novamente, como pode ser visto na coluna da esquerda da Figura 38. Aplicou-se o método MECE sobre a descrição do subsistema de gerência de reclamação do Telestrada e obteve-se: (1) a definição dos casos de uso do subsistema com seus cenários primários, alternativos e excepcionais, (2) a definição da arquitetura do subsistema com a especificação normal e excepcional de cada componente, e (3) a proposta de uso de conectores simples para interligar componentes.

Na fase de provisionamento, reutilizou-se os componentes implementados no primeiro incremento, aplicando diretrizes das estratégias de estruturação arquitetural para tratamento de exceções apresentadas na subseção 2.6. Após o provisionamento, integrou-se os componentes e obteve-se uma nova versão do subsistema de gerência de reclamações do Telestrada. Aproveitou-

se estas fases de provisionamento e integração do segundo incremento para realização de diferentes exercícios de reutilização de componentes. Inicialmente foram realizados três exercícios associados ao caso de uso de registro de reclamações do subsistema de gerência de reclamações.

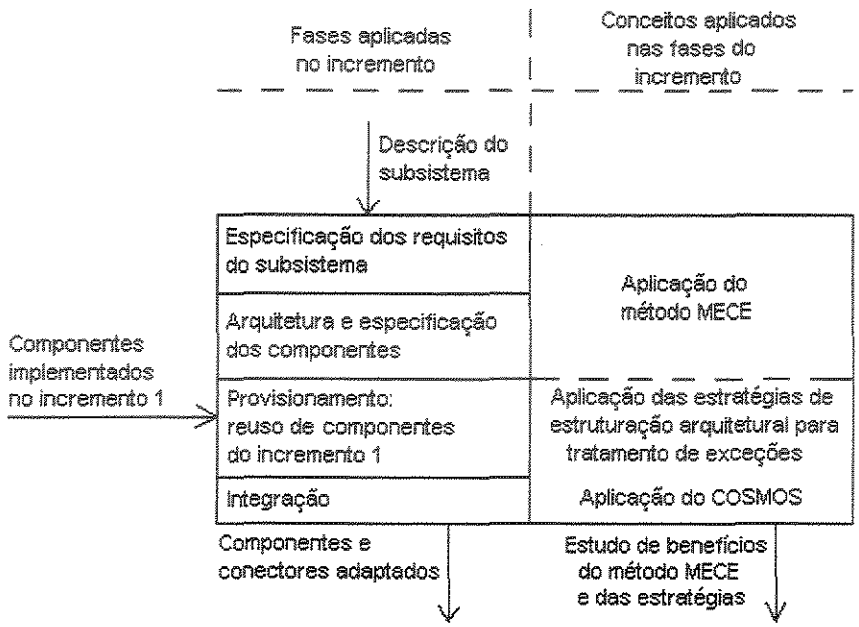

Figura 38 – Fases do Incremento 2 do Estudo de Caso

Como pode ser visto na Figura 39, num primeiro exercício considerou-se que o integrador teria acesso ao código de um componente reutilizável da camada de sistema, implementado no primeiro incremento. Num segundo exercício considerou-se que o integrador não teria acesso ao código de um componente reutilizável da camada de negócios, também implementado no primeiro incremento. Um terceiro exercício envolveu a integração destes componentes, com uso de um conector, para se obter a funcionalidade de registro de reclamação do subsistema de gerência de reclamações.

Além destes três exercícios, outros foram realizados. Exercícios de injeção de falhas em componentes e conectores foram realizados para se observar a reação do subsistema na presença de falhas, durante o registro de uma reclamação. Exercícios para se estudar especificamente os benefícios da estratégia inter-componente foram realizados, considerando-se que um integrador não tivesse acesso aos códigos de componentes tanto da camada de sistema quanto da camada de negócios.

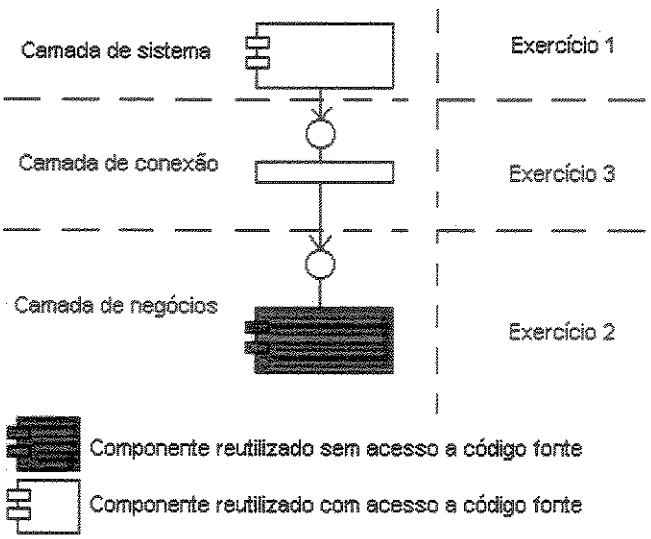

Figura 39 – Exercícios de Reutilização do Incremento 2 do Estudo de Caso

Com este conjunto de exercícios do segundo incremento, foi possível estudar situações de reuso de componentes tendo acesso ao código fonte de um componente e não tendo acesso ao código fonte, além de observar a reação dos componentes na presença de falhas. Desta forma foi possível estudar: (1) os benefícios de se aplicar o método MECE para se definir arquitetura com especificação normal e excepcional, permitindo que os componentes pudessem ser implementados ou identificados para reutilização, e (2) os benefícios das estratégias de estruturação arquitetural para tratamento de exceções, permitindo que as adaptações necessárias fossem feitas nos componentes reutilizáveis e nos conectores. As próximas subseções apresentam as atividades realizadas no primeiro e no segundo incremento.

4.3 Estudo de Caso – Incremento 1

Com base na descrição do subsistema de gerência de reclamações em 4.1, iniciou-se a aplicação das fases de especificação de requisitos e de componentes, definidas no processo UML Components. Posteriormente teve-se as fases de provisionamento, onde os componentes foram implementados de acordo com o modelo COSMOS, e a fase de integração destes componentes.

4.3.1 Especificação dos Requisitos do Subsistema

Com base na descrição do Telestrada definiu-se o processo de negócio, o modelo conceitual de negócio e os casos de uso. O modelo conceitual e os casos de uso são apresentados a seguir.

Através do modelo conceitual de negócio, que pode ser visto na Figura 40, pode-se observar que: uma rodovia pode pertencer a várias unidades de federação (pode passar por vários estados); uma unidade de federação pertence a um país; uma rodovia possui vários trechos; um trecho pode ter várias reclamações associadas a ele e cada reclamação deve ser de um determinado tipo de reclamação; um tipo de reclamação pode estar associado a várias reclamações; um usuário do sistema com o papel de usuário de rodovia pode registrar várias reclamações; um usuário do sistema com o papel de responsável por trecho de rodovia, pode tomar conta de vários trechos. Além disso: uma reclamação pode fazer parte de um processo de reclamações; um processo está associado a várias reclamações e a um determinado tipo de reclamação; um processo tem um meio de ser encaminhado para um responsável pelo trecho com reclamações.

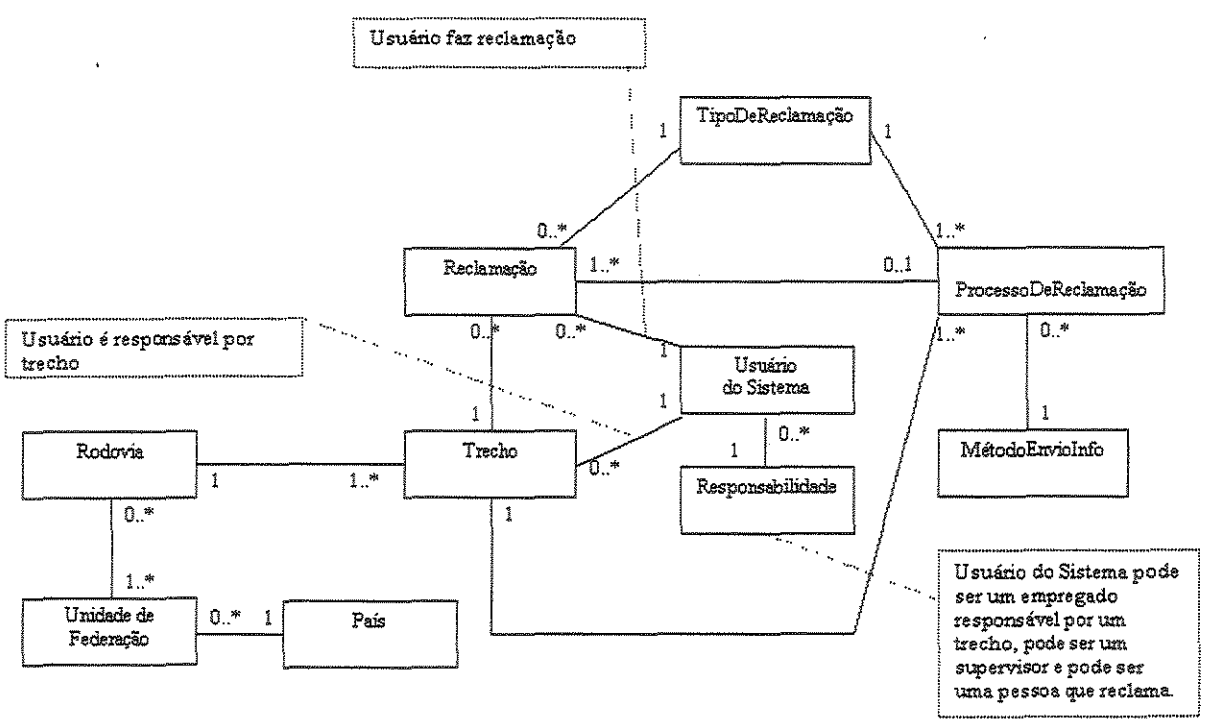

Figura 40 – Modelo Conceitual de Negócios do Subsistema de Gerência de Reclamações do Telestrada

Foram definidos nove casos de uso para o subsistema de gerência de reclamações do Telestrada. Dois deles são apresentados aqui pois foram os casos de uso implementados nos estudos de caso. São eles: 'registro de reclamação' e 'consulta de reclamação'. Os demais casos de uso são apresentados no Apêndice A.

Para efeito de simplificação, as pré e pós-condições são descritas informalmente, sem uma linguagem específica para este fim. Para as extensões dos casos de uso, ou cenário alternativos, utiliza-se os indicadores de parada adotados pelo UML Components [Chessman+01], que são os apresentados a seguir. Se nenhum deles é utilizado, significa que o próximo passo a ser executado (após o cenário alternativo) é o passo do cenário primário, a partir de onde se originou o primeiro passo do cenário alternativo.

- *Fail*: Indica que o cenário alternativo terminou sem atingir o objetivo do caso de uso.
- *Stop*: Indica que o cenário alternativo terminou e atingiu o objetivo do caso de uso.
- *Resume to Step N*: Indica que o cenário alternativo terminou e o próximo passo é o passo N do cenário primário.

O caso de uso de registro de reclamação com seu cenário primário e alternativo é apresentado no Quadro 1, abaixo. Optou-se por já apresentar as pré e pós-condições.

Caso de Uso: Registro de Reclamação
Iniciador: Usuário de rodovia
Objetivo: Permitir a inclusão de reclamação a respeito de trecho de rodovia.
Cenário de Sucesso:

1. Usuário visita página do Telestrada.
2. Usuário seleciona a opção 'registrar reclamação'.
3. Sistema exibe lista de rodovias.
4. Usuário seleciona uma rodovia.
5. Sistema exibe trechos da rodovia selecionada e tipos de reclamação cadastrados.
6. Usuário seleciona um trecho e um tipo de reclamação.
7. Usuário seleciona próximo passo.
8. Sistema verifica a existência de uma resposta padrão para o dado trecho e tipo da reclamação do usuário.
9. Sistema apresenta resposta padrão para usuário (se existir), juntamente com um formulário que permita o usuário preencher a descrição da reclamação e seus dados pessoais (Nome e e-mail).
10. Usuário preenche os dados do formulário.
11. Usuário seleciona opção 'incluir reclamação no sistema'.
12. Sistema inclui reclamação, exibe número de registro da reclamação e instruções para usuário acompanhar o status da reclamação no sistema.

Cenário Alternativo:

9. Usuário fica satisfeito com resposta padrão e
a. seleciona opção de cancelar inclusão de reclamação.
b. *Stop*

Pré-condições:

- Devem existir rodovias no sistema.
- Devem existir tipos de reclamação no sistema.
- Devem existir trechos para um rodovia selecionada pelo usuário, com identificador (id) válido.
- Identificadores de trecho e tipo devem ser válidos para pesquisa de resposta padão.
- Dados de reclamação e usuário não são vazios para a inserção.

Pós-condições:

- Rodovias devem ser selecionadas e retornadas.
- Tipos de reclamação devem ser selecionados.
- Trechos devem ser selecionados e retornados.
- Uma resposta padrão deve ser mostrada na tela.
- Ao se inserir dados de usuário, e antes de inserir a reclamação, deve ser gerado um id único para usuário. Um novo usuário deve ser inserido no banco de dados. Uma nova reclamação deve ser inserida no banco de dados. Um id único de reclamação deve ser apresentado para o usuário.

Quadro 1 - Caso de Uso para Registro de Reclamação

O caso de uso de consulta de reclamação com seu cenário primário e cenários alternativos é apresentado no Quadro 2, abaixo.

Caso de Uso: Consulta de Reclamação
Iniciador: Usuário de rodovia
Objetivo: Permitir a consulta sobre dados de reclamação.
Cenário de Sucesso:

1. Usuário acessa página do Telestrada.
2. Usuário seleciona a opção 'consultar reclamação'.
3. Usuário informa número da reclamação.
4. Sistema apresenta reclamação para usuário, contendo: trecho de rodovia e situação em que se encontra a reclamação ('nova', 'sendo analisada' ou 'resolvida') e dados do usuário que incluiu reclamação.
5. Usuário escolhe imprimir a reclamação.
6. Sistema formata relatório e envia para impressora do usuário.

Cenários Alternativos:

5. Usuário escolhe receber por e-mail os dados da reclamação.
   a. Sistema abre formulário para usuário definir endereço do destinatário a receber o mail.
   b. Sistema formata e envia o mail.
   c. *Stop*.
5. Usuário pode escolher opção de alterar suas informações e/ou informações da reclamação.
   a. Usuário salva as informações.
   b. *Stop*.

Pré-condições:

- Id da reclamação inserido pelo usuário deve ser um número válido.
- Impressora deve estar configurada corretamente.
- Se usuário escolher enviar mail, dados válidos devem ser fornecidos.
- Se usuário escolher alterar dados, dados válidos de usuário e reclamação são passados para alteração.

Pós-condições:

- Se Id da reclamação válido, a reclamação é apresentada para usuário. Se Id da reclamação não válido, a reclamação não é apresentada para usuário.
- Reclamação é impressa.
- Se usuário escolher enviar mail, mail é enviado.
- Se usuário escolher alterar dados. os mesmos são alterados sistema.

Quadro 2 - Caso de Uso para Consulta de Reclamação

Após a definição destes casos de uso, iniciou-se a atividade de verificação se novos casos de uso seriam necessários para lidar com informações que deveriam estar cadastradas no sistema. Neste momento definiu-se mais dois casos de uso para registro e remoção de tipo de reclamação, que também podem ser vistos no Apêndice A.

Uma vez definidos os casos de uso e o modelo conceitual de negócios, iniciou-se a fase de especificação de componentes do subsistema de gerência de reclamações.

4.3.2 Arquitetura e Especificação dos Componentes

O subsistema de gerência de reclamação é uma aplicação *Web*. Sua arquitetura de software é baseada no estilo arquitetural em camadas. A camada de mais alto nível é a camada de Interface

para lidar com interface gráfica apresentada para usuário. Em seguida tem-se uma camada de Diálogo com usuário para lidar com controle de sessões de usuários, seguida das camadas de Serviços de Sistema, de Serviços de Negócios, e de acesso a banco de dados, como pode ser visto na Figura 41.

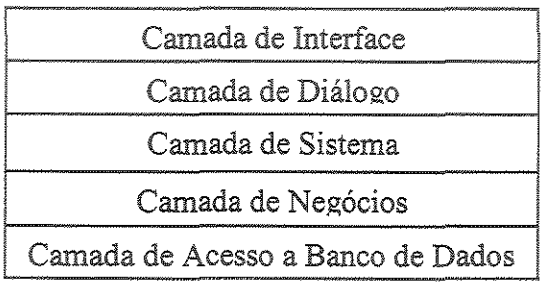

Figura 41 – Camadas da Arquitetura do Subsistema de Gerência de Reclamações do Telestrada

A camada de Interface e de Diálogo foram projetadas e implementadas através do *framework Struts* [JakartaWeb] que permite o rápido desenvolvimento de páginas *Web* e jsp [JSPWeb] com formulários integrados a ações que são executadas no servidor de aplicações. A camada de acesso a banco foi projeta e implementada com o uso do padrão de projeto DAO (Database Access Object) [Alur+01], que permite flexibilidade para alterações de banco de dados a serem utilizados.

As camadas de Sistema e Negócios foram projetadas de acordo com o processo UML Components. Esta subseção apresenta os estágios de identificação, interação e especificação dos componentes das camadas de sistema e negócios do subsistema de gerência de reclamações.

Identificação dos Componentes

Este estágio foi iniciado com a definição das interfaces da camada de sistema com base na análise dos cenários dos casos de uso. As interfaces da camada de sistema e suas operações associadas aos casos de uso de registro e consulta de reclamação são apresentadas a seguir. Neste estágio ainda não se tem a definição dos parâmetros das operações das interfaces da camada de sistema.

- Interfaces IRegistroReclamacao e IOperacoesGenericas

O caso de uso de registro de reclamação possui vários passos onde o sujeito da oração é o sistema. O primeiro deles é consultar rodovias registradas no sistema. Este passo deu origem a

operação consultarRodovias() que deveria pertencer à interface IRegistroReclamacao. Mas sabendo-se que esta operação também seria oferecida por outros componentes, optou-se por colocá-la em uma interface genérica chamada de IOperacoesGenericas. Outros passos são os de selecionar trechos de uma rodovia e selecionar tipos de reclamação. Estes passos levaram a definição das operações consultarTrechos() e consultarTiposReclamacao() da interface IOperacoesGenericas. Os outros dois passos que são para consultar de resposta padrão e efetivar registro levaram a definição das operações consultarRespostaPadrão() e efetivarRegistro() da interface IRegistroReclamacao.

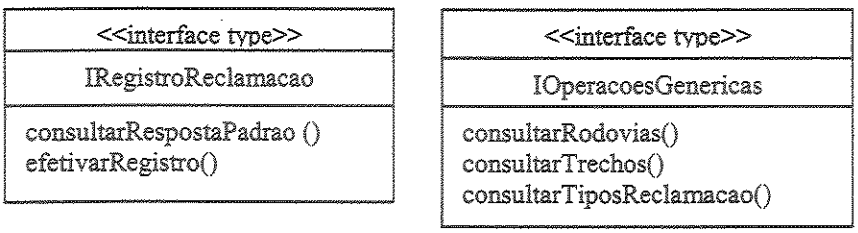

Figura 42 – Interfaces IRegistroReclamacao e IOperacoesGenericas

- Interfaces IConsultaReclamacao e IAlteracaoReclamacao

Estas interfaces estão associadas ao caso de uso de consulta de reclamação. Os passos em que o sistema exibe os dados de uma reclamação, permite impressão dos dados de uma reclamação e permite envio de uma mensagem com dados de uma reclamação levaram a definição, respectivamente, das operações exibirDadosReclamacao(), imprimirReclamacao(), enviarEmailReclamacao(), da interface IConsultaReclamacao. O cenário alternativo que permite alterar dados de uma reclamação tem um passo que levou a definição da operação alterarDadosReclamacao() da interface IAlteracaoReclamacao. A interface IAlteracaoReclamacao foi definida pois sabia-se que seria utilizada em outro caso de uso.

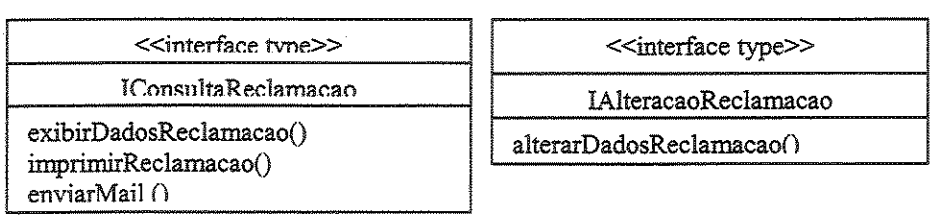

Figura 43 – Interfaces IConsultaReclamacao e IAlteracaoReclamacao

Após a identificação das interfaces da camada de sistema, definiu-se as interfaces da camada de negócios. Para tal, teve-se como base o modelo de tipos de negócios, que é uma

evolução do modelo conceitual de negócios, como é explicado em [Chessman+01]. Aqui ele é apresentado sem os atributos de cada entidade, para efeito de simplificação, como pode ser visto na Figura 44.

Neste momento identificou-se quem são os tipos fundamentais. Para cada tipo fundamental definiu-se um tipo de interface da camada de negócios. Neste caso, teve-se os tipos fundamentais: Reclamação, Processo de Reclamação, Usuário, e as interfaces IReclamacaoMgt, IProcessoReclamacaoMgt, IUsuarioMgt, respectivamente. Optou-se por criar uma interface ITipoReclamacaoMgt para distribuir a responsabilidade que estava com a IReclamacaoMgt. Além destes tipos, as entidades rodovia e trecho foram consideradas tipos fundamentais, pois outros subsistemas do Telestrada poderão lidar apenas com interfaces da camadas de negócios que gerenciam informações de rodovias e trechos. Definiu-se então as interfaces: IRodoviaMgt e ITrechoMgt.

Assumiu-se nesta especificação, que já existem interfaces da camada de negócios para envio de menssagem e impressão: IEmailMgt e IImpressoraMgt. Neste subsistema não se utilizou interfaces para lidar com País e Unidade Federal.

Para oferecer as interfaces da camada de negócios, definiu-se componentes que possuem o sufixo Mgr: ReclamacaoMgr, UsuarioMgr, ProcessoReclamacaoMgr, RodoviaMgr e TrechoMgr. Assumiu-se que os componentes IImpressoraMgr e IEmailMgr já existiam e não foram implementados neste estudo de caso.

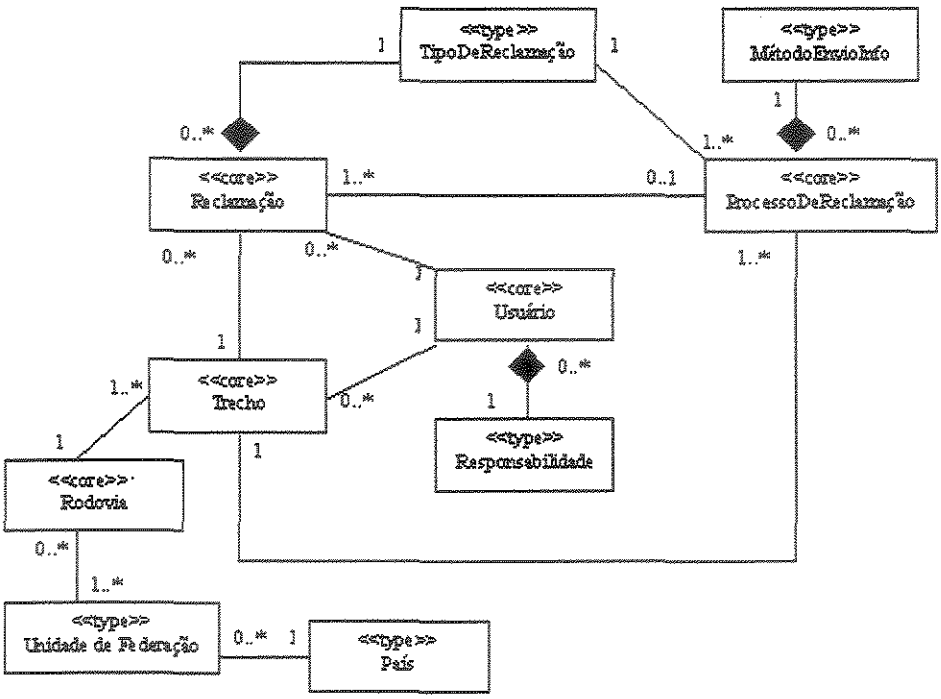

Figura 44 – Modelo de Tipos de Negócios para o Subsistema de Gerência de Reclamações

Uma vez definidas as interfaces das camadas de sistema e negócios, iniciou-se a especificação da arquitetura inicial. A seguir são apresentados os componentes, e as interfaces que os mesmos oferecem, para os casos de uso de registro e consulta de reclamação. Para os diagramas associados aos demais casos de uso, consultar Apêndice B. As camada de interface e diálogo, definida com *Struts*, são abstraídas nas figuras, na forma de um componente, e a camada de banco utilizada pela camada de negócios não é apresentada para efeito de simplificação. Normalmente cada componente deveria ser apresentado com estereótipo <<comp spec>>, mas optou-se nesta monografia por apresentá-los com um estereótipo que indicasse a que camada da arquitetura o componente pertence, ou <<apresentação>>, ou <<sistema>>, ou <<negócios>>.

- Registro de Reclamação

Durante um registro, ocorrem as seguintes requisições a partir da camada de apresentação do subsistema de gerência de reclamações:

- consultar rodovias: leva a uma requisição da lista de rodovias para um componente da camada de sistema que oferece a interface IOperacoesGenericas, que por sua vez faz

uma requisição para um componente da camada de negócios, que oferece a interface IRodoviaMgt.

- consultar trechos: leva a uma requisição da lista de trechos para um componente da camada de sistema que oferece a interface IOperacoesGenericas, que por sua vez faz uma requisição para um componente da camada de negócios, que oferece a interface ITrechoMgt.
- consultar tipos de reclamação: leva a uma requisição da lista de tipos para um componente da camada de sistema que oferece a interface IOperacoesGenericas, que por sua vez faz uma requisição para um componente da camada de negócios, que oferece a interface ITipoReclamacaoMgt.
- registro da reclamação: leva a uma requisição de registro de reclamação para um componente da camada de sistema que oferece a interface IRegistroReclamacao. Este componente faz duas outras requisições: uma para um componente da camada de negócios que oferece a interface IReclamacaoMgt para registro de reclamação, e outra para um componente da camada de negócios que oferece a interface IUsuarioMgt para o registro do usuário que faz uma reclamação.

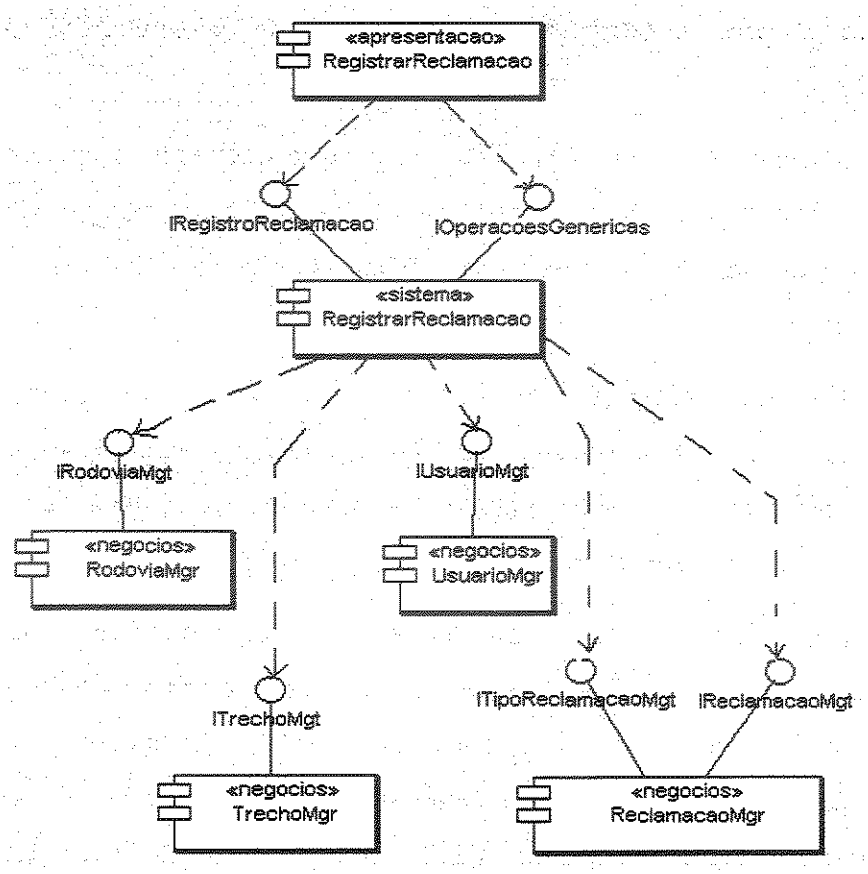

Figura 45 – Arquitetura Inicial de Registro de Reclamação

Um possível refinamento para a arquitetura seria ter um componente separado para oferecer a interface IOperacoesGenericas e outro separado para oferecer a interface ITipoReclamacaoMgt, podendo ser reaproveitados para a especificação de diferentes casos de uso.

- Consulta de Reclamação

A consulta de reclamação também permite que o usuário altere dados do próprio usuário ou da reclamação. Nesta primeira versão dos requisitos, não existe definição de que status uma reclamação deve estar para que seus dados possam ser alterados. Durante uma consulta, ocorrem as seguintes requisições a partir da camada de apresentação do subsistema de gerência de reclamações:

- selecionar reclamação: leva a uma requisição de consulta de uma reclamação para um componente da camada de sistema que oferece a interface IConsultaReclamacao, que

por sua vez faz uma requisição para um componente da camada de negócios, que oferece a interface IReclamacaoMgt.

- alterar dados de reclamação ou usuário: leva a uma requisição de alteração de dados para um componente da camada de sistema que oferece a interface IAlteracaoReclamacao. Este componente faz outras duas requisições: uma para um componente da camada de negócios que oferece a interface IReclamacaoMgt para alteração de dados de uma reclamação, e outra para um componente que oferece a interface IUsuarioMgt para alteração de dados de um usuário.
- enviar dados de reclamação por mensagem eletrônica: leva a uma requisição de envio de dados de uma reclamação para um componente da camada de sistema que oferece a interface IConsultaReclamacao, que por sua vez faz uma requisição para um componente da camada de negócios, que oferece a interface IEmailMgt.
- imprimir dados de reclamação: leva a uma requisição de impressão de dados de reclamação para um componente da camada de sistema que oferece a interface IConsultaReclamacao, que por sua vez faz uma requisição para um componente da camada de negócios, que oferece a interface IImpressoraMgt.

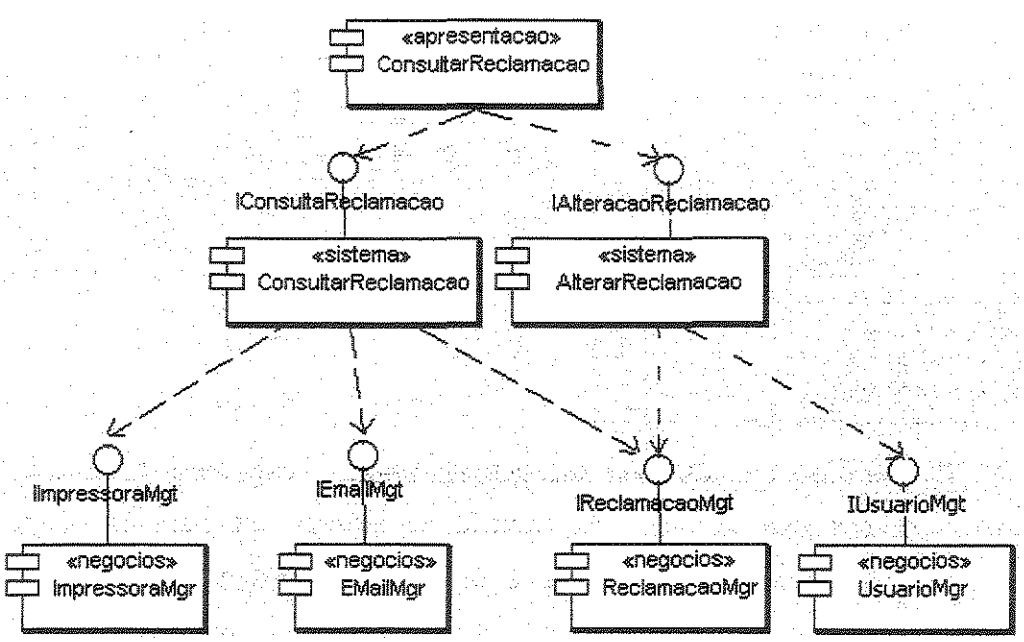

Figura 46 – Arquitetura Inicial de Consulta de Reclamação

Interação dos Componentes

Uma vez definida a arquitetura inicial dos componentes e interfaces, bem como as operações das interfaces da camada de sistema, iniciou-se o estágio de interação dos componentes do processo UML Components. Definiu-se os tipos de dados utilizados como parâmetros e retornos das operações e criou-se diagramas de colaboração com interações entre as camadas de sistema e negócios, identificando seus parâmetros e retornos. Desta forma foi possível se definir as operações das interfaces da camada de negócios.

Para os casos de uso de registro e consulta de reclamação, definiu-se os seguintes tipos de dados, representados pelo estereótipo <<data type>>:

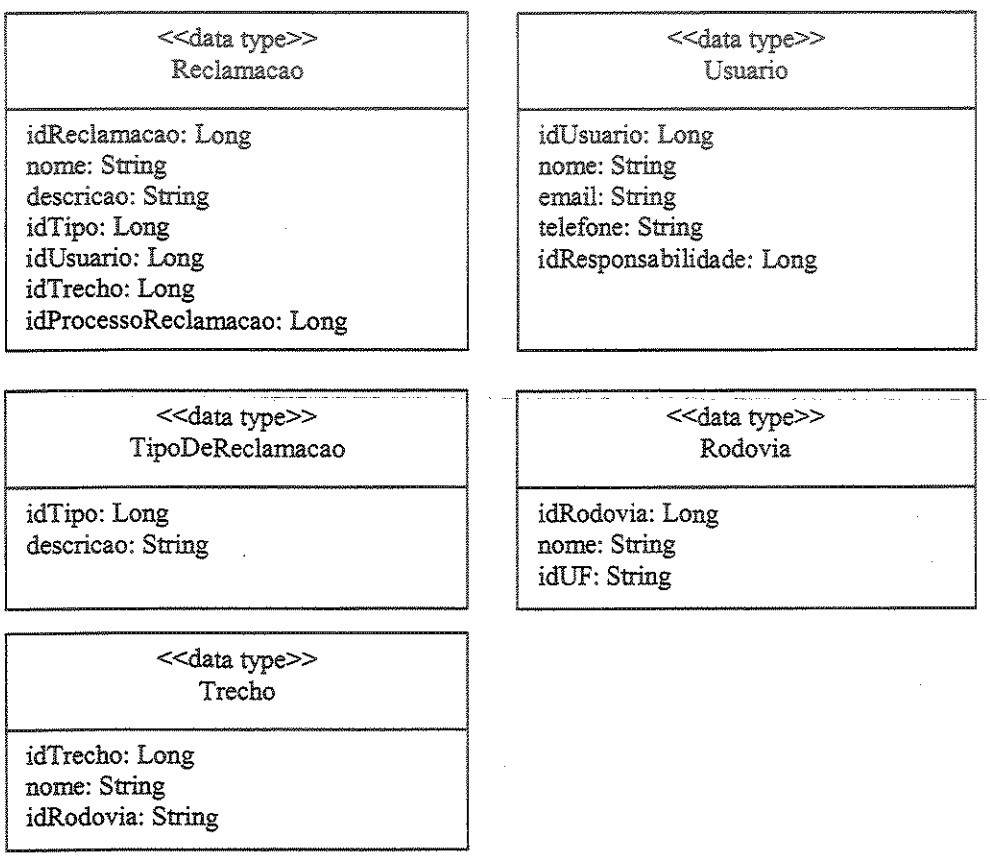

Figura 47 – Tipos de Dados Utilizados nos Casos de Uso de Registro e Consulta de Reclamação

Para cada operação que se inicia na camada de sistema criou-se um diagrama de colaboração indicando que operações o componente da camada de sistema requer da camada de negócios. A seguir apresenta-se os diagramas de colaboração definidos para os casos de uso de registro e consulta de reclamação.

- Registro de Reclamação

Um componente da camada de sistema ou negócios é apresentado, nos diagramas, na forma de: "/TipoDeInterfaceOferecida:NomeComponente". As próximas figuras apresentam os diagramas de colaboração para as consultas de rodovias, trechos e tipos respectivamente, pertencentes ao caso de uso de registro de reclamação.

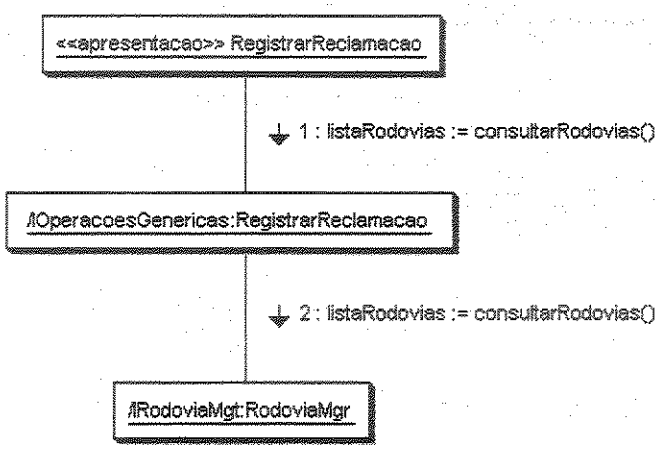

Figura 48 – Diagrama de Colaboração para Consultar Rodovias

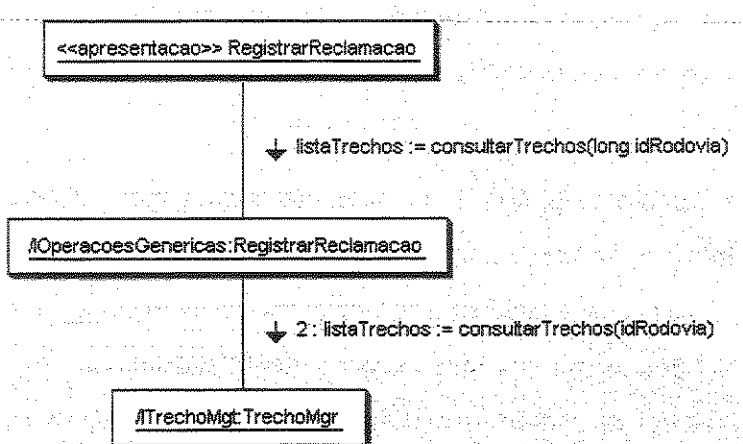

Figura 49 – Diagrama de Colaboração para Consultar Trechos

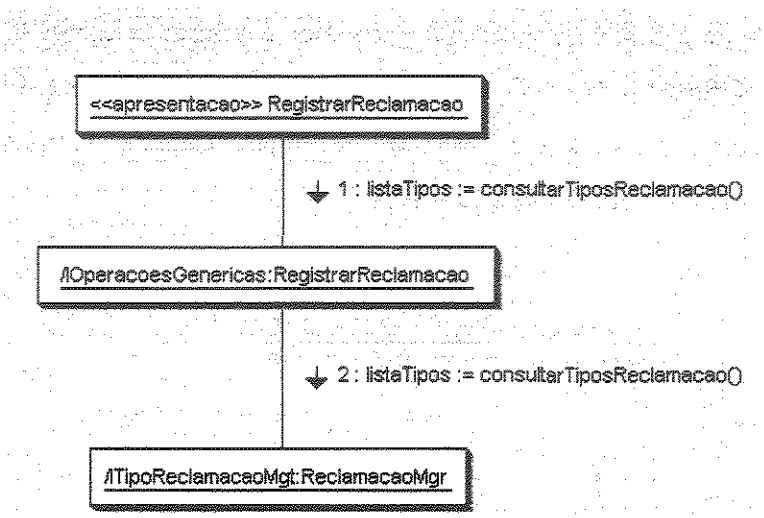

Figura 50 – Diagrama de Colaboração para Consultar Tipos

Estes três diagramas apresentados nas Figura 48, Figura 49 e Figura 50 apresentam métodos novos para as interfaces da camada de negócios: consultarRodovias() para a interface IRodoviaMgt, consultarTrechos() para a interface ITrechoMgt e consultarTiposReclamacao() para a interface ITipoReclamacaoMgt. O método consultarRodovias() retorna uma lista de tipos Rodovia. O método consultarTrechos(long idRodovia) retorna uma lista de tipos Trecho para uma dada rodovia. O método consultarTiposReclamacao() retorna uma lista de tipos TipoDeReclamacao.

A Figura 51 apresenta o diagrama de colaboração para o método de consulta de resposta padrão para um determinado tipo e trecho de rodovia. Uma nova operação para a interface IReclamacaoMgt foi definida: consultarRespostaPadrao(long idTipo, long idTrecho). Este método recebe como parâmetros tipos primitivos indicando os identificadores de tipo de reclamação e de um trecho, e retorna uma resposta padrão associada ao trecho e ao tipo de reclamação. Este retorno é do tipo primitivo *String*.

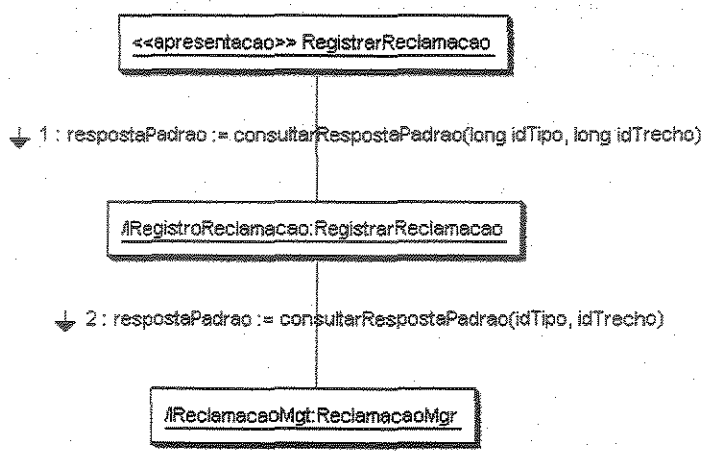

Figura 51 – Diagrama de Colaboração para Consultar Resposta Padrão

A seguir, tem-se o diagrama para efetivar o registro de uma reclamação. De acordo com o diagrama, primeiramente registra-se o usuário. Caso o registro seja realizado sem erro, registra-se a reclamação. Percebe-se que o método efetivarRegistro da interface IRegistroReclamacao possui dois argumentos: Usuário e Reclamação. O diagrama apresenta uma nova operação para a interface IUsuarioMgt: registrarUsuario(Usuario: u) que retorna um identificador de usuário. Também apresenta uma nova operação para a interface IReclamacaoMgt: registrarReclamacao(long idUsuario, Reclamacao: r) que registra a reclamação para um dado usuário e retorna um identificador da reclamação que é então retornado para a camada de sistemas. Este identificador será apresentado para o usuário pela camada de Interface do sistema e permitirá que o usuário consulte sua reclamação no sistema.

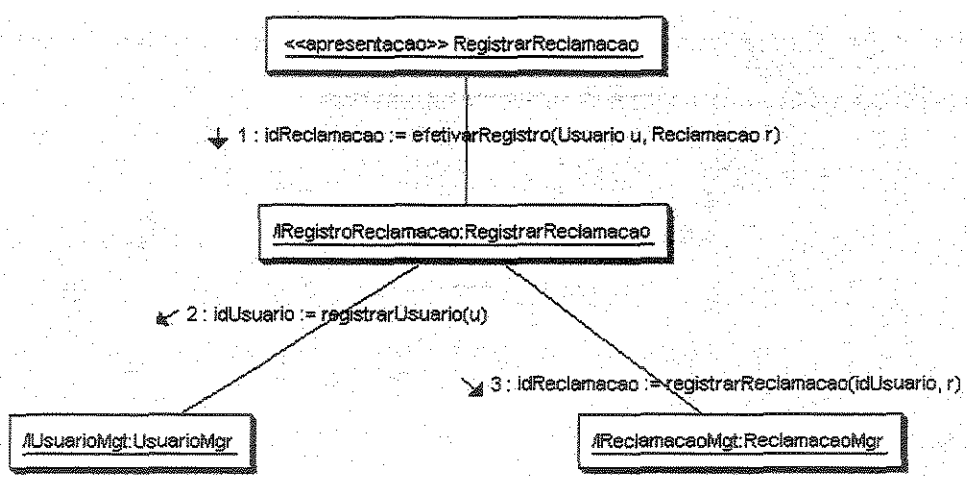

Figura 52 – Diagrama de Colaboração para Registro de Reclamação

- Consulta de Reclamação

A seguir são apresentados os diagramas de colaboração associados ao caso de uso de consultar reclamação. A Figura 53 apresenta uma nova operação para a interface IReclamacaoMgt: exibirDadosReclamacao, que recebe como parâmetro o identificador da reclamação e retorna os dados da reclamação.

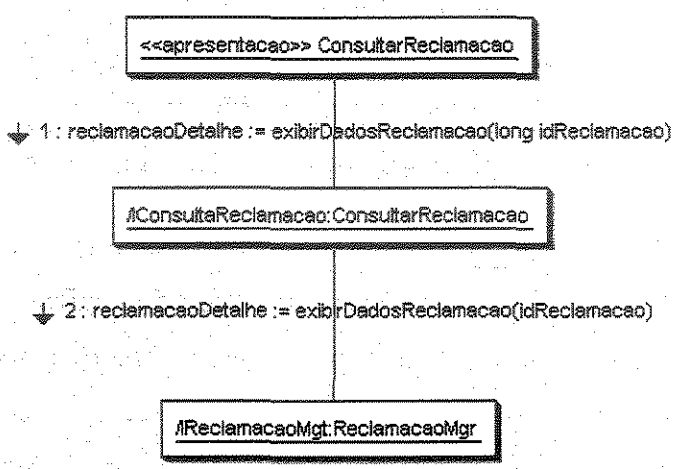

Figura 53 – Diagrama de Colaboração para Exibição de Dados de Reclamação

A Figura 54 apresenta uma nova operação para a interface IReclamacaoMgt: alterarDadosReclamacao, que recebe como parâmetro um tipo Reclamacao que contém um identificador da reclamação a ser alterada. Este diagrama também define uma nova operação para a interface IUsuarioMgt: alterarDadosUsuario, que recebe como parâmetro um tipo Usuario que contém um identificador do usuário cujos dados serão alterados.

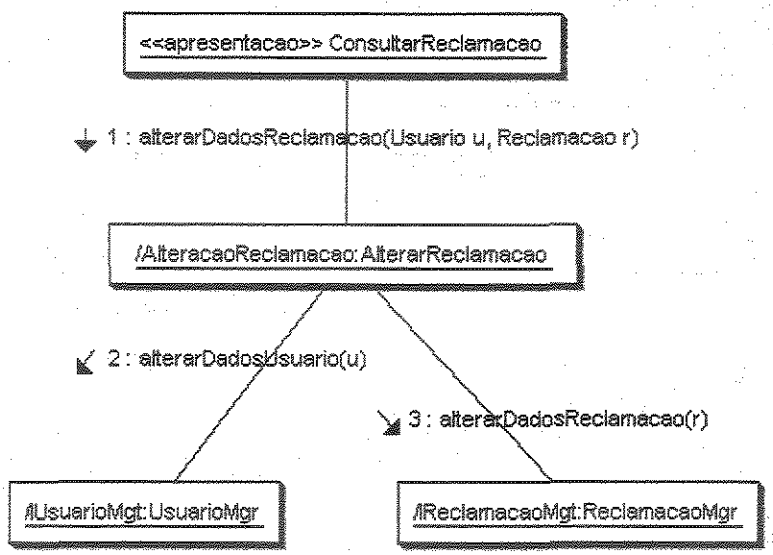

Figura 54 – Diagrama de Colaboração para Alteração de Reclamação Após Consulta

A Figura 55 apresenta uma nova operação para a interface IImpressoraMgt: imprimirReclamacao, que recebe como parâmetro uma String formatada contendo a reclamação.

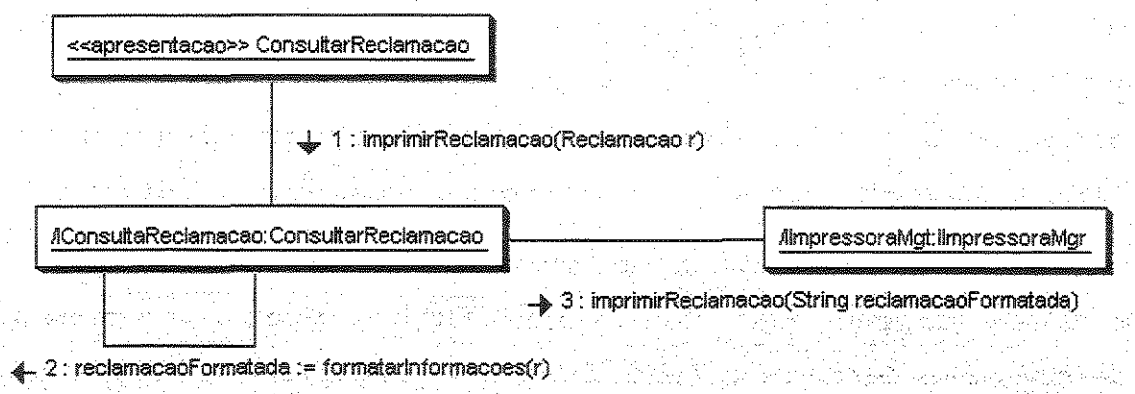

Figura 55 – Diagrama de Colaboração para Impressão de Dados de Reclamação

A Figura 56 apresenta uma nova operação para a interface IEmailMgt: enviarMail, que recebe como parâmetro uma String formatada contendo a mensagem com a reclamação.

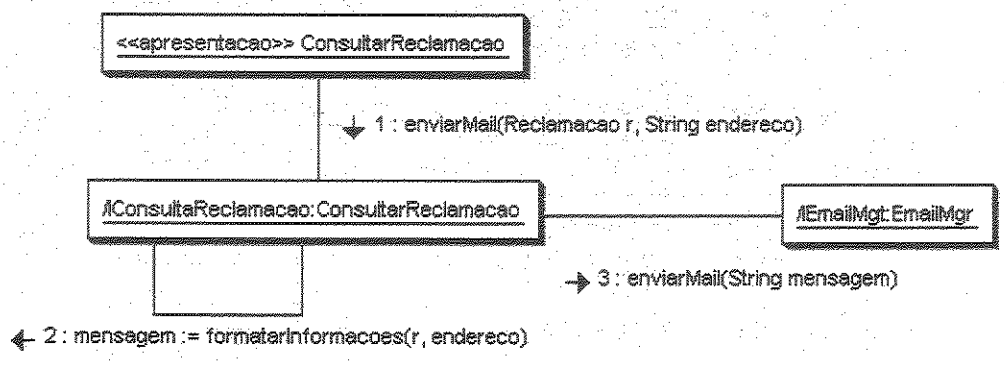

Figura 56 – Diagrama de Colaboração para Envio de Mensagem sobre Reclamação

Desta forma, após os estágios de identificação e interação de componentes, teve-se as interfaces das camadas de sistema e negócios, com suas operações, e componentes que oferecem estas interfaces. A partir deste momento iniciou-se a especificação dos componentes.

Especificação dos Componentes

Neste estágio definiu-se os contratos de uso e de realização de cada componente da arquitetura. Os contratos são apresentados aqui de forma simplificada. O contrato de uso será apresentado aqui com as interfaces providas por cada componente, as operações destas interfaces com suas pré e pós-condições. Não se apresenta aqui as alterações dos estados dos componentes durante as chamadas de operações. O contrato de realização é apresentado aqui com as interfaces providas de cada componente e com as operações das interfaces requeridas por cada componente.

O contrato de uso dos componentes será apresentado em forma de tabela e as pré e pós-condições de modo não formal, como pode ser visto nas Tabela 5 e Tabela 6. Na primeira coluna tem-se os componentes associados aos casos de uso de registro e consulta de reclamação. A segunda coluna apresenta as interfaces providas por estes componentes. A terceira coluna apresenta operações associadas a estas interfaces. A quarta coluna e a quinta apresentam pré e pós-condições respectivamente para cada operação.

A Tabela 5 apresenta as interfaces e componentes da camada de sistema. As pré e pós-condições são baseadas nas pré e pós-condições identificadas nos casos de uso.

Componente que oferece Interface	Interface Provida	Operação	Pré-Condição	Pós-Condição
RegistrarReclamacao	IOperacoesGenericas	listaRodovias: consultarRodovias()	Rodovias devem existir no sistema	Uma lista de rodovias é retornada.
		listaTrechos: consultarTrechos(long idReclamacao)	Trechos devem existir no sistema	Uma lista de trechos da rodovia especificada é retornada.
		listaTipos: consultarTiposReclamacao()	Tipos de Reclamação devem existir no sistema	Uma lista de tipos é retornada.
	IRegistroReclamacao	resposta: consultarRespostaPadrao(long idTrecho, long idTipo)	Identificadores válidos de trecho e tipo devem ser passados.	Uma resposta padrão ou string vazia é retornada.
		idReclamacao: efetivarRegistro(Reclamacao r, Usuario u)	Os dados de usuário e reclamação devem estar devidamente preenchidos.	Um identificador de usuário é definido pelo sistema. Dados de Usuário e Reclamação são inseridos no sistema.
ConsultarReclamacao	IConsultaReclamacao	reclamacaoDetalhe: exibirDadosReclamacao(long idReclamacao)	Identificador válido de reclamação é passado.	Dados da reclamação, associada ao identificador, são retornados.
AlterarReclamacao	IAlteracaoReclamacao	alterarDadosReclamacao(Reclamacao r, Usuario u)	Dados de reclamação e usuário válidos são passados.	Usuário e Reclamação são alterados no sistema.

Tabela 5 – Especificação de Componentes da Camada de Sistema

A Tabela 6 apresenta as interfaces e componentes da camada de negócios. Como não se implementou os componentes IImpressoraMgr e IEmailMgr, os mesmos não são apresentados aqui.

Componente que Oferece Interface	Interface Provida	Operação	Pré-Condição	Pós-Condição
RodoviaMgr	IRodoviaMgt	listaRodovias: consultarRodovias()	Banco de dados deve conter rodovias	Uma lista de rodovias é retornada.
TrechoMgr	ITrechoMgt	listaTrechos: consultarTrechos(long idReclamacao)	Banco de dados deve conter trechos para a rodovia informada	Uma lista de trechos da rodovia é retornada.
ReclamacaoMgr	ITipoReclamacaoMgt	listaTipos: consultarTiposReclamacao()	Banco de dados deve conter tipos de reclamação.	Uma lista de tipos é retornada.
ReclamacaoMgr	IReclamacaoMgt	resposta: consultarRespostaPadrao(long idTrecho, long idTipo)	Identificadores de trecho e tipos devem ter formato válidos para o sistema	Uma resposta padrão deve ser retornada. Caso não exista, uma string vazia deve ser apresentada.
		idReclamacao: registrarReclamacao(Reclamacao r)	Dados de reclamação devem ser válidos.	Uma nova reclamação deve existir no banco de dados e ter um identificador único e um usuário associados.
		reclamacaoDetalhe: exibirDadosReclamacao (long idReclamacao)	Identificador de reclamação deve ser válido.	Dados de reclamação devem ser lidos do banco e retornados.
		alterarDadosReclamacao(Reclamacao r)	Dados de reclamação devem ser válidos.	Reclamação já existente no banco de dados tem seus dados alterados.

UsuarioMgr	IUsuarioMgt	idUsuario: registrarUsuario(Usuario u)	Dados de usuário devem ser válidos.	Um novo usuário deve existir no banco de dados e ter um identificador único associado.
		alterarDadosUsuario(Usuario u)	Dados de usuário devem ser válidos.	Usuário já existente no banco de dados tem seus dados alterados.

Tabela 6 – Especificação de Componentes da Camada de Negócios

O contrato de realização também é apresentado na forma de tabela. Sabe-se que um componente da camada de sistema requer interfaces da camada de negócios. Neste estudo de caso, os componentes da camada de acesso ao banco não foram especificados da mesma forma que os componentes da camada de sistema e negócios, ou seja, contendo explicitamente suas interfaces providas. Sabe-se que componentes da camada de negócios requerem interfaces da camada de acesso a banco, mas isto não será apresentado aqui.

Na Tabela 7 a primeira coluna apresenta um componente cujo contrato de realização está sendo especificado. A segunda coluna apresenta a interface provida deste componente e a terceira coluna apresenta a operação desta interface que pode ser chamada por um componente da camada de Diálogo com usuário. Para executar a operação apresentada na terceira coluna, um componente da camada de sistema requer interfaces da camada de negócios, e mais especificamente requer operações destas interfaces. As interfaces requeridas são apresentadas na quarta coluna e as operações na quinta coluna.

Componente	**Interface Provida**	**Operação da Camada de Sistema**	**Interface Requerida**	**Operação Requerida na Camada de Negócios**
RegistrarReclamacao	IOperacoesGenericas	listaRodovias: consultarRodovias()	IRodoviaMgt	listaRodovias: consultarRodovias()

		listaTrechos: consultarTrechos(long idReclamacao)	ITrechoMgt	listaTrechos: consultarTrechos(long idReclamacao)
		listaTipos: consultarTiposReclamacao()	ITipoReclamacaoMgt	listaTipos: consultarTiposReclamacao()
	IRegistroReclamacao	resposta: consultarRespostaPadrao(long idTrecho, long idTipo)	IReclamacaoMgt	resposta: consultarRespostaPadrao(long idTrecho, long idTipo)
		idReclamacao: efetivarRegistro(Reclamacao r, Usuario u)	IReclamacaoMgt	registrarReclamacao(Reclamacao r)
			IUsuarioMgt	registrarUsuario(Usuario u)
ConsultarReclamacao	IConsultaReclamacao	reclamacaoDetalhe: exibirDadosReclamacao(long idReclamacao)	IReclamacaoMgt	reclamacaoDetalhe: exibirDadosReclamacao(long idReclamacao)
AlterarReclamacao	IAlteracaoReclamacao	alterarDadosReclamacao(Reclamacao r, Usuario u)	IReclamacaoMgt	alterarReclamacao(Reclamacao r)
			IUsuarioMgt	alterarUsuario(Usuario u)

Tabela 7 – Contrato de Realização de Componentes

Uma vez tendo especificado os componentes, continuou-se com as fases do incremento 1, com o provisionamento e a integração de componentes que são apresentados a seguir.

4.3.3 Provisionamento dos Componentes

Nesta fase do primeiro incremento implementou-se todos os componentes para os casos de uso de registro e consulta de reclamação de acordo com as especificações. Para a implementação utilizou-se o modelo COSMOS e a linguagem Java. Com base no modelo de tipos de negócios da Figura 44, foi definido um *sript* de criação de tabelas de banco de dados para o subsistema de gerência de reclamações. Este *sript* pode ser visto no apêndice D.

Cada camada da arquitetura foi mapeada para um pacote, como pode ser visto Figura 57. O pacote *'WebContent'* representa a camada de interface com usuário e possui as páginas da aplicação *Web* que implementa o subsistema de gerência de reclamações. O pacote 'apresentação' representa a camada de diálogo com usuário e possui as classes que implementam as ações que são executadas no servidor de aplicação e os formulários, utilizados pelo *framework Struts*. O pacote 'db' representa a camada de acesso a banco de dados e possui as classes que implementam o acesso ao banco seguindo o padrão de projeto DAO [Alur+01]. Os pacotes 'sistema' e 'negócios' representam respectivamente as camadas de sistema e negócios da arquitetura e contêm os componentes especificados na subseção anterior.

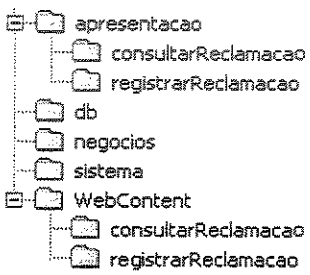

Figura 57 – Pacotes da Implementação do Subsistema de Gerência de Reclamações

A Figura 58 apresenta os componentes implementados em seus pacotes seguindo o sub-modelo de especificação de componentes do modelo COSMOS. As interfaces providas e requeridas foram criadas. Como neste estudo de caso a camada de acesso a banco não foi especificada de acordo com o modelo COSMOS, não se definiu as interfaces requeridas para os componentes da camada de negócios.

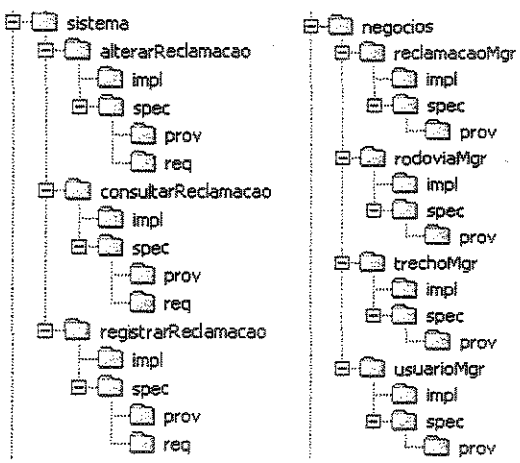

Figura 58 – Componentes do Subsistema de Gerência de Reclamações em seus Pacotes

A partir deste momento cada componente foi implementado seguindo o sub-modelo de implementação de componente do COSMOS. Nesta subseção são apresentados exemplos de projeto e implementação de componentes para o caso de uso de registro de reclamação.

A Figura 59 apresenta o componente RegistrarReclamacao da camada de sistema com o conteúdo de seu pacote *spec* e *impl*. A Figura 60 apresenta as classes do pacote *impl* deste componente.

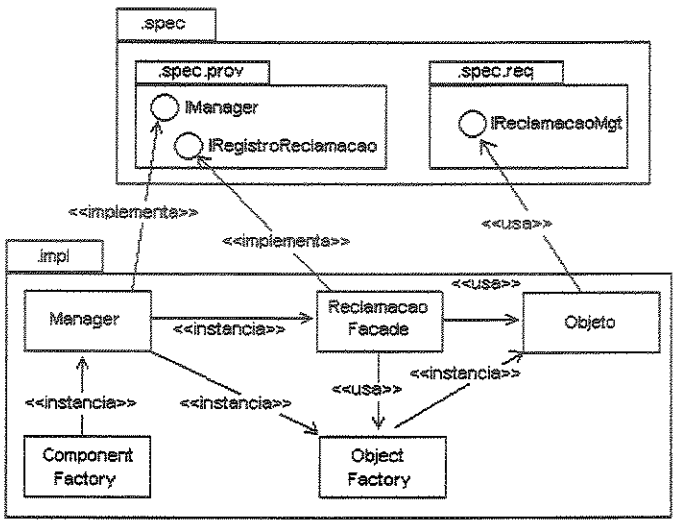

Figura 59 – Pacotes do Componente RegistrarReclamacao

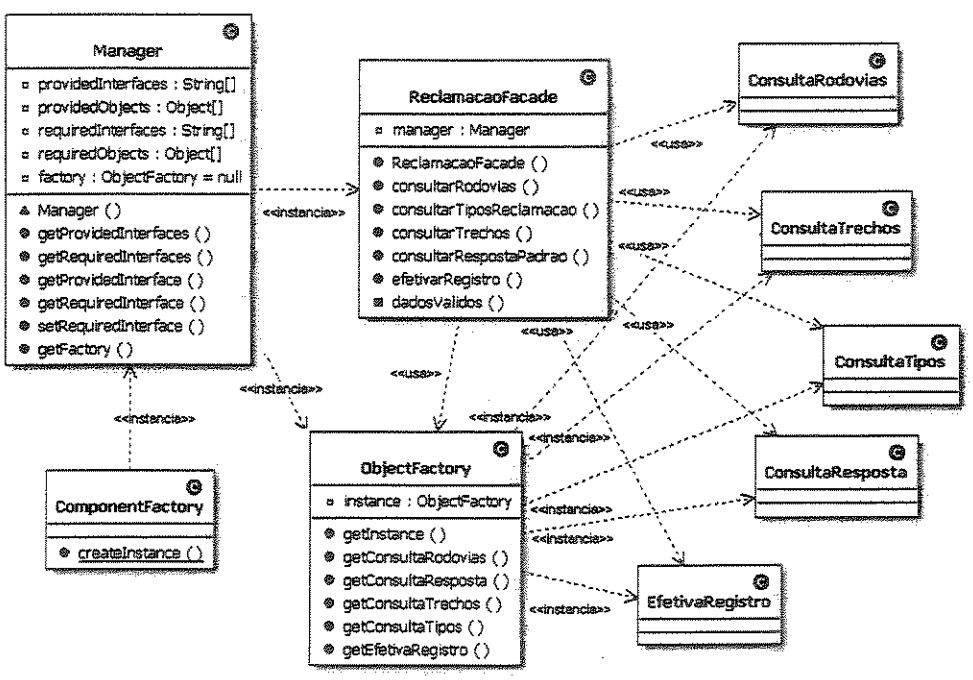

Figura 60 – Componente RegistrarReclamacao e sua Implementação

A Figura 61 apresenta as classes do pacote *impl* do componente ReclamacaoMgr e a Figura 62 apresenta as classes do pacote *impl* do componente UsuarioMgr, ambos da camada de negócios.

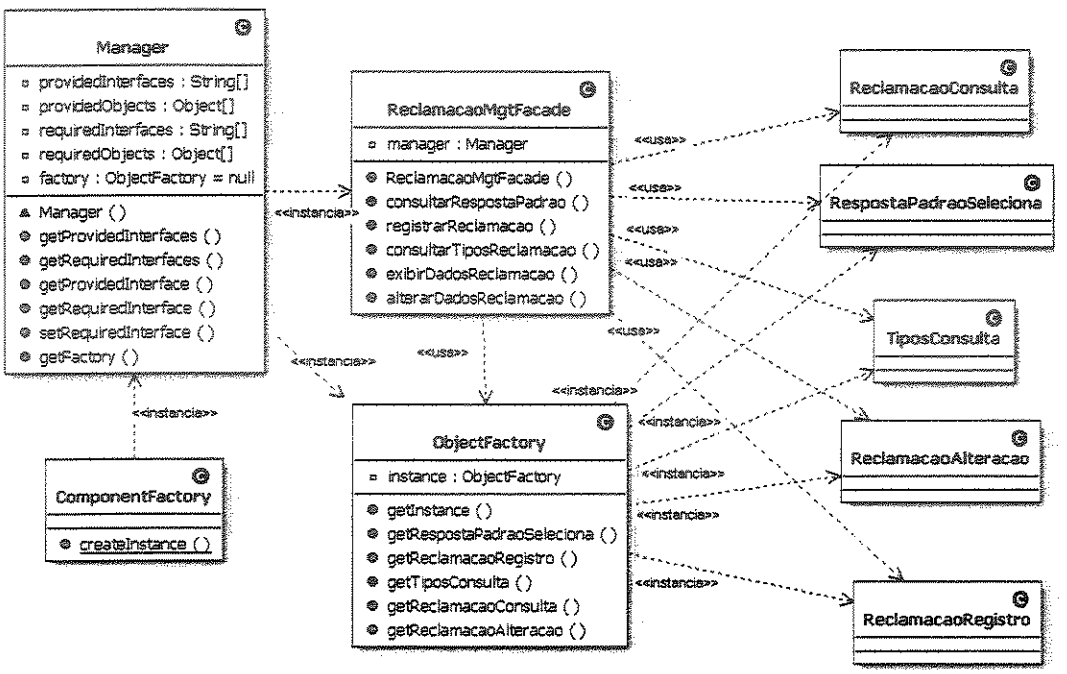

Figura 61 – Componente ReclamacaoMgr e sua Implementação

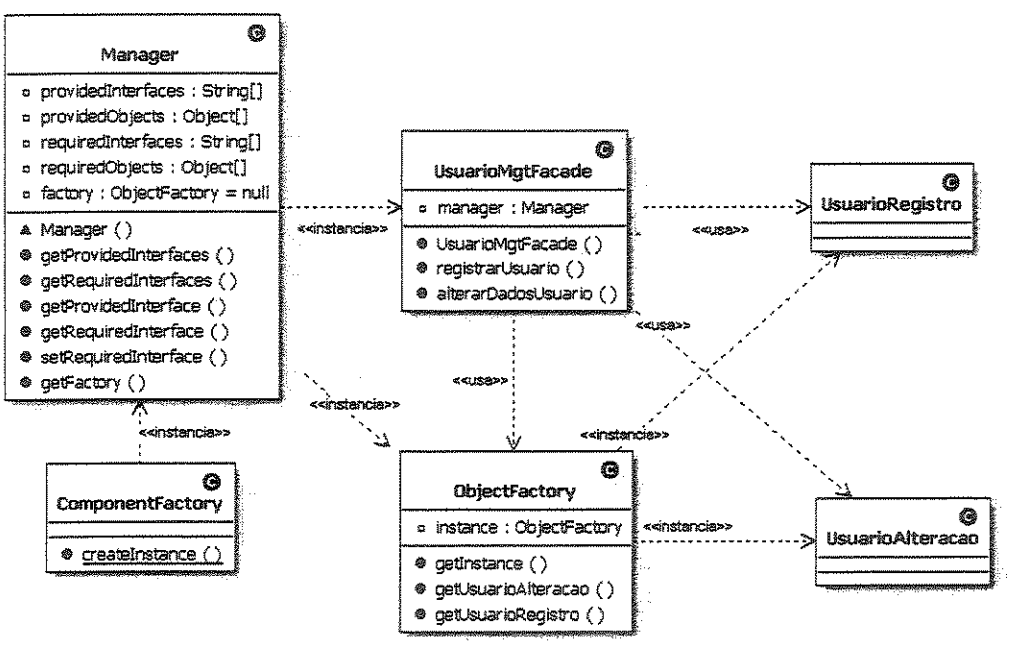

Figura 62 – Componente UsuarioMgr e sua Implementação

A implementação dos componentes neste incremento não levou em consideração especificações excepcionais. Desta forma, não se preocupou em tratar as condições excepcionais, ou seja, exceções que ocorressem na camada de banco de dados eram propagadas até a camada de sistema. Como o estudo de caso foi implementado em Java, a exceção SQLException do pacote java.sql seria propagada da camada de acesso a banco até a camada de sistema. O sistema não estava preparado para tratar condições excepcionais não antecipadas.

Ao se terminar a implementação dos componentes, iniciou-se a fase de integração que é apresentada a seguir.

4.3.4 Integração dos Componentes

Nesta fase realizou-se a integração entre os componentes implementados durante o provisionamento. Para a integração optou-se por aplicar o sub-modelo de conectores do modelo COSMOS. Para cada componente da camada de sistema que utiliza um componente da camada de negócios, foi criado um conector para interligá-los.

Os conectores foram criados em um pacote chamado connNeg. Seguindo com o exemplo para o caso de uso de registro de reclamação, a Figura 63 apresenta o conector ReclamacaoMgrConn interligando os componentes RegistrarReclamacao da camada de sistema com o componente ReclamacaoMgr da camada de negócios, e o conector UsuarioMgrConn interligando o componente RegistrarReclamacao com o componente UsuarioMgr.

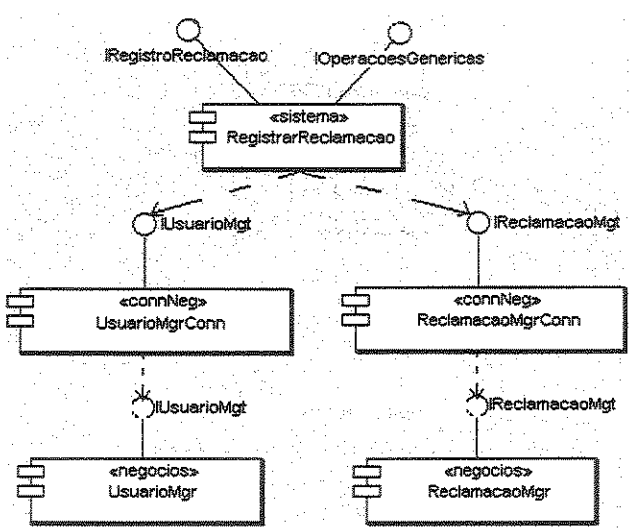

Figura 63 – Conectores ReclamacaoMgrConn e UsuarioMgrConn

O diagrama de seqüência apresentado na Figura 64 mostra o fluxo de execução durante a chamada da operação efetivarRegistro(...) da interface IRegistroReclamacao da camada de sistema. O componente RegistrarReclamacao oferece esta interface e utiliza os conectores ReclamacaoMgrConn e UsuarioMgrConn para acessar os componentes ReclamacaoMgr e UsuarioMgr respectivamente. O método efetivarRegistro(...) é chamado após o usuário ter selecionado uma rodovia, um trecho, um tipo de reclamação, ter digitado os dados de uma reclamação e seus os seus próprios dados, e ter realizado uma requisição para que estes dados fossem registrados no sistema.

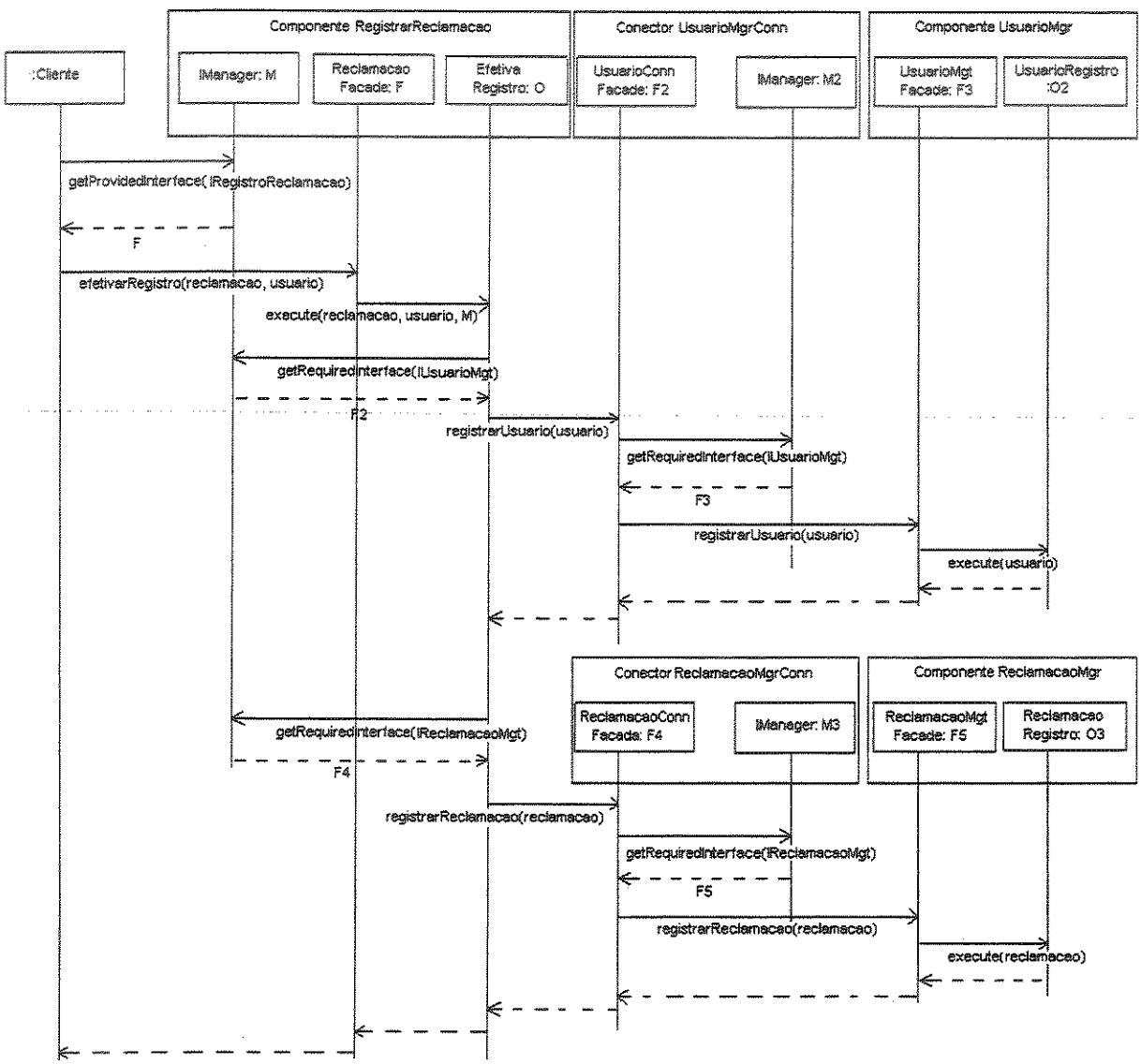

Figura 64 – Diagrama de Seqüência para Registro de Reclamação

Considerando a Figura 64, quando o usuário faz a requisição de registro, a camada de interface do subsistema de gerência de reclamação passa a requisição para a camada de diálogo. A camada de diálogo sabe que o componente RegistrarReclamacao é responsável pelos serviços de registro de dados de reclamação, e tem acesso a interface IManager deste componente. A interface IManager é implementada pelo objeto Manager M do componente RegistrarReclamacao. Assim, a camada de diálogo chama o método getProvidedInterface da interface IManager passando como parâmetro a interface requerida pela camada de diálogo: IRegistroReclamacao. O Manager M do componente RegistrarReclamacao sabe que existe uma fachada deste componente responsável por lidar com a interface IRegistroReclamacao, e retorna a fachada F para a camada de diálogo. A camada de diálogo chama o método efetivarRegistro da fachada F passando como parâmetros um objeto do tipo Reclamacao e um objeto do tipo Usuario. A fachada F delega a execução desta operação para um objeto O, do tipo EfetivaRegistro. Este objeto pergunta para o objeto Manager M do componente RegistrarReclamacao quem é responsável pela interface requerida IUsuarioMgt. O Manager M retorna uma fachada F2 do conector UsuarioMgrConn. A fachada F2 do conector pergunta para o Manager M2 do conector quem é responsável pela interface requerida IUsuarioMgt, e tem como resposta uma fachada F3 do componente UsuarioMgr. A fachada F2 do conector faz uma requisição para a fachada F3 do componente UsuarioMgr para que os dados do usuário sejam registrados. A fachada F3 do componente UsuarioMgr delega o pedido para um objeto O2 do tipo UsuarioRegistro. Este objeto faz requisições para objetos da camada de acesso ao banco de dados para registrar efetivamente os dados de usuário no banco de dados.

Após o registro do usuário, deve-se realizar o registro dos dados da reclamação. Para isto, o objeto O do componente RegistrarReclamacao pergunta para o Manager M deste componente quem é responsável pela interface IReclamacaoMgt. Recebe como retorno a fachada F4 do conector ReclamacaoMgrConn. A fachada F4 pergunta para o Manager M2 do conector quem é responsável pela interface IReclamacaoMgt. O Manager M2 retorna a fachada F5 do componente ReclamacaoMgr. A fachada F4 chama o método registrarReclamacao passando como parâmetro os dados da reclamação. A fachada F5 delega a requisição para o objeto O3 do tipo ReclamacaoRegistro que utiliza a camada de acesso a banco para efetivar o registro.

Quando o subsistema de gerência de reclamações é inicializado no servidor de aplicação, ocorre a ligação dos componentes e conectores com base na configuração da arquitetura. Nesta ligação define-se que:

- A fachada do conector UsuarioMgrConn fica como responsável pela interface requerida do componente RegistrarReclamacao: IUsuarioMgt. Isto liga o componente RegistrarReclamacao ao conector UsuarioMgrConn.
- A fachada do componente UsuarioMgr fica como responsável pela interface requerida do conector UsuarioMgrConn: IUsuarioMgt. Isto liga o componente UsuarioMgr ao conector UsuarioMgrConn.
- A fachada do conector ReclamacaoMgrConn fica como responsável pela interface requerida do componente RegistrarReclamacao: IReclamacaoMgt. Isto liga o componente RegistrarReclamacao ao conector ReclamacaoMgrConn.
- A fachada do componente ReclamacaoMgr fica como responsável pela interface requerida do conector ReclamacaoMgrConn: IReclamacaoMgt. Isto liga o componente ReclamacaoMgr ao conector ReclamacaoMgrConn.

Estas ligações são definidas através dos objetos do tipo Manager de componentes e conectores, ou seja, para se definir que a fachada F2 do conector UsuarioMgrConn ficará como responsável pela interface requerida IUsuarioMgt do componente RegistrarReclamacao, deve-se chamar a operação setRequiredInterface(IUsuarioMgt, F2) do Manager M do componente RegistrarReclamacao. Desta forma, quando o objeto O, do tipo EfetivaRegistro, perguntar para o Manager M do componente RegistrarReclamacao quem é o responsável pela interface IUsuarioMgt, ele receberá como retorno a fachada F2 do conector UsuarioMgrConn.

Além disso, na implementação de uma classe Manager de um componente, são definidos os responsáveis pelas interfaces providas deste componente. O mesmo se aplica a uma classe Manager de um conector. Por exemplo, na implementação da classe Manager M do componente RegistrarReclamacao define-se que a fachada F será a responsável pela interface provida IRegistroReclamacao.

Exemplos de Implementação de Componentes e Conectores

A seguir, apresenta-se trechos de código de classes dos componentes RegistrarReclamacao, ReclamacaoMgr e do conector ReclamacaoMgrConn, associados a seqüência de registro de dados de reclamação.

O Quadro 3 apresenta um trecho de código da classe ReclamacaoFacade do componente RegistrarReclamacao, com o seu método efetivarRegistro. Este método obtém uma referência para o objeto da classe EfetivaRegistro através do objeto ObjectFactory, que é apresentado no Quadro 4. Um trecho de código da classe EfetivaRegistro é apresentado no Quadro 5. As linhas em negrito da classe EfetivaRegistro apresentam as requisições que a classe faz para o Manager de seu componente para saber quem são os responsáveis pelas interfaces requeridas IUsuarioMgt e IReclamacaoMgt.

```
package sistema.registrarReclamacao.impl;
//seção de importação de pacotes
class ReclamacaoFacade implements IRegistroReclamacao, IOperacoesGenericas{
  private Manager manager;
  public ReclamacaoFacade(Manager manager){
    this.manager = manager;
  }
  //public ArrayList consultarRodovias()...
  //public ArrayList consultarTiposReclamacao()...
  //public ArrayList consultarTrechos(long idRodovia)...
  //public String consultarRespostaPadrao(Reclamacao reclamacao)...
  public String efetivarRegistro(Reclamacao reclamacao, Usuario usuario) throws Exception{
    ObjectFactory fac = manager.getFactory();
    EfetivaRegistro obj = fac.getEfetivaRegistro();
    String idReclamacao = obj.execute(usuario, reclamacao, manager);
    return idReclamacao;
  }
  ...
}
```

Quadro 3 – Código da Classe ReclamacaoFacade do Componente RegistrarReclamacao

```
package sistema.registrarReclamacao.impl;
class ObjectFactory {
 private ObjectFactory instance;
 public synchronized ObjectFactory getInstance(){
  if (instance == null){
   //O ObjectFactory é um singleton
   instance = new ObjectFactory();
  }
  return instance;
 }
 public ConsultaRodovias getConsultaRodovias(){
  return new ConsultaRodovias();
 }
 public ConsultaResposta getConsultaResposta(){
  return new ConsultaResposta();
 }
 public ConsultaTrechos getConsultaTrechos(){
  return new ConsultaTrechos();
 }
 public ConsultaTipos getConsultaTipos(){
  return new ConsultaTipos();
 }
 public EfetivaRegistro getEfetivaRegistro(){
  return new EfetivaRegistro();
 }
}
```

Quadro 4 – Código da Classe ObjectFactory do Componente RegistrarReclamacao

```
package sistema.registrarReclamacao.impl;
//seção de importação de pacotes
public class EfetivaRegistro {
  public String execute(Usuario usuario, Reclamacao reclamacao, Manager manager) throws Exception{
    //Requisita a factory de banco de dados
    DAOFactory factory = DAOFactory.getDAOFactory(DAOFactory.DB);
    //GenericOperations é uma classe que encapsulará conexão a banco de dados.
    GenericOperations genOp = null;
    //Requisita o DataSource de acesso a banco de dados, de onde se obtem a conexão para banco de dados
    genOp = factory.getDSGenericOperations();
    //Inicia o processo de registro de dados.
    boolean erroUser = false;
    String userId;
    Exception userException = null;
    //Faz a pergunta para o manager do componente sobre quem é responsável pela
    //interface IUsuárioMgt. É retornado a fachada do conector em tempo de execução.
    sistema.registrarReclamacao.spec.req.IUsuarioMgt iUsuarioMgt =
      (IUsuarioMgt)manager.getRequiredInterface("sistema.registrarReclamacao.spec.req.IUsuarioMgt");
    try{
      //Faz a requisição para registro de dados de usuário e reclamação.
      userId = iUsuarioMgt.registrarUsuario(factory.getUsuarioDAO(), genOp, usuario);
    }catch (Exception e){ //Caso ocorra um erro ao registrar usuario
      erroUser = true;
      userException = e;
    }
    String reclamacaoId;
    boolean erroRec = false;
    Exception recException = null;
    sistema.registrarReclamacao.spec.req.IReclamacaoMgt iReclamacaoMgt;
    if (!erroUser){
      iReclamacaoMgt = (IReclamacaoMgt)manager.getRequiredInterface("sistema.
        registrarReclamacao.spec.req.IReclamacaoMgt");
      try{
        reclamacaoId = iReclamacaoMgt.registrarReclamacao(userId, factory.getReclamacaoDAO(),
        genOp, reclamacao);
      }catch (Exception e){ //Caso ocorra um erro ao registrar reclamacao
            erroRec = true;
            recException = e;
      }
    }
    if (erroUser || erroRec) { //Caso ocorra um erro no registro de usuário ou reclamação faz-se um rollback.
      genOp.rollback();
      genOp.close();
      if (erroUser) {throw userException;
      }else if (erroRec){
        throw recException;
      }
    }else{//Em caso de sucesso, faz-se um commit.
      genOp.commit();
      genOp.close();
    }
    return reclamacaoId;
}
```

Quadro 5 – Código da Classe EfetivaRegistro do Componente RegistrarReclamacao

O Quadro 6 apresenta a fachada do conector ReclamacaoMgrConn. As linhas em negrito apresentam a requisição que a classe faz para o Manager do conector para saber quem é responsável pela interface requerida IReclamacaoMgt.

```
package connNeg.reclamacaoMgrConn.impl;
//Seção de importação de pacotes
class ReclamacaoConnFacade
 implements sistema.registrarReclamacao.spec.req.IReclamacaoMgt,
        sistema.registrarReclamacao.spec.req.ITipoReclamacaoMgt,
        sistema.consultarReclamacao.spec.req.IReclamacaoMgt,
        sistema.alterarReclamacao.spec.req.IReclamacaoMgt {
 private Manager manager = null;
 public ReclamacaoConnFacade(Manager manager) {
  this.manager = manager;
 }
 public String registrarReclamacao(String userId, ReclamacaoDAO dao,
  GenericOperations aGenOp, Reclamacao reclamacao)
   throws Exception {
  negocios.reclamacaoMgr.spec.prov.IReclamacaoMgt iReclamacaoMgt;
  iReclamacaoMgt = (negocios.reclamacaoMgr.spec.prov.IReclamacaoMgt)
   manager.getRequiredInterface("negocios.reclamacaoMgr.spec.prov.IReclamacaoMgt");
  String reclamacaoId = null;
  reclamacaoId = iReclamacaoMgt.registrarReclamacao(userId, dao, aGenOp, reclamacao);
  return reclamacaoId;
 }
 ...
}
```

Quadro 6 – Código da Classe ReclamacaoConnFacade do Conector ReclamacaoMgrConn

O Quadro 7 apresenta um trecho de código da classe ReclamacaoMgtFacade do componente ReclamacaoMgr, com o seu método registrarReclamacao. Este método obtem uma referência para o objeto da classe ReclamacaoRegistro através do objeto ObjectFactory que é apresentado no Quadro 8. Um trecho de código da classe ReclamacaoRegistro é apresentado no Quadro 9.

```
package negocios.reclamacaoMgr.impl;
//Seção de importação de pacotes
public class ReclamacaoMgtFacade implements IReclamacaoMgt, ITipoReclamacaoMgt{
  private Manager manager;
  public ReclamacaoMgtFacade(Manager manager){
    this.manager = manager;
  }
  public String registrarReclamacao(String userId, ReclamacaoDAO dao,
    GenericOperations aGenOp, Reclamacao reclamacao) throws Exception {
    ObjectFactory fac = manager.getFactory();
    ReclamacaoRegistro obj = fac.getReclamacaoRegistro();
    String reclamacaoId = obj.execute(dao, userId, aGenOp, reclamacao);
    return reclamacaoId;
  }
  ...
}
```

Quadro 7 – Código da Classe ReclamacaoMgtFacade do Componente ReclamacaoMgr

```
package negocios.reclamacaoMgr.impl;
//Seção de importação de pacotes
class ObjectFactory {
  private ObjectFactory instance;
  public synchronized ObjectFactory getInstance(){
    if (instance == null){
      instance = new ObjectFactory();
    }
    return instance;
  }
  public RespostaPadraoSeleciona getRespostaPadraoSeleciona(){
    return new RespostaPadraoSeleciona();
  }
  public ReclamacaoRegistro getReclamacaoRegistro(){
    return new ReclamacaoRegistro();
  }
  public TiposConsulta getTiposConsulta(){
    return new TiposConsulta();
  }
  public ReclamacaoConsulta getReclamacaoConsulta(){
    return new ReclamacaoConsulta();
  }
  public ReclamacaoAlteracao getReclamacaoAlteracao(){
    return new ReclamacaoAlteracao();
  }
}
```

Quadro 8 – Código da Classe ObjectFactory do Componente ReclamacaoMgr

```
package negocios.reclamacaoMgr.impl;
//Seção de importação de pacotes
public class ReclamacaoRegistro{
  public String execute(ReclamacaoDAO dao, String userId,
  GenericOperations aGenOp, Reclamacao reclamacao) throws Exception{
    String id = null;
    //Nesta operação de insert pode ocorrer uma exceção SQLException
    id = dao.insert(userId, aGenOp, reclamacao);
    if (id == null || id == ""){
      SQLException ex = new SQLException("Id inválido");
      throw ex;
    }
    return id;
  }
}
```

Quadro 9 – Código da Classe ReclamacaoRegistro do Componente ReclamacaoMgr

4.4 Estudo de Caso – Incremento 2

Com base na descrição do subsistema de gerência de reclamações em 4.1, iniciou-se a aplicação das fases de especificação de requisitos e de componentes, definidas no método MECE. Posteriormente teve-se as fases de provisionamento e integração, onde exercícios de reuso de componentes, desenvolvidos no incremento 1, foram realizados.

4.4.1 Especificação dos Requisitos do Subsistema

As primeiras atividades da fase de especificação de requisitos do método MECE são atividades definidas pelo processo UML Components. Desta forma, com base na descrição do Telestrada definiu-se o processo de negócio, o modelo conceitual de negócio e os casos de uso da mesma forma como foi apresentado na seção 4.3.1.

Assim como no incremento 1, o foco do incremento 2 foi nos casos de uso de registro de reclamação e consulta de reclamação. Com base nestes casos de uso verificou-se, para cada passo dos cenários primários e alternativos, quais seriam as possíveis condições excepcionais que poderiam ocorrer no sistema. Para cada condição excepcional definiu-se um cenário excepcional. Seguindo com o exemplo do caso de uso de registro de reclamação, a seguir são apresentados alguns cenários excepcionais deste caso de uso. Estes cenários serão utilizados nesta subseção para ilustrar a aplicação do método MECE.

Cenário Recuperável 1:
Passo de Cenário Associado: Passo 3 do cenário primário.
Violação de pré-condição: Não existem rodovias no sistema.
Sinal: Lista vazia de rodovias.
Tratador: Preparar mensagem de ausência de rodovias e apresentar para usuário.
Pós-Condição: Uma mensagem é apresentada para o usuário dizendo que não existem rodovias no sistema.

Cenário Recuperável 2:
Passo de Cenário Associado: Passo 5 do cenário primário.
Violação de pré-condição: Não existem tipos de reclamação no sistema.
Sinal: Lista vazia de tipos de reclamação.
Tratador: Preparar mensagem de ausência de tipos de reclamação e apresentar para usuário.
Pós-Condição: Uma mensagem é apresentada para o usuário dizendo que não existem tipos de reclamação no sistema.

Cenário Recuperável 3:
Passo de Cenário Associado: Passo 5 do cenário primário.
Violação de pré-condição: Não existem trechos para rodovia selecionada.
Sinal: Lista vazia de trechos.
Tratador: Preparar mensagem de ausência de trechos e apresentar para usuário.
Pós-Condição: Uma mensagem é apresentada para o usuário dizendo que não existem trechos para rodovia selecionada.

Cenário Recuperável 4:
Passo de Cenário Associado: Passos 8 e 9 do cenário primário.
Violação de pré-condição: Identificador de trecho ou tipo inválido.
Sinal: Percepção que formato de um dos identificadores não é reconhecido pelo sistema, ou seu valor é inválido.
Tratador: Preparar mensagem sobre erro para ser apresentada para usuário.
Pós-Condição: Uma mensagem é apresentada para o usuário dizendo que o valor do identificador de trecho ou tipo não é valido.

Cenário Recuperável 5:
Passo de Cenário Associado: Passo 12 do cenário primário.
Violação de pré-condição: Dados de usuário ou reclamação não são preenchidos corretamente para inserção.
Sinal: Dados vazios ou incorretos.
Tratador: Preparar mensagem sobre dados incorretos e apresentar para usuário.
Pós-Condição: Uma mensagem é apresentada para o usuário dizendo que dados a serem inseridos são inválidos. Dados não são inseridos no sistema.

Cenário Recuperável 6:
Passo de Cenário Associado: Passo 3 do cenário primário.
Violação de pós-condição: Rodovias não são retornadas.
Sinal: Problema ao acessar servidor de banco de dados durante seleção de rodovias.

Tratador: Preparar mensagem sobre erro de acesso ao servidor de banco de dados e apresentar mensagem para usuário.
Pós-Condição: Uma mensagem é apresentada para o usuário dizendo que houve um erro ao selecionar rodovias no sistema e para ele tentar novamente.

Cenário Recuperável 7:
Passo de Cenário Associado: Passo 5 do cenário primário.
Violação de pós-condição: Trechos não são retornados.
Sinal: Problema ao acessar servidor de banco de dados durante seleção de trechos.
Tratador: Preparar mensagem sobre erro de acesso ao servidor de banco de dados e apresentar mensagem para usuário.
Pós-Condição: Uma mensagem é apresentada para o usuário dizendo que houve um erro ao selecionar trechos no sistema e para ele tentar novamente.

Cenário Recuperável 8:
Passo de Cenário Associado: Passo 5 do cenário primário.
Violação de pós-condição: Tipos de reclamação não são retornadas.
Sinal: Problema ao acessar servidor de banco de dados durante seleção de tipos de reclamação.
Tratador: Preparar mensagem sobre erro de acesso ao servidor de banco de dados e apresentar mensagem para usuário.
Pós-Condição: Uma mensagem é apresentada para o usuário dizendo que houve um erro ao selecionar tipos de reclamação no sistema e para ele tentar novamente.

Cenário Recuperável 9:
Passo de Cenário Associado: Passos 8 e 9 do cenário primário.
Violação de pós-condição: Resposta padrão não selecionada no sistema.
Sinal: Problema de acesso ao servidor de banco de dados ao selecionar resposta padrão para trecho e tipo de reclamação.
Tratador: Preparar mensagem sobre erro para ser apresentada para usuário.
Pós-Condição: Uma mensagem é apresentada para o usuário dizendo que houve um erro ao selecionar uma resposta padrão no sistema.

Cenário Recuperável 10:
Passo de Cenário Associado: Passo 12 do cenário primário.
Violação de pós-condição: Usuário não inserido no sistema.
Sinal: Identificador de usuário não é retornado pelo servidor de banco de dados devido a a falha de comunicação com servidor de banco de dados.
Tratador: Prepara mensagem sobre erro para ser apresentada para usuário e não registra dados do usuário nem de dados de reclamação.
Pós-Condição: Uma mensagem é apresentada para o usuário dizendo que houve um erro ao inserir dados de usuário no sistema e que também não foi possível inserir dados de reclamação. Dados de usuário e reclamação não são inseridos no sistema. Identificador de reclamação não é retornado para usuário do sistema.

Cenário Recuperável 11:
Passo de Cenário Associado: Passo 12 do cenário primário.

Violação de pós-condição: Reclamação não inserida no sistema.
Sinal: Problema de acesso ao servidor de banco de dados ao registrar dados da reclamação no sistema.
Tratador: Prepara mensagem sobre erro de registro de dados de reclamação e não registra a reclamação.
Pós-Condição: Uma mensagem é apresentada para o usuário dizendo que houve um erro ao registrar reclamação no sistema. Dados de reclamação e de usuário não são inseridos. Identificador de reclamação não é retornado para usuário.

Após se definir cenários excepcionais, inicia-se o estágio de detalhamento dos comportamentos normal e excepcional do sistema. Este detalhamento não é apresentado aqui para efeito de simplificação. Após o detalhamento pode-se formalizar os requisitos e iniciar o estágio de definição de arquitetura com a especificação de componentes.

4.4.2 Arquitetura e Especificação dos Componentes

Nesta fase aplicou-se os três estágios propostos pelo método MECE.

Identificação dos Componentes

Este estágio foi iniciado com a definição das interfaces da camada de sistema e de suas operações. Para estas atividades, o método MECE utiliza atividades do processo UML Components. Desta forma, no segundo incremento utilizou-se as interfaces e operações da camada de sistema que são apresentados em 4.3.2.

A partir deste momento definiu-se quais exceções poderiam ser lançadas em cada operação de cada interface da camada de sistema. Sabendo que (1) um passo de um cenário primário ou alternativo pode ser mapeado para uma operação de uma interface e que (2) um passo de um cenário primário ou alternativo pode ter um cenário excepcional associado a ele, realizou-se a atividade "Definições de exceções que operações podem lançar", apresentada na Figura 32. Por exemplo, o passo 12 do cenário primário do caso de uso de registro de reclamação tem o cenário excepcional 11 associado a ele. Este passo 12 está associado a operação efetivarRegistro(...) da interface IRegistroReclamacao. Pôde-se então definir uma condição excepcional associada a esta operação, baseada no cenário excepcional 11, que seria sinalizada pela exceção E_ReclamacaoNaoRegistradaException. A Tabela 8 apresenta exceções que foram definidas para as operações associadas ao caso de uso de registro de reclamação. Pode-se perceber que se mapeou

pré e pós-condições de passos de caso de uso para as operações que implementam estes passos de caso de uso.

Interface	Operação	Pré-Condição	Pós-Condição	Exceção Associada a Pré-Condição	Exceção Associada a Pós-Condição
IOperacoesGenericas	listaRodovias: consultarRodovias()	Rodovias devem existir no sistema	Uma lista de rodovias é retornada.	E_RodoviasNaoExistentesException	E_RodoviaConsultaException
	listaTrechos: consultarTrechos(long idReclamacao)	Trechos devem existir no sistema	Uma lista de trechos da rodovia especificada é retornada.	E_TrechosNaoExistentesException	E_TrechoConsultaException
	listaTipos: consultarTiposReclamacao()	Tipos de Reclamação devem existir no sistema	Uma lista de tipos é retornada.	E_TiposNaoExistentesException	E_TipoConsultaException
IRegistroReclamacao	resposta: consultarRespostaPadrao(long idTrecho, long idTipo)	Identificadores de trecho e tipo devem ser válidos.	Uma resposta padrão ou string vazia é retornada.	E_DadosRegistroInvalidosException	E_SelecionarRespostaPadraoException
	idReclamacao: efetivarRegistro(Reclamacao r, Usuario u)	Os dados de usuário e reclamação devem estar devidamente preenchidos.	Um identificador de usuário é definido pelo sistema.	E_DadosRegistroInvalidosException	E_IdUsuarioNaoRetornadoException
			Dados de usuário são registrados no sistema.	-	E_UsuarioNaoRegistradoException

			Dados de reclamação são registrados no sistema.	-	E_ReclamacaoNaoRegistradaException

Tabela 8 – Exceções que Operações da Camada de Sitema podem Lançar

Neste estágio também definiu-se as interfaces da camada de negócios como apresentado na subseção 4.3.2. E por fim refinou-se a arquitetura inicial com uso de componentes ideais tolerantes a falhas, como o exemplo da Figura 65. Este exemplo apresenta a operação efetiverRegistro(...) da interface IRegistroReclamacao do componente RegistrarReclamacao, e as exceções de defeito (E_IdUsuarioNaoRetornadoException, E_UsuarioNaoRegistradoException e E_ReclamacaoNaoRegistradaException) e de interface (E_DadosRegistroInvalidosException) que podem ser lançadas pelo componente ao executar esta operação. Caso se identifique exceções internas que possam ocorrer em operações dos componentes, pode-se indicá-las também durante este refinamento. Até este momento não foi possível definir as exceções de interface e de defeito que alcançam este componente, a partir de camadas inferiores da arquitetura. Para tal, iniciou-se o estágio de interação de componentes.

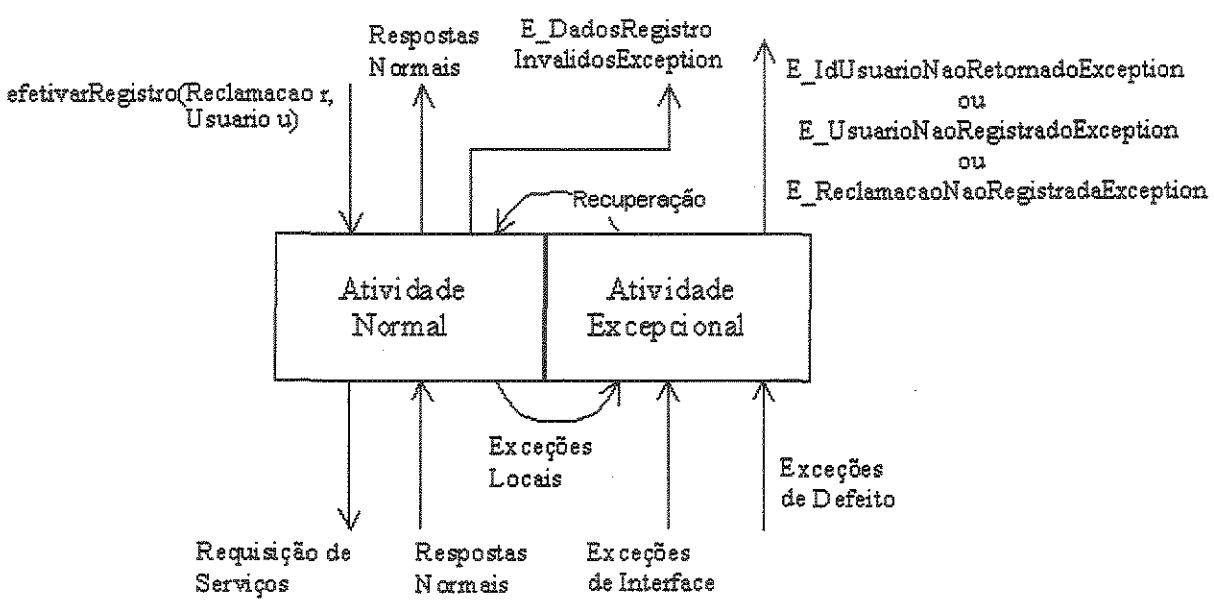

Figura 65 – Exemplo de Componente Ideal Tolerante a Falhas

Interação dos Componentes

Este estágio foi iniciado com a aplicação das atividades apresentadas na Figura 33. Obteve-se os diagramas de colaboração apresentados na seção 4.3.2. Para cada operação de interface da camada de negócios, apresentada nestes diagramas, definiu-se que exceções poderiam ser lançadas. Estas exceções são apresentadas na Tabela 9.

Interface	Operação	Pré-Condição	Pós-Condição	Exceção Associada a Pré-Condição	Exceção Associada a Pós-Condição
IRodoviaMgt	listaRodovias: consultarRodovias()	Banco de dados deve conter rodovias	Uma lista de rodovias é retornada.	E_RodoviasNaoExistentesNoBancoException	E_ConsultaDeRodoviasNoBancoException
ITrechoMgt	listaTrechos: consultarTrechos(long idReclamacao)	Banco de dados deve conter trechos para a rodovia informada	Uma lista de trechos da rodovia é retornada.	E_TrechosNaoExistentesException	E_ConsultaDeTrechoNoBancoException
ITipoReclamacaoMgt	listaTipos: consultarTiposReclamacao()	Banco de dados deve conter tipos de reclamação.	Uma lista de tipos é retornada.	E_TiposNaoExistentesException	E_ConsultaDeTiposNoBancoException
IReclamacaoMgt	resposta: consultarRespostaPadrao(long idTrecho, long idTipo)	Identificadores de trecho e tipos devem ter formato válidos para o sistema	Uma resposta padrão deve ser retornada. Caso não exista, uma string vazia deve ser apresentada.	E_DadosNaoValidosParaSelecaoRespostaPadraoException	E_ConsultaRespostaPadraoNoBancoException
	idReclamacao: registrarReclamacao(Reclamacao r)	Os dados de reclamação devem ser válidos	Uma nova reclamação deve existir no banco de dados e ter um identificador único e um usuário associados.	E_DadosNaoValidosParaRegistroDeReclamacaoException	E_RegistroReclamacaoNoBancoException

IUsuarioMgt	idUsuario: registrarUsuario(Usuario u)	Os dados de usuário devem ser válidos	Um novo usuário deve existir no banco de dados e ter um identificador único associado.	E_DadosNaoValidosParaRegistroDeUsuarioExcepcion	E_RegistroUsuarioNoBancoException e E_IdUsuarioNaoRetornadoException

Tabela 9 – Exceções que Operações da Camada de Negócios podem Lançar

Após definir as exceções que poderiam ser lançadas pelas operações das interfaces da camada de negócios, refinou-se a arquitetura com componentes ideais tolerantes a falhas para a camada de negócios. Como dito anteriormente, neste estudo de caso não se especificou explicitamente as interfaces requeridas para componentes da camada de negócios.

Neste momento foi possível definir também quais exceções deveriam ser tratadas nas interfaces requeridas dos componentes da camada de sistema, em função das exceções que seriam lançadas pelas operações da camada de negócios. A Figura 73 apresenta um cenário para o caso de uso de registro de reclamação. Considerando que a operação efetivarRegistro(Reclamacao r, Usuario u) do componente RegistrarReclamacao é chamada, pode ocorrer uma exceção de interface devido a dados inválidos de usuário ou de reclamação, violando uma pré-condição. Caso os dados de usuário sejam válidos, eles são inseridos no sistema e em seguida, o componente RegistrarReclamacao chama a operação registrarReclamacao(Reclamacao r) da interface IReclamacaoMgt do componente ReclamacaoMgr. Neste momento pode ocorrer uma exceção de interface caso os dados de reclamação não sejam válidos: E_DadosNaoValidosParaRegistroDeReclamacaoException. Durante a execução da operação registrarReclamacao(Reclamacao r) pode ocorrer um erro ao se acessar o servidor de banco de dados. Este erro pode levar a uma exceção de defeito do componente ReclamacaoMgr, chamada E_RegistroReclamacaoNoBancoException. As exceções E_DadosNaoValidosParaRegistroDeReclamacaoException e E_RegistroReclamacaoNoBancoException devem ser tratadas pela parte de Atividade Excepcional do componente RegistrarReclamacao. O comportamento excepcional do componente RegistrarReclamacao, neste caso, é lançar uma exceção E_ReclamacaoNaoRegistradaException para o componente que requisitou a operação efetivarRegistro(Reclamacao r, Usuario u).

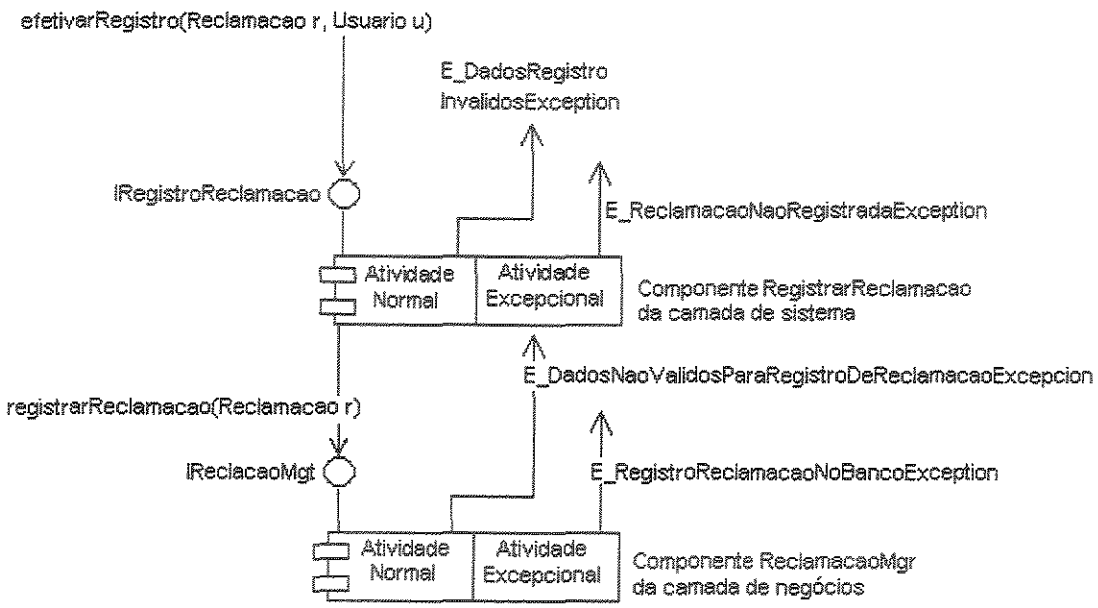

Figura 66 – Refinamento da Arquitetura com Componentes Ideais Tolerantes a Falhas

Durante o estudo de caso, pôde-se questionar a finalidade de se definir pré-condições parecidas para componentes da camada de sistema e negócios, como por exemplo a pré-condição que diz que os dados de reclamação devem ser válidos. Esta pré-condição existe tanto na especificação do componente RegistrarReclamacao, quanto na especificação do componente ReclamacaoMgr. Preferiu-se adotar uma postura mais conservadora, pois caso o componente RegistrarReclamacao tivesse uma falha de projeto que deixasse passar dados inválidos de reclamação, e o componente ReclamacaoMgr não estivesse preparado para lidar com os dados inválidos, poderia haver uma condição excepcional não antecipada no sistema ao se tentar registrar dados inválidos.

No estudo de caso, optou-se por refinar a arquitetura com o uso de conectores, como pode ser visto na Figura 67. Conectores simples foram considerados para apenas passar a requisição de um componente da camada de sistema para um componente da camada de negócios. Além disso, um conector possui uma parte para atividade excepcional que além de propagar exceções, também poderia ser usada para: (1) transformar exceções lançadas pelo componente da camada de negócios em exceções que o componente da camada de sistema possa entender e tratar, e (2) para lidar com condições excepcionais não antecipadas, transformando exceções não declaradas, lançadas pelo

componente da camada de negócios, em exceções definidas em hierarquia de tipos abstratos de exceções especificada para o sistema.

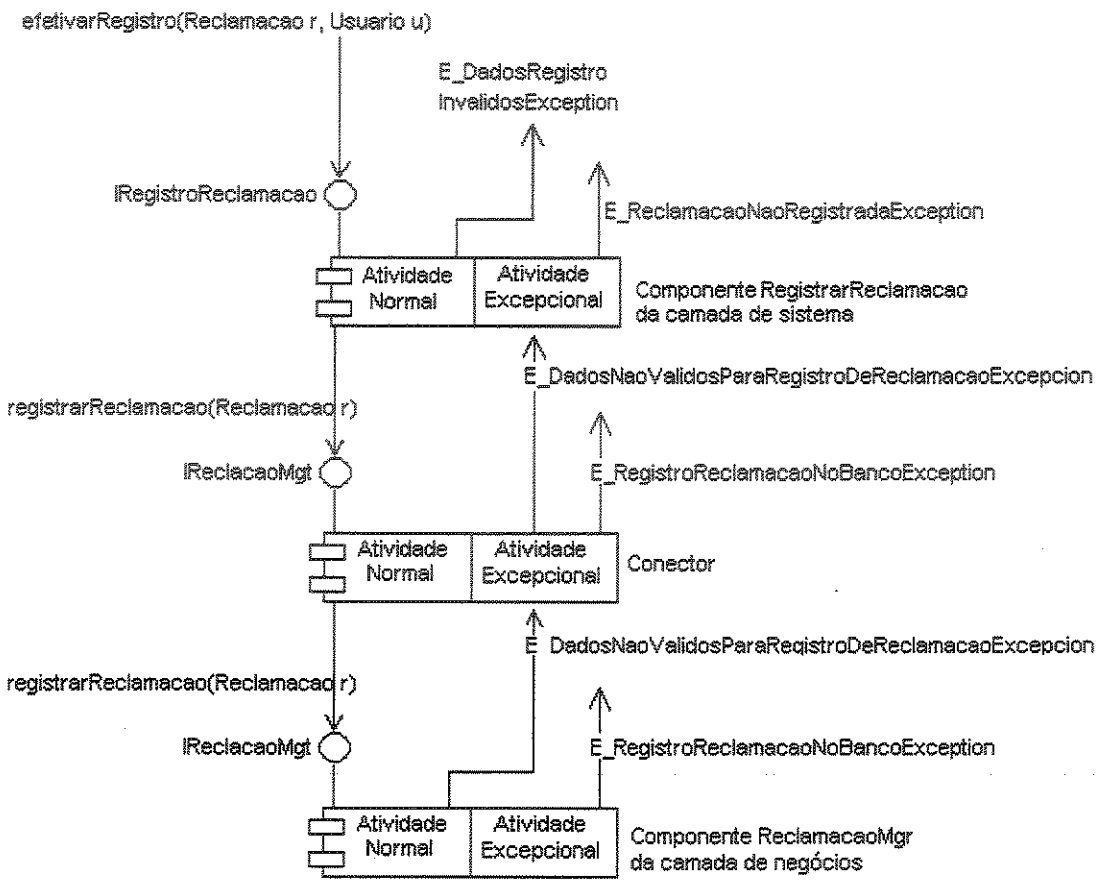

Figura 67 – Refinamento da Arquitetura com Conector

Especificação dos Componentes

Neste estágio definiu-se os contratos de uso e de realização de cada componente associado aos casos de uso de registro e consulta de reclamação. Inicialmente teve-se a especificação normal dos componentes conforme apresentadas nas Tabela 5 e Tabela 6. Após a especificação normal, iniciou-se a atividade "Elaboração da especificação excepcional", da Figura 35. Seguindo as orientações do método MECE, definiu-se as tabelas TECEp e TECEr para os componentes.

Seguindo com o exemplo do caso de uso de registro de reclamações, a seguir são apresentadas, as tabelas TECEp e TECEr para o componente RegistrarReclamacao.

TECEp - Componente RegistrarReclamacao / Camada de Sistema							
Interface Provida	**Operação**	**Pré-Condição**	**Pós-Condição**	**Exceção Associada a Pré-Condição (que pode ser lançada)**	**Exceção Associada a Pós-Condição (que pode ser lançada)**	**Pós-Condição Excepcional Associada a Violação da Pré-Condição**	**Pós-Condição Excepcional Associada a Violação da Pós-Condição**
IOperacoes Genericas	listaRodovias: consultarRodovias()	Rodovias devem existir no sistema	Uma lista de rodovias é retornada.	E_RodoviasNaoExistentesException	E_RodoviaConsultaException	Componente vai para um estado onde não permite consulta de rodovias enquanto novas rodovias não são inseridas no sistema.	Componente retorna para um estado onde aceita nova consulta de rodovias.
	listaTrechos: consultarTrechos(long idReclamacao)	Trechos devem existir no sistema	Uma lista de trechos da rodovia especificada é retornada.	E_TrechosNaoExistentesException	E_TrechoConsultaException	Componente vai para um estado onde não permite consulta de trechos enquanto novos trechos não são inseridos no sistema.	Componente retorna para um estado onde aceita nova consulta de trechos.
	listaTipos: consultarTiposReclamacao()	Tipos de Reclamação devem existir no sistema	Uma lista de tipos é retornada.	E_TiposNaoExistentesException	E_TipoConsultaException	Componente vai para um estado onde não permite consulta de tipos enquanto novos tipos não são inseridos no sistema.	Componente retorna para um estado onde aceita nova consulta de tipos de reclamação.

IRegistroReclamacao	resposta: consultarRespostaPadrao(long idTrecho, long idTipo)	Identificadores de trecho e tipo devem ser válidos.	Uma resposta padrão ou string vazia é retornada.	E_DadosRegistroInvalidosException	E_SelecionarRespostaPadraoException	Componente retorna para um estado onde aceita nova consulta de resposta padrão.	Componente retorna para um estado onde aceita nova consulta de resposta padrão.
	idReclamacao: efetivarRegistro(Reclamacao r, Usuario u)	Os dados de usuário e reclamação devem estar devidamente preenchidos.	Um identificador de usuário é definido pelo sistema.	E_DadosRegistroInvalidosException	E_IdUsuarioNaoRetornadoException	Componente retorna para um estado onde aceita nova tentativa de registro de reclamação com dados da reclamação e de eusuário.	Componente retorna para um estado onde aceita nova tentativa de registro de reclamação
			Dados de usuário são registrados no sistema.		E_UsuarioNaoRegistradoException		Componente retorna para um estado onde aceita nova tentativa de registro de reclamação
			Dados de reclamação são registrados no sistema.		E_ReclamacaoNaoRegistradaException		Componente retorna para um estado onde aceita nova tentativa de registro de reclamação

Tabela 10 – TECEp para Componente RegistrarReclamacao

TECEr - Componente RegistrarReclamacao / Camada de Sistema					
Interface Requerida	**Operação**	**Pré-Condição**	**Pós-Condição**	**Exceção Associada a Pré-Condição (que deve ser tratada)**	**Exceção Associada a Pós-Condição (que deve ser tratada)**
IReclamacaoMgt	resposta: consultarRespostaPadrao(long idTrecho, long idTipo)	Identificadores de trecho e tipos devem ter formato válidos para o sistema	Uma resposta padrão deve ser retornada. Caso não exista, uma string vazia deve ser apresentada.	E_DadosNaoValidosParaSelecaoRespostaPadraoExcepcion	E_ConsultaRespostaPadraoNoBancoException
	idReclamacao: registrarReclamacao(Reclamacao r)	Os dados de reclamação devem ser válidos	Uma nova reclamação deve existir no banco de dados e ter um identificador único e um usuário associados.	E_DadosNaoValidosParaRegistroDeReclamacaoExcepcion	E_RegistroReclamacaoNoBancoException
IUsuarioMgt	idUsuario: registrarUsuario(Usuario u)	Os dados de usuário devem ser válidos	Um novo usuário deve existir no banco de dados e ter um identificador único associado.	E_DadosNaoValidosParaRegistroDeUsuarioExcepcion	E_RegistroUsuarioNoBancoException e E_IdUsuarioNaoRetornadoException

Tabela 11 – TECEr para o Componente RegistrarReclamacao

Uma vez tendo especificado os contratos de uso e de realização de cada componente iniciou-se a fase de provisionamento dos componentes seguida da fase de integração.

4.4.3 Provisionamento e Integração dos Componentes

Nestas fases, realizou-se exercícios de reutilização de componentes desenvolvidos no primeiro incremento. O primeiro exercício estudou o reuso de um componente da camada de

sistema tendo acesso ao seu código. O segundo exercício estudou o reuso de um componente da camada de negócios sem acesso ao seu código. No terceiro exercício integrou-se estes componentes. Tanto no primeiro exercício quanto no segundo exercício, aplicou-se a estratégia intra-componente (Ver subseção 2.6.2). No terceiro exercício aplicou-se a estratégia inter-componente (Ver subseção 2.6.3). Os componentes usados nestes três exercícios foram: RegistrarReclamacao, da camada de sistema, e ReclamacaoMgr da camada de negócios.

Além dos três exercícios comentados acima, outros exercícios foram realizados: (1) para injeção de falhas em componentes e conectores, associados ao caso de uso de registro de reclamação, e (2) para simular o reuso de componentes da camada de sistema e negócios, associados ao caso de uso de consulta de reclamação, sem acesso aos códigos destes componentes.

Exercícios de Reuso de Componentes

O componente RegistrarReclamacao foi reutilizado levando-se em consideração a estratégia intra-componente apresentada na subseção 2.6.2 e tomando-se como base o contrato de realização do componente definido na subseção 4.4.2. Criou-se um tratador BLE para a fachada do componente, e tratadores ALE para as classes de implementação. Como este componente foi implementado no primeiro incremento de acordo com o modelo COSMOS e considerou-se que o integrador teria acesso ao seu código, pôde-se aplicar a estratégia intra-componente como se estivesse desenvolvendo um novo componente. A Figura 68 apresenta como ficaram os pacotes *spec* e *impl*. do componente.

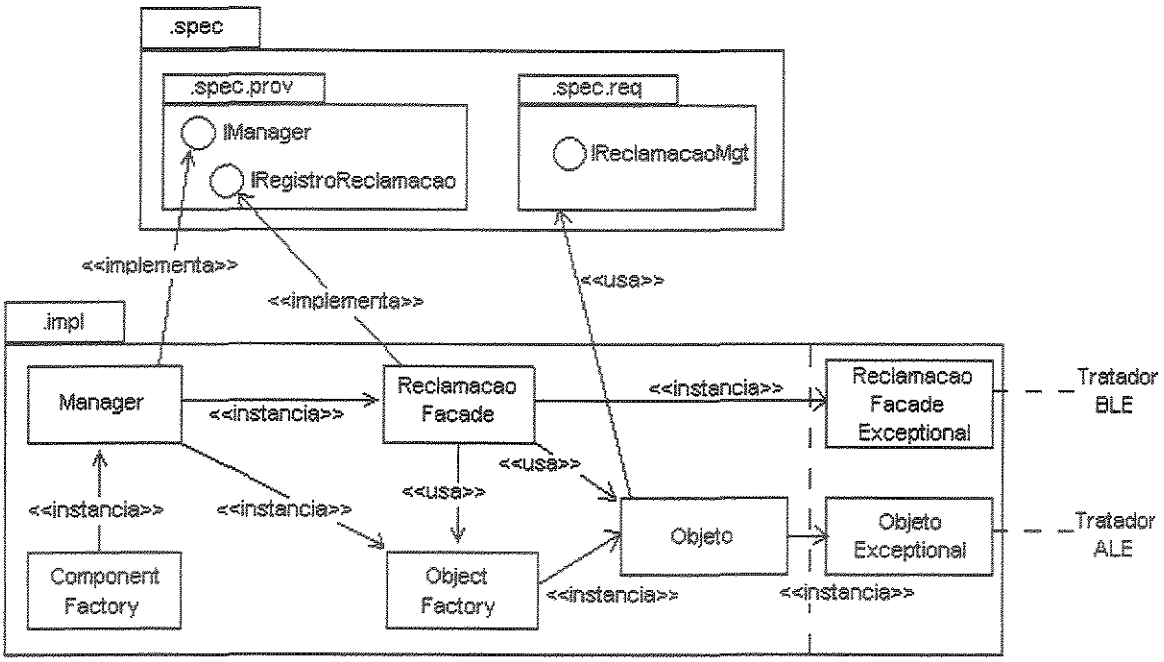

Figura 68 – Componente RegistrarReclamacao com tratadores

Neste exercício o componente ReclamacaoMgr foi reutilizado levando-se em consideração a estratégia intra-componente apresentada na subseção 2.6.2 e tomando-se como base a sua especificação. Mas para o reuso deste componente, considerou-se que o integrador não teria acesso ao código fonte. Desta forma, a estratégia intra-componente para reuso de componentes foi aplicada. A Figura 69 apresenta como ficou a adaptação deste componente com a criação de um *wrapper*.

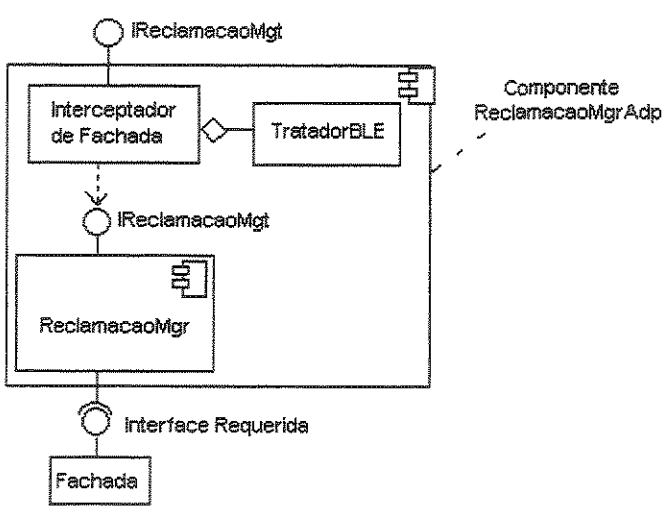

Figura 69 – Componente ReclamacaoMgr adaptado

O novo componente criado, como um *wrapper*, foi chamado de ReclamacaoMgrAdp (Onde o sufixo "Adp" é abreviação de "Adaptação"). Este novo componente teve um pacote *spec.prov* e um pacote *impl*. Em seu pacote *spec.prov* foi definida a interface IManager e IReclamacaoMgt. Em seu pacote *impl*, foi definida classe a Manager, um interceptador de fachada, chamado ReclamacaoMgtFacadeAdp, e um tratador BLE associado ao interceptador. O interceptador de fachada foi definido para utilizar a interface provida IReclamacaoMgt do componente ReclamacaoMgr.

Como apresentado na Figura 63, o conector ReclamacaoConnMrg interligava os componentes RegistrarReclamacao e ReclamacaoMgr. Para que o conector passasse a interligar os componentes RegistrarReclamacao e ReclamacaoMgrAdp, e para que o componente ReclamacaoMgrAdp passasse a utilizar o componente ReclamacaoMgr, como apresentado na Figura 70, foram feitas alterações na configuração dos componentes, a qual é utilizada pelo sistema em sua inicialização para ligar os componentes e conectores. Nestas adaptações, a interface IReclamacaoMgt requerida pelo conector passou a ficar sob responsabilidade da fachada ReclamacaoMgtFacadeAdp. Além disso, teve-se um passo a mais de configuração que foi deixar a fachada do componente ReclamacaoMgr responsável pela interface requerida do interceptador ReclamacaoMgtFacadeAdp, a IReclamacaoMgt.

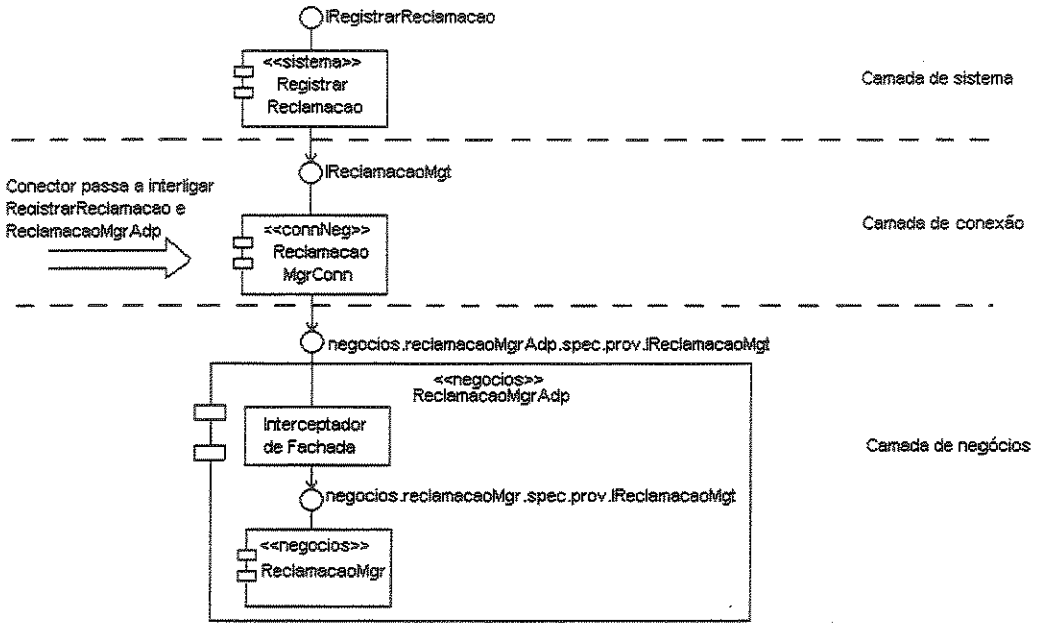

Figura 70 – Conector ReclamacaoMgrConn Interligando Componentes RegistrarReclamacao e ReclamacaoMgrAdp

O diagrama de seqüência da Figura 71 apresenta um fluxo de execução diferente do fluxo apresentado na Figura 64, a partir do momento que o conector ReclamacaoMgrConn procura o responsável pela interface requerida IReclamacaoMgt. Neste novo fluxo, quando o ReclamacaoConnFacade do conector ReclamacaoMgrConn pergunta para o Manager M3, do conector, quem é responsável pela interface IReclamacaoMgt, o ReclamacaoConnFacade tem como retorno o interceptador de fachada F5_, do componente ReclamacaoMgrAdp, e não mais a fachada F5 do componente ReclamacaoMgr. Quando o interceptador de fachada F5_ recebe a requisição para registrar os dados de reclamação, ele pergunta ao Manager M4, do componente ReclamacaoMgrAdp, quem é responsável pela interface IReclamacaoMgt e tem como retorno a fachada F5 do componente ReclamacaoMgr.

Quando o interceptador de fachada ReclamacaoMgtFacadeAdp pergunta para o Manager M4 quem é responsável pela interface IReclamacaoMgt, ele espera como retorno um objeto que implemente a interface IReclamacaoMgt provida pelo componente ReclamacaoMgr.

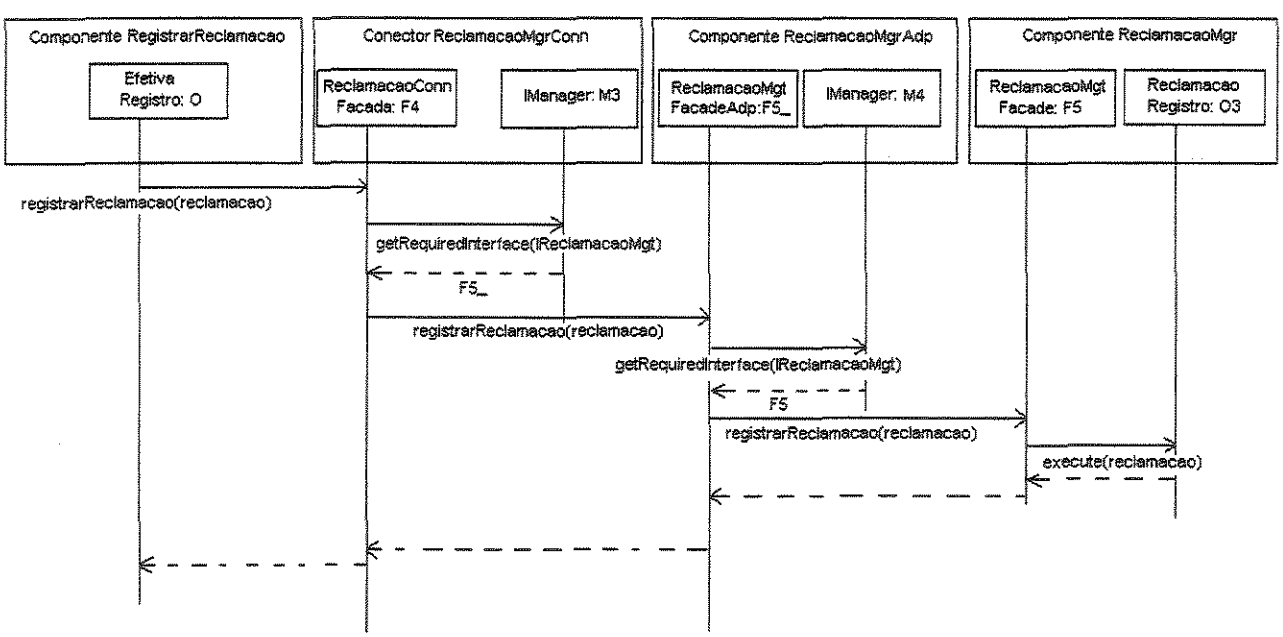

Figura 71 – Diagrama de Seqüência Após Adapatação para Reuso do Componente ReclamacaoMgr

O conector ReclamacaoMgrConn foi alterado de acordo com a estratégia inter-componente da subseção 2.6.3, passando a ter um tratador CLE.

Com a criação de um novo componente RegistrarReclamacao, com a criação de um novo tratador para o conector ReclamacaoMgrConn, e com as adaptações realizadas para se reutilizar o

componente ReclamacaoMgr, foi possível identificar e integrar componentes que atendessem as especificações excepcionais definidas na arquitetura.

No caso do fluxo de execução para registro de dados de reclamação, o componente implementado RegistrarReclamacao passou a ter um tratador ALE. Este tratador ficou preparado para tratar as exceções que poderiam ocorrer na operação efetivarRegistro(...) da interface requerida IReclamacaoMgt: E_DadosNaoValidosParaRegistroDeReclamacaoExcepcion e E_RegistroReclamacaoNoBancoException. As classes de implementação do componente RegistrarReclamacao ficaram responsáveis por detectar estas exceções. Além disso, o tratador BLE deste componente ficou preparado para lidar com condições excepcionais não antecipadas que poderiam ser detectadas pela fachada do componente.

O componente ReclamacaoMgrAdp ficou preparado para lançar as exceções E_DadosNaoValidosParaRegistroDeReclamacaoExcepcion e E_RegistroReclamacaoNoBancoException, considerando que o componente ReclamacaoMgr pode lançar exceções do tipo SQLException. Ao lidar com uma exceção SQLException, o interceptador de fachada do componente ReclamacaoMgrAdp ativa o tratador associado a ele, que por sua vez transforma esta exceção SQLException em E_RegistroReclamacaoNoBancoException, que então é lançada. Caso haja violação de pré-condição na chamada do método para registro de reclamação, o interceptador de fachada lança a exceção E_DadosNaoValidosParaRegistroDeReclamacaoExcepcion.

O conector ficou preparado para lidar com condições excepcionais não antecipadas, através do uso de um tratador CLE. Este tratador ficou responsável por tratar condições excepcionais não antecipadas. Caso uma exceção não declarada alcance o conector, o tratador CLE transforma a exceção não declarada em uma exceção definida na hierarquia de tipos abstratos de exceções (Ver subseção 2.6.1).

Exemplos de Implementação de Componentes e Conectores

A seguir, apresenta-se trechos de código de classes dos componentes RegistrarReclamacao, ReclamacaoMgrAdp, ReclamacaoMgr e do conector ReclamacaoMgrConn, associados ao caso de uso de registro de reclamação. O Quadro 10 apresenta parcialmente o código da classe ReclamacaoFacade. Em negrito pode-se ver as exceções que a operação efetivarRegistro pode

lançar, o bloco try/catch que detecta uma condição excepcional não antecipada e inicializa o tratador BLE, e o bloco que lida com violação de pré-condição e inicializa o tratador BLE.

```
package sistema.registrarReclamacao.impl;
//Seção para importação de pacotes
class ReclamacaoFacade implements IRegistroReclamacao, IOperacoesGenericas{
  private Manager manager;
  public ReclamacaoFacade(Manager manager){
    this.manager = manager;
  }
  public String efetivarRegistro(Reclamacao reclamacao,
     Usuario usuario) throws E_DadosRegistroInvalidosException,
     E_IdUsuarioNaoRetornadoException, E_UsuarioNaoRegistradoException,
     E_ReclamacaoNaoRegistradaException, RecoveredFailureException,
     UnrecoveredFailureException, RequestRejectedException{
    ObjectFactory fac = manager.getFactory();
    EfetivaRegistro obj = fac.getEfetivaRegistro();
    String idReclamacao = null;
    try{
      //Esta operação pode lançar as exceções: E_ReclamacaoNaoRegistradaException,
      //E_IdUsuarioNaoRetornadoException, E_UsuarioNaoRegistradoException
      idReclamacao = obj.execute(usuario, reclamacao, manager);
    }catch (java.lang.RuntimeException e){
      //Inicializa o tratador para lidar com exceção não esperada
      //O tratador poderá lançar uma RecoveredFailureException ou UnrecoveredFailureException
      //dependendo do estado deste componente
      ReclamacaoFacade_Exceptional ex = new ReclamacaoFacade_Exceptional();
      ex.handle(e);
    }
    if (!dadosValidos(usuario, reclamacao)){
      //Caso haja uma exceção não esperada durante a validação dos dados dos objetos 'usuario' e
      //'reclamacao', uma exceção RequestRejectedException é lançada
      //Caso não haja exceção ao validar dados, mas os mesmos sejam inválidos, inicializa o tratador
      //que poderá lançar uma E_DadosRegistroInvalidosException
      ReclamacaoFacade_Exceptional ex = new ReclamacaoFacade_Exceptional();
       ex.handle();
    }
    return idReclamacao;
  }
  ...
}
```

Quadro 10 – Código da Classe ReclamacaoFacade do Componente RegistrarReclamacao

O Quadro 11 apresenta parcialmente a classe EfetivaRegistro do componente RegistrarReclamacao. Em negrito pode-se ver o tratador ALE sendo inicializado para tratar uma condição excepcional esperada. Em caso de erro, o tratador ALE deve executar um *rollback* e lançar uma das exceções: E_IdUsuarioNaoRetornadoException,

E_UsuarioNaoRegistradoException, E_ReclamacaoNaoRegistradaException, dependendo do erro ocorrido.

```
package sistema.registrarReclamacao.impl;
//Seção de importação de pacotes
public class EfetivaRegistro {
  public String execute(Usuario usuario, Reclamacao reclamacao, Manager manager)
   throws  E_IdUsuarioNaoRetornadoException, E_UsuarioNaoRegistradoException,
           E_ReclamacaoNaoRegistradaException, RecoveredFailureException, UnrecoveredFailureException {
   ...
   sistema.registrarReclamacao.spec.req.IUsuarioMgt iUsuarioMgt =
    (IUsuarioMgt)manager.getRequiredInterface("sistema.registrarReclamacao.spec.req.IUsuarioMgt");
   boolean erroUserId = false;
   boolean erroUserNaoRegistrado = false;
   String userId = new String();
   E_IdUsuarioNaoRetornadoException erroUserIdEx = null;
   E_RegistroUsuarioNoBancoException erroUserNaoRegistradoEx = null;
   try{
    userId = iUsuarioMgt.registrarUsuario(factory.getUsuarioDAO(), genOp, usuario);
   }catch (E_IdUsuarioNaoRetornadoException e){
    erroUserId = true;
    erroUserIdEx = e;
   }catch (E_RegistroUsuarioNoBancoException e){
    erroUserNaoRegistrado = true;
    erroUserNaoRegistradoEx = e;
   }
   String reclamacaoId = new String();
   boolean erroRec = false;
   E_RegistroReclamacaoNoBancoException recException = null;
   sistema.registrarReclamacao.spec.req.IReclamacaoMgt iReclamacaoMgt = null;
   if (!erroUserId && !erroUserNaoRegistrado){
    try{
     iReclamacaoMgt =  (IReclamacaoMgt)manager.getRequiredInterface("sistema.registrarReclamacao.
                       spec.req.IReclamacaoMgt");
     reclamacaoId = iReclamacaoMgt.registrarReclamacao(userId, factory.getReclamacaoDAO(), genOp,
                                                       reclamacao);
    }catch (E_RegistroReclamacaoNoBancoException e){
        erroRec = true;
        recException = e;
    }
   }
   if (erroUserId || erroUserNaoRegistrado || erroRec) {
    //Inicializa o tratador
    EfetivaRegistro_Exceptional ex = new EfetivaRegistro_Exceptional();
    ex.handle(genOp, erroUserId, erroUserIdEx, erroUserNaoRegistrado,
              erroUserNaoRegistradoEx, erroRec, recException);
   }else{
    genOp.commit();
    genOp.close();
   }
   return reclamacaoId;
  }
  ...
}
```

Quadro 11 – Código da Classe EfetivaRegistro do Componente RegistrarReclamacao

O Quadro 12 apresenta parcialmente a classe ReclamacaoConnFacade. O primeiro trecho em negrito mostra que a fachada do conector pergunta ao Manager do conector quem é o responsável pela interface requerida negocios.reclamacaoMgrAdp.spec.prov.IReclamacaoMgt. A fachada recebe, em tempo de execução, a instância do interceptador de fachada do componente ReclamacaoMgrAdp.

O segundo trecho em negrito apresenta a detecção de condições excepcionais não antecipadas. Caso ocorra uma condição excepcional não antecipada, o tratador poderá lançar a exceção RecoveredFailureException ou UnrecoveredFailureException.

```
package connNeg.reclamacaoMgrConn.impl;
//Seção de importação de pacotes
class ReclamacaoConnFacade
 implements sistema.registrarReclamacao.spec.req.IReclamacaoMgt,
        sistema.registrarReclamacao.spec.req.ITipoReclamacaoMgt,
        sistema.consultarReclamacao.spec.req.IReclamacaoMgt,
        sistema.alterarReclamacao.spec.req.IReclamacaoMgt {
 private IManager manager = null;
 public ReclamacaoConnFacade(IManager manager) {
  this.manager = manager;
 }
 public String registrarReclamacao(String userId, ReclamacaoDAO dao,
  GenericOperations aGenOp, Reclamacao reclamacao) throws E_RegistroReclamacaoNoBancoException,
    RecoveredFailureException, UnrecoveredFailureException {
  negocios.reclamacaoMgrAdp.spec.prov.IReclamacaoMgt iReclamacaoMgt;
  iReclamacaoMgt = (negocios.reclamacaoMgrAdp.spec.prov.IReclamacaoMgt) manager.
          getRequiredInterface("negocios.reclamacaoMgr.spec.prov.IReclamacaoMgt");
  String reclamacaoId = null;
  try{
   //Esta operação pode lançar E_RegistroReclamacaoNoBancoException
   reclamacaoId = iReclamacaoMgt.registrarReclamacao(userId, dao, aGenOp, reclamacao);
  }catch(java.lang.RuntimeException e){
   ReclamacaoConnFacade_Exceptional ex = new ReclamacaoConnFacade_Exceptional();
   ex.handle(e);
  }
  return reclamacaoId;
 }
 ...
}
```

Quadro 12 – Código da Classe ReclamacaoConnFacade do Conector ReclamacaoMgrConn

O Quadro 13 apresenta a classe ReclamacaoMgtFacadeAdp do componente ReclamacaoMgrAdp. No primeiro trecho em negrito a fachada ReclamacaoMgtFacadeAdp pergunta ao manager do componente ReclamacaoMgrAdp, quem é o responsável pela interface negocios.reclamacaoMgr.spec.prov.IReclamacaoMgt. A fachada do componente

ReclamacaoMgrAdp tem como retorno a fachada do componente ReclamacaoMgr, que implementa a interface negocios.reclamacaoMgr.spec.prov.IReclamacaoMgt. A fachada do componente ReclamacaoMgrAdp chama a operação registrarReclamacao(...) da fachada do componente ReclamacaoMgr. Esta operação pode lançar a exceção SQLException. Caso uma exceção SQLException seja detectada, o tratador BLE do componente ReclamacaoMgrAdp é inicializado e transforma a SQLException em uma exceção do tipo E_RegistroReclamacaoNoBancoException.

As classes do componente ReclamacaoMgr não são mostradas aqui pois são as mesmas dos Quadro 7, Quadro 8 e Quadro 9.

```
package negocios.reclamacaoMgrAdp.impl;
//Seção de importação de pacotes
public class ReclamacaoMgtFacadeAdp
 implements negocios.reclamacaoMgrAdp.spec.prov.IReclamacaoMgt{

 private negocios.reclamacaoMgrAdp.spec.prov.IManager manager;
 public ReclamacaoMgtFacadeAdp(negocios.reclamacaoMgrAdp.spec.prov.IManager manager){
  this.manager = manager;
 }
 public String registrarReclamacao(String userId, ReclamacaoDAO dao,
  GenericOperations aGenOp, Reclamacao reclamacao) throws E_RegistroReclamacaoNoBancoException{
  negocios.reclamacaoMgr.spec.prov.IReclamacaoMgt iReclamacaoMgt =
    (negocios.reclamacaoMgr.spec.prov.IReclamacaoMgt)manager.
     getRequiredInterface("negocios.reclamacaoMgr.spec.prov.IReclamacaoMgt");
  String reclamacaoId = new String();
  try{
   reclamacaoId = iReclamacaoMgt.registrarReclamacao(userId, dao, aGenOp, reclamacao);
  }catch (SQLException e){
   ReclamacaoMgtFacadeAdp_Exceptional handler = new ReclamacaoMgtFacadeAdp_Exceptional();
   handler.handle(e);
  }
  return reclamacaoId;
 }
}
```

Quadro 13 – Código da Classe ReclamacaoMgtFacadeAdp do Componente ReclamacaoMgrAdp

Exercícios de Injeção de Falhas

Ao se terminar a integração dos componentes, realizou-se exercícios de injeção de falhas nos componentes para verificar como o subsistema de gerência de reclamações reagiria diante de condições excepcionais não antecipadas, durante o registro de uma reclamação. Os exercícios de

injeção de falhas foram realizados através da alteração de código dos componentes fazendo com que os mesmos lançassem determinadas exceções.

A primeira injeção simulou uma falha no tratador BLE do componente ReclamacaoMgrAdp. Alterou-se o código de modo que fosse lançada uma exceção NullPointerException de Java. Compilou-se o componente e iniciou-se o sistema. A exceção NullPointerException foi lançada e detectada pelo conector ReclamacaoMgrConn. A exceção foi tratada pelo tratador CLE, que por sua vez lançou uma exceção RecoveredFailureException.

A segunda injeção simulou, num contexto independente da primeira simulação de injeção, uma falha no conector ReclamçãoMgrConn logo antes deste conector retornar, para o componente RegistrarReclamacao, o resultado da operação registrarReclamacao(reclamacao), que pode ser vista na Figura 64. Esta falha se propagou até a fachada do componente RegistrarReclamacao e foi tratada pelo tratador BLE deste componente, que por sua vez realizou o *rollback* da operação de registro de dados de usuário e de reclamação, e lançou uma exceção RecoveredFailureException.

Além de exercícios de injeção de falhas associados a condições excepcionais não antecipadas, uma terceira injeção de falha forçou o lançamento de uma SQLException no componente ReclamacaoMgr, que foi transformada na exceção E_RegistroReclamacaoNoBancoException pelo tratador BLE do componente ReclamacaoMgrAdp. Esta terceira injeção foi realizada num contexto independente das duas primeiras injeções de falhas.

Considerações sobre Declaração de Exceções e Uso de Conectores

Um sistema de software baseado em componentes pode prover um pacote público com declarações e definições de exceções que seus componentes podem lançar e tratar. Por outro lado, um componente pode prover um pacote público com declarações e definições de exceções que pode lançar devido a erros em suas operações. Estas declarações e definições normalmente são implementadas na forma de classes abstratas de exceções, nestes pacotes.

Nos exercícios apresentados até este momento, assumiu-se que as exceções definidas na especificação excepcional de cada componente eram declaradas e especificadas em um pacote público e acessível pelos componentes do sistema. Outros exercícios foram realizados considerando que componentes teriam um pacote público com declarações de exceções que poderiam ser lançadas por eles. Nestes outros exercícios, foram definidos: o pacote *spec.prov.types*

para o componente AlterarReclamacao da camada de sistema, e o pacote *spec.prov.types* para o componente UsuarioMgr da camada de negócios. Além disso, foi considerado um cenário onde o integrador não tinha acesso ao código de componentes reutilizáveis, ou seja, ele deveria integrar componentes já existentes que atendessem as especificações de AlterarReclamacao e do UsuarioMgr, mas sem alterar seus códigos. Ao aplicar a estratégia intra-componente para estes componentes, criou-se um *wrapper* com o nome AlterarReclamacaoAdp e outro com o nome de UsuarioMgrAdp. O tratador BLE do componente AlterarReclamacaoAdp ficou responsável por tratar exceções que o componente AlterarReclamacao pudesse lançar, transformando-as em exceções declaradas em seu pacote .spes.prov.types. O tratador BLE do componente UsuarioMgrAdp ficou responsável por tratar exceções que o componente UsuarioMgr pudesse lançar, transformando-as em exceções declaradas em seu pacote .spes.prov.types. Ao aplicar a estratégia inter-componente, foi necessário fazer com que o tratador CLE do conector traduzisse exceções especificadas no pacote *spec.prov.types* do componente UsuarioMgrAdp, para exceções que pudessem ser tratadas pelo componente AlterarReclamacao.

Durante o estudo de caso, e com base neste último cenário apresentado, percebeu-se que a estratégia inter-componente seria tão importante quanto a estratégia intra-componente. Com avanços nos exercícios pôde-se perceber que mesmo que as exceções, que componentes pudessem lançar, estivessem declaradas em um pacote público para todos os componentes do sistema, o uso de conectores seria fundamental para o caso onde o integrador não tivesse acesso ao código de componentes reutilizáveis da camada de sistema. Isto ocorre pois caso o integrador tenha que reutilizar um componente na camada de sistema sem ter acesso ao seu código, ele não tem como alterar o componente para que este componente passe a tratar exceções que são lançadas por um componente da camada de negócios, apenas aplicando a estratégia intra-componente.

4.5 Análise de Resultados do Estudo de Caso

Durante o primeiro incremento do estudo de caso, aplicou-se o processo UML Components e o modelo COSMOS. Pôde-se definir uma arquitetura com especificação normal de seus componentes, implementá-los e testar o comportamento normal do sistema. O primeiro incremento atingiu seus objetivos: (1) aprendizado do processo UML Components, (2) aprendizado do modelo COSMOS, e (3) criação de componentes para serem reutilizados no segundo incremento.

Durante o segundo incremento do estudo de caso, aplicou-se o método MECE, as estratégias de estruturação arquitetural e continuou-se seguindo o modelo COSMOS. Pôde-se definir uma arquitetura com especificação normal e excepcional de seus componentes. Na fase de provisionamento, reutilizou-se componentes implementados no primeiro incremento. Foi possível a realização de diferentes exercícios de reuso de componentes, considerando que o integrador tinha acesso ao código de alguns componentes e considerando que o integrador não tinha acesso ao código de outros componentes.

No exercício que envolveu o reuso de um componente com acesso ao seu código, pôde-se perceber a importância de se ter uma arquitetura bem organizada com especificação normal e excepcional de seus componentes. Para que se pudesse preparar um componente de modo que ele lançasse e tratasse as exceções definidas no contrato de realização, as tabelas TECEp e TECEr (ver subseção 3.1.2) foram fundamentais por apresentarem as informações de modo organizado.

No exercício que envolveu o reuso de um componente da camada de negócios, sem acesso ao seu código, percebeu-se que uma especificação excepcional bem organizada é importante para que se possa criar um *wrapper* adequado que lance exceções definidas na especificação. No exercício que envolveu o reuso de um componente da camada de sistema, sem acesso ao seu código, a especificação excepcional foi fundamental para que se pudesse criar um conector que traduzisse exceções lançadas pela camada de negócios em exceções que o componente reutilizado na camada de sistema pudesse tratar.

Durante os exercícios de reuso de componentes, pôde-se perceber os benefícios de se aplicar as estratégias intra e inter-componente. Com a aplicação da estratégia intra-componente em um componente reutilizável cujo código era acessível, pôde-se preparar o código com tratadores ALE e BLE e adicionar o tratamento de condições excepcionais não antecipadas. Com a aplicação da estratégia intra-componente em um componente reutilizável cujo código não era acessível, pôde-se: criar um *wrapper* para o componente reutilizado, e criar um tratador BLE associado a fachada deste *wrapper* para lidar com condições excepcionais não antecipadas e para traduzir as exceções que o componente reutilizado pudesse lançar em exceções definidas na especificação excepcional do componente.

A estratégia inter-componente foi aplicada em duas situações: (1) quando se tinha acesso ao código do componente reutilizável da camada de sistema mas não se tinha acesso ao código de

um componente reutilizável da camada de negócios, e (2) quando não se tinha acesso ao código do componente reutilizável da camada de sistema e também não se tinha acesso ao código de um componente reutilizável da camada de negócios.

Na primeira situação, a estratégia intra-componente aplicada ao componente da camada de sistema, permitiu preparar este componente para lidar com exceções que poderiam ocorrer devido a erros em operações de suas interfaces requeridas. Desta forma, a estratégia inter-componente foi aplicada para lidar apenas com condições excepcionais não antecipadas. Na segunda situação, não foi possível alterar o componente da camada de sistema para lidar com exceções que pudessem ocorrer em suas interfaces requeridas, com a estratégia intra-componente. Desta forma, a estratégia inter-componente foi aplicada para lidar com tradução de exceções entre componente da camada de negócio e o componente da camada de sistema, além de lidar com condições excepcionais não antecipadas. Tanto na primeira situação quanto na segunda, um conector foi criado contendo um tratador CLE.

A especificação normal e excepcional de componentes e a aplicação das estratégias intra e inter-componente transcorreram sem dificuldades durante os exercícios do segundo incremento. Pode-se dizer que o segundo incremento atingiu seus objetivos: (1) estudo dos benefícios do método MECE, e (2) estudo dos benefícios das estratégias intra e inter-componente.

Ao se alterar um componente para que o mesmo atendesse a especificação excepcional pôde-se observar que: (1) a alteração envolveu mais tempo para se codificar o tratamento de exceções, ou seja, por um lado gastou-se mais tempo, mas por outro lado teve-se um sistema onde exceções eram detectadas e tratadas de acordo com especificação excepcional definida na arquitetura, que por sua vez foi definida com base em cenários excepcionais definidos na fase de especificação de requisitos. Caso existissem ferramentas para geração de código, com base em especificação excepcional, o tempo para a codificação de tratamento de exceções poderia ser reduzido; (2) o impacto da implementação de tratamento de exceções na estrutura de código dos componentes para quais se tinha acesso ao código, foi pequeno. Este impacto ocorreu quando blocos de try/catch foram adicionados às classes.

4.6 Considerações do Capítulo

Este capítulo apresentou um estudo de caso com a aplicação da solução proposta neste trabalho em um sistema de software real. Neste estudo de caso pôde-se aplicar o método MECE e as estratégias de estruturação arquitetural para tratamento de exceções, e observar seus benefícios.

O estudo de caso foi dividido em dois incrementos. O primeiro incremento envolveu fases de definição de requisitos, especificação de componentes, provisionamento e integração, com o foco no comportamento normal do sistema. O segundo incremento também envolveu todas estas fases mencionadas, mas teve o foco no comportamento excepcional do sistema.

As atividades de cada fase do primeiro incremento e do segundo incremento foram apresentadas. Este capítulo mostrou como aplicar o processo UML Components nas fases de definição de requisitos e especificação de componentes do primeiro incremento. Durante as fases de provisionamento e integração do primeiro incremento, foi mostrado como aplicar o modelo COSMOS. Este capítulo mostrou como aplicar o método MECE nas fases de definição de requisitos e especificação de componentes do segundo incremento. Durante as fases de provisionamento e integração do segundo incremento, foi mostrado como aplicar as estratégias de estruturação arquitetural para tratamento de exceções.

Durante a fase de provisionamento e integração do segundo incremento, diversos exercícios de reuso de componentes, implementados no primeiro incremento, foram realizados. Pôde-se também realizar exercícios de injeção de falhas para se estudar o comportamento do sistema na presença de falhas. Estes exercícios permitiram o estudo dos benefícios de se ter uma arquitetura bem organizada com especificação normal e excepcional de seus componentes e de se aplicar as estratégias de estruturação arquitetural para tratamento de exceções.

Por fim, este capítulo apresentou uma análise dos resultados obtidos durantes os incrementos do estudo de caso.

Capítulo 5

Contribuições, Trabalhos Futuros e Conclusões

Este capítulo apresenta as contribuições deste trabalho, possíveis trabalhos futuros e conclusões. Entre as contribuições destaca-se a elaboração de um artigo publicado na Euromicro 2004 *Conference*.

5.1 Contribuições

Publicação de Artigo na Euromicro 2004 *Conference*

Durante os estudos de caso com o Telestrada, as estratégias de estruturação arquitetural para tratamento de exceções foram aplicadas conforme apresentado na subseção 4.4. O estudo de caso fez parte do artigo "*Structuring Exception Handling for Dependable Component-Based Software Systems*" [Guerra+03], submetido para simpósios da área e aceito no *Workshop on Component Models for Dependable Systems,* da *EUROMICRO 2004 Conference.* A implementação contribuiu para análises e discussões sobre a proposta das estratégias e para sua evolução, descrita em [Guerra04].

Definição do Método MECE

O método MECE envolve a definição de requisitos e especificação normal e excepcional dos componentes de uma arquitetura. O método integra um processo utilizado em mercado, o UML Components, com a metodologia MDCE, além de propor outras atividades complementares. O método MECE contribui para desenvolvimento de sistemas de software baseados em componentes e confiáveis, podendo ser aplicado em sistemas *Web* caracterizados por arquiteturas em camadas e por requisitos de confiabilidade, mas especificamente de fiabilidade, disponibilidade e segurança. Sistemas *Web* são amplamente utilizados no mercado.

Proposta da Solução Arquitetural com Tratamento de Exceções

Com o método MECE tem-se orientações para definição de requisitos e especificação de componentes para sistemas de software baseados com componentes e confiáveis. Com a proposta de estruturação arquitetural para tratamento de exceções tem-se orientações para as fases de provisionamento e integração de componentes destes sistemas. Com a aplicação de ambos, tem-se

uma solução centrada na arquitetura para desenvolvimento de sistemas de software baseados em componentes e confiáveis.

Aprimoramento do COSMOS

Durante os estudos de caso com o Telestrada, foi possível ajudar alunos do Instituto de Computação nos estudos e na aplicação do COSMOS. Permitiu a observação das vantagens apresentadas pelo COSMOS e a definição de propostas de aprimoramento para o mesmo, como: (1) a definição de padrão para uso de tipos de dados comuns a componentes, que pode ser visto em [Silva03], e (2) a definição de que exceções que podem ser lançadas por um componente podem estar declaradas no pacote *spec.prov.types*, como classes abstratas.

Estudos de Testes e Injeção de Falhas em Componentes

O estudo de caso do Telestrada, foi utilizado para auxiliar estudos sobre injeção de falhas em componentes e testes destes componentes com o uso da ferramenta Jaca, que é baseada em *Javassist* [Chiba00]. Estes estudos envolvem uma proposta de priorização de componentes a serem testados em um sistema. O grupo de estudos de testes, do Instituto de Computação, escreveu um artigo [Moraes03] sobre estes estudos fazendo referência ao sistema Telestrada.

Ferramentas para Desenvolvimento Baseado em Componentes

Durante os estudos de caso com o Telestrada, foi possível estudar o Eclipse, o WebShere Server e o WebSphere Studio, e apresentar para o grupo de estudos de ferramentas os módulos destas ferramentas que pudessem contribuir para desenvolvimento baseado em componentes, como *plug-in* de modelagem de arquitetura do Eclipse, facilidades de manipulação de componentes do WebSphere, entre outros. Além disso, discutiu-se possibilidades de criação de novos *plug-ins* para geração de código Java, com base em arquitetura de componentes e seguindo o modelo COSMOS, e para testes de componentes com injeção de falhas.

5.2 Trabalhos Futuros

Novos Plug-Ins para Eclipse

Durante o estudo de caso do Telestrada, percebeu-se a necessidade e oportunidade de se desenvolver novos *plug-ins* para o Eclipse, ou para WebPhere Studio, para: (1) permitir especificação de comportamento excepcional de componentes, (2) geração de código para

componentes com base em especificação do comportamento normal e excepcional, (3) geração de código para componentes seguindo o modelo COSMOS, (4) auxiliar injeção de falhas em componentes em tempo de projeto e execução, (5) mater a rastreabilidade entre requisitos definidos, implementados e testados. Desta forma, pode-se contribuir para um ambiente com ferramentas de desenvolvimento de sistemas de software baseado em componentes e confiáveis.

Novos Estudos de Caso para a Solução Arquitetural

Um possível trabalho futuro seria terminar o desenvolvimento do subsistema de gerência de reclamações do Telestrada, aplicando a solução arquitetural proposta nesta monografia. Outro trabalho seria aplicar a solução proposta a outros sistemas. Estes trabalhos poderiam estudar: (1) o uso de conectores mais complexos entre componentes de diferentes camadas, (2) a especificação de componentes para as camadas de interface e diálogo com usuário e para camadas de mais baixo nível. De modo mais genérico, pesquisar uma proposta de se especificar todos os componentes de todas as camadas de uma arquitetura.

Avaliação de Confiabilidade em Sistemas

Um trabalho futuro seria avaliar o quanto um sistema se torna mais confiável com aplicação da abordagem proposta neste trabalho. Este trabalho futuro poderia envolver uma comparação entre uma versão V1 de um sistema, implementada sem a abordagem proposta neste trabalho, com uma versão V2 do mesmo sistema, implementada de acordo com a abordagem proposta neste trabalho. Esta comparação poderia ser realizada com base em injeção de falhas nas versões V1 e V2. Poderia-se injetar falhas e contabilizar o número de falhas tratadas devidamente por cada uma das versões. Além disto, poderia-se estudar qual o esforço necessário para se criar a versão V1, e comparar com o esforço necessário para se criar a versão V2. O estudo de caso com o Telestrada poderia ser utilizado nesta avalição.

Evolução e Gerência de Configuração de Componentes

Este trabalho considera que durante a fase de provisionamento e integração de componentes, os mesmos estão disponíveis para que integradores possam encontrá-los e reutilizá-los. Com o crescimento da prática de desenvolvimento baseado em componentes, empresas e universidades tendem a criar os seus repositórios de componentes, controlar a evolução dos componentes nestes repositórios e disponibilizá-los de forma controlada para que integradores de

diferentes organizações de software possam utilizá-los. Um integrador pode construir um sistema a partir de componentes de diferentes repositórios de empresas e universidades.

Para disponibilizar um repositório de forma controlada garantindo a confiabilidade de seus componentes, deve-se seguir boas práticas de gerência de configuração, tais como: controlar e gerar versões de componentes, definir e descrever atributos de qualidade de serviço para componentes que estarão disponíveis, aceitar novas versões de componentes ou novos componentes apenas após verificações de que os mesmos atendem suas especificações, entre outras. Por outro lado, para se utilizar componentes de diferentes repositórios, um integrador também deve aplicar boas práticas de gerência de configuração, como manter a informação de qual versão de um componente ele está usando e de qual repositório, manter registro de adaptações realizadas em componentes, entre outras.

Com base nos cenários apresentados acima pode-se iniciar diferentes frentes de estudos e trabalhos futuros em: (1) evolução e gerência de configuração de componentes do ponto de vista de desenvolvedores e do ponto de vista de integradores, e (2) evolução e confiabilidade de repositórios de componentes.

5.3 Conclusões

Este trabalho apresentou fundamentos teóricos sobre: desenvolvimento baseado em componentes, confiabilidade, arquitetura de software, o processo de desenvolvimento de software baseado em componentes UML Components, a Metodologia de Definição de Comportamento Excepcional (MDCE), as estratégias de estruturação arquitetural para tratamento de exceções, e um modelo de estruturação de componentes que promove a conformidade da arquitetura de um sistema com sua implementação (COSMOS). Além disso, este trabalho apresentou um método para especificação de componentes da arquitetura de um sistema e de seu comportamento excepcional (MECE). Este método é caracterizado por atividades de definição de requisitos, especificação de componentes e implementação, resultantes da integração entre a metodologia MDCE e o processo UML Components. Além destas atividades, o método MECE propõe outras atividades que orientam a identificação de exceções que podem ser lançadas e tratadas por componentes, e que orientam o uso de conectores na especificação da arquitetura.

Foi apresentado neste trabalho que o desenvolvimento de sistemas de software baseados em componentes tem fases de: definição de requisitos, especificação da arquitetura com componentes, provisionamento, integração de componentes, testes e instalação. Durante a fase de provisionamento, integradores procuram componentes que podem ser reutilizados em um sistema, ou desenvolvedores implementam novos componentes, ambos com base na arquitetura do sistema que deve apresentar a especificação normal e excepcional de seus componentes. Caso não se tenha uma especificação organizada de componentes, pode-se comprometer o trabalho de desenvolvimento destes componentes ou de identificação e integração de componentes já existentes.

Foi mostrado neste trabalho que o método MECE apresenta orientações importantes para se obter uma arquitetura bem organizada com especificação normal e excepcional de cada componente. Na especificação de comportamento normal, apresenta-se os componentes com suas interfaces providas e suas operações, suas interfaces requeridas, e os conectores que interligam os componentes. Na especificação do comportamento excepcional, apresenta-se que exceções podem ser lançadas por cada operação de cada interface provida de um componente, a pós-condição excepcional do componente após lançar uma exceção, e as exceções que um componente deve tratar devido a erros que possam ocorrer em operações de interfaces requeridas.

Foi mostrado também que mesmo com uma especificação completa de arquitetura, pode ser que: (1) um componente reutilizável apresente uma falha de projeto que o faça lançar um exceção não definida pela especificação do sistema onde é integrado, ou que (2) um componente reutilizável não lance e trate exceções definidas em uma especificação excepcional. Para lidar com estes problemas, este trabalho mostrou que deve-se aplicar as estratégias de estruturação arquitetural para tratamento de exceções.

Desta forma, este trabalho propôs uma solução arquitetural composta pela aplicação do método MECE e pela aplicação das estratégias de estruturação arquitetural para tratamento de exceções. Durante este trabalho aplicou-se a solução arquitetural proposta em um estudo de caso real e pôde-se analisar suas vantagens e limitações. Com a aplicação do método MECE teve-se uma especificação arquitetural adequada para que se pudesse identificar componentes prontos para reuso e para que se pudesse desenvolver novos componentes. Quanto ao uso das estratégias de estruturação arquitetural para tratamento de exceções, foi possível verificar: (1) a sua contribuição

ao se injetar falhas em componentes do sistema, forçando o lançamento de exceções não declaradas e observando que o sistema pôde transformar estas exceções em novas exceções que foram devidamente tratadas, e (2) o uso de tratadores BLE e CLE cuidando da incompatibilidade de tipos de exceções ao se reutilizar um componente sem acesso ao seu código.

Quanto a limitações da solução, percebeu-se que ela é específica para sistemas com arquiteturas em camadas, e que mais estudos devem ser realizados para especificação de componentes de camadas de interface com usuário.

Conclui-se que a solução arquitetural proposta neste trabalho contribui para o desenvolvimento de sistemas de software baseados em componentes e confiáveis, através de orientações de definição de requisitos, especificação de arquitetura, provisionamento e integração de componentes.

Referências Bibliográficas

[Alur+01] ALUR, Deepak; CRUPI, John; MALKS, Dan. Core J2EE Patterns. Sun Microsustems, Inc, 2001.

[Bass+02] BASS, Len; CLEMENTS, Paul; KASMAN, Rick. Software Architecture in Practice - SEI Series in Software Engineering. 2002.

[Chambers96] CHAMBERS, C. Towards Reusable, Extensible Components. ACM Computing Surveys, 28, December 1996.

[Cheesman+01] CHESSMAN, John; DANIEL John. UML Components: A Simple Process for Specifying Component-Based Software. Addison-Wesley, 2001.

[Chiba00] CHIBA, Shigueru. Getting Started with Javassist, 2000. Disponível em http://www.csg.is.titech.ac.jp/~chiba/javassist/ (Acessada em 10 de Dezembro de 2003).

[CockburnWeb] COCKBURN, Alistar. Structuring Use Cases with Goals, 1995. Disponível em http://alistair.cockburn.us/crystal/articles/sucwg/structuringucswithgoals.htm (Acessada em 8 de Maio de 2004).

[Cristian89] CRISTIAN, Flaviu. Exception Handling – Dependability of Resilient Computers (ed. T. Anderson), BlackWell Scientific Publications, 1989.

[Dellarocas97] DELLAROCAS, Chrysanthos. Toward Exception Handling Infrastructures for Component-Based Software. 1997. Disponível em http://www.sei.cmu.edu/cbs/icse98/papers/p23.html (Acessada em 5 de Maio de 2004).

[Desmond98] SOUZA, Desmond; WILL, A.C. Objects,Components and Frameworks with UML. The Catalysis Approach. Addisson-Wesley, 1998.

[EclipseWeb] The eclipse project. Disponível em http://www.eclipse.org (Acessada em 10 de Novembro de 2003).

[Ferreira01] FERREIRA, Gisele R. M. Tratamento de Exceções no Desenvolvimento de Sistema Baseados em Componentes. Dissertação de Mestrado – Instituto de Computação da Unicamp, 2001.

[Gamma+95] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. Design Patterns-Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[Guerra+03] GUERRA, Paulo Asterio de C.; FILHO, Fernando Castor; PAGANO, Vinicius Asta; RUBIRA, Cecília Mary F. Structuring Exception Handling for Dependable Component-Based Software Systems. 2003. Artigo publicado na conferência Euromicro 2004, no Workshop on Component Models for Dependable Systems.

[Guerra04] GUERRA, Paulo Asterio de C. Uma Abordagem Arquitetural para Tolerância a Falhas em Sistemas de Software Baseados em Componentes. Tese de Doutorado – Instituto de Computação da Unicamp, 2004.

[IBMWeb] IBM WebSphere Software Plataform. Disponível em http://www.ibm.com/websphere (Acessada em 10 de Novembro de 2003).

[Issarny+01] ISSARNY Valérie; BANÂTRE, Jean-Pierre. Architecture-Based Exception Handling. Proceedings of the 34th Hawaii International Conference on System Sciences. 2001. Disponível em http://portal.acm.org/citation.cfm?id=644769&dl=ACM&coll=portal (Acessada em 3 de Julho de 2004).

[J2EEWeb] Sun Microsystems. Java 2 platform, enterprise edition. Disponível em http://java.sun.com/j2ee/learning/tutorial/index.html (Acessada em 10 de Maio de 2004).

[J2SEWeb] Sun Microsystems. Java 2 platform, standard edition. Disponível em http://java.sun.com/docs/books/tutorial/index.html (Acessada em 10 de Maio de 2004).

[Jacobson+99] JACOBSON, I.; RUMBAUGJ, J.; BOOCH G. Unified Software Development Process - Addison-Wesley, Reading -MA, 1999.

[JakartaWeb] Jakarta Struts Project. Referência principal em http://jakarta.apache.org/struts (Acessada em 15 de Novembro de 2003). Referência para *download* em http://jakarta.apache.org/site/binindex.cgi (Acessada em 3 de Julho de 2004).

[JSPWeb] Sun Microsystems. Javaserver pages specification version 1.2. Disponível em http://java.sun.com/products/jsp (Acessada em 15 de Novembro de 2003).

[Kruchten95] KRUCHTEN, Philippe B. The 4+1 View Model of Software Architecture. Disponível em http://www.cs.bilkent.edu.tr/~bedir/CS411/Papers/ArchitecturalBlueprintsKruchten.pdf. (Acessada em 3 de Julho de 2004).

[Lee+90] LEE, P. A.; ANDERSON, T. Fault Tolerance: Principles and Practice, Springer-Verlag, 1990.

[Monroe+97] MONROE, Robert T.; KOMPANEK, Andrew; MELTON, Ralph; GARLAN, David. Architectural Styles, Design Patterns and Objects, IEEE Software, 1997.

[Moraes03] MORAES, Regina Lúcia de O. A Strategy for Testing Component-based Systems using Interface Fault Injection. 2003.

[OMondoWeb] OMondo. Referência principal em http://www.omondo.com (Acessada em 10 de Novembro de 2003). Referência para *download* em http://www.omondo.com/download/index.jsp (Acessada em 12 de Novembro de 2003).

[Pressman01] PRESSMAN, Roger S. Software Engineering: A Practitioner's Approach. McGrawHill, 2001.

[Randell+95] RANDELL, Brian; XU, Jie. The Evolution of the Recovery Block Concept. John Wiley Sons Ltd., 1995.

[Romanovsky01] ROMANOVSKY, Alexander. Exception Handling in Component-Based System Development. 2001. Disponível em http://www.cs.ncl.ac.uk/research/pubs/trs/papers/724.pdf (Acessada em 14 de Maio de 2004).

[Rubira+03] RUBIRA, Cecília Mary Fischer; GUERRA, Paulo Asterio de Castro. Desenvolvimento Baseado em Componentes e Confiabilidade. In: Desenvolvimento Baseado em Componentes. EDUEM (Editora da Universidade Estadual de Maringá), 2003. pp.10-34.

[Silberschatz+01] SILBERSCHATZ, Abraham; GALVIN, Peter; GAGNE, Greg. Sistemas Operacionais. Conceitos e Aplicações. Editora Campus – 2001.

[Silva03] SILVA, Moacir. COSMOS - Um Modelo de Estruturação de Componentes para Sistemas Orientados a Objetos. Dissertação de Mestrado – Instituto de Computação da Unicamp, 2003.

[Sommerville01] SOMMERVILLE, Ian. Software Engineering. Addison-Wesley – 2001.

[SunWeb] Tutorial sobre Tratamento de Exceções da *Sun Microsystems*. Disponível em http://java.sun.com/docs/books/tutorial/essential/exceptions/definition.html (Acessada em 20 de Maio de 2004).

[Szyperski97] SZYPERSKI, C. Component Software - Beyond Object Oriented Programming, Addison-Wesley, 1997.

[Szyperski00] SZYPERSKI, C. Component Software and the Way Ahead, In Foundations of Component-Based Systems, N.Y.: Cambridge University Press, 2000.

[Uml99] UML Revision Task Force, OMG Unified Modeling Language Specification, version 1.3, document ad/99-06-08. Object Management Group, June 1999.

Apêndice A – Casos de Uso do Telestrada

Neste apêndice apresenta-se os casos de uso não implementados do Telestrada. As informações dos apêndices podem servir para trabalhos futuros de continuidade de implementação do Telestrada.

Caso de Uso: Registro de Tipo de Reclamação
Iniciador: Supervisor do sistema
Objetivo: Definir um tipo de reclamação do sistema
Cenário de Sucesso:

1. Supervisor acessa página do Telestrada na Internet.
2. Supervisor seleciona opção 'registrar tipo de reclamação'.
3. Supervisor define um tipo de reclamação.
4. Supervisor seleciona opção para inserir tipo no sistema.
5. Sistema valida dados, verificando se o tipo de reclamação já existe.
6. Sistema armazena tipo.

Caso de Uso: Remoção de Tipo de Reclamação
Iniciador: Supervisor do sistema
Objetivo: Remover um tipo de reclamação do sistema
Cenário de Sucesso:

1. Supervisor acessa página do Telestrada na Internet.
2. Supervisor seleciona opção 'remover tipo de reclamação'.
3. Supervisor escolhe o tipo de reclamação.
4. Supervisor seleciona opção para remoção do tipo.
5. Sistema remove tipo.

Caso de Uso: Alteração de Reclamação
Iniciador: Supervisor do sistema
Objetivo: Editar reclamações inseridas por usuários de rodovia.
Cenário de Sucesso:

1. Supervisor acessa página do Telestrada
2. Supervisor escolhe 'alterar reclamação'.
3. Sistema apresenta lista de rodovias.
4. Supervisor escolhe rodovia.
5. Sistema apresenta lista de trechos da rodovia selecionada e tipos de reclamações.
6. Supervisor selecionar trecho e tipo de reclamação.
7. Sistema apresenta todas as reclamações para o dado trecho e tipo.
8. Supervisor seleciona reclamações que devem fazer parte de outro trecho e/ou tipo.
9. Supervisor seleciona opção alterar reclamações.
10. Supervisor seleciona trecho e/ou tipo corretos para as reclamações selecionadas no passo 8 e escolhe opção efetivar alteração.
11. Sistema transfere as reclamações de seu trecho e/ou tipo antigos para novos.

Caso de Uso: Definição de Processo de Reclamação
Iniciador: Supervisor do sistema
Objetivo: Criar processos de reclamações
Cenário de Sucesso:

1. Supervisor acessa página do Telestrada
2. Supervisor escolhe 'definir processo de reclamação'.
3. Sistema apresenta lista de rodovias, para supervisor.
4. Supervisor escolhe rodovia.

5. Sistema apresenta lista de trechos da rodovia selecionada e tipos de reclamações.
6. Supervisor seleciona trecho e tipo de reclamação.
7. Sistema apresenta todas as reclamações para o dado trecho e tipo.
8. Supervisor seleciona reclamações que devem fazer parte do processo a ser criado.
9. Supervisor seleciona opção 'criar processo de reclamações'.
10. Sistema cria novo processo de reclamações para o trecho e tipo de reclamações.

Caso de Uso: Alteração de Processo de Reclamação
Iniciador: Supervisor do sistema
Objetivo: Alterar processos de reclamações
Cenário de Sucesso:

1. Supervisor acessa página do Telestrada.
2. Supervisor escolhe 'alterar processo de reclamação'.
3. Sistema lista todos os processos existentes no sistema.
4. Supervisor seleciona um processo de uma lista de processos.
5. Sistema exibe todas as reclamações do processo, seu trecho e tipo.
6. Sistema exibe todas as reclamações associadas ao trecho e tipo, mas que atualmente não fazem parte do processo em questão.
7. Supervisor adiciona reclamações ao processo (a partir da lista do passo 6).
8. Sistema adiciona reclamações ao processo.

Cenário Alternativo

7. Supervisor remove reclamações do processo (o que faz as mesmas irem para a lista do passo 6).
   a. Sistema desassocia reclamações do processo.
   b. *Stop*.
7. Supervisor seleciona reclamações do processo da lista do passo 4.
   a. Supervisor seleciona opção para incluir reclamações, selecionadas, em outro processo.
   b. Sistema, remove reclamações do processo atual e as move para um novo processo, alterando assim o tipo e trecho das reclamações, se necessário.
   c. *Stop*.

Caso de Uso: Remoção de Processo de Reclamação
Iniciador: Supervisor do sistema
Objetivo: Remover processos de reclamações
Cenário de Sucesso:

1. Supervisor acessa página do Telestrada e escolhe 'remover processo de reclamação'.
2. Sistema lista todos os processos existentes no sistema.
3. Supervisor seleciona um processo da lista de processos.
4. Sistema exibe todas as reclamações do processo, seu trecho e tipo.
5. Supervisor escolhe opção para remover processo.
6. Sistema desassocia reclamações do processo e remove processo.

Caso de Uso: Encaminhamento de Processo de Reclamação
Iniciador: Supervisor do sistema
Objetivo: Encaminhar processos de reclamações
Cenário de Sucesso:

1. Supervisor acessa página do Telestrada e escolhe 'encaminhar processo de reclamação'.
2. Sistema seleciona processos que ainda não foram encaminhados para responsáveis pelos trechos (trechos estes, associados aos processos).
3. Para cada processo, o supervisor escolhe um responsável para lidar com o processo, a partir de uma lista pré-definida no sistema.
4. Para cada processo, o supervisor escolhe a forma de encaminhamento para responsável pelo trecho, entre: e-mail, fax ou impressão.
5. Supervisor escolhe opção efetivar encaminhamento.
6. Sistema envia cada processo a seu responsável de acordo com forma de encaminhamento.
7. Sistema atualiza cada reclamação de cada processo com o status de 'sendo analisada'.

Cenário Alternativo:

4. Supervisor pode selecionar um processo e ver seus detalhes (reclamações, tipo, trecho), em uma nova janela do sistema.

Caso de Uso: Atualização de Processo de Reclamação
Iniciador: Responsável por trecho
Objetivo: Atualizar processos de reclamações
Cenário de Sucesso:

1. Responsável por trecho acessa página do Telestrada e escolhe 'atualizar processo de reclamação'.
2. Sistema exibe os processos atribuídos ao responsável.
3. Responsável seleciona um processo para atualização.
4. Sistema exibe todas as reclamações do processo, seu trecho e tipo de reclamações.
5. Responsável atualiza processo de reclamação alterando o status das reclamações pertencentes ao processo, para 'resolvida'.
6. Responsável pode preencher um campo de comentário para cada reclamação, e um campo para o processo de reclamações em questão, se desejar.
7. Responsável pode também atribuir o comentário de um campo para todos os demais. (O campo da reclamação é apresentado para usuário final quando este consulta reclamação. O campo do processo é apresentado para o supervisor quando este visualiza o processo em qualquer caso de uso).
8. Sistema atualiza cada reclamação de cada processo com o status de 'resolvida'.

Caso de Uso: Definição de Resposta Padrão para Trecho e Tipo
Iniciador: Supervisor do sistema (ou Responsável por trecho).
Objetivo: Definir uma resposta padrão para um trecho e tipo de reclamação.
Cenário de Sucesso:

1. Supervisor acessa página do Telestrada e escolhe opção 'definir resposta padrão'.
2. Sistema exibe rodovias e tipos de reclamação.
3. Supervisor seleciona rodovia.
4. Sistema apresenta trechos da rodovia selecionada.
5. Supervisor seleciona um trecho e um tipo de reclamação.
6. Sistema apresenta um formulário com um campo para descrição de resposta padrão.
7. Supervisor escreve resposta padrão para trecho e tipo e escolhe opção para armazenar resposta padrão no sistema.
8. Sistema armazena resposta padrão.

Apêndice B – Diagramas da Arquitetura do Telestrada

Neste apêndice são apresentados os diagramas de componentes dos casos de uso do Telestrada, complementares aos já apresentados na monografia. Estes diagramas ainda não possuem a camada de conexão.

Registro e Remoção de Tipo de Reclamação: Na figura a seguir a camada de apresentação utiliza interfaces providas pelo componente TipoReclamacao que pertence a camada de sistema, que por sua vez utiliza a interface provida pela camada de negócios.

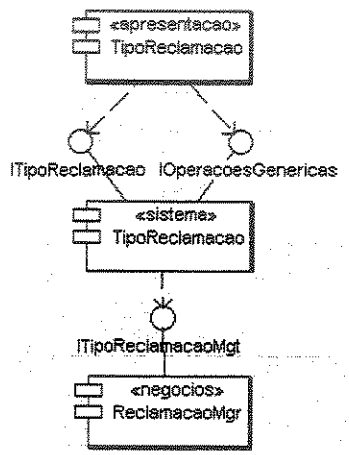

Figura 72 – Arquitetura Inicial de Registro e Remoção de Tipo de Reclamação

Alteração de Reclamação:

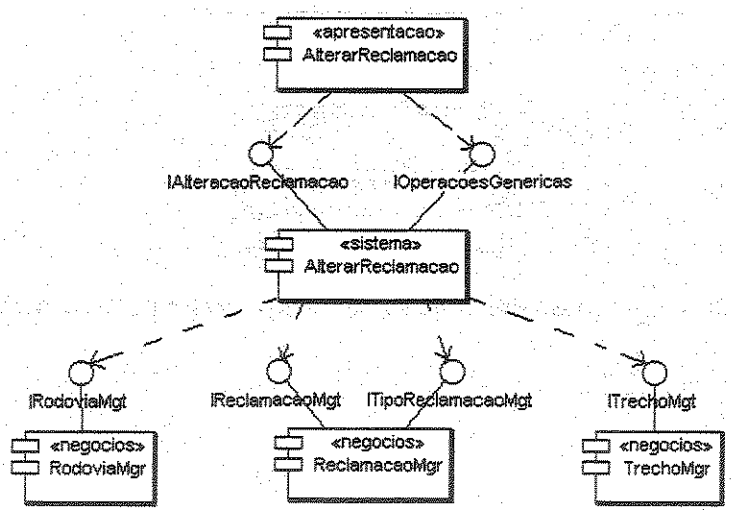

Figura 73 – Arquitetura Inicial de Alteração de Reclamação

Definição de Processo de Reclamação:

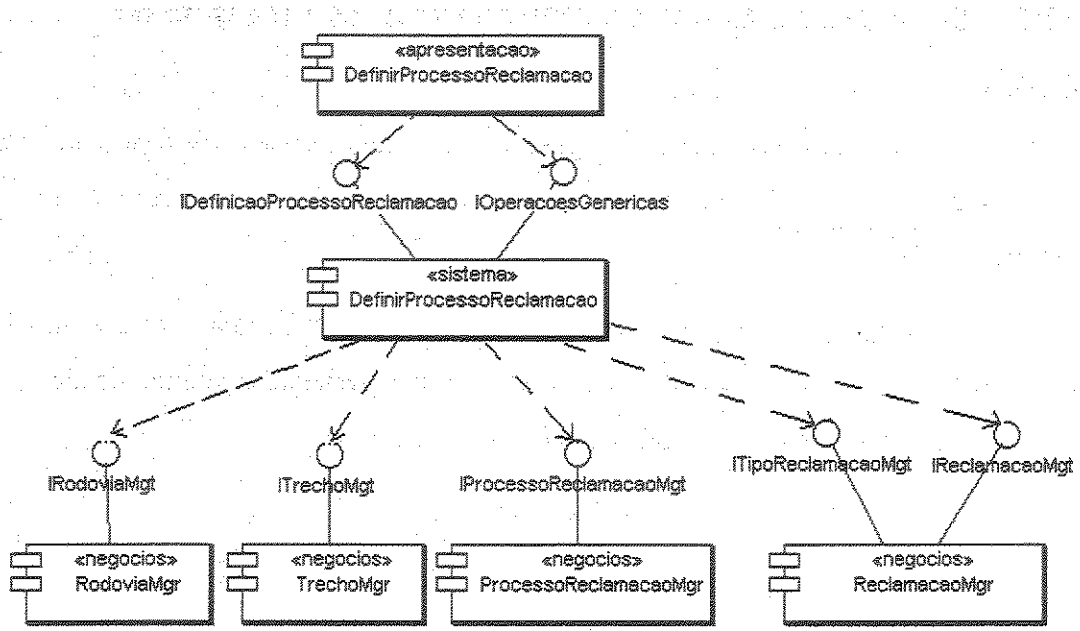

Figura 74 – Arquitetura Inicial de Definição de Processo de Reclamação

Remoção de Processo de Reclamação:

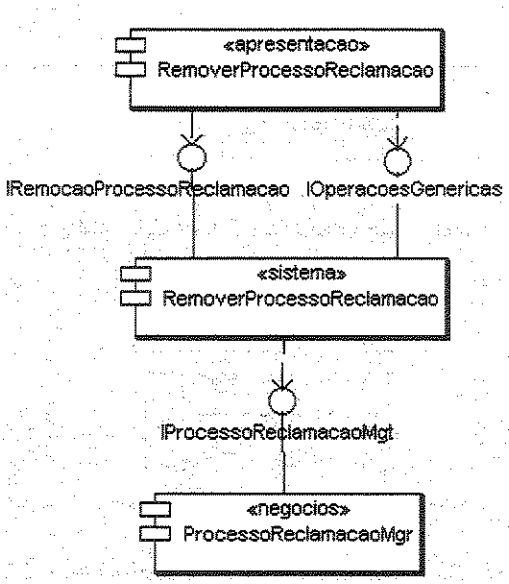

Figura 75 – Arquitetura Inicial de Remoção de Processo de Reclamação

Alteração de Processo de Reclamação:

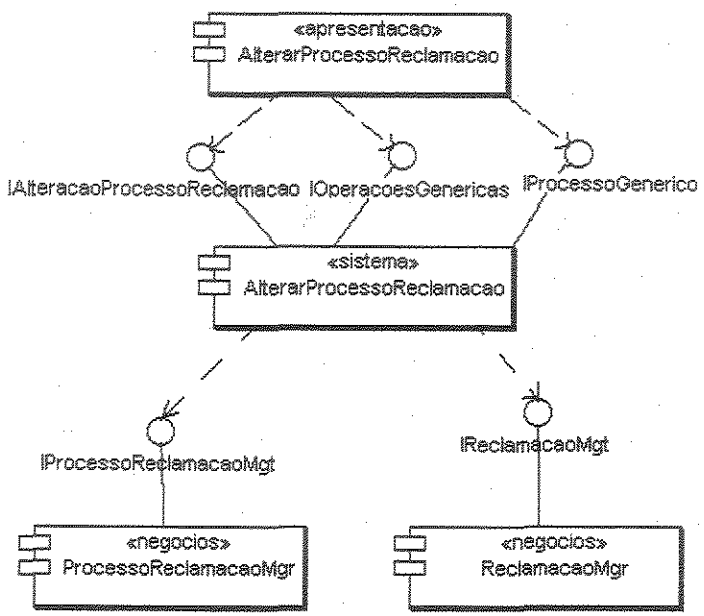

Figura 76 – Arquitetura Inicial de Alteração de Processo de Reclamação

Encaminhamento de Processo de Reclamação:

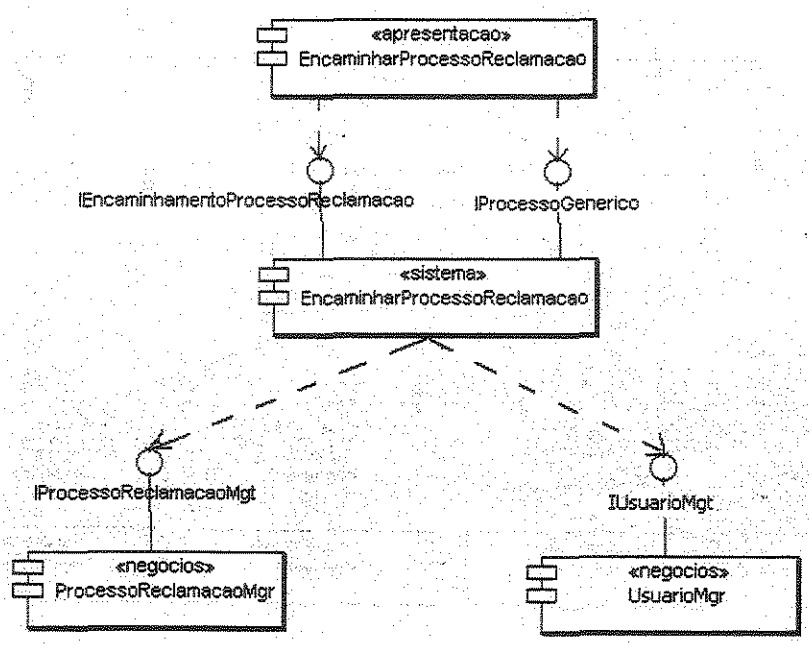

Figura 77 – Arquitetura Inicial de Encaminhamento de Processo de Reclamação

Atualização de Processo de Reclamação:

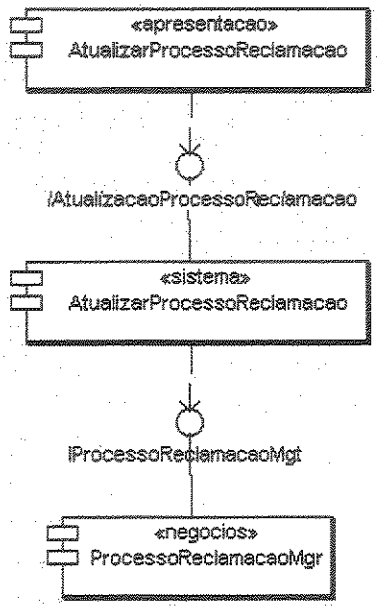

Figura 78 – Arquitetura Inicial de Atualização de Processo de Reclamação

Definição de Resposta Padrão para Trecho e Tipo:

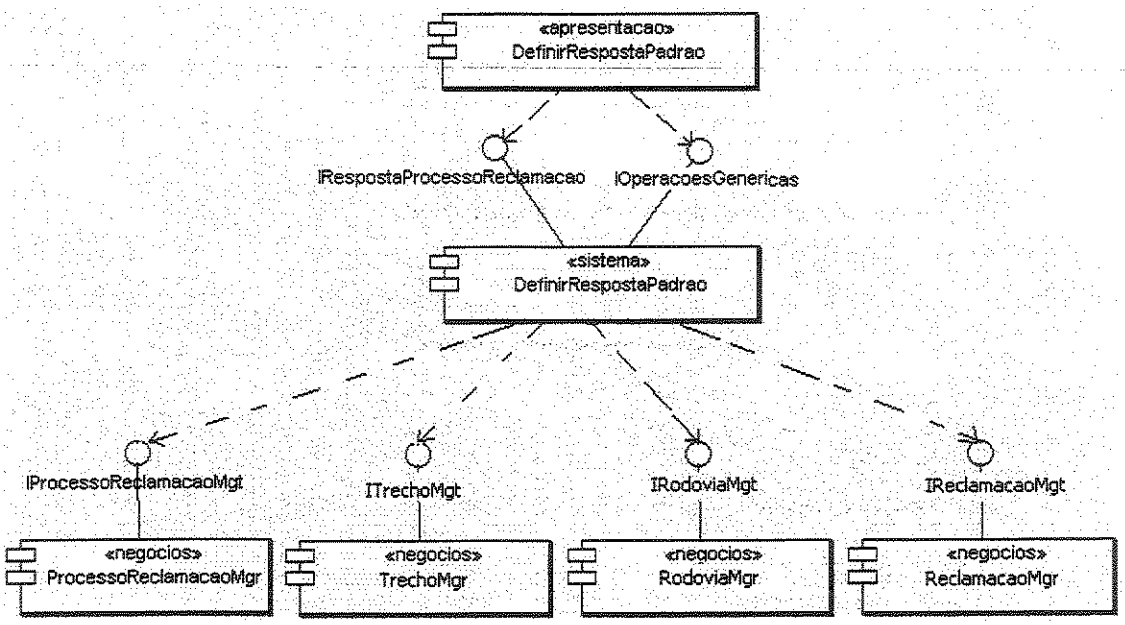

Figura 79 – Arquitetura Inicial de Definição de Resposta Padrão

Apêndice C – Telas do Telestrada

Neste apêndice apresenta-se as telas implementadas dos casos de uso de registro e consulta de reclamação do sub sistema de gerência de reclamações do Telestrada.

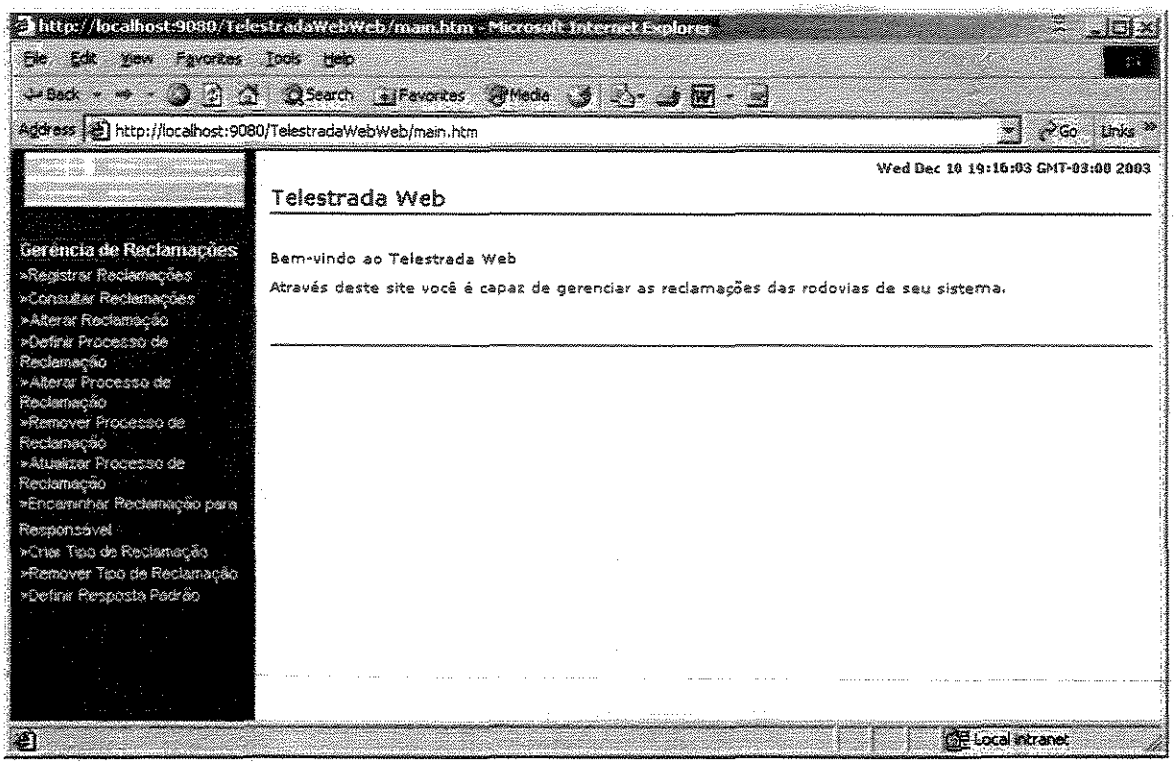

Figura 80 – Página Principal do Telestrada

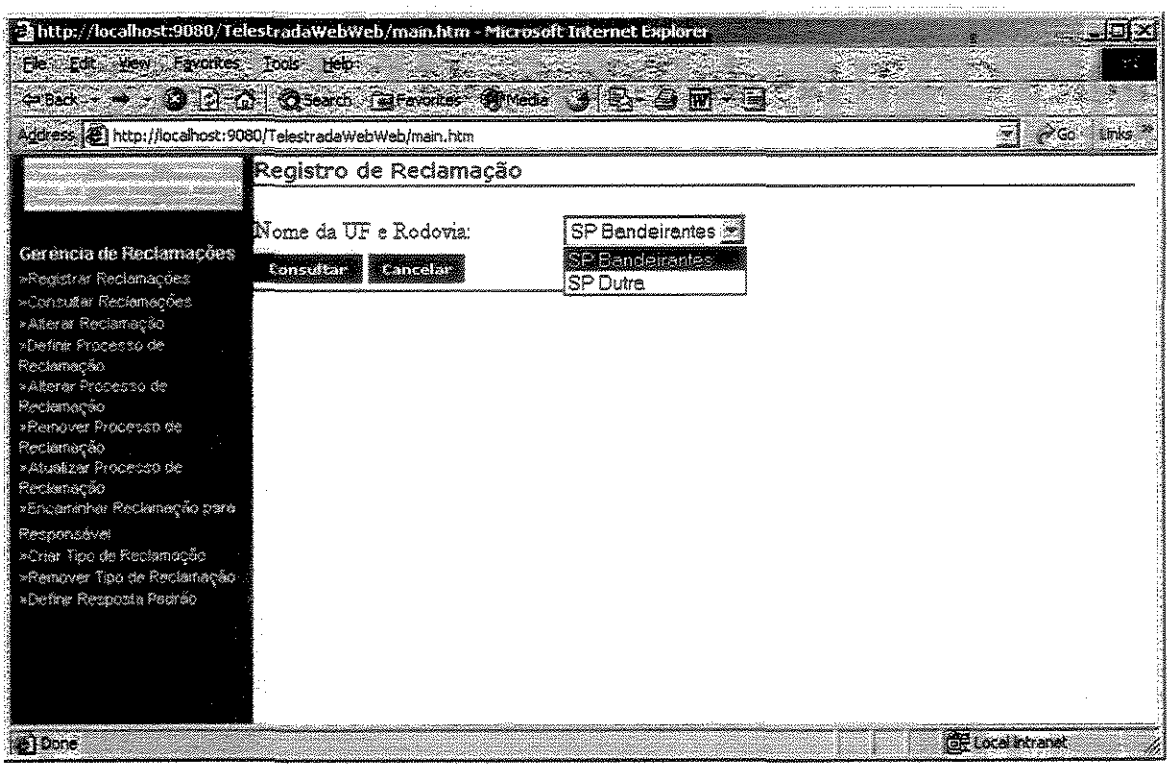

Figura 81 – Página para Seleção de Rodovias

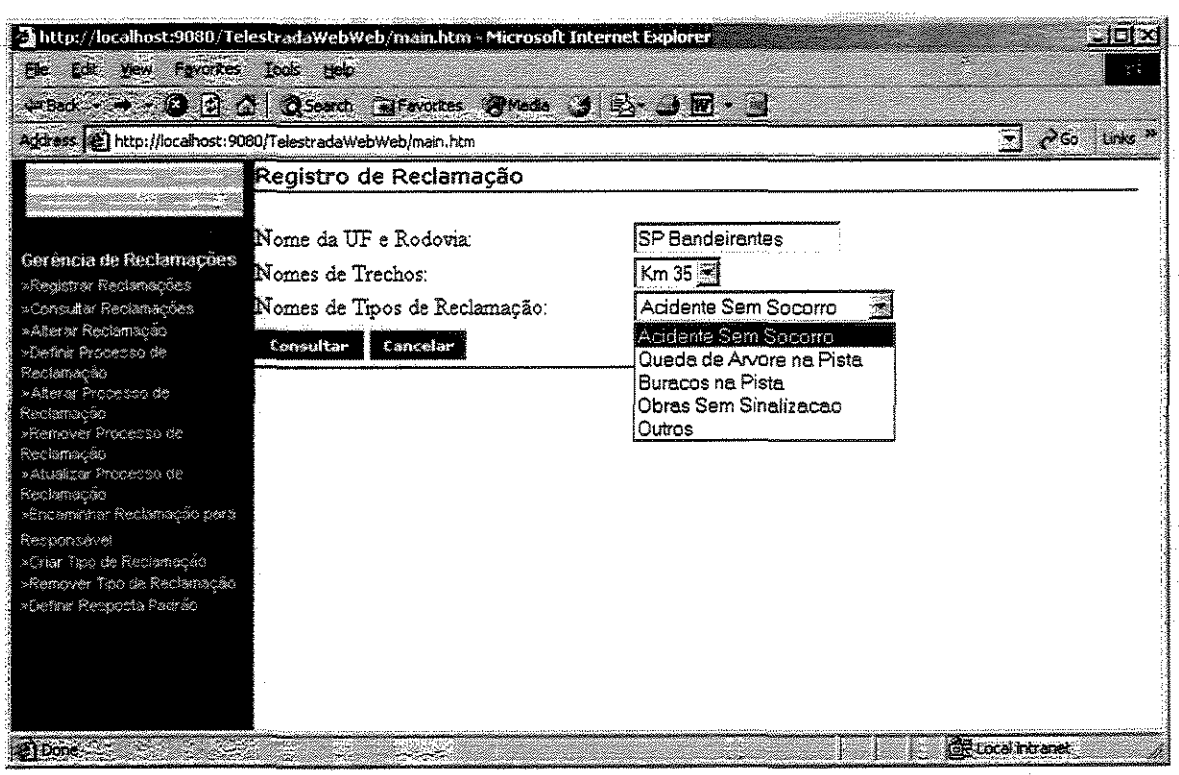

Figura 82 – Página para Seleção de Trechos e Tipos de Reclamações

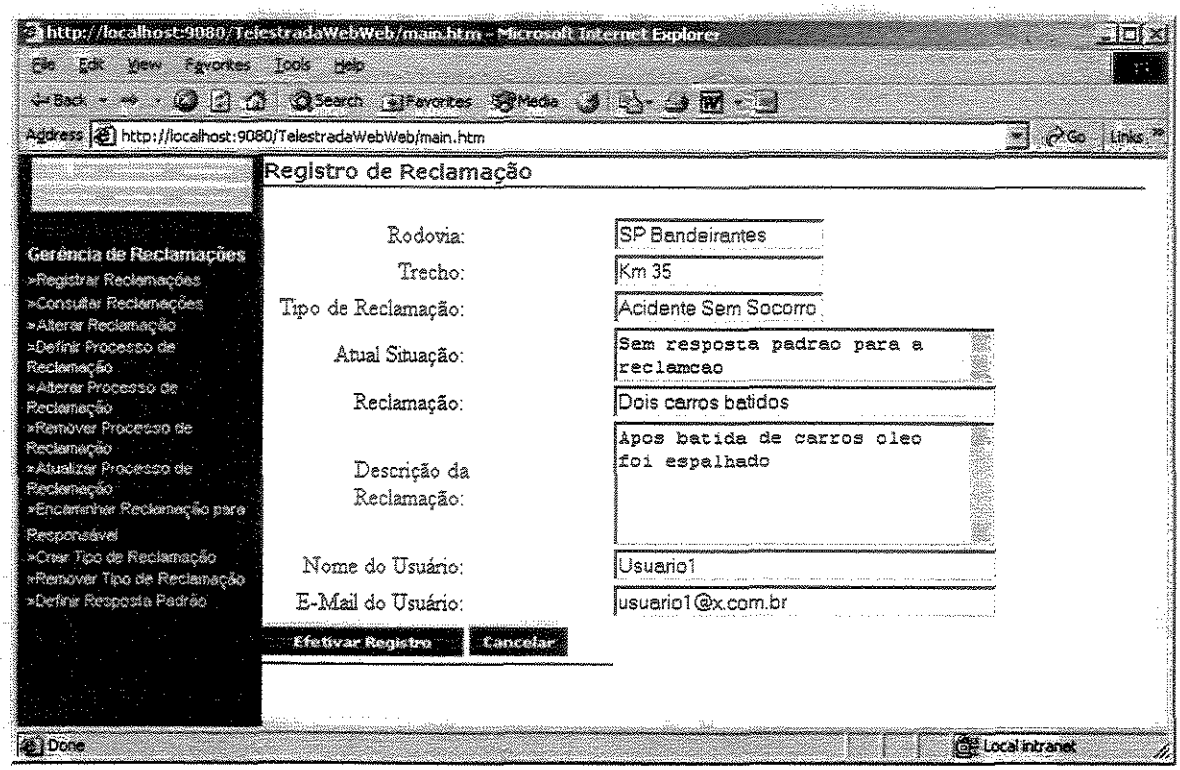

Figura 83 – Página para Entrada de Dados de Reclamação e Usuário

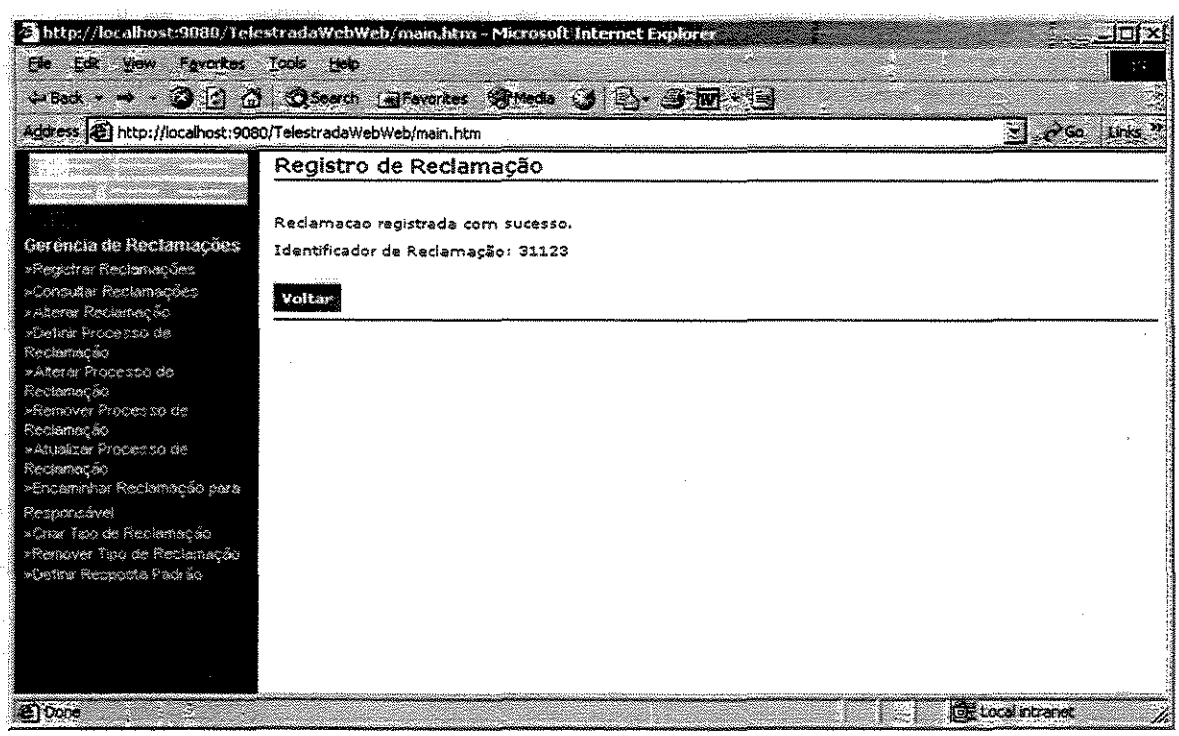

Figura 84 – Página Apresentando Identificador da Reclamação Registrada

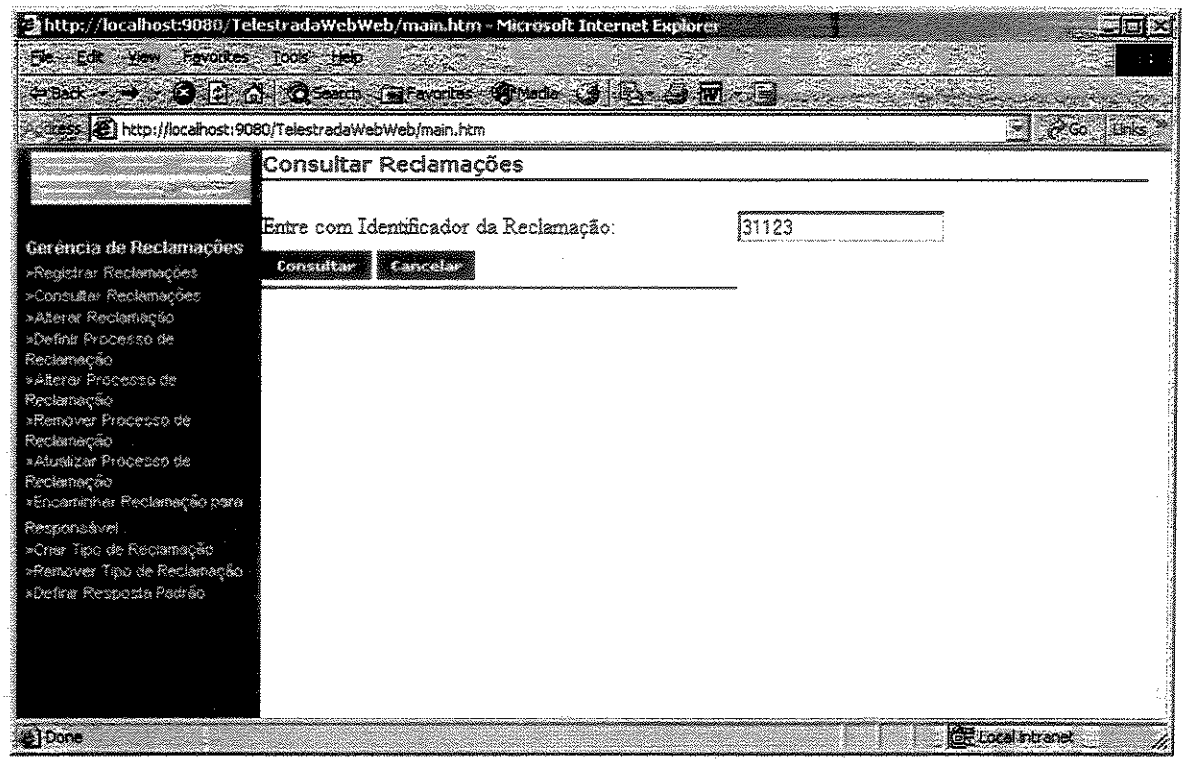

Figura 85 – Página Inicial de Consulta de Reclamações

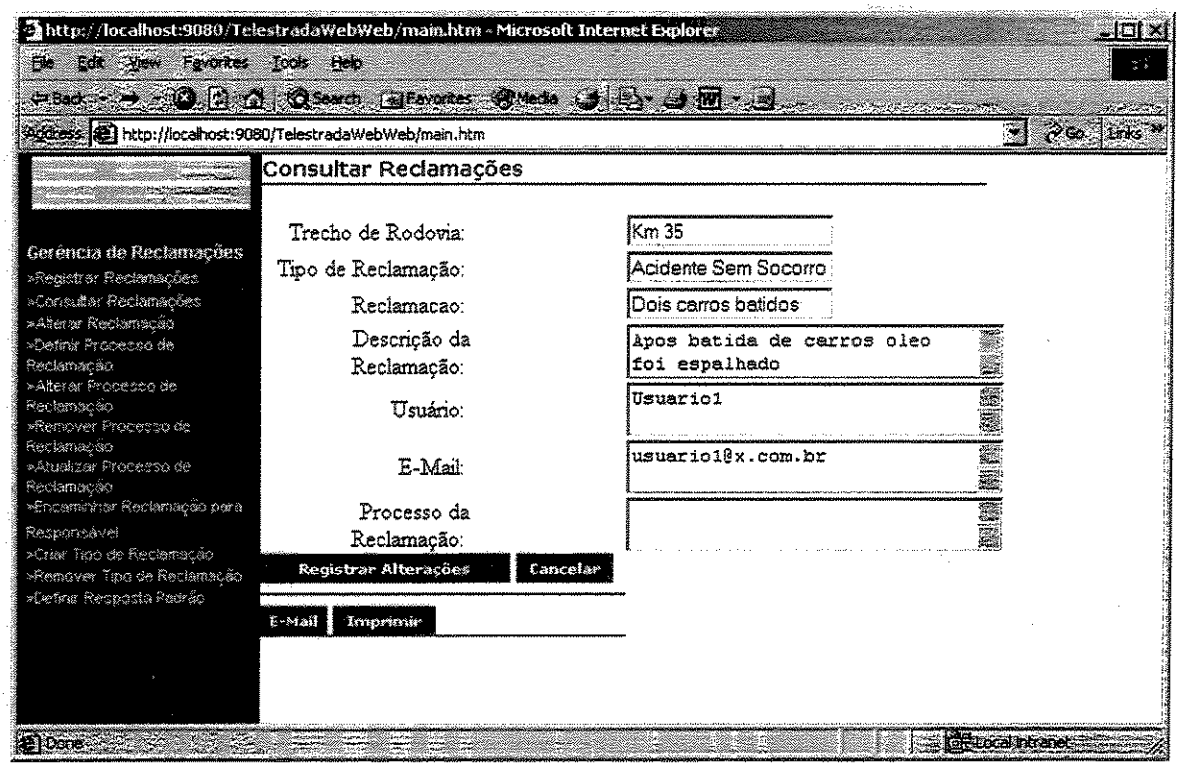

Figura 86 – Página para Consulta de Reclamação com Opção para Alteração de Dados

Apêndice D – *Scripts* de Criação de Tabelas de Banco de Dados

Neste apêndice é apresentado o *script* para criação de tabelas de banco de dados para o sub sistema de gerência de reclamações.

```
CREATE TABLE PAIS (
 OID NUMBER(10) NOT NULL,
 NOME VARCHAR2(30) NOT NULL,
 CONSTRAINT OID_PAIS PRIMARY KEY (OID));
CREATE TABLE UF (
 OID NUMBER(10) NOT NULL,
 NOME VARCHAR2(30) NOT NULL,
 OID_PAIS NUMBER(10) NOT NULL,
 CONSTRAINT OID_UF PRIMARY KEY (OID),
 CONSTRAINT OID_PAIS_FK FOREIGN KEY (OID_PAIS) REFERENCES PAIS (OID));
CREATE TABLE RODOVIA (
 OID NUMBER(10) NOT NULL,
 NOME VARCHAR2(30) NOT NULL,
 CONSTRAINT OID_RODOVIA PRIMARY KEY (OID));
CREATE TABLE UF_RODOVIA (
 OID NUMBER(10) NOT NULL,
 OID_UF NUMBER(10) NOT NULL,
 OID_RODOVIA NUMBER(10) NOT NULL,
 CONSTRAINT OID_UF_RODOVIA PRIMARY KEY (OID),
 CONSTRAINT OID_UF_FK FOREIGN KEY (OID_UF) REFERENCES UF (OID),
 CONSTRAINT OID_RODOVIA_FK FOREIGN KEY (OID_RODOVIA) REFERENCES RODOVIA (OID));
CREATE TABLE RESPONSABILIDADE(
 OID NUMBER(10) NOT NULL,
 NOME VARCHAR2(30) NOT NULL,
 CONSTRAINT OID_RESPONSABILIDADE PRIMARY KEY (OID));
CREATE TABLE USUÁRIO(
 OID NUMBER(10) NOT NULL,
 OID_RESPONSABILIDADE NUMBER(10) NOT NULL,
 NOME VARCHAR2(30) NOT NULL,
 EMAIL VARCHAR2(30),
 TELEFONE VARCHAR2(30),
 CONSTRAINT OID_USUÁRIO PRIMARY KEY (OID),
 CONSTRAINT OID_RESPONSABILIDADE_FK FOREIGN KEY (OID_RESPONSABILIDADE) REFERENCES RESPONSABILIDADE (OID));
CREATE TABLE TRECHO (
 OID NUMBER(10) NOT NULL,
 OID_UF_RODOVIA NUMBER(10) NOT NULL,
 OID_RESPONSAVEL_POR_TRECHO NUMBER(10),
 NOME VARCHAR2(30) NOT NULL,
 KM VARCHAR2(30),
 CONSTRAINT OID_TRECHO PRIMARY KEY (OID),
 CONSTRAINT OID_UF_RODOVIA_FK FOREIGN KEY (OID_UF_RODOVIA) REFERENCES UF_RODOVIA (OID),
 CONSTRAINT OID_RESPONSAVEL_POR_TRECHO_FK FOREIGN KEY (OID_RESPONSAVEL_POR_TRECHO) REFERENCES USUÁRIO (OID));
CREATE TABLE TIPO_RECLAMACAO(
 OID NUMBER(10) NOT NULL,
```

```
  NOME VARCHAR2(30) NOT NULL,
  DESCRICAO VARCHAR2(50) NOT NULL,
  CONSTRAINT OID_TIPO_RECLAMACAO PRIMARY KEY (OID));
CREATE TABLE METODO_ENCAMINHAMENTO(
  OID NUMBER(10) NOT NULL,
  NOME VARCHAR(30) NOT NULL,
  CONSTRAINT OID_METODO_ENCAMINHAMENTO PRIMARY KEY (OID));
CREATE TABLE PROCESSO_RECLAMACAO(
  OID NUMBER(10) NOT NULL,
  NOME VARCHAR2(30) NOT NULL,
  OID_METODO_ENCAMINHAMENTO NUMBER(10),
  OID_TIPO_RECLAMACAO NUMBER(10) NOT NULL,
  OID_TRECHO NUMBER(10) NOT NULL,
  CONSTRAINT OID_PROCESSO_RECLAMACAO PRIMARY KEY (OID),
  CONSTRAINT OID_METODO_ENCAMINHAMENTO_FK FOREIGN KEY (OID_METODO_ENCAMINHAMENTO) REFERENCES
METODO_ENCAMINHAMENTO (OID),
  CONSTRAINT OID_TIPO_RECLAMACAO_FK FOREIGN KEY (OID_TIPO_RECLAMACAO) REFERENCES TIPO_RECLAMACAO (OID),
  CONSTRAINT OID_TRECHO_FK FOREIGN KEY (OID_TRECHO) REFERENCES TRECHO (OID));
CREATE TABLE HISTORICO_PROCESSO(
  OID NUMBER(10) NOT NULL,
  NOME VARCHAR2(30) NOT NULL,
  OID_PROCESSO_RECLAMACAO NUMBER(10) NOT NULL,
  DATA DATE NOT NULL,
  CONSTRAINT OID_HISTORICO_PROCESSO PRIMARY KEY (OID),
  CONSTRAINT OID_PROCESSO_RECLAMACAO_FK FOREIGN KEY (OID_PROCESSO_RECLAMACAO) REFERENCES
PROCESSO_RECLAMACAO (OID));
CREATE TABLE RECLAMACAO(
  OID NUMBER(10) NOT NULL,
  OID_TIPO_RECLAMACAO NUMBER(10) NOT NULL,
  OID_USUÁRIO NUMBER(10) NOT NULL,
  OID_TRECHO NUMBER(10) NOT NULL,
  OID_PROCESSO_RECLAMACAO NUMBER(10),
  NOME VARCHAR2(30) NOT NULL,
  DESCRICAO VARCHAR2(50) NOT NULL,
  CONSTRAINT OID_RECLAMACAO PRIMARY KEY (OID),
  CONSTRAINT OID_R_USUÁRIO_FK FOREIGN KEY (OID_USUÁRIO) REFERENCES USUÁRIO (OID),
  CONSTRAINT OID_R_TIPO_RECLAMACAO_FK FOREIGN KEY (OID_TIPO_RECLAMACAO) REFERENCES TIPO_RECLAMACAO (OID),
  CONSTRAINT OID_R_TRECHO_FK FOREIGN KEY (OID_TRECHO) REFERENCES TRECHO (OID),
  CONSTRAINT OID_R_PROCESSO_RECLAMACAO_FK FOREIGN KEY (OID_PROCESSO_RECLAMACAO) REFERENCES
PROCESSO_RECLAMACAO (OID));
CREATE TABLE RESPOSTA_PADRAO(
  OID NUMBER(10) NOT NULL,
  NOME VARCHAR2(30) NOT NULL,
  OID_TIPO_RECLAMACAO NUMBER(10) NOT NULL,
  OID_TRECHO NUMBER(10) NOT NULL,
  CONSTRAINT OID PRIMARY KEY (OID),
  CONSTRAINT OID_RP_TIPO_RECLAMACAO_FK FOREIGN KEY (OID_TIPO_RECLAMACAO) REFERENCES TIPO_RECLAMACAO (OID),
  CONSTRAINT OID_RP_TRECHO_FK FOREIGN KEY (OID_TRECHO) REFERENCES TRECHO (OID));
```