FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO "EURÍPIDES DE MARÍLIA" – UNIVEM
PROGRAMA DE MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

LISANDRO ROGÉRIO MODESTO

TESTE FUNCIONAL BASEADO EM DIAGRAMAS DA UML

MARÍLIA
2006

LISANDRO ROGÉRIO MODESTO

TESTE FUNCIONAL BASEADO EM DIAGRAMAS DA UML

Dissertação apresentada ao Programa de Mestrado do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do Título de Mestre em Ciência da Computação. (Área de Concentração: Engenharia de Software).
Orientador: Dr. Edmundo Sérgio Spoto.

MARÍLIA
2006

Lisandro Rogério Modesto

Teste Funcional baseado em Diagrama da UML

Banca examinadora da dissertação apresentada ao Programa de Mestrado da UNIVEM / FEESR, para obtenção do Título de Mestre em Ciência da Computação.

A Comissão Julgadora:

Prof. Dr. Edmundo Sérgio Spoto

Profa. Dra Maria Istela Cagnin Machado

Profa. Dra Rogéria Cristiane Gratão de Souza

Marília, 20 de abril de 2006.

Dedico esse trabalho aos meus pais, às minhas irmãs, à minha afilhada, ao meu cunhado, à minha noiva e aos meus amigos, sem os quais essa caminhada jamais teria sido concluída.

AGRADECIMENTOS

Agradeço primeiramente a Deus que, com sua sabedoria e bondade divina, iluminou-me em todos os meus aprendizados.

Ao meu orientador, Prof. Dr. Edmundo Sérgio Spoto, que com sua experiência, sempre me orientou em qual caminho deveria prosseguir, fazendo isso com segurança e transmitindo a mim a confiança necessária durante toda a elaboração desta dissertação.

À minha família, em especial aos meus pais, Vicente e Aparecida, às minhas irmãs, Carla e Elizângela, à minha afilhada Rafaela e ao meu cunhado Flávio, que sempre me apoiaram e me encorajaram, principalmente nas horas em que mais precisei.

À minha noiva Juliana, pelo carinho, amor, compreensão e dedicação que sempre me concedeu no decorrer deste curso.

À FAP – Faculdade de Apucarana, que me apoiou financeira e profissionalmente para que eu pudesse vencer mais essa etapa de minha vida profissional.

Aos amigos André Casavechia, Andréa Santiago, Giofranceno da Silva Pujólli e Rodrigo Tazima, que gentilmente cederam o material usado como Estudo de Caso nesta dissertação.

Ao amigo Lucas Franco, pela colaboração no desenvolvimento do Sistema SOFTHOTEL e da Ferramenta FuTeBOOL.

A todas as pessoas que de uma forma ou de outra contribuíram para que eu conseguisse concluir mais esta etapa de minha vida.

MODESTO, Lisandro Rogério. **Teste Funcional Baseado em Diagramas da UML.** 2006. 116 f. Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares Rocha, Marília, 2006.

RESUMO

Com a explosão de projetos de software usando o paradigma Orientado a Objetos, entende-se que a necessidade de melhorar as etapas de teste para um projeto é fundamental e cada vez mais necessária para a obtenção da qualidade. Neste trabalho, é feita uma proposta de novas técnicas de Teste Funcional, visando derivar elementos requeridos de testes a partir da fase de projeto usando a UML. A partir da especificação, baseando-se em Diagramas de Casos de Uso, Seqüência, Colaboração e Classes, são gerados um conjunto de elementos requeridos funcionais intra-classe (métodos que agem numa mesma classe) e inter-classe (métodos que interagem em duas ou mais classes). São descritos os principais requisitos necessários em cada Caso de Uso relacionados com os Diagramas de: Casos de Uso e Seqüência ou Colaboração; os quais foram confrontados com as principais especificações para que uma tarefa A ou B seja concluída com sucesso e validada pelo sistema. São apresentados: um estudo de caso para ilustrar as propostas deste projeto, bem como a construção de uma ferramenta que a partir dos Diagramas da UML sejam derivados os elementos requeridos de teste funcional. Essa ferramenta cria elementos requeridos válidos e inválidos e apresenta a possibilidade de verificação de quais elementos foram exercitados (cobertura obtida).

Palavras-Chave: Teste Funcional. UML. Elementos Requeridos.

MODESTO, Lisandro Rogério. **Teste Funcional Baseado em Diagramas da UML.** 2006. 116 f. Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares Rocha, Marília, 2006.

ABSTRACT

With the great number of software projects using the Object Oriented paradigm, it is understood that the necessity of improving the test stages for a project is fundamental and more and more necessary for quality achievement. This work proposes new techniques of Functional Test, aiming to derive required test elements from a project stage using the UML. From specification, based on Use Cases, Sequence, Collaboration and Classes Diagrams it is generated a required functional elements set intra class (methods that act in a same class) and inter class (methods that interact in two or more classes). It is described the main necessary requirements in each Use Case related with the diagrams of: Use Cases and Sequence or Collaboration; which were compared with the main specifications so that an A or B task can be successfully concluded and validated by the system. It is presented a case study to illustrate this project proposals, as well as the construction of a tool which considering the UML diagrams the required elements of functional test are derived. This tool creates valid and invalid required elements and presents the possibility of checking which elements were practiced (obtained covering).

Keywords: Functional Test. UML. Required Elements.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

SUMÁRIO

INTRODUÇÃO

Durante o processo de desenvolvimento de software existem vários métodos, técnicas e ferramentas que visam garantir qualidade ao produto final. Mesmo assim, os problemas no produto final podem persistir se não houver um apoio das técnicas de obtenção da qualidade.

O teste de software é uma das técnicas utilizadas para obtenção da qualidade do software, tornando-se indispensável para a identificação e eliminação de problemas que possam vir a ocorrer durante este processo (PRESSMAN, 2002).

O teste pode ser considerado como uma das últimas fases de revisão do processo de desenvolvimento do software. Teste é o processo pelo qual se executa um determinado produto de software com a finalidade de detectar erros no software (MYERS, 1979).

Myers (1979) define caso de teste como sendo o conjunto formado pela entrada de dados do software, as saídas esperadas e as saídas obtidas durante a execução do software, e descreve um bom caso de teste sendo aquele que tem alta probabilidade de revelar um defeito ainda não descoberto e se, de fato isto ocorrer, este teste é considerado bem sucedido.

De acordo com Pressman (2002), duas abordagens de teste podem ser identificadas: teste baseado nas especificações (caixa-preta), cujo objetivo é demonstrar que o software possui deficiências relativas às especificações externas; e teste baseado nos programas (caixa-branca), que tem por finalidade identificar a ocorrência de código problemático.

Nos testes de programas orientados a objetos, a estrutura dos programas provoca alterações tanto na estratégia quanto na tática de teste. Baseado nisso, Binder (1999) afirma: “Cada reuso é um novo contexto de utilização e a retestagem é prudente. Parece provável que mais testes, ao invés de menos, vão ser necessários para alcançar alta confiabilidade em sistemas orientados a objetos”.

Quando se leva em consideração sistemas orientados a objetos, existem cinco diferenças em relação aos sistemas clássicos, que apontam como as abordagens tradicionais deveriam ser alteradas para atender às situações orientadas a objetos: níveis adicionais de abstração, estados dos objetos, heranças, polimorfismo e encapsulamento.

Em virtude da necessidade de atribuir qualidade ao software, faz-se necessário executar tarefas de teste desde as primeiras fases do desenvolvimento do produto.

O Teste Estrutural também é conhecido como Teste Caixa Branca e algumas vezes chamado Teste Caixa de Vidro, é um método de projeto de casos de teste que usa a estrutura de controle do projeto procedimental para derivar casos de teste. Usando métodos de teste Caixa Branca, o engenheiro de software pode derivar casos de teste que garantam que todos os caminhos independentes de um módulo tenham sido exercitados pelo menos uma vez, exercitam todas as decisões lógicas em seus lados verdadeiro e falso, executam todos os ciclos nos seus limites e dentro de seus intervalos operacionais e exercitam as estruturas de dados internas para garantir sua validade (PRESSMAN, 2002).

O Teste Funcional também é conhecido como Teste Caixa Preta, devido ao fato de o software ser tratado como uma caixa, na qual é visível e disponível apenas a interface, ou seja, o lado externo (BEIZER, 1990).

O Teste Funcional focaliza os requisitos funcionais do software, isto é, permite ao engenheiro de software derivar das especificações um conjunto de condições de entrada que vão exercitar plenamente todos os requisitos funcionais de um programa (PRESSMAN, 2002). Sendo assim, Teste Funcional baseado em Diagramas da UML é uma alternativa complementar ao Teste Funcional tradicional para atribuir qualidade ao produto de software desenvolvido.

Este teste é feito sem preocupações com formulações matemáticas ou com a linguagem de programação na qual o software foi desenvolvido, a preocupação está centrada

em fazer o teste tendo como base a própria descrição (cenário) passada pelo cliente ou levantada pelo desenvolvedor.

O Teste Funcional tradicional é embasado nas verificações feitas através das variáveis de domínio, enquanto o Teste Funcional Baseado em Diagramas da UML além de executar o teste sobre as variáveis de domínio, também se baseia em Dependências Funcionais, podendo-se testar uma ou mais classes, possibilitando avaliar interações numa mesma classe (intra-classe) ou entre classes distintas (inter-classes), bem como testar heranças e dependências.

Harrold & Rothermel (1994) explicam teste intra e inter-classes da seguinte forma:

- No teste intra-classe, são testadas interações entre métodos públicos fazendo chamada a esses métodos em diferentes seqüências. O objetivo é identificar possíveis seqüências de ativação de métodos inválidas que levem o objeto a um estado inconsistente;
- No teste inter-classe, o mesmo conceito de invocação de métodos públicos em diferentes seqüências é utilizado, entretanto, esses métodos públicos não necessitam estar na mesma classe.

Porém, neste trabalho, não estão sendo separados teste intra-classe de teste inter-classes, uma vez que o foco é testar as ações do programa sem preocupações com código do mesmo mas sob o efeito funcional.

Pressman (2002) afirma que o Teste Funcional não é uma alternativa às técnicas de Teste Estrutural, pelo contrário, é uma abordagem complementar, que mais provavelmente descobrirá uma classe diferente de erros das técnicas de teste da caixa-branca.

Trabalhos desenvolvidos anteriormente também tinham a intenção de desenvolver o teste funcional baseado em alguns diagramas da UML (*Unified Modeling Language*). Como por exemplo, o trabalho que efetua o teste funcional baseado em Casos de Usos, desenvolvido

por Chaim *et al* (2003), porém tal trabalho tinha o foco no teste de variáveis de domínio. Outro exemplo é o trabalho desenvolvido por Souza (2003), que determina procedimentos que definem subsídios para a realização de testes e que contribuem para a redução dos esforços necessários para a realização dos testes de programas orientados a objetos, para isso foram abordadas características de testabilidade nos diagramas de análise da UML.

A seguir são apresentados a motivação e objetivos deste trabalho, visando contribuir com uma proposta de ampliar o enfoque do teste funcional em Sistemas Orientado a Objetos sob a visão das dependências funcionais extraídas dos diagramas da UML.

Esta proposta de teste é complementar ao Teste Funcional tradicional e possibilita a automatização na geração dos Elementos Requeridos e na geração de estatísticas que podem servir como base para análises futuras.

Motivação

Com a crescente busca pela qualidade no produto final, faz-se necessário o teste para atribuir qualidade ao produto de software desenvolvido, uma vez que, se desde o início o software já estiver sendo testado e avaliado, o número de problemas, geralmente encontrados depois do software estar desenvolvido, poderá cair significativamente (PRESSMAN, 2002).

Vários motivos levaram a este trabalho entre eles:

a) necessidade de atestar qualidade ao produto de software comercialmente desenvolvido;
b) crescente busca pela perfeição no desenvolvimento de softwares comercialmente desenvolvidos;
c) proposta de uma nova técnica para teste funcional de software, sem a

preocupação com a linguagem na qual o software foi desenvolvido;

d) ampliação da visão do teste funcional, visto que o teste funcional tradicional baseia-se em variáveis de domínio. Este trabalho aponta que além de basear o teste funcional em variáveis de domínio é possível basear também em dependências funcionais, podendo testar uma ou mais classes, possibilitando envolver o teste em níveis *intra-classe e inter-classes*;
e) possibilidade de automação da técnica de teste funcional;
f) possibilidade de auxiliar na avaliação de maneira automática a partir de estatísticas de resultados do teste funcional;
g) propor uma derivação dos elementos requeridos de teste na técnica de teste funcional, a partir dos diagramas da UML.

Objetivos

Este trabalho tem como objetivo fazer um estudo sobre teste de software e encontrar fundamentos para propor Teste Funcional baseado em Diagramas da UML.

Este estudo engloba Teste de Software, Teste de Software Orientado a Objetos, Critérios de Teste Funcional e Orientados a Objetos, Teste de Programas Orientados a Objetos e um estudo de caso que exemplifica a proposta apresentada.

Os principais objetivos deste trabalho são:

a) proposta de técnica de Teste Funcional baseado em Diagramas da UML;
b) desenvolvimento de uma ferramenta para auxiliar o Teste Funcional baseado em Diagramas da UML;
c) geração de elementos requeridos para Teste Funcional baseado em Diagramas da

UML;

d) análise de resultados da fase de Teste Funcional.

Organização do Trabalho

Nesta seção, são apresentados a introdução, a motivação e os objetivos.

No Capítulo 1, encontra-se o Levantamento Bibliográfico e uma contextualização sobre Critérios de Teste, tanto funcionais quanto baseados em Diagramas da UML.

No Capítulo 2 destaca-se a Ferramenta FuTeBOOL – Tool (*Functional Testing Building on Object Oriented Language – Tool*), Ferramenta para Geração de Teste Funcional Baseada na Linguagem Orientada a Objetos, bem como suas funcionalidades e particularidades.

No Capítulo 3 apresenta-se um estudo de caso denominado *"Sistema de Gerenciamento de Hotéis - SOFTHOTEL"*, que serve como base para apresentação dos Diagramas da UML que serão utilizados como fundamentação para execução da FuTeBOLL – Tool.

No Capítulo 4 são apresentados os resultados obtidos através do Estudo da Técnica de Teste Funcional Baseado em Diagramas da UML e do uso da Ferramenta FuTeBOOL – Tool.

E por fim, apresenta-se a conclusão do trabalho, dando uma visão geral do projeto desenvolvido para a obtenção do título de mestre em Ciência da Computação, bem como as possibilidades para trabalhos futuros.

1 TESTE DE SOFTWARE

Teste de software é um elemento crítico da garantia de qualidade de software e representa a revisão final da especificação, projeto e geração de código. A crescente visibilidade do software como um elemento do sistema e os "custos" de atendimento associados com uma falha são forças motivadoras para o teste rigoroso e bem planejado.

Não é raro uma organização de desenvolvimento de software gastar entre 30% e 40% do esforço total do projeto no teste. A rigor, o teste de software que envolve vidas (por exemplo, controle de vôo, monitoramento de reatores nucleares) pode custar de três a cinco vezes mais do que todos os outros passos de engenharia combinados. O processo de teste focaliza os aspectos lógicos internos do software, garantindo que todos os comandos sejam testados (teste estrutural), e os aspectos externos funcionais; isto é, conduz testes para descobrir erros e garantir que entradas definidas produzirão resultados reais, que concordam com os resultados exigidos (PRESSMAN, 2002).

O teste expõe uma anomalia interessante para o engenheiro de software. Durante as primeiras atividades de engenharia de software, o engenheiro tenta construir um software a partir de um conceito abstrato até um produto tangível, depois vem o teste.

O engenheiro cria uma série de casos de testes, que são destinados a "demolir" o software que foi construído. De fato, o teste é um passo do processo de software que poderia ser visto como destrutivo em vez de construtivo.

Engenheiros de software são por natureza pessoas construtivas, o teste exige que o desenvolvimento reveja noções preconcebidas da "corretividade" do software recém desenvolvido e supere um conflito de interesses que ocorre quando erros são descobertos (PRESSMAN, 2002).

Embora durante o processo de desenvolvimento de software possam ser utilizadas

técnicas, critérios e ferramentas a fim de evitar que erros sejam introduzidos no produto de software, a atividade de teste continua sendo de fundamental importância para eliminação de erros que persistem (MALDONADO, 1991). Por isso, o teste de software é um elemento crítico para a garantia da qualidade do produto e representa a última revisão de especificação, projeto e codificação (PRESSMAN, 2000).

Segundo Myers (1979), o objetivo da fase de teste é o processo de executar um programa com a intenção de descobrir um erro através de um bom caso de teste. Um bom caso de teste pode ser definido como aquele que tem alta probabilidade de revelar defeitos no software, e um caso de teste bem sucedido é aquele capaz de revelar defeitos ainda não descobertos.

Esses objetivos implicam uma dramática mudança de ponto de vista. Eles vão contra a idéia de que um teste bem sucedido é aquele no qual não são encontrados defeitos. O objetivo é projetar testes que descobrem sistematicamente diferentes classes de erros e que possam fazê-lo com uma quantidade mínima de tempo e esforço. Se o teste for conduzido de maneira bem sucedida, ele descobrirá erros no software.

Como benefício secundário, os testes demonstram que as funções do software parecem estar funcionando de acordo com a especificação de que os requisitos de comportamento e desempenho parecem estar sendo satisfeitos.

Apesar de não ser possível, por meio de testes, provar que um programa está correto, os testes, se conduzidos sistemática e criteriosamente, contribuem para aumentar a confiança de que o software desempenha as funções especificadas e evidenciar algumas características mínimas do ponto de vista da qualidade do produto (VINCENZI, 1998).

Idealmente um programa deveria ser exercitado com todos os valores possíveis do domínio de entrada. Sabe-se, entretanto, que o teste exaustivo é impraticável devido a restrições de tempo e custo para realizá-lo.

Dessa forma, é necessário determinar quais casos de teste utilizar, de modo que a maioria dos erros existentes possa ser encontrada e que o número de casos de teste não seja tão grande a ponto de ser impraticável (MALDONADO, 1997; MALDONADO *et al*, 1998).

Além disso, os dados coletados à medida que o teste é conduzido fornecem uma boa indicação da confiabilidade do software e alguma indicação da qualidade de todo software.

O teste não pode mostrar a ausência de erros e defeitos, mas apenas mostrar que erros e defeitos de software estão presentes.

Nesse sentido, técnicas e critérios de teste têm sido elaborados com o objetivo de fornecer uma maneira sistemática e rigorosa para selecionar um subconjunto do domínio de entrada e ainda assim ser eficiente para apresentar os erros existentes, respeitando-se as restrições de tempo e custo associado a um projeto de software (VINCENZI, 1998).

As principais técnicas de teste para o desenvolvimento deste trabalho são descritas na Seção 1.1 e os principais critérios de teste para o desenvolvimento deste trabalho são descritos na Seção 1.2.

1.1 Técnicas de Teste

As técnicas e critérios de teste fornecem ao desenvolvedor uma abordagem sistemática e teoricamente fundamentada para se conduzir e avaliar a qualidade do teste de software. Dentre as várias técnicas existentes, pode-se destacar as técnicas de teste estrutural, funcional e baseada em erros.

Segundo Howden (1987), é importante ressaltar que as técnicas de teste devem ser vistas como complementares e a questão está em como utilizá-las de forma que as vantagens de cada uma sejam melhor exploradas, possibilitando uma estratégia que leve a uma atividade

de teste de boa qualidade, ou seja, eficaz e de baixo custo.

Apresenta-se, a seguir, uma definição para cada uma das técnicas citadas acima (PRESSMAN, 2002):

- **Teste Estrutural**: Também chamado de caixa-branca, é um método de projeto de casos de teste que usa a estrutura de controle do projeto procedimental para derivar casos de teste. Usando métodos de teste caixa-branca, o engenheiro de software pode derivar casos de teste que: (1) garantam que todos os caminhos independentes de um módulo tenham sido exercitados pelo menos uma vez; (2) exercitam todas as decisões lógicas em seus lados verdadeiros e falsos; (3) executam todos os ciclos nos seus limites e dentro de seus intervalos operacionais; e (4) exercitam as estruturas de dados internas para garantir a sua validade.
- **Teste Funcional**: Também chamado de caixa-preta ou comportamental, focaliza os requisitos funcionais do software. O teste caixa-preta permite ao engenheiro de software derivar conjuntos de condições de entrada que vão exercitar plenamente todos os requisitos funcionais de um programa.
- **Baseada em erros**: O objetivo do teste baseado em erros em um sistema é projetar testes que tenham uma grande probabilidade de descobrir erros sutis. Como o produto ou sistema deve satisfazer a requisitos do cliente, o planejamento preliminar necessário para realizar este tipo de teste começa com o modelo de análise. O testador procura erros sutis (aspectos da implementação do sistema que podem resultar em defeitos), para determinar se esses erros existem, casos de teste são projetados para exercitar o projeto ou código.

Para este trabalho será abordada apenas a técnica de Teste Funcional, sendo que a mesma servirá como base para a proposta da Técnica de Teste Funcional Baseada em

Diagramas da UML.

1.2 Critérios de Teste

Um critério de teste é o que define quais propriedades precisam ser testadas, a fim de encontrar o maior número possível de erros. Como é impossível garantir inexistência, o conceito é utilizado, na prática, para definir uma qualidade mínima que será validada pelo teste.

De um modo bem específico, serve para direcionar a atividade de teste e tomar decisões relativas ao teste. Os critérios de teste podem ser usados de duas maneiras:

- **Critério de seleção** (também chamado de critério de geração): quando o critério é utilizado para selecionar um conjunto de dados de teste;
- **Critério de adequação** (também chamado de critério de cobertura): quando o critério é utilizado para avaliar a qualidade de um conjunto de dados de teste.

Formalmente, critérios de teste definem qual é o conjunto de elementos requeridos do software que devem ser exercitados (ROCHA *et al*, 2001).

1.2.1 Teste Funcional

O Teste Funcional também é conhecido como Teste Caixa Preta, devido ao fato do software ser tratado como uma caixa, na qual é visível e disponível apenas a interface, ou seja, o lado externo (BEIZER, 1990).

O conteúdo (implementação) não é utilizado pelo testador que, ao invés disso, utiliza-se da especificação para derivar os requisitos de teste. Por este motivo, pode-se dizer que o Teste Funcional analisa o produto sob o ponto de vista macroscópico.

Segundo Pressman (2002), o Teste Funcional procura, entre outras coisas, mostrar que os requisitos funcionais do software são satisfeitos, que a entrada é adequadamente aceita, que a saída esperada é produzida e que a integridade das informações externas é mantida; por isso, não existe preocupação com a estrutura lógica interna do sistema.

O Teste Funcional exercita o sistema do ponto de vista do usuário, isto é, não considera a estrutura interna ou a forma de implementação do sistema. Este é o único tipo de teste possível quando não se dispõe do código-fonte, por exemplo.

Esta é a mais conhecida forma de testes. O objetivo é verificar se o sistema executa corretamente suas funções normais. Portanto, os casos de testes serão desenvolvidos e introduzidos no sistema, as saídas serão examinadas para testar sua correção (PRESSMAN, 2002).

O teste funcional, também, não é uma alternativa às técnicas da Caixa-Branca ou Teste Estrutural. Pelo contrário, é uma abordagem complementar que mais provavelmente descobrirá classes diferentes de erros não descobertas pelas técnicas de Caixa Branca.

O teste funcional concentra seus esforços nos requisitos funcionais do software. Por meio do teste funcional, é possível verificar erros nas entradas e saídas de cada unidade do software. O teste funcional preocupa-se com o que, e não com o como uma determinada unidade do software está sendo executada (BEIZER, 1990).

O teste funcional tem, basicamente, a finalidade de descobrir (PRESSMAN, 2000):

- funções incorretas ou ausentes;
- erros de interface;
- erros nas estruturas de dados ou no acesso a bancos de dados externos;

- erros de desempenho;
- erros de inicialização e término.

Para se realizar o teste funcional, leva-se em consideração o estado dos objetos que estão sobre teste. O teste deve ser aplicado de maneira que alcance todo o seu ciclo de vida, sendo assim as operações dos objetos não podem ser testadas de maneira individual. Diferente do teste Caixa-Branca, que é realizado no início do processo de teste, o teste funcional tende a ser aplicado durante os últimos estágios do teste. Como o teste funcional descarta a estrutura de controle, a atenção é focalizada no domínio da informação. A seguir, são apresentados alguns critérios de teste funcional (PRESSMAN, 2002):

- **Particionamento de Equivalência**: divide o domínio de entrada de um programa em classes de equivalência válidas e inválidas, a partir das condições de entrada de dados identificadas na especificação. Depois, seleciona casos de teste supondo que um elemento de dada classe representaria a classe toda, lembrando que, para classes inválidas, devem ser gerados casos de teste distintos. O uso desse critério permite restringir o número de casos de teste necessários.
- **Análise de Valor Limite**: este critério é complementar ao critério de particionamento de equivalência. Ao invés de se fazer a seleção de qualquer elemento de uma classe de equivalência, a análise de valor limite leva à seleção de casos de testes nas extremidades da classe, pois nesses pontos pode estar uma grande concentração de erros. Em vez de se concentrar somente nas condições de entrada, a análise de valor limite deriva os casos de teste também do domínio de saída.
- **Grafo Causa-Efeito**: estabelece requisitos de teste baseados nas possíveis combinações das condições de entrada que os critérios de Particionamento de

Equivalência e Análise de Valor Limite não exploram. Primeiramente, são levantadas as possíveis condições de entrada (causa) e as possíveis ações (efeitos) do programa; em seguida, é desenvolvido um grafo relacionando causas e efeitos, que é convertido em tabela de decisão a partir da qual são derivados os casos de teste.

Vale lembrar que, como esses critérios baseiam-se nas especificações do software para derivar os requisitos de teste, estes critérios podem ser aplicados tanto a programas procedimentais quanto a programas orientados a objetos (HOFFMAN and STROOPER, 1997).

Este trabalho apresenta uma forma de aplicar a técnica de Teste Funcional baseando-se nos Diagramas da UML.. Na realidade, depois de feita a análise, o testador faz uso da Ferramenta FuTeBOOL (*Functional Testing Building on Object Oriented Language*), que será apresentada no Capítulo 2, para que possam ser criados os Elementos-Requeridos para o teste, a fim de encontrar erros funcionais, independente da linguagem de programação que será usada.

Para tanto, é necessário o uso dos Diagramas de Casos de Uso, Diagramas de Classe, Diagrama de Seqüência e Diagrama de Colaboração para que as associações funcionais possam ser identificadas podendo assim encontrar erros oriundos da análise.

1.2.2 Teste Estrutural

O Teste Estrutural, também chamado de caixa-branca, é uma filosofia de projeto de casos de teste que usa a estrutura de controle descrita como parte do projeto ao nível de componentes para derivar casos de teste. Usando métodos de teste caixa-branca, o engenheiro

de software pode derivar casos de teste que: (1) garantam que todos os caminhos independentes de um módulo tenham sido exercitados pelo menos uma vez; (2) exercitam todas as decisões lógicas em seus lados verdadeiros e falsos; (3) executam todos os ciclos nos seus limites e dentro de seus intervalos operacionais; e (4) exercitam as estruturas de dados internas para garantir a sua validade (PRESSMAN, 2006).

Uma pergunta razoável pode ser feita neste ponto: "Por que gastar tempo e energia preocupando-se com (e testando) minúcias lógicas, quando poderíamos empregar melhor o esforço garantindo que os requisitos do programa foram satisfeitos?" Dito de outro modo, por que não gastamos toda nossa energia em testes caixa-preta? A resposta está na natureza dos defeitos de software (PRESSMAN, 2002):

- Erros lógicos e pressupostos incorretos são inversamente proporcionais à probabilidade de que um caminho de programa vai ser executado. Os erros tendem a penetrar no nosso trabalho quando projetamos e implementamos funções, condições ou controle que estão fora da função principal. Um processamento cotidiano tende a ser bem entendido (e bem examinado), enquanto um processamento "de casos especiais" tende a cair pelas frestas.
- Freqüentemente acreditamos que um caminho lógico não é provável de ser executado quando, na realidade, ele pode ser executado em base regular. O fluxo lógico de um programa é algumas vezes contra-intuitivo, significando que nossos pressupostos inconscientes sobre o fluxo de controle e dados podem nos levar a cometer erros de projeto que são descobertos apenas quando o teste de caminhos começa.
- Erros tipográficos são aleatórios. Quando um programa é traduzido em código-fonte, numa linguagem de programação, é provável que ocorram alguns erros de digitação. Muitos serão descobertos por mecanismos de verificação de sintaxe e

ortografia, mas outros podem continuar não detectados ate que o teste comece. É provável que um erro tipográfico exista tanto num caminho lógico obscuro quanto num caminho principal.

Cada uma dessas razoes fornece argumento para a condução de testes caixa-branca. O teste caixa preta, independentemente de quão rigoroso, pode não encontrar as espécies de erro mencionadas aqui. O teste caixa-branca tem muito mais probabilidade de descobri-los.

1.3 UML – Unified Modeling Language

A *Unified Modeling Language* (UML), Linguagem de Modelagem Unificada, é uma linguagem gráfica para visualização, especificação, construção e documentação de artefatos de sistemas complexos de software (BOOCH, 2000).

A UML é o resultado da unificação dos métodos de Booch, OMT (Rumbaugh) e OOSE (Jacobson), que dá origem a uma linguagem padronizada para a modelagem de sistemas de software orientados a objetos, sendo adotada pela indústria de software como linguagem padrão, e também por fornecedores de Ferramentas CASE, conforme mostrado na Figura 1.1.

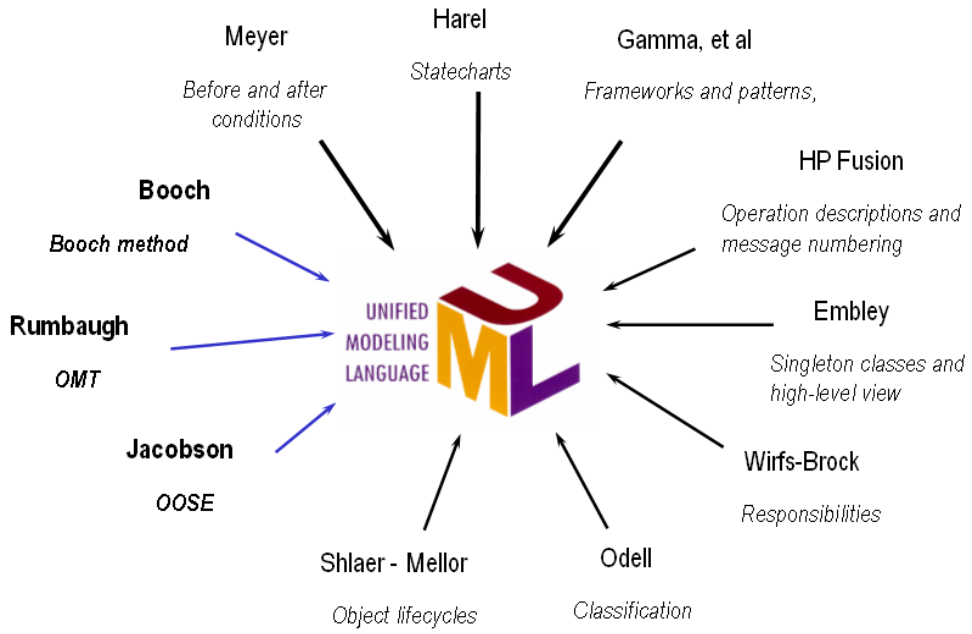


Figura 1.1 – Origens da UML (BOOCH, 2000)

Os trabalhos para a criação da UML iniciaram-se em 1994 com Grady Booch da Rational Software (método Booch) e James Rumbaugh (método OMT), que combinaram seus dois métodos mais populares.

O grande problema do desenvolvimento de novos sistemas utilizando a orientação a objetos nas fases de análise de requisitos, análise de sistemas e projeto é que não existia uma notação padronizada e realmente eficaz que abrangesse qualquer tipo de aplicação que se desejasse desenvolver.

Cada simbologia existente possuía seus próprios conceitos, gráficos e terminologias, resultando numa grande confusão, especialmente para aqueles que queriam utilizar a orientação a objetos não só sabendo para que lado aponta a seta de um relacionamento, mas sabendo criar modelos de qualidade para ajudá-los a construir e manter sistemas cada vez mais eficazes.

Quando a "*Unified Modeling Language*" (UML) foi lançada, os desenvolvedores da área da orientação a objetos ficaram entusiasmados já que essa padronização proposta era o tipo de força que eles sempre esperaram.

Por isso, além de outras razões, o bom entendimento da UML não é apenas aprender a simbologia e o seu significado, mas também significa aprender a modelar orientado a objetos no estado da arte. A UML foi a junção do que havia de melhor nas metodologias de Booch, Rumbaugh e Jacobson.

Segundo Booch (2000), a arquitetura de um sistema complexo de software pode ser descrita mais adequadamente por cinco visões interligadas. Cada visão constitui uma projeção na organização e estrutura dos sistemas, cujo foco está voltado para determinado aspecto desse sistema, conforme ilustrado na Figura 1.2.

Figura 1.2 – Visões da UML (BOOCH, 2000)

No Quadro 1.1 apresenta-se a descrição e os diagramas utilizados para a modelagem de aspectos estáticos e dinâmicos do sistema (BOOCH, 2000):

Quadro 1.1 – Diagramas utilizados na modelagem de aspectos estáticos e dinâmicos.

Visão	Descrição	Modelagem de Aspectos Estáticos	Modelagem de Aspectos Dinâmicos
Visão de Casos de Uso	Abrange os casos que descrevem o comportamento do sistema conforme são vistos pelos usuários finais, analistas e pessoal de teste.	- Diagrama de Casos de Uso.	- Diagrama de Interação; - Diagrama de estados; - Diagrama de Atividades.
Visão de Projeto	Abrange classes, interfaces e colaborações que formam o vocabulário do problema e de sua solução.	- Diagrama de Classes; - Diagrama de Objetos.	- Diagrama de Interação; - Diagrama de estados; - Diagrama de Atividades.
Visão de Processo	Abrange as tarefas e processos que formam os mecanismos de concorrência e sincronização do sistema.	- Diagrama de Classes; - Diagrama de Objetos.	- Diagrama de Interação; - Diagrama de estados; - Diagrama de Atividades.
Visão de Implementação	Abrange os componentes utilizados para montar e liberar o sistema físico.	- Diagrama de Componentes.	- Diagrama de Interação; - Diagrama de estados; - Diagrama de Atividades.
Visão de Implantação	Abrange os nós que formam a topologia de hardware em que o sistema é executado.	- Diagrama de Implantação.	- Diagrama de Interação; - Diagrama de estados; - Diagrama de Atividades.

1.3.1 Diagramas da UML

Segundo Booch (2000), um diagrama é a representação gráfica de um conjunto de elementos do sistema. Esses diagramas são desenhados para permitir a visualização de um sistema sob diferentes perspectivas, sendo assim um diagrama constitui uma projeção de um determinado sistema. Em todos os sistemas, com exceção dos mais triviais, um diagrama

representa uma visão parcial dos elementos que o compõem. Um mesmo elemento pode aparecer em todos os diagramas, em alguns ou em nenhum diagrama.

Teoricamente, um diagrama pode conter qualquer combinação de itens e de relacionamentos, porém na prática, aparecerá um pequeno número de combinações comuns, que são consistentes com as cincos visões (Figura 1.3) mais úteis da arquitetura de um sistema complexo de software. A UML disponibiliza nove desses diagramas que permitem representar diferentes partes do modelo de um sistema (BOOCH, 2000):

- Diagrama de Caso de Uso;
- Diagrama de Classes;
- Diagrama de Objetos;
- Diagrama de Componentes;
- Diagrama de Implantação.
- Diagrama de Interação (Seqüência e Colaboração);
- Diagrama de Estados;
- Diagrama de Atividades.

Para os estudos apresentados neste trabalho foram usados apenas os Diagramas de Casos de Uso, Diagramas de Classe, Diagramas de Seqüência e Diagramas da Colaboração, conforme as Seções 1.3.1.1, 1.3.1.2, 1.3.1.3 e 1.3.1.4.

1.3.1.1 Diagrama de Caso de Uso

O diagrama de casos de uso consiste em um diagrama utilizado para modelar os aspectos estáticos de um sistema. Os diagramas de casos de uso modelam o comportamento de um sistema por meio de atores, casos de uso e relacionamentos.

Um caso de uso consiste em uma interação entre um usuário e o sistema (BOOCH, 2000). É um modo específico que define processos genéricos que o sistema deve ser capaz de manipular, estabelecendo assim, um conjunto de funcionalidades inter-relacionadas do sistema (JACOBSON, 1992).

Na UML, os detalhes levantados na especificação de requisitos, que são gerados na documentação produzida na fase de análise e projeto e nas especificações, podem servir como fontes de informações para o projeto de casos de teste.

Os casos de uso do sistema, que podem gerar um documento chamado listas de casos de uso, devem ser identificados e listados utilizando um identificador para cada função, e a especificação informal deve seguir a regra de frases curtas, usando frases do tipo "O sistema deve...", conforme visto no Quadro 1.2.

Quadro 1.2 - Funções de um sistema para Gerenciamento de Recursos de um projeto.

Identificador	Funcionalidade
UC1	O sistema deve manter um registro dos recursos alocados no projeto, permitindo adicionar, remover ou atualizar informações sobre os recursos.
UC2	Manter um registro de pré-requisitos, permitindo associar um recurso ao pré-requisito.
UC3	O sistema deve permitir consultar um pré-requisito.

Segundo Larman (2004), uma excelente técnica para melhorar o entendimento dos requisitos é a transformação das funcionalidades em casos de uso. Casos de uso são definidos pela UML e podem ser visualmente representados em Diagramas de Caso de Uso.

Na Figura 1.3, apresenta-se o Diagrama de Caso de Uso do Gerenciamento de

Recursos de um projeto.

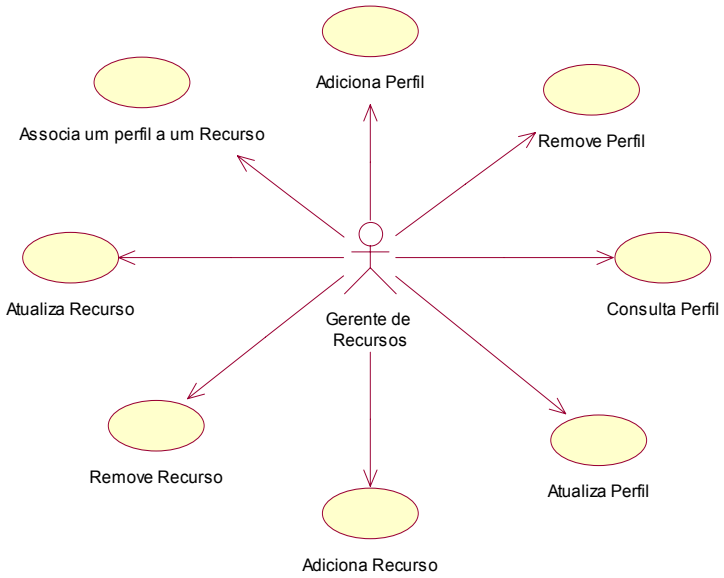


Figura 1.3 – Diagrama de Caso de Uso do sistema de Gerenciamento de Recursos de um projeto.

As elipses no diagrama da Figura 1.3 representam os Casos de Uso, as linhas representam a comunicação entre os atores, que podem ser humanos ou outros sistemas, e os Casos de Uso. Cada Caso de Uso requer um documento que descreva a seqüência de eventos de um ator que é um agente externo (Figura 1.4) usando o sistema.

Figura 1.4 – Exemplos de Atores

É muito útil inicia a descrição dos casos de uso com uma visão de alto nível para rapidamente obter algum entendimento dos principais processos do negócio. Pode-se associar cada caso de uso por meio de uma seção de referência cruzada com todas as funções identificadas e relacionadas no documento de requisitos ou lista de Casos de Uso.

Isso providencia uma informação importante em termos de rastreabilidade entre funções, Casos de Uso, implementação e testes. No Quadro 1.3 apresenta-se uma estrutura do texto narrativo para o Caso de Uso Adiciona Recurso:

- **Caso de Uso** – Adiciona Recurso
- **Atores** – Gerente de Recursos
- **Propósito** – Adicionar um Recurso no sistema.
- **Função** – UC1

Um sistema especificado como casos de uso, conforme mostrado no Quadro 1.2, fornece algumas informações necessárias para testar o sistema. Jacobson (1992) sugere quatro tipos de testes que podem ser derivados de casos de uso:

- teste do curso básico, ou fluxo esperado dos eventos;
- teste de todos os outros fluxos de eventos;
- teste de qualquer item requerido, rastreável pelo caso de uso;
- teste das funções descritas na documentação do usuário.

Quadro 1.3 – Especificações do Caso de Uso Adiciona Recurso

Identificador	**Ações dos atores**	**Resposta do sistema**
1	Este caso de uso inicia quando o Gerente de Recursos seleciona a opção *Adiciona Recurso*.	O sistema apresenta uma tela para digitação dos dados.
2	O Gerente de Recursos entra com os dados no sistema.	O sistema valida as informações digitadas e retorna uma mensagem confirmando o registro.
Fluxo Alternativo		
3	Mensagem do sistema informando inconsistência nos dados.	

Segundo Binder (2000), Casos de Uso como definidos na UML, OOSE e outros

métodos orientados a objetos, são necessários, mas não suficientes para o projeto de teste do sistema. Deve-se determinar os seguintes itens adicionais:

- domínio de cada variável que participa do Caso de Uso. Variáveis operacionais podem ser derivadas de Casos de Uso. Nos Casos de Uso do sistema Gerenciamento de Recursos e Projetos, algumas variáveis operacionais são identificadas no texto narrativo;
- os relacionamentos de entrada/saída requeridos entre as variáveis e caso de uso;
- a freqüência de ocorrência das interações de cada caso de uso;
- dependências de seqüências entre os casos de uso.

1.3.1.2 Diagrama de Classes

Durante a fase de análise, tem-se a criação do modelo de análise que é o mais importante resultado desta fase. Este modelo dá suporte aos conceitos do cenário, suas associações e atributos. Um rico e expressivo conjunto de objetos, levantados na análise, dão subsídio para as fases de projeto e implementação.

Uma técnica simples e eficaz para identificação de objetos é separar os substantivos das especificações dos cenários que dão origem aos casos de uso. Esses substantivos poderão transformar-se em objetos ou atributos. Vale lembrar que os objetos levantados nesta fase são referentes ao cenário e não ao software desenvolvido.

Assim sendo, o primeiro passo para transformar a especificação do sistema que descreve o que o sistema deve fazer em como o sistema será feito, é identificar a partir dessa especificação as classes e atributos do sistema por meio da análise dos substantivos da especificação, em seguida é possível criar o Diagrama de Classes, conforme mostrado na

Figura 1.5.

As classes de análise são usadas para representar os papéis dos elementos do modelo que oferecem o comportamento necessário para atender aos requisitos funcionais especificados pelos casos de uso e aos requisitos não funcionais descritos nas especificações (cenário). À medida que a ênfase do sistema se desloca para o projeto, esses papéis desenvolvem um conjunto de elementos de projeto que realizam os casos de uso.

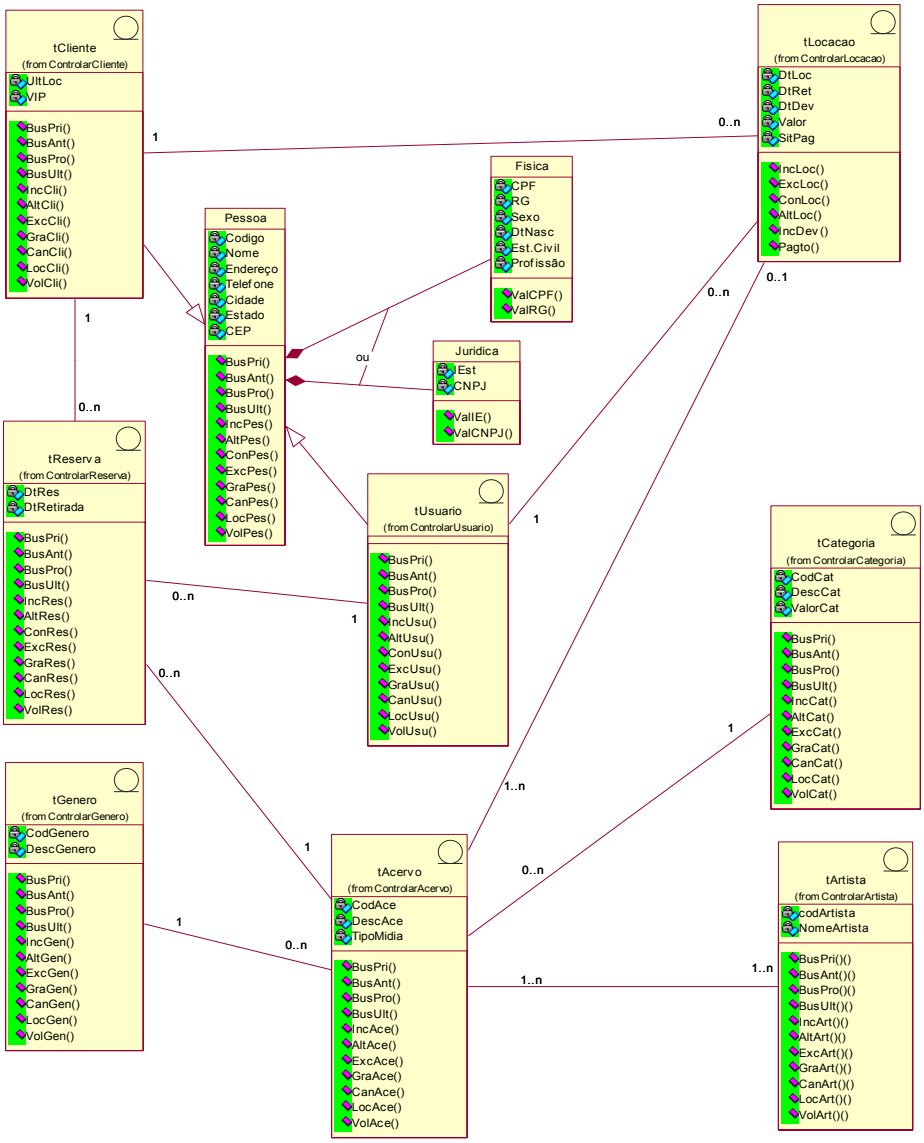


Figura 1.5 – Exemplo de Diagrama de Classes

1.3.1.3 Diagrama de Seqüência

Um Diagrama de Seqüência tem como foco a ordem temporal das mensagens, mostrando um conjunto de objetos e mensagens enviadas e recebidas por esses objetos. Tipicamente os objetos são instâncias nomeadas ou anônimas de classes, mas também podem representar instâncias de outros itens, como colaborações, componentes e nós. O Diagrama de Seqüência é usado para ilustrar a visão dinâmica de um sistema (BOOCH, 2000).

Na maior parte do tempo, os objetos que são mostrados participando de uma interação existem ao longo de toda a duração da interação. Entretanto, em algumas interações, os objetos poderão ser criados (especificados por uma mensagem *create*) e destruídos (especificados por uma mensagem *destroy*).

O mesmo é verdadeiro em relação aos vínculos: os relacionamentos entre objetos poderão ocorrer e desaparecer

Durante uma interação, um objeto muda os valores de seus atributos, seu estado e seus papéis. Pode-se representar as modificações de um objeto, replicando-o na interação (possivelmente com valores diferentes para os atributos, estados ou papéis).

Em um Diagrama de Seqüência, cada variante do objeto poderá ser colocada na mesma linha de vida. Em um diagrama de interação, conectar-se-á cada variante a uma mensagem *become*.

Ao fazer a modelagem de uma interação, incluim-se objetos (cada um desempenhando um papel específico) e mensagens (cada uma representando a comunicação entre objetos, com alguma ação como resultado).

Esses objetos e mensagens envolvidos em uma interação podem ser visualizados de duas maneiras:

- com ênfase na ordenação temporal de suas mensagens;

- com ênfase na organização estrutural dos objetos que enviam e recebem mensagens.

Na UML (*Unified Modeling Language*), o primeiro tipo de representação é chamado Diagrama de Seqüência e o segundo tipo de representação é chamado de Diagrama de Colaboração. Tanto o Diagrama de Seqüência quanto o Diagrama de Colaboração são tipos de Diagramas de Interação.

Os Diagramas de Seqüência e Colaboração são isomórficos, isso significa que um poderá ser transformado no outro sem qualquer perda de informação, entretanto, existem algumas diferenças visuais.

Em primeiro lugar, o Diagrama de Seqüência permite fazer a modelagem da linha de vida de um objeto, que representa a existência desse objeto em um determinado período, possivelmente abrangendo sua criação e destruição.

Em segundo lugar, o Diagrama de Colaboração permite fazer a modelagem de vínculos estruturais que possam existir entre os objetos em uma interação.

Os Diagramas de Seqüência devem ser construídos com base na informação disponível nos casos de uso, dessa forma diz-se que diagramas de seqüência desenham os casos de uso (BOOCH, 2000). Na Figura 1.6 apresenta-se um exemplo de Diagrama de Seqüência.

Como um caso de uso descreve toda a vertente de uma dada funcionalidade (incluindo casos de erro e exceção) é mais fácil construir um diagrama de seqüência por cenário.

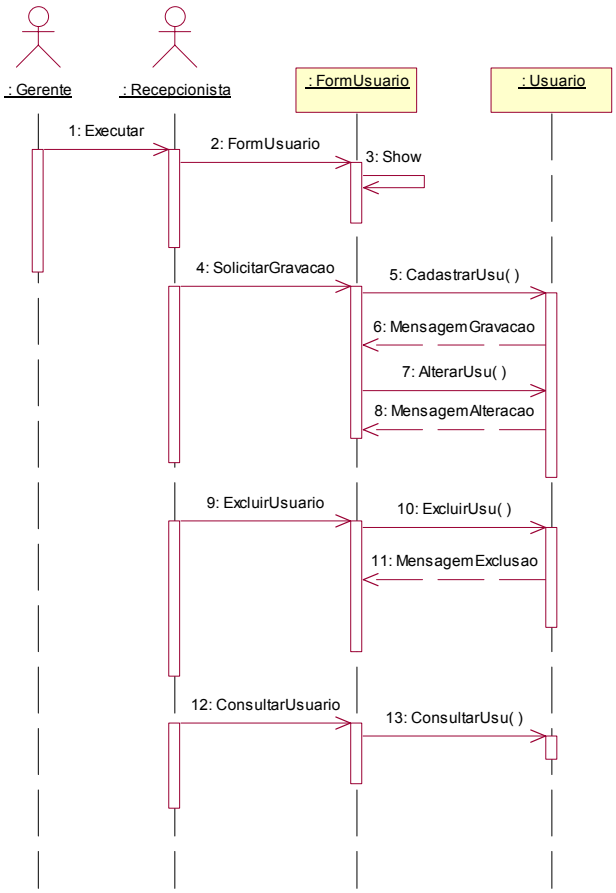


Figura 1.6 – Exemplo de Diagrama de Seqüência

De acordo com a Figura 1.6, um diagrama de seqüência é uma tabela que mostra graficamente (BOOCH, 2000):

- os objetos organizados ao longo do eixo X;
- mensagens ordenadas temporalmente no eixo Y;
- a linha de vida de um objeto: linha tracejada que representa a existência de um objeto num período de tempo;
- o fluxo de controle de execução: retângulo estreito que mostra o período de tempo desde que o objeto recebe a mensagem até o momento em que fornece uma resposta (tempo de execução de uma operação), este tempo pode incluir tempos de execução de operações subordinadas.

1.3.1.4 Diagrama de Colaboração

A ênfase deste diagrama está na organização estrutural dos objetos que enviam e recebem mensagens, os objetos da colaboração são vértices de um grafo, os vínculos são os arcos e contém as mensagens que os objetos enviam e recebem.

O diagrama de colaboração define uma estrutura de comunicação entre os objetos, pode-se identificar todas as conexões objeto-a-objeto em um único diagrama de colaboração. Cada mensagem é numerada para documentar a ordem na qual ela ocorre, conforme mostrado na Figura 1.7.

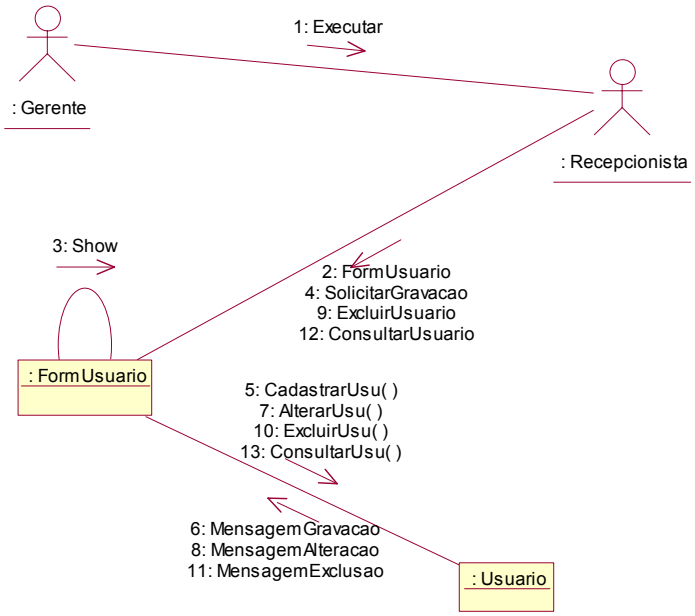


Figura 1.7 – Exemplo de Diagrama de Colaboração.

Ao contrário do diagrama de seqüência, nesse diagrama, o foco não recai sobre o tempo, mas na organização dos objetos. Por isso, esse diagrama mostra explicitamente as conexões entre os objetos (o que o diagrama de seqüência não faz), enquanto ele deve acrescentar números de seqüência às mensagens para indicar a ordem de chamada.

Uma outra diferença é que no diagrama de seqüência se mostra explicitamente o retorno das mensagens enquanto o diagrama de colaboração não o faz.

1.4 Considerações Finais

Pode-se observar neste capítulo que a UML, por meio de seus diagramas, possibilita uma visão completa no que diz respeito ao embasamento para os desenvolvedores de software. Com esses diagramas os analistas podem ter uma visão mais clara e eficiente de todas as especificações do software, incluindo seus requisitos funcionais e não-funcionais.

2 FERRAMENTA DE GERAÇÃO DE ELEMENTOS REQUERIDOS DE TESTE FUNCIONAL

Neste capítulo, são apresentadas as funcionalidades e particularidades da Ferramenta FuTeBOOL – Tool (*Functional Testing Building on Object Oriented Language – Tool*), Ferramenta para Geração de Teste Funcional Baseada na Linguagem Orientada a Objetos.

2.1 Diagrama de Classes da Ferramenta FuTeBOOL

Na Figura 2.1, apresenta-se o Diagrama de Classes da Ferramenta FuTeBOOL. Este Diagrama de Classes é composto pelas seguintes entidades: Sistema, Classe, Sentido, Método e Ferramenta. A partir do Diagrama de Classes apresentado na Figura 2.1 serão construídas as regras (metadados[1]) e o Banco de dados utilizados pela Ferramenta "FuTeBOOL".

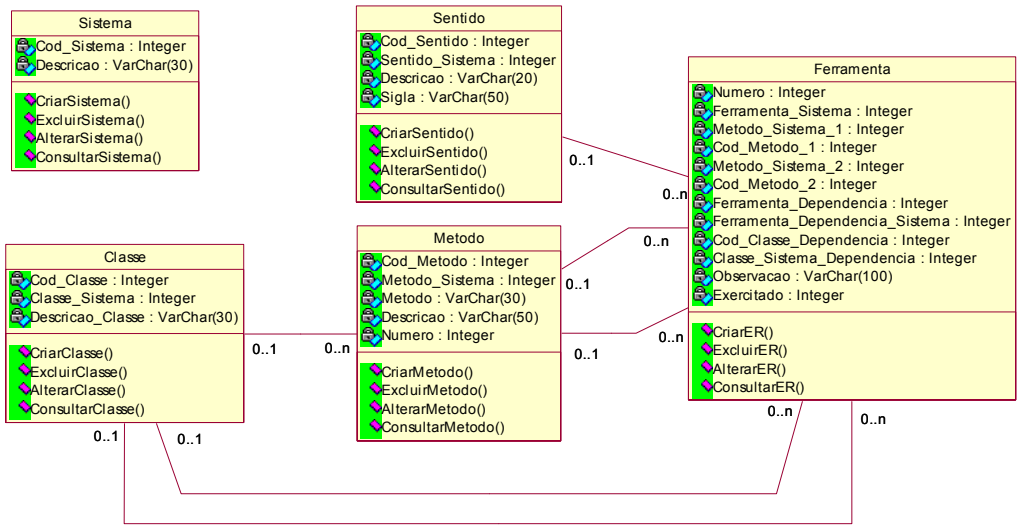


Figura 2.1 – Diagrama de Classes da Ferramenta FuTeBOOL

[1] É um catálogo que contém informações detalhadas como: variáveis de relações, índices, grupos de usuários, restrições de integridade, restrições de segurança, etc (DATE, 2000).

2.2 Funcionalidades da Ferramenta FuTeBOOL

A Ferramenta FuTeBOOL é caracterizada por funcionalidades que facilitam o processo de geração dos Elementos Requeridos para Teste Funcional Baseado em Diagramas da UML. As principais funcionalidades englobadas pela ferramenta são:

- **Cadastro de Sistema**: Esta funcionalidade é necessária uma vez que podemos ter mais de um projeto sendo avaliado e testado ao mesmo tempo; para isso tem-se o cadastro de sistema.

- **Cadastro de Classes**: Esta funcionalidade é necessária para que as classes do sistema em questão possam ser cadastradas. Nesta funcionalidade, é possível incluir, excluir, alterar e consultar classes.

- **Cadastro de Métodos**: Esta funcionalidade é necessária para que os métodos das classes possam ser cadastrados.

- **Cadastro de Sentido**: Esta funcionalidade é necessária para que os sentidos das associações entre os métodos de cada classe possam ser cadastrados; com isso é possível saber de que forma um método depende do outro, se é Verificação, Informação, Execução, Execução-Informação ou Verificação-Informação.

- **Alteração de Sistema**: Esta funcionalidade é necessária para que se possa alterar para o projeto com o qual se queria trabalhar, sem perda de dados no projeto atual.

- **Relatórios**: A ferramenta conta com relatórios de Classes, Métodos e Sentidos.

- **Geração de Elementos Requeridos**: Por meio desta funcionalidade, é possível gerar os Elementos Requeridos de Teste Funcional. Quando um elemento requerido é gerado e salvo, automaticamente é criado o Elemento Requerido Válido e o Elemento Requerido Inválido.

- ***Check-list***: Depois que todos os Elementos Requeridos são criados, é possível

gerar uma lista com todos esses elementos, possibilitando a marcação dos que foram exercitados ou não exercitados, possibilitando assim a geração da Estatística. Ainda é possível observar a existência de uma coluna indicando os resultados esperados, com isso o resultado do teste fica mais efetivo e mais fácil de observar, se um dado de teste satisfez ou não o elemento requerido e se o resultado esperado foi ou não atendido.

2.3 Arquitetura da Ferramenta FuTeBOOL

A Ferramenta FuTeBOOL foi desenvolvida para que pudesse automatizar a Geração de Elementos Requeridos para Teste Funcional baseada em Diagramas da UML. Os dados necessários para utilizar a ferramenta são extraídos dos Diagramas de Casos de Uso, Classes, Seqüência ou Colaboração, e alimentam um Banco de Dados.

A ferramenta foi desenvolvida utilizando o ambiente de desenvolvimento Delphi 7 e Banco de Dados Interbase 6.5.

Na Figura 2.2, é mostrada a Arquitetura da Ferramenta FuTeBOOL. Esta ferramenta baseia-se inicialmente na Especificação de um software. A partir dessa Especificação, o analista desenvolverá os Diagramas de Casos de Uso, Classes, Seqüência e Colaboração que são a base para a automatização da geração de elementos requeridos para o teste funcional.

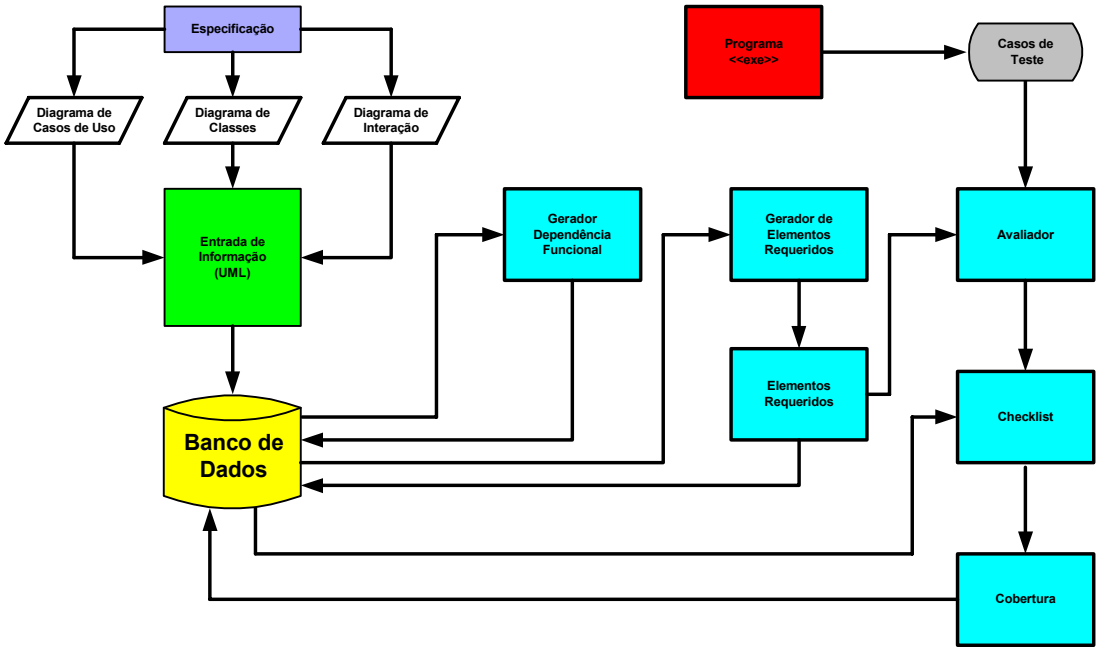


Figura 2.2 – Arquitetura da Ferramenta FuTeBOOL

Essa etapa de análise poderá ser feita com auxílio de uma Ferramenta CASE, porém como a alimentação dos dados de entrada da mesma é feita de forma manual, essa análise pode ser feita sem o auxílio de tais ferramentas.

Após os Diagramas serem criados começa, a etapa que se denomina de entrada de informação, e a partir dos Diagramas de Casos de Uso, é possível desenvolver os demais diagramas. Analisando o Diagrama de Classes, é possível alimentar na ferramenta o cadastro de Classe e Métodos. Através da análise dos Diagramas de Seqüência ou Colaboração é possível determinar a qual Classe um método é pertencente. Ainda é necessário efetuar o cadastro de Sentidos que serão usados na geração dos elementos requeridos. A partir do cadastro das Classes, dos Métodos e dos Sentidos no Banco de Dados da ferramenta iniciam-se os trabalhos de geração dos elementos funcionais para o teste.

Após a alimentação de todas as informações necessárias, é possível executar o módulo de Geração das Dependências funcionais. Nesse módulo são geradas todas as dependências funcionais existentes no projeto, levam-se em consideração a dependência de uma determinada classe com outras por meio da dependência criada pelos seus métodos.

Também é importante identificar pré e pós-condições dessas dependências e uma observação é cadastrada, na qual se descreve qual é a dependência para o elemento requerido que será gerado. Conforme são identificadas essas dependências o Banco de Dados é alimentado.

Depois dessas dependências serem criadas, é possível, de maneira automática, gerar os elementos requeridos para o teste funcional, que são enviados para o Banco de Dados. Após a geração de todos os elementos requeridos e o envio dessas informações para o Banco de Dados, inicia-se a geração dos casos de testes para serem executados no programa em teste.

Cada caso de teste marca um número referente à execução do programa em teste e a partir destes casos de teste, pode-se requerer uma avaliação da quantidade de Elementos Requeridos que foram exercitados pelos respectivos Casos de Teste executados até o momento. A avaliação pode ser requerida sempre que houver necessidade da obtenção de cobertura dos Elementos Requeridos. O módulo avaliador gera um *check-list* que é baseado em informações constantes no Banco de Dados, neste *check-list* é possível marcar quais foram os Elementos Requeridos exercitados.

Após a marcação dos Elementos Requeridos exercitados, o módulo de Cobertura faz análise dessas marcações e gera uma estatística dos resultados, o que mostra quantitativamente os resultados do projeto de teste para o software desenvolvido. Essas estatísticas ficam armazenadas no Banco de Dados.

2.4 Resultados Gerados

O resultado obtido com a Ferramenta FuTeBOOL é a automação na Geração de Elementos Requeridos de Teste Funcional, como a Rational Rose não exporta os métodos

criados e utilizados em um projeto desenvolvido utilizando a mesma, fez-se necessário o desenvolvimento da Ferramenta FuTeBOOL, para que fosse simplificada a etapa de Geração destes elementos.

A Ferramenta FuTeBOOL auxilia não somente na geração dos elementos requeridos, mas também auxilia na marcação e nas verificações de quais elementos foram exercitados e quais elementos não foram exercitados, e se um dado de teste satisfez ou não o elemento requerido e se o resultado esperado foi ou não atendido; podendo, ao final desta etapa, gerar uma estatística, em níveis percentuais, do projeto de Teste Funcional aplicado baseado em Diagramas da UML e auxiliado pela Ferramenta FuTeBOOL.

2.5 Considerações Finais

Com esta ferramenta, será possível diminuir o tempo da elaboração dos Elementos Requeridos do Teste Funcional e tornar esta elaboração mais segura. Todo o trabalho desenvolvido utilizando a Ferramenta FuTeBOOL, é feito baseando-se em Diagramas da UML..

Conclui-se que, com a Ferramenta FuTeBOOL, o Teste Funcional poderá ser aplicado e baseado em Elementos Requeridos reais, tornando o mesmo mais confiável e seguro. Vale lembrar que esta técnica de teste pode ser aplicada a qualquer software, independente da linguagem de programação em que foi desenvolvido.

Depois do software desenvolvido é que estes Elementos Requeridos e a Ferramenta FuTeBOOL serão usados atestando assim uma maior qualidade ao produto de software comercialmente desenvolvido.

3 ESTUDO DE CASO DA APLICABILIDADE DA FERRAMENTA FuTeBOOL.

Para o estudo de caso, utilizou-se um sistema de Gerenciamento de Hotéis – SOFTHOTEL que foi desenvolvido para servir como ferramenta gerencial e de controle de hotéis e pousadas. A escolha deste exemplo de uso se deu pela quantidade de diagramas desenvolvidos na etapa de projeto, bem como por possuir inúmeras funcionalidades que facilitam a visibilidade da aplicabilidade do uso da Ferramenta FuTeBOOL.

3.1 Diagrama de Classes do SOFTHOTEL

Na Figura 3.1, é mostrado o Diagrama de Classes do Sistema de Gerenciamento de Hotéis (SOFTHOTEL).

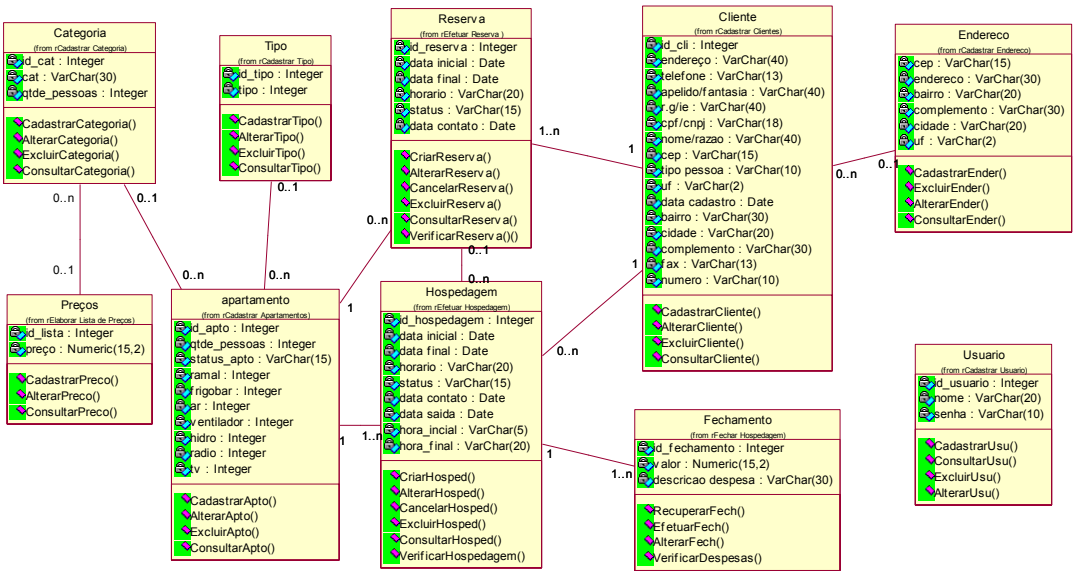


Figura 3.1– Diagrama de Classes do Sistema de Gerenciamento de Hotéis (SOFTHOTEL)

3.2 Diagramas da UML para o Sistema de Gerenciamento de Hotéis (SOFTHOTEL)

Nesta seção, são apresentados os Diagramas da UML utilizados no desenvolvimento dos projetos, através da Ferramenta FuTeBOOL.

Os diagramas utilizados no uso da Ferramenta FuTeBOOL são:

- Diagrama de Caso de Uso;
- Diagrama de Seqüência;
- Diagrama de Colaboração;
- Diagrama de Classes.

3.2.1 Diagrama de Caso de Uso do Sistema de Gerenciamento de Hotéis.

Na Figura 3.2, é apresentado o Diagrama de Casos de Uso do Sistema de Gerenciamento de Hotéis que possui os seguintes atores: Gerente, Atendente e Recepcionista.

O Sistema de Gerenciamento de Hotéis possui os seguintes casos de uso: Cadastrar Tipo, Cadastrar Categoria, Cadastrar Apartamentos, Elaborar Lista de Preços, Efetuar Reserva, Efetuar Hospedagem, Cadastrar Clientes, Cadastrar Usuário, Fechar Hospedagem e Cadastrar Endereço.

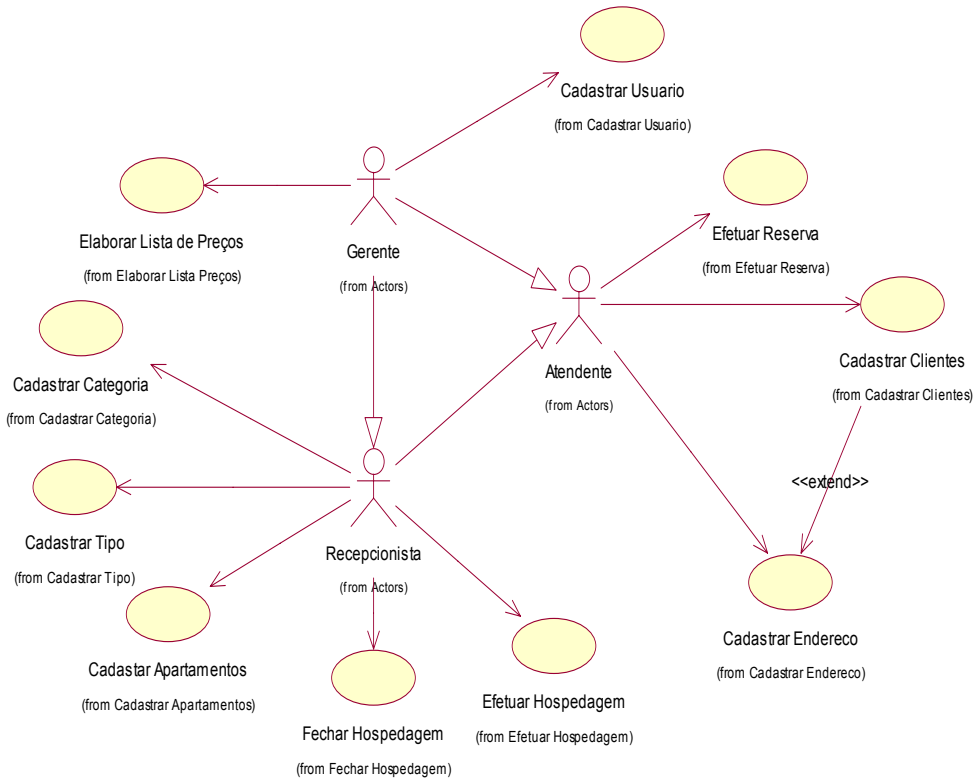


Figura 3.2 – Diagrama de Casos de Uso do Sistema SOFTHOTEL.

3.2.2 Diagrama de Seqüência de Cadastrar Apartamentos (Apartamento: Classe 1000)

Na Figura 3.3, é apresentado o Diagrama de Seqüência de Cadastrar Apartamentos. Esse diagrama mostra a interação da classe Apartamento com as classes Tipo, Categoria, Hospedagem e Reserva. Para cadastrar ou alterar um Apartamento, é necessário que Tipo e Categoria já estejam previamente cadastrados. Para excluir um Apartamento, é necessário verificar se não existe Hospedagem e Reserva criadas para o Apartamento em questão.

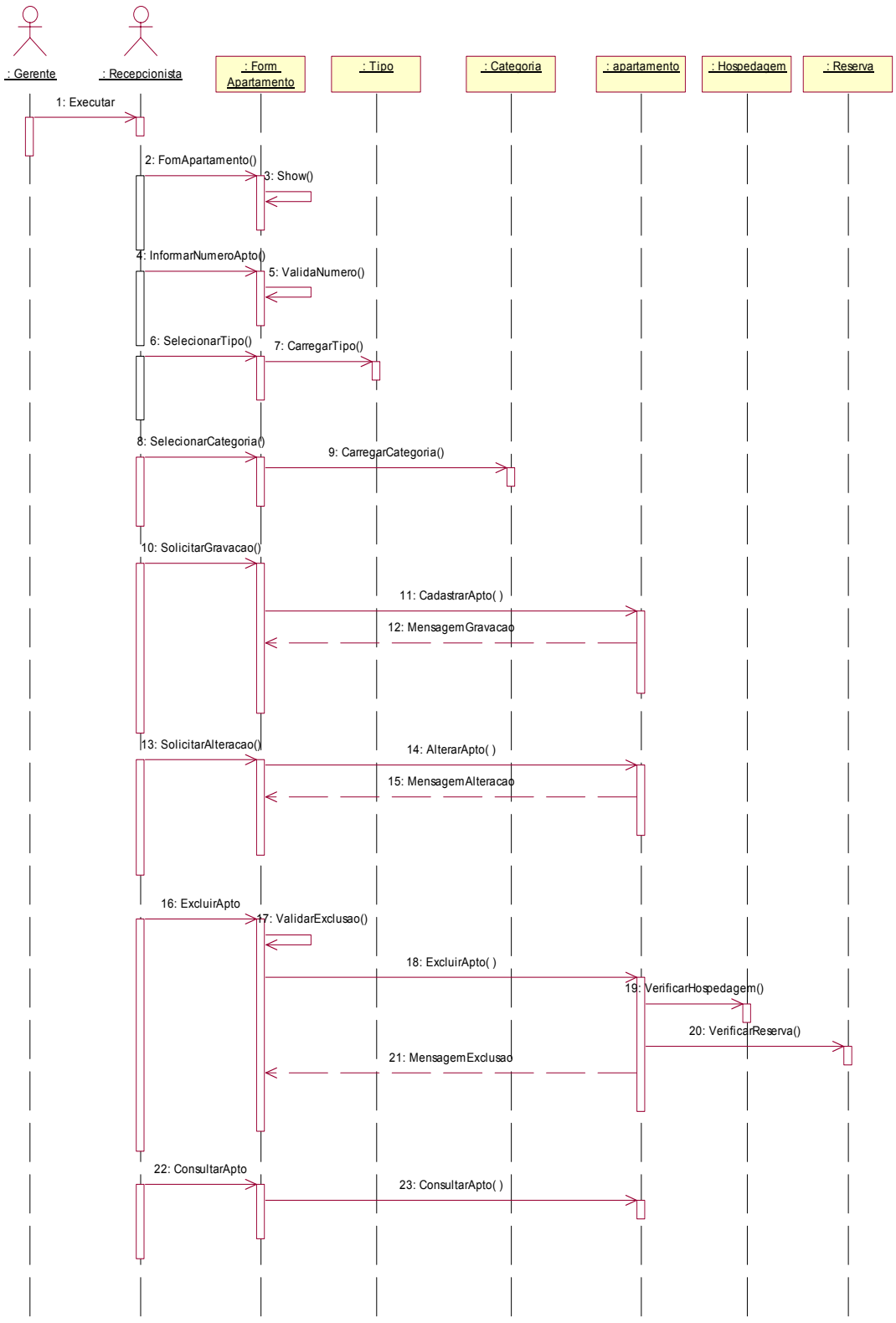


Figura 3.3 – Diagrama de Seqüência de Cadastrar Apartamento

3.2.3 Diagrama de Colaboração de Cadastrar Apartamentos (Apartamento: Classe 1000)

A Figura 3.4 apresenta o Diagrama de Colaboração de Cadastrar Apartamentos. Este diagrama dá ênfase à organização estrutural dos objetos que enviam e recebem mensagens, mostrando a relação dos objetos através dos métodos.

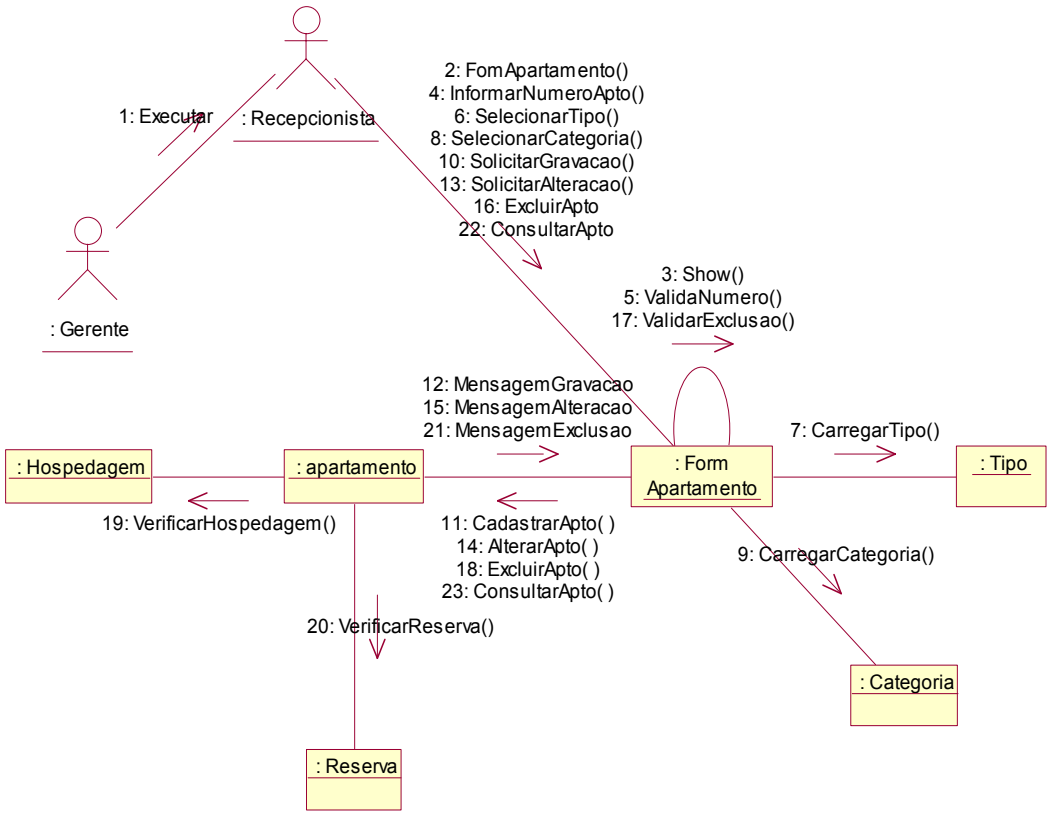


Figura 3.4 – Diagrama de Colaboração de Cadastrar Apartamento

3.2.4 Diagrama de Seqüência de Cadastrar Categoria (Categoria: Classe 2000)

Na Figura 3.5, apresenta-se o Diagrama de Seqüência de Cadastrar Categorias. Esse diagrama mostra a interação da classe Categoria com a classe Apartamento. Para cadastrar um apartamento, é necessário o cadastro de Categoria. Para alterar uma Categoria, é necessário verificar se não existe Apartamento cadastrado com a Categoria em questão. Para excluir uma Categoria, é necessário verificar se não existe Apartamento cadastrado com a Categoria em questão.

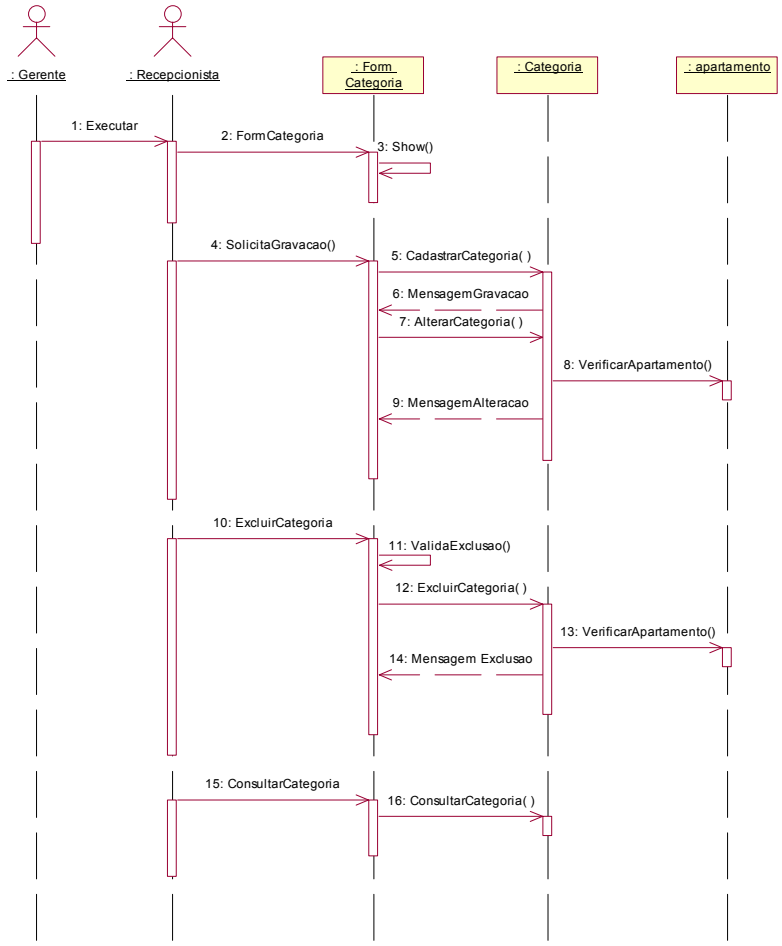


Figura 3.5 – Diagrama de Seqüência de Cadastrar Categorias.

3.2.5 Diagrama de Colaboração de Cadastrar Categoria (Categoria: Classe 2000)

Na Figura 3.6, destaca-se o Diagrama de Colaboração de Cadastrar Categorias. Este diagrama dá ênfase à organização estrutural dos objetos que enviam e recebem mensagens, mostrando a relação dos objetos através dos métodos.

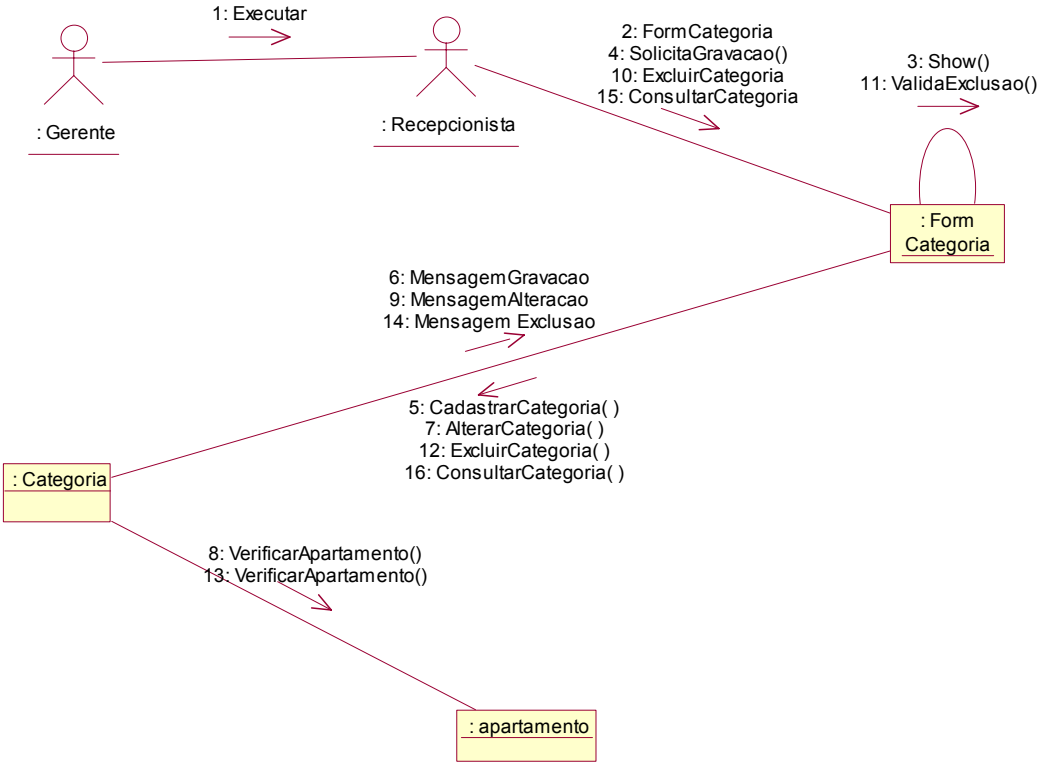


Figura 3.6 – Diagrama de Colaboração de Cadastrar Categorias

3.2.6 Diagrama de Seqüência de Cadastrar Tipo (Tipo: Classe 3000)

Na Figura 3.7, é apresentado o Diagrama de Seqüência de Cadastrar Tipos. Esse diagrama mostra a interação da classe Tipo com a classe Apartamento. Para cadastrar um apartamento, é necessário o cadastro de Tipo. Para alterar um Tipo, é necessário verificar se não existe Apartamento cadastrado com o Tipo em questão. Para excluir um Tipo, é necessário verificar se não existe Apartamento cadastrado com o Tipo em questão.

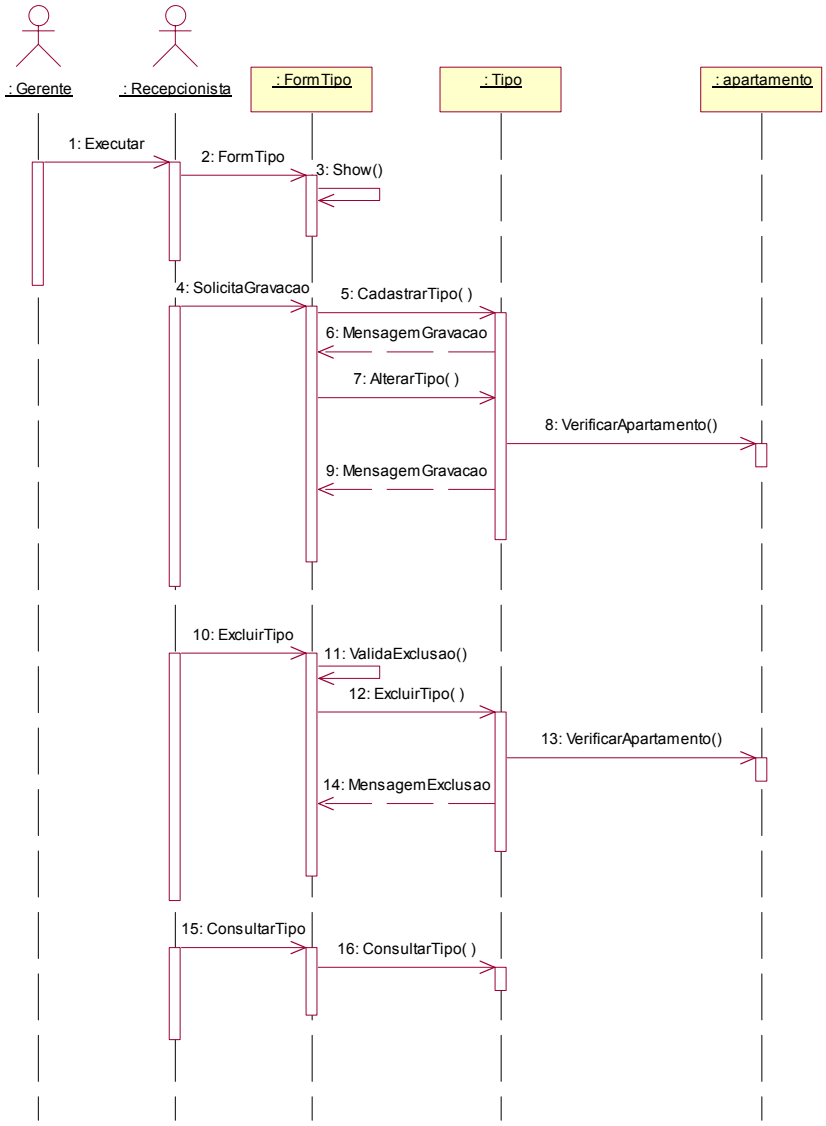


Figura 3.7 – Diagrama de Seqüência de Cadastrar Tipos

3.2.7 Diagrama de Colaboração de Cadastrar Tipo (Tipo: Classe 3000)

A Figura 3.8 apresenta o Diagrama de Colaboração de Cadastrar Tipos. Este diagrama dá ênfase à organização estrutural dos objetos que enviam e recebem mensagens, mostrando a relação dos objetos através dos métodos.

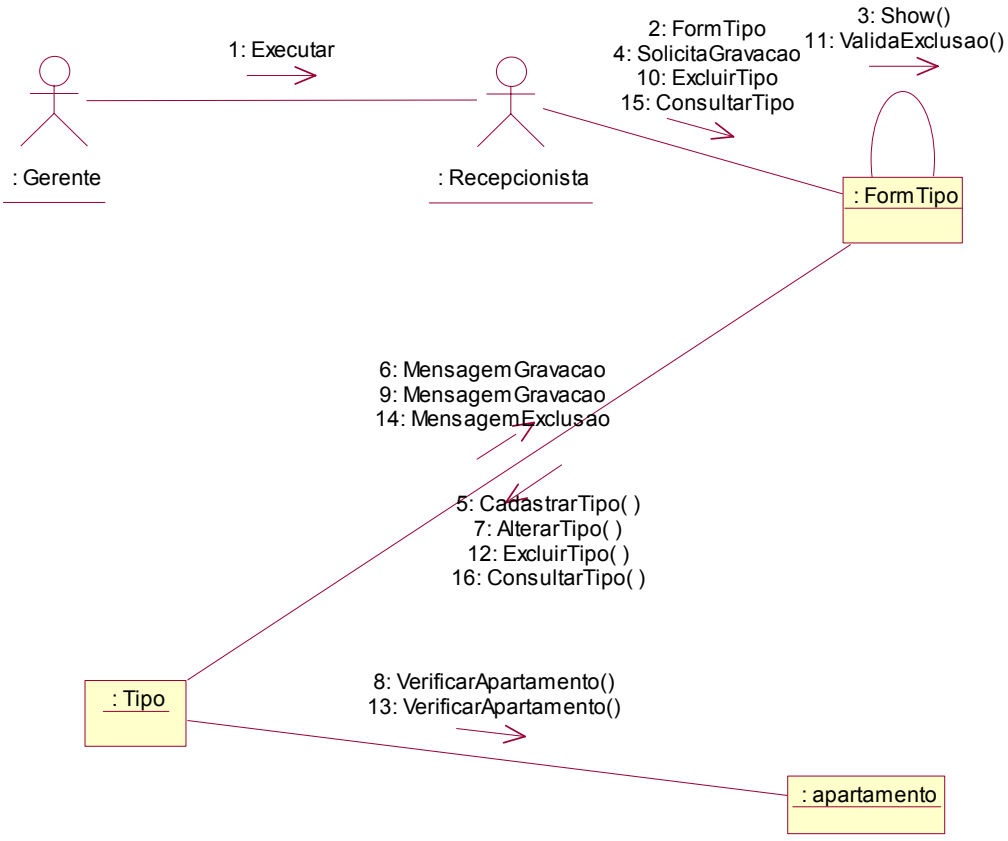


Figura 3.8 – Diagrama de Colaboração de Cadastrar Tipos

3.2.8 Diagrama de Seqüência de Cadastrar Clientes (Cliente: Classe 4000)

Na Figura 3.9, apresenta-se o Diagrama de Seqüência de Cadastrar Clientes. Esse diagrama mostra a interação da classe Cliente com as classes Hospedagem, Reserva e Endereço. Para cadastrar ou Alterar um Cliente , é necessário que o Endereço já esteja previamente cadastrado. A consulta de Cliente pode ser feita através da Hospedagem ou Reserva. Para excluir um Cliente, é necessário verificar se não existe Hospedagem ou Reserva criadas para o Cliente em questão.

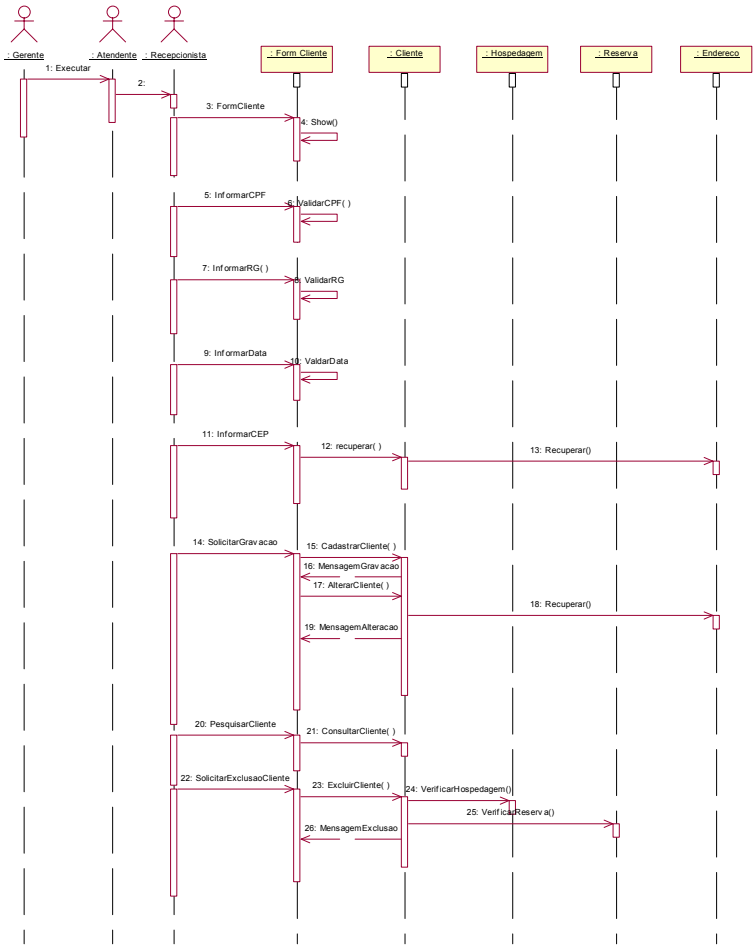


Figura 3.9 – Diagrama de Seqüência de Cadastrar Clientes.

3.2.9 Diagrama de Colaboração de Cadastrar Clientes (Cliente: Classe 4000)

Na Figura 3.10, é apresentado o Diagrama de Colaboração de Cadastrar Clientes. Este dá ênfase à organização estrutural dos objetos que enviam e recebem mensagens mostrando a relação dos objetos através dos métodos.

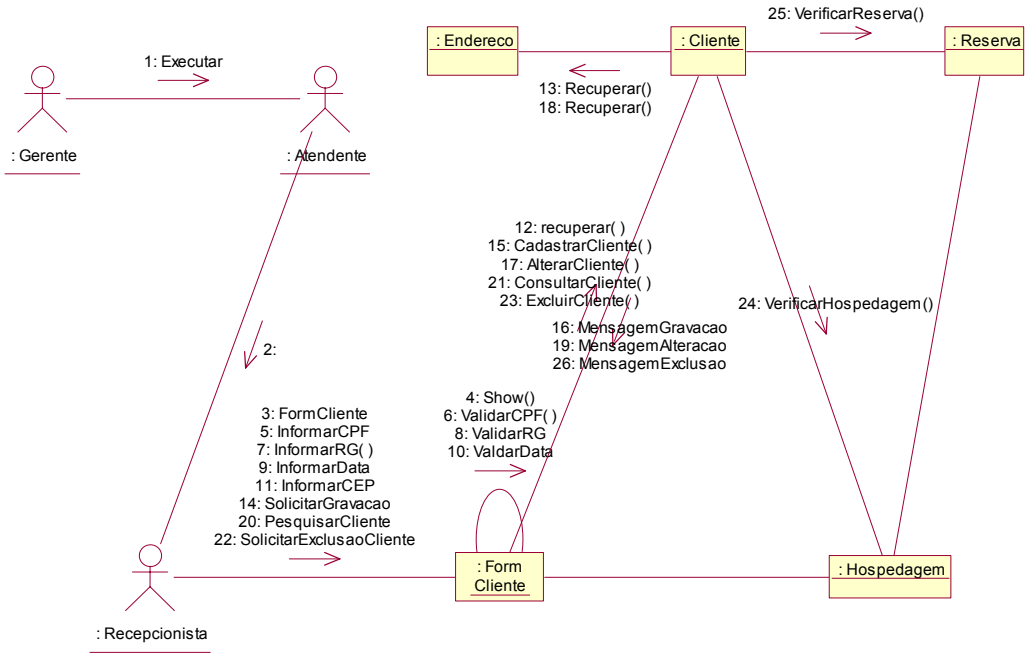


Figura 3.10 – Diagrama de Colaboração de Cadastrar Clientes.

3.2.10 Diagrama de Seqüência de Efetuar Hospedagem (Hospedagem: Classe 5000)

Na Figura 3.11, apresenta o Diagrama de Seqüência de Efetuar Hospedagens. Esse diagrama mostra a interação da classe Hospedagem com as classes Cliente, Apartamento, Reserva e Fechamento. Para efetuar uma Hospedagem, é necessário verificar se existe quarto cadastrado e se esse quarto não está alocado a uma reserva no mesmo período. Para alterar uma Hospedagem, é necessário verificar se existe quarto cadastrado e se esse quarto não está alocado a uma reserva no mesmo período. Para cancelar ou excluir uma hospedagem, é necessário verificar se foi efetuado o fechamento das despesas da mesma.

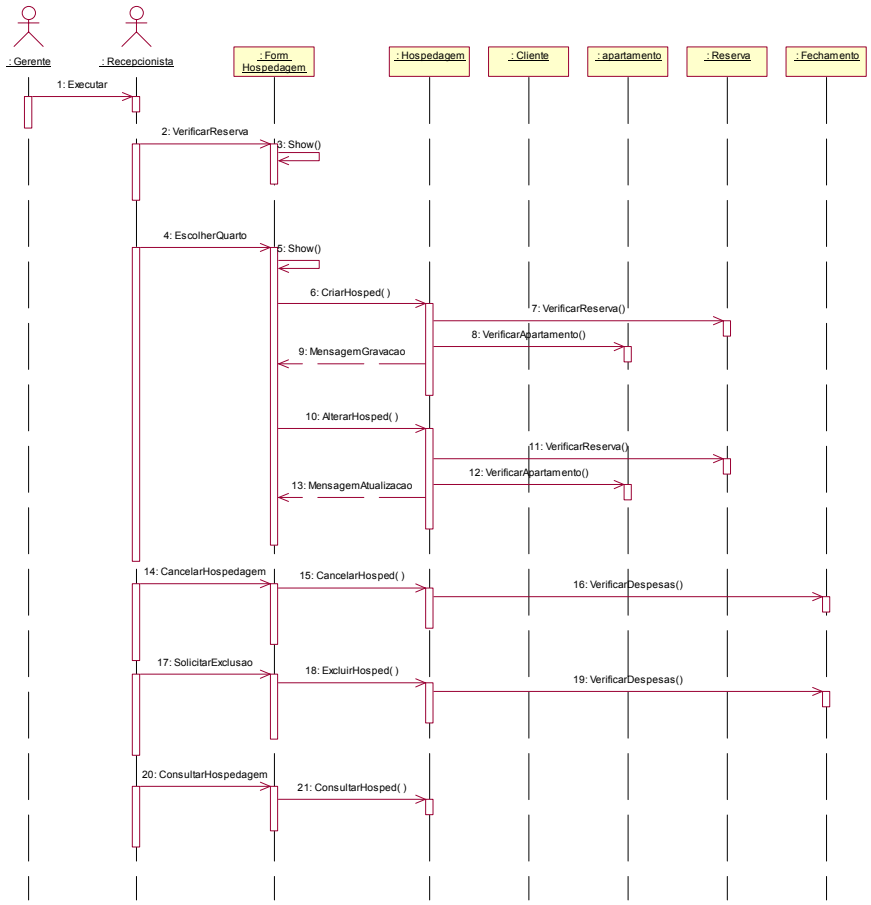


Figura 3.11 – Diagrama de Seqüência de Efetuar Hospedagens.

3.2.11 Diagrama de Colaboração de Efetuar Hospedagem (Hospedagem: Classe 5000)

A Figura 3.12 destaca o Diagrama de Colaboração de Efetuar Hospedagens. Este diagrama dá ênfase à organização estrutural dos objetos que enviam e recebem mensagens, mostrando a relação dos objetos através dos métodos.

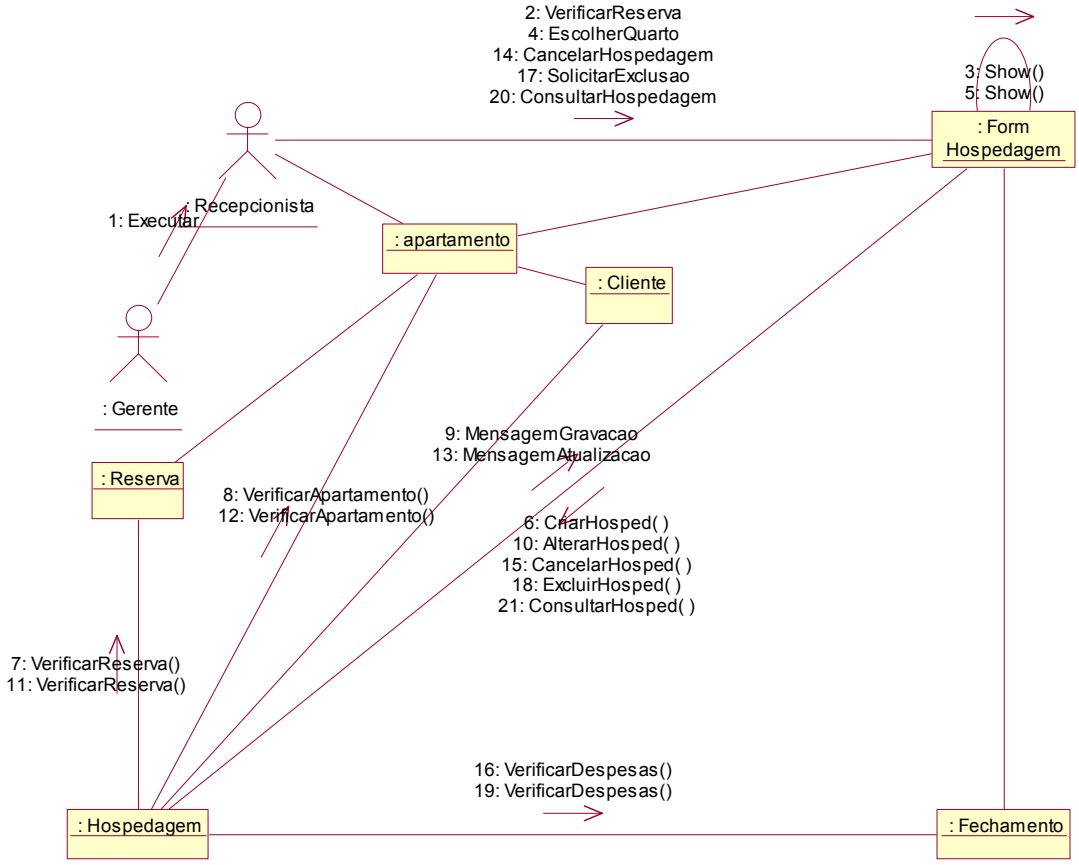


Figura 3.12 – Diagrama de Colaboração de Efetuar Hospedagem.

3.2.12 Diagrama de Seqüência de Efetuar Reserva (Reserva: Classe 6000)

Na Figura 3.13, é apresentado o Diagrama de Seqüência de Efetuar Reservas. Para efetuar uma Reserva, é necessário verificar se não existe uma reserva efetuada para o mesmo período levando em consideração o quarto da reserva. Para alterar uma Reserva, é necessário verificar se não existe uma reserva efetuada para o mesmo período levando em consideração o quarto da reserva. Para excluir ou cancelar uma Reserva, é necessário que a mesma tenha sido efetuada.

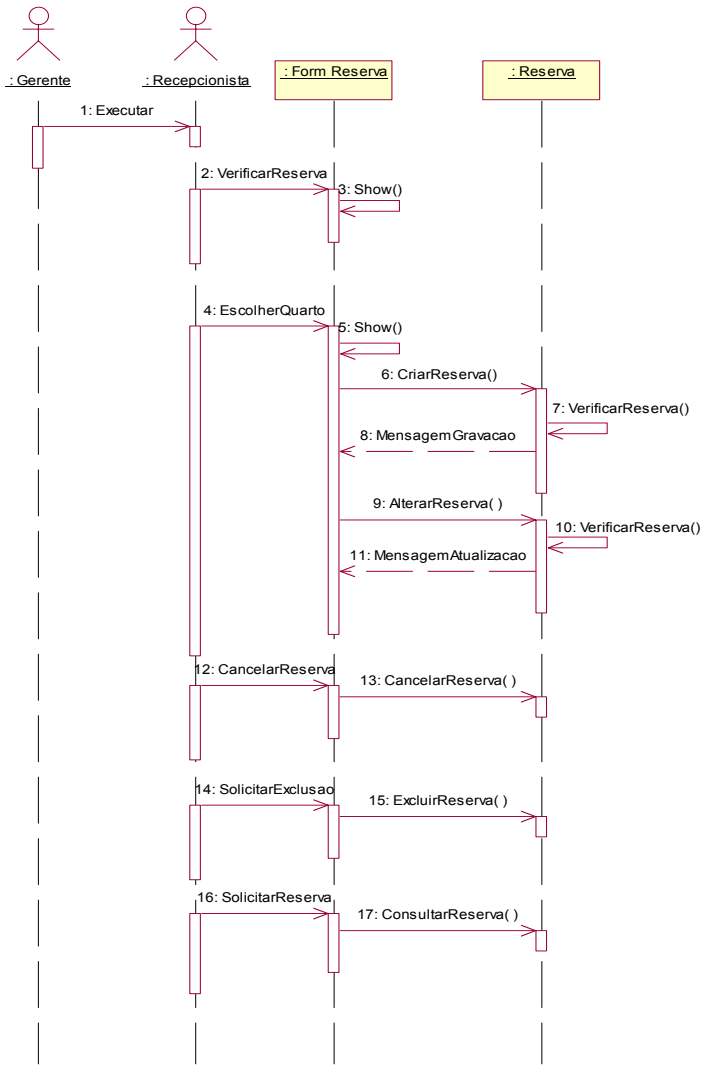


Figura 3.13 – Diagrama de Seqüência de Efetuar Reservas.

3.2.13 Diagrama de Colaboração de Efetuar Reserva (Reserva: Classe 6000)

Na Figura 3.14 apresenta-se o Diagrama de Colaboração de Efetuar Reservas. Este diagrama dá ênfase à organização estrutural dos objetos que enviam e recebem mensagens, mostrando a relação dos objetos através dos métodos.

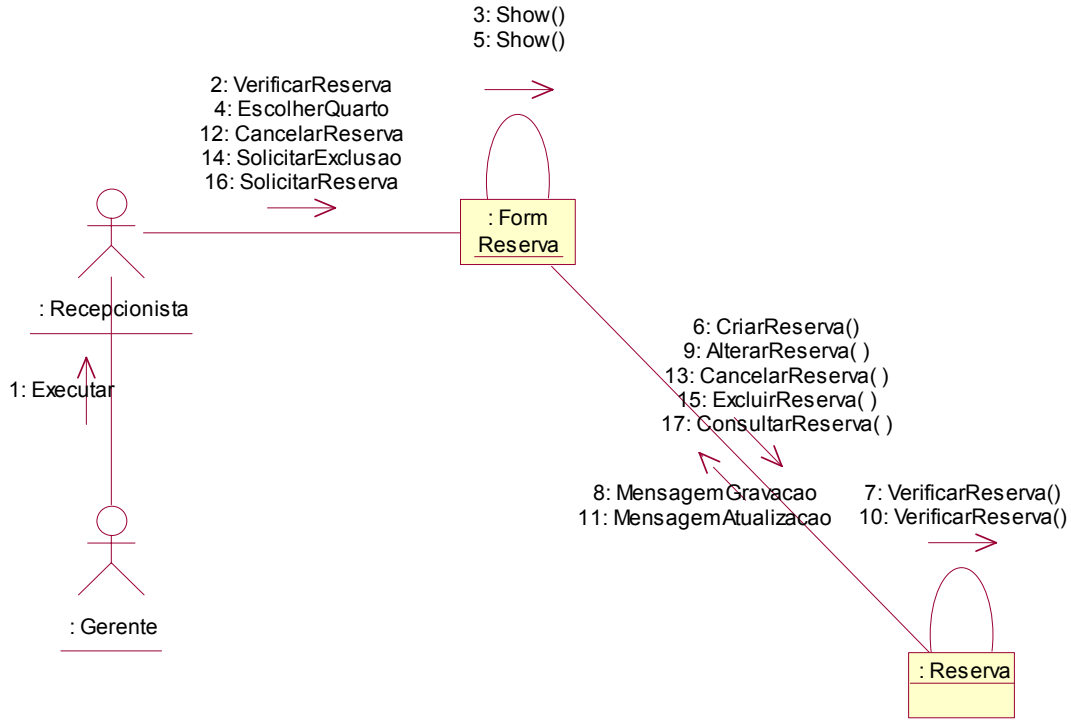


Figura 3.14 – Diagrama de Colaboração de Efetuar Reservas

3.2.14 Diagrama de Seqüência de Efetuar Lista de Preços (Preços: Classe 7000)

A Figura 3.15 demonstra o Diagrama de Seqüência de Efetuar Lista de Preços. Esse diagrama mostra a interação da classe Preços com as classes Categoria e Hospedagem. Para cadastrar um Preço, é necessário que a categoria já esteja criada, pois este preço está vinculado à categoria do quarto. Para alterar um Preço é necessário que a categoria já esteja cadastrada, pois este preço está vinculado à categoria do quarto, é preciso verificar se não há hospedagem que use quartos cadastrados da categoria em que se deseja alterar o preço.

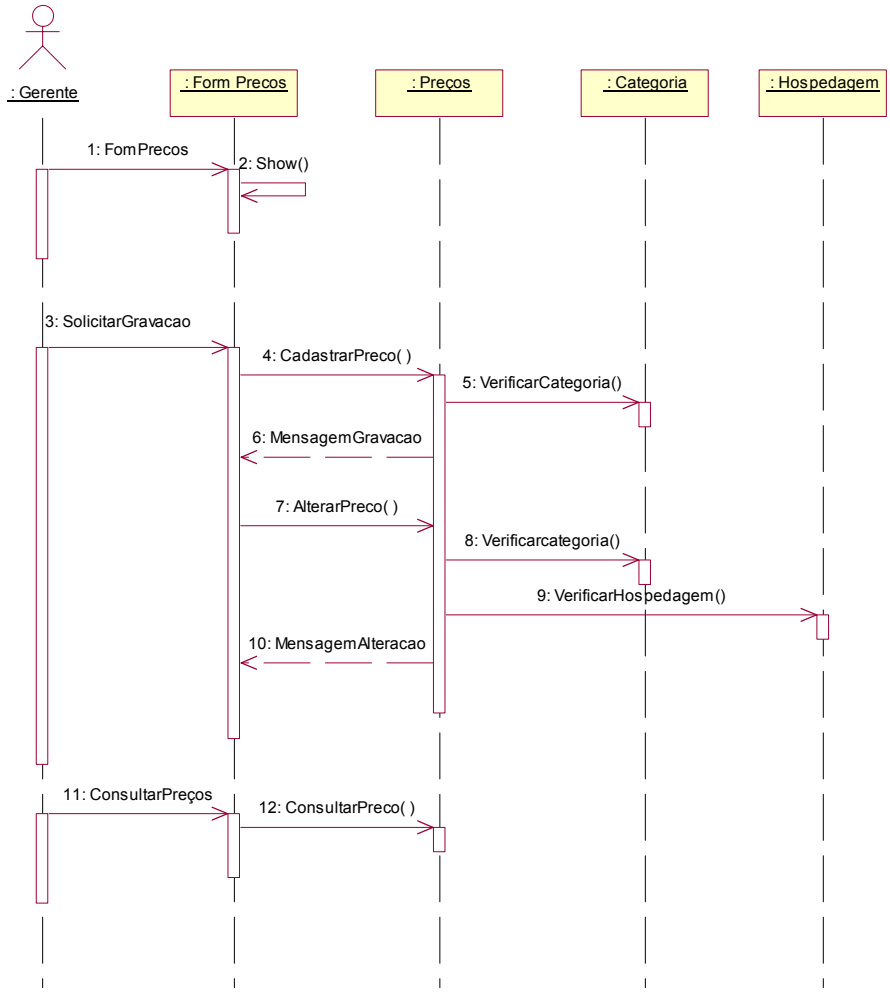


Figura 3.15 – Diagrama de Seqüência de efetuar Lista de Preços.

3.2.15 Diagrama de Colaboração de Efetuar Lista de Preços (Preços: Classe 7000)

Na Figura 3.16 é apresentado o Diagrama de Colaboração de Efetuar Lista de Preços. Este diagrama enfatiza a organização estrutural dos objetos que enviam e recebem mensagens, mostrando a relação dos objetos através dos métodos.

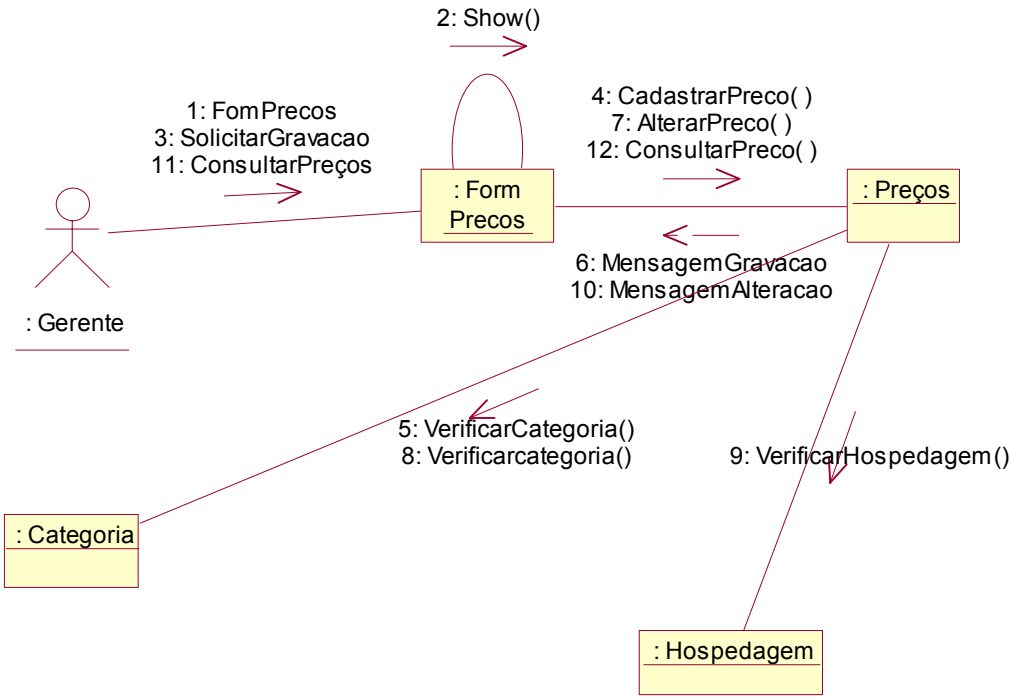


Figura 3.16 – Diagrama de Colaboração de efetuar Lista de Preços.

3.2.16 Diagrama de Seqüência de Fechar Hospedagem (Fechamento: Classe 8000)

O Diagrama de Seqüência de Fechamento de Hospedagens encontra-se na Figura 3.17. Para efetuar ou alterar um Fechamento, é necessário verificar as despesas extras.

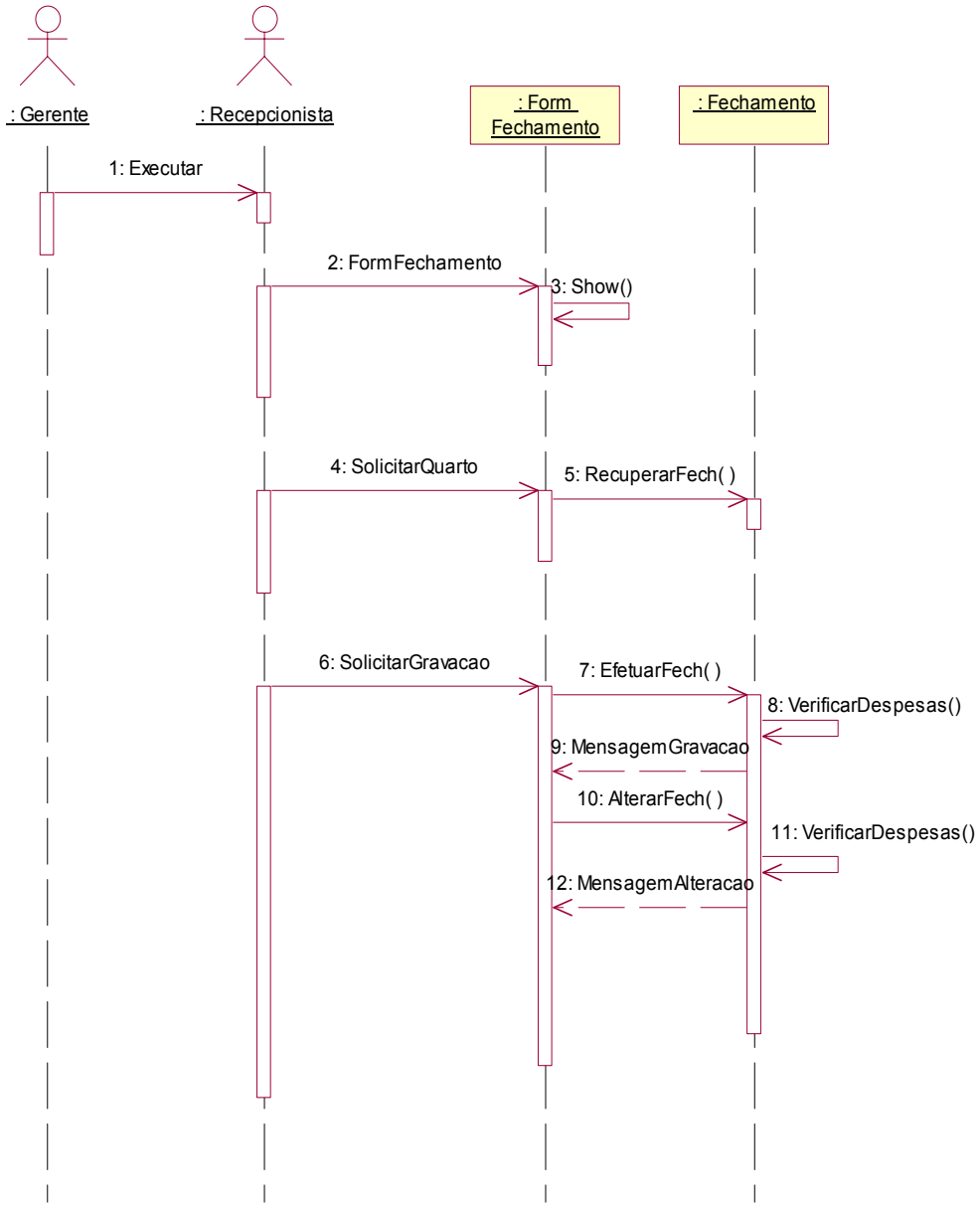


Figura 3.17 – Diagrama de Seqüência de Fechamento de Hospedagens.

3.2.17 Diagrama de Colaboração de Fechar Hospedagem (Fechamento: Classe 8000)

Na Figura 3.18, tem-se o Diagrama de Colaboração de Fechamento de Hospedagens, cuja ênfase está na organização estrutural dos objetos que enviam e recebem mensagens, mostrando a relação dos objetos através dos métodos.

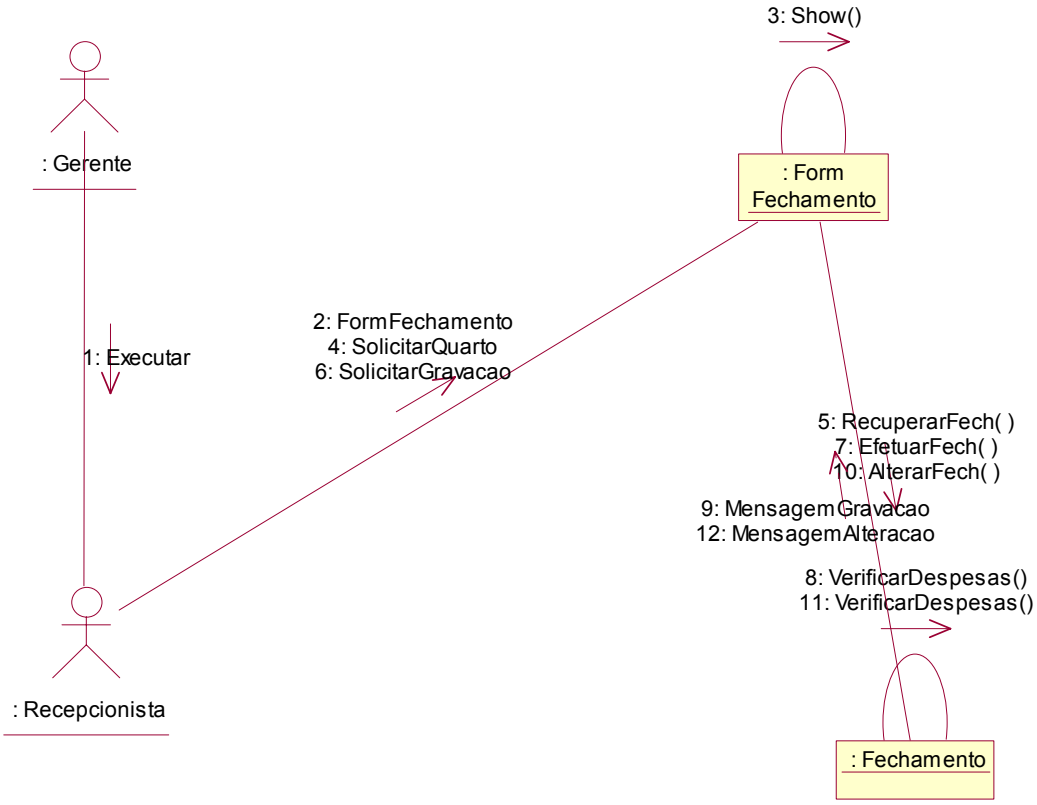


Figura 3.18 – Diagrama de Colaboração de Fechamento de Hospedagens.

3.2.18 Diagrama de Seqüência de Cadastrar Usuário (Usuário: Classe 9000)

A Figura 3.19 apresenta o Diagrama de Seqüência de Cadastrar Usuários. Não há restrições quanto à inserção, alteração, exclusão ou consulta de usuários.

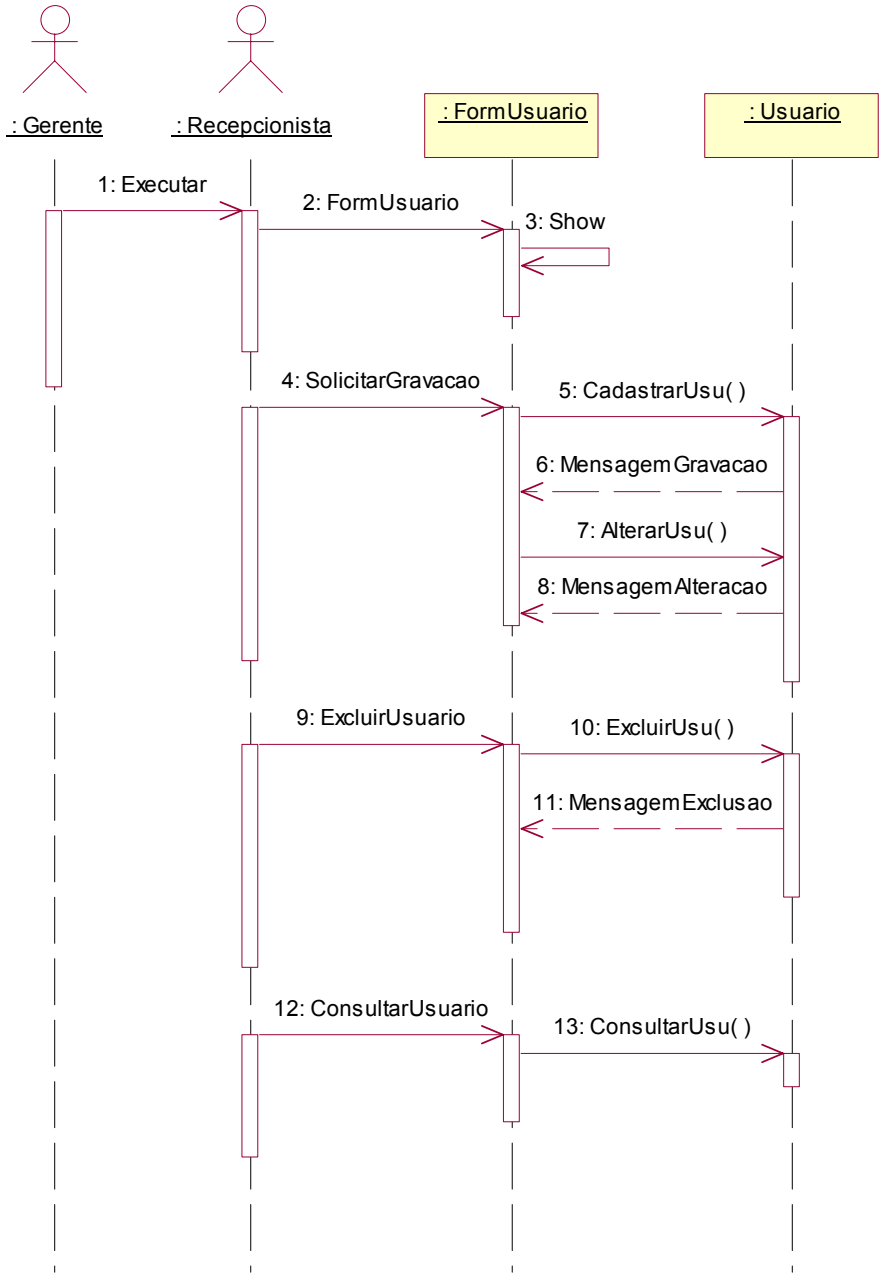


Figura 3.19 – Diagrama de Seqüência de Cadastrar Usuários.

3.2.19 Diagrama de Colaboração de Cadastrar Usuário (Usuário: Classe 9000)

A Figura 3.20 destaca o Diagrama de Colaboração de Cadastrar Usuários. Este diagrama dá ênfase à organização estrutural dos objetos que enviam e recebem mensagens, mostrando a relação dos objetos através dos métodos.

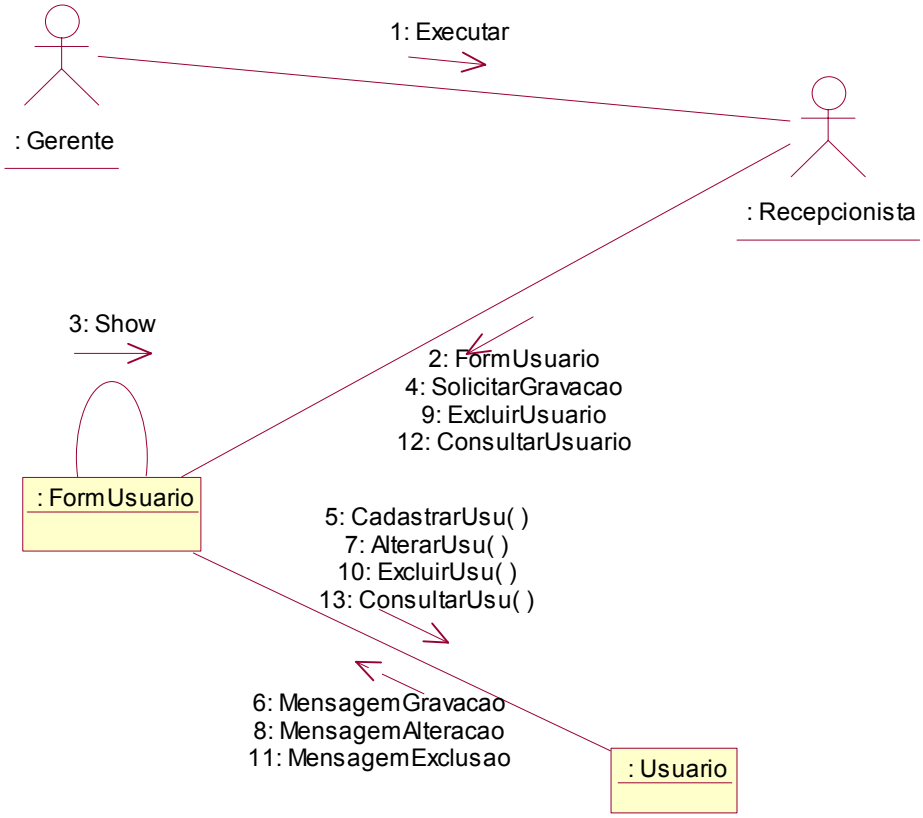


Figura 3.20 – Diagrama de Colaboração de Cadastrar Usuários.

3.2.20 Diagrama de Seqüência de Cadastrar Endereço (Endereço: Classe 10000)

Na Figura 3.21, é apresentado o Diagrama de Seqüência de Cadastrar Endereços. Esse diagrama mostra a interação da classe Endereço com a classe Cliente. Um Endereço será cadastrado antes do cadastro de Cliente, ou seja, no cadastro de cliente é necessário um endereço previamente cadastrado. Para alterar ou excluir um Endereço, é necessário verificar se não existe cliente cadastrado com o Endereço em questão.

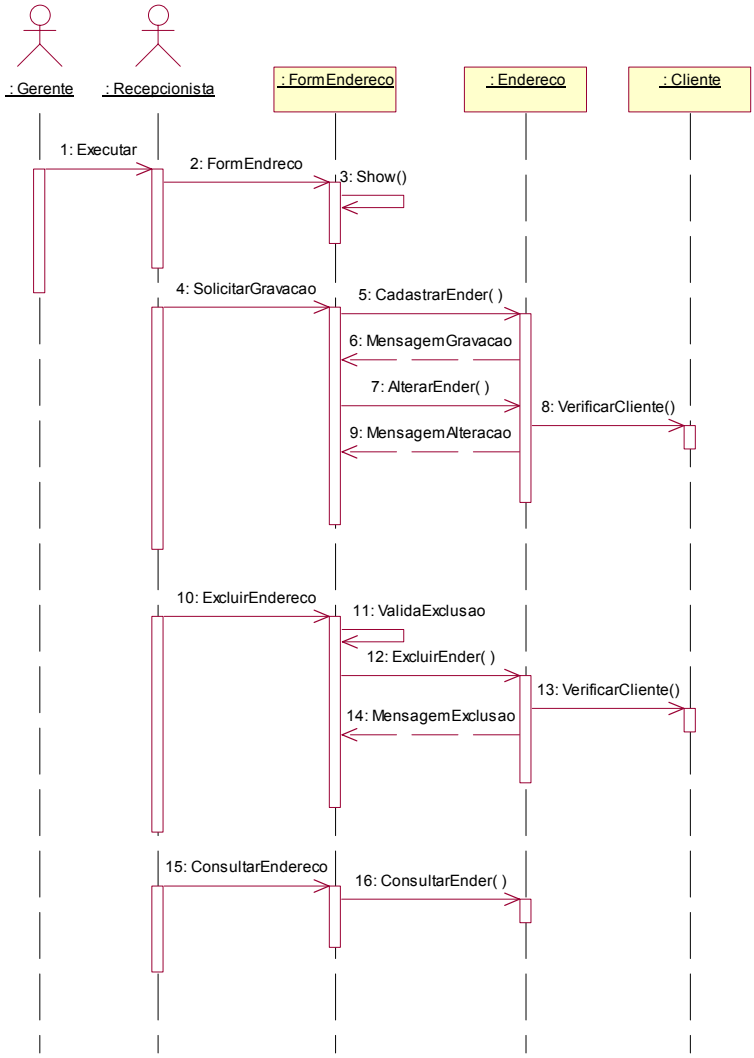


Figura 3.21 – Diagrama de Seqüência de Cadastrar Endereços.

3.2.21 Diagrama de Colaboração de Cadastrar Endereço (Endereço: Classe 10000)

Na Figura 3.22, é apresentado o Diagrama de Colaboração de Cadastrar Endereços, cuja ênfase está na organização estrutural dos objetos que enviam e recebem mensagens, mostrando a relação dos objetos através dos métodos.

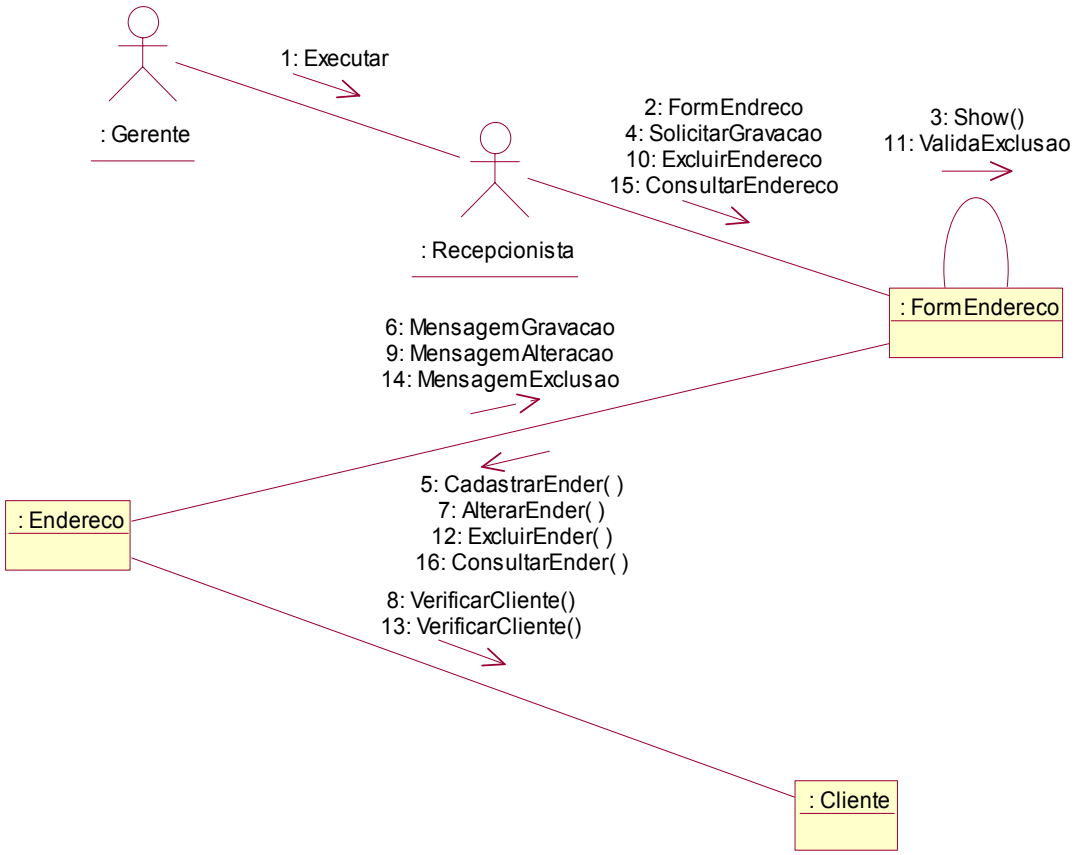


Figura 3.22 – Diagrama de Colaboração de Cadastrar Endereços.

3.2.22 Diagrama de Classes do Sistema de Gerenciamento de Hotéis

O Diagrama de Classes do Sistema de Gerenciamento de Hotéis é destaque na Figura 3.23. Neste diagrama, são apresentadas as Classes, os Relacionamentos e as Multiplicidades entre as classes.

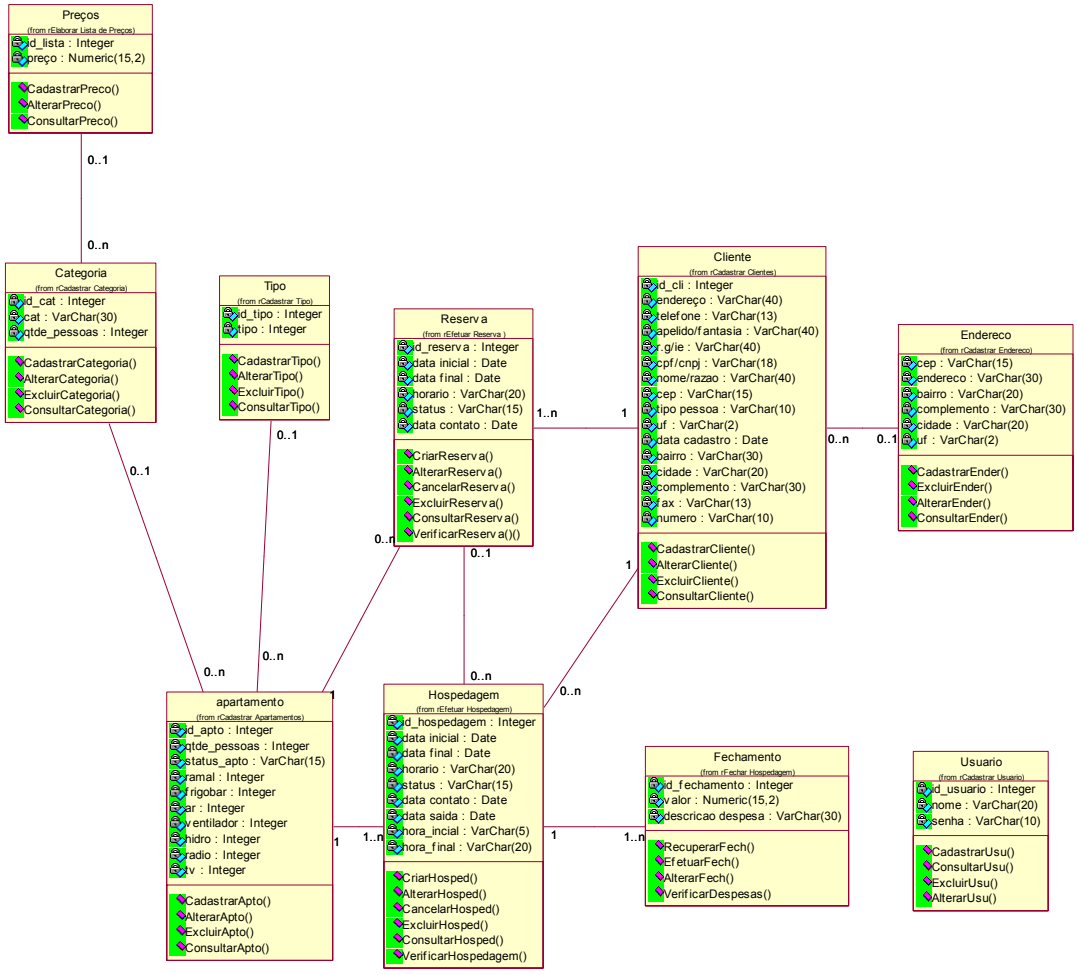


Figura 3.23 – Diagrama de Classes do Sistema de Gerenciamento de Hotéis.

3.3 Extraindo as informações dos Diagramas da UML

A Ferramenta FuTeBOOL tem como objetivo a geração de Elementos Requeridos Baseados em Diagramas da UML, para tanto possui funcionamento simples e organizado.

A seguir, são apresentados os passos para efetivar a extração desses Elementos Requeridos, criação de um *check-list* e ageração de estatísticas para futuras análises, por meio de Diagramas da UML, que foram desenvolvidos para o Sistema de Gerenciamento de Hotéis e Pousadas – SOFTHOTEL.

- Passo 1 – Cadastro de Sistema: Esta funcionalidade é necessária uma vez que podemos ter mais de um projeto sendo avaliado e testado ao mesmo tempo, para isso temos um cadastro de sistema. Quando a ferramenta é executada, o primeiro passo é escolher o projeto no qual se queira trabalhar, se for um projeto já existente basta selecioná-lo em uma lista; caso contrário, basta criá-lo, clicando no botão CADASTRAR, conforme mostrado na Figura 3.24. No cadastro de sistemas, é possível incluir (1), alterar (2), excluir (3) e consultar (4) sistemas, conforme mostrado na Figura 3.25. Sem um sistema previamente cadastrado não é possível usar a ferramenta. Após o cadastro de sistema, é possível verificar quais sistemas estão cadastrados, conforme pode ser verificado na Figura 3.26.

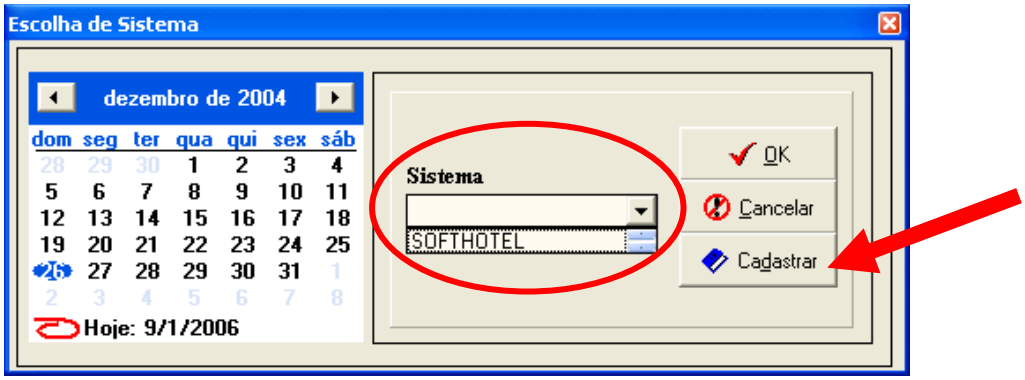


Figura 3.24 – Tela de Seleção de Sistemas

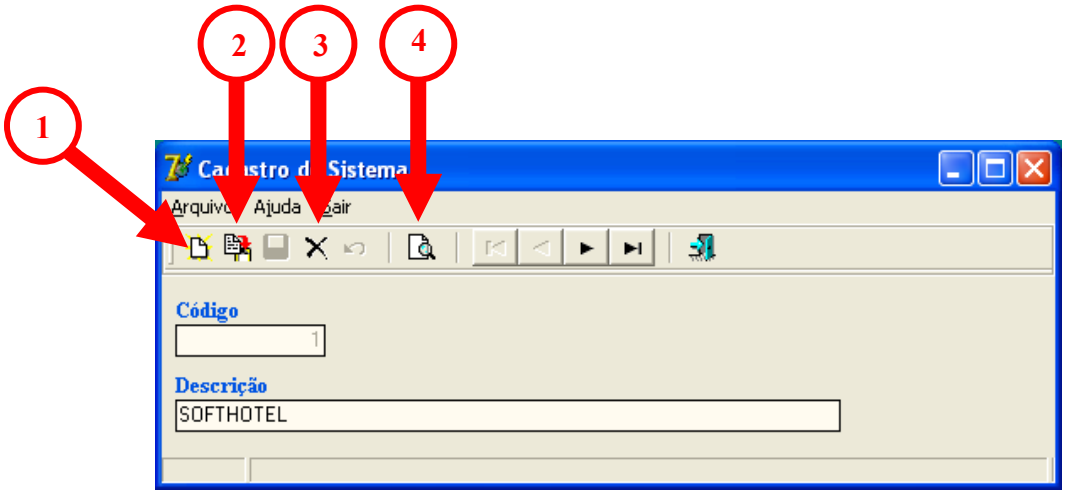


Figura 3.25 – Tela de Cadastro de Sistemas

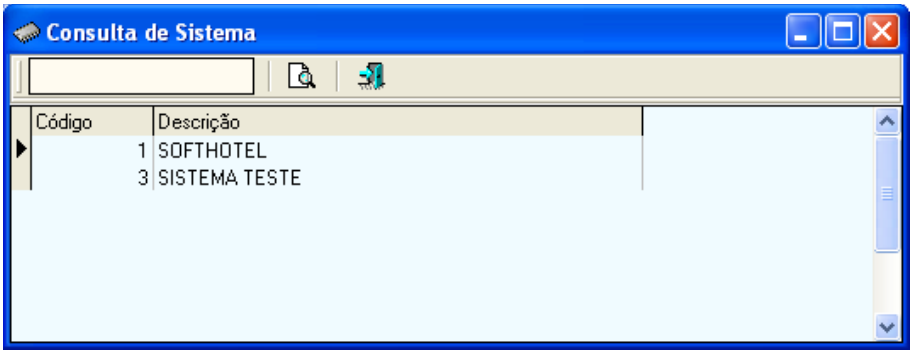


Figura 3.26 – Tela de Consulta de Sistemas

- Passo 2 – Cadastro de Classes: Esta funcionalidade é necessária para que as classes do sistema em questão possam ser cadastradas. No cadastro de classes, é possível incluir (1), alterar (2), excluir (3) e consultar (4) classes, conforme mostrado na Figura 3.27. Sem classes, previamente cadastradas, não é possível cadastrar métodos. O cadastro das classes é feito a partir do Diagrama de Classes do SOFTHOTEL, mostrado na Figura 3.23. Após o cadastro das classes, é possível verificar quais classes estão cadastradas, conforme pode ser verificado na Figura 3.28.

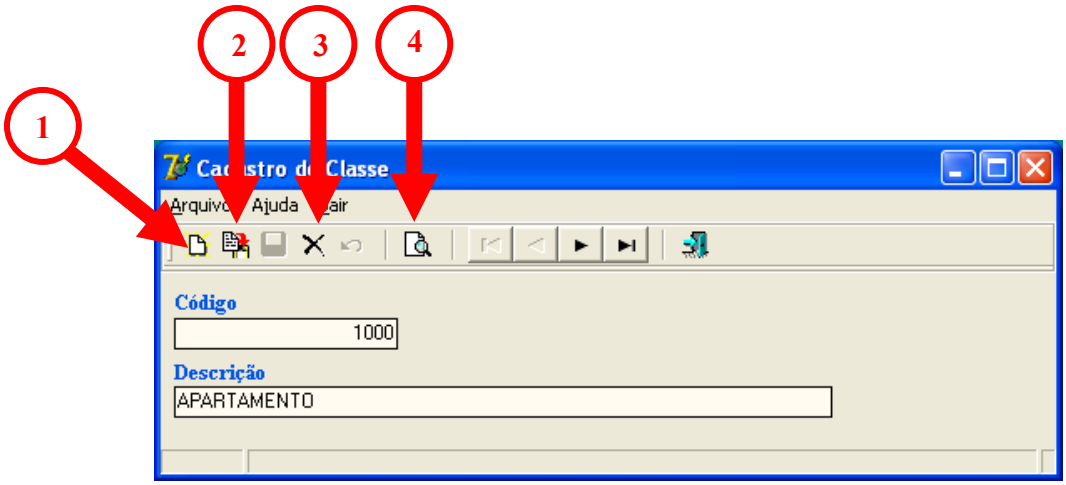


Figura 3.27 – Tela de Cadastro de Classes

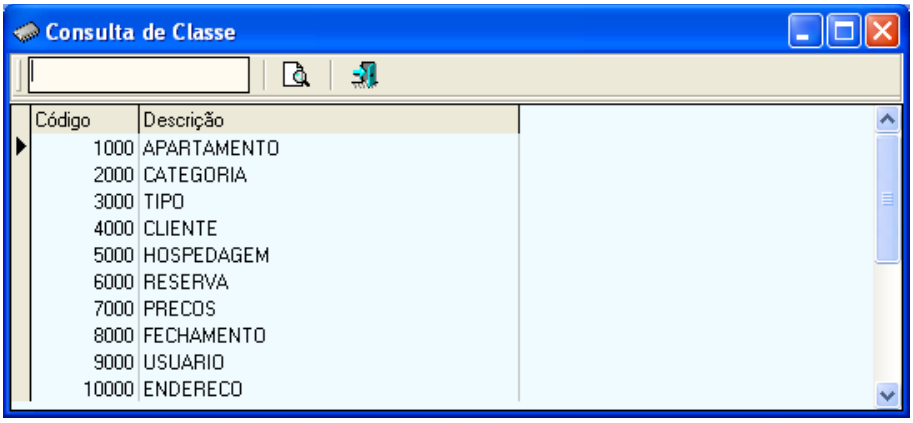


Figura 3.28 – Tela de Consulta de Classes

- Passo 3 – Cadastro de Métodos: Esta funcionalidade é necessária para que os métodos das classes, previamente cadastradas (1) possam ser cadastrados; no cadastro de métodos é possível incluir (2), alterar (3), excluir (4) e consultar (5) métodos, conforme mostrado na Figura 3.29. Sem as classes previamente cadastradas, não é possível cadastrar os métodos. O cadastro dos métodos é feita a partir do Diagrama de Classes do SOFTHOTEL, que é mostrado na Figura 3.23. Após o cadastro das classes, é possível verificar quais métodos estão cadastrados, conforme pode ser verificado na Figura 3.30. Neste cadastro, existe o campo Método, onde é colocado um nome simplificado para o método que está sendo cadastrado; no campo Descrição aparece a descrição do método em questão; e no campo

Número é colocado o número do método. Esse número é composto da seguinte forma: o primeiro número mostra a unidade/dezena de milhar da Classe cadastrada e os três últimos números mostram o número do método que é gerado pela Rational Rose (conforme mostrado na Figura 3.31). Exemplo Método 10007: Classe 10000 (Endereço) e Método 007 (Alterar Endereço).

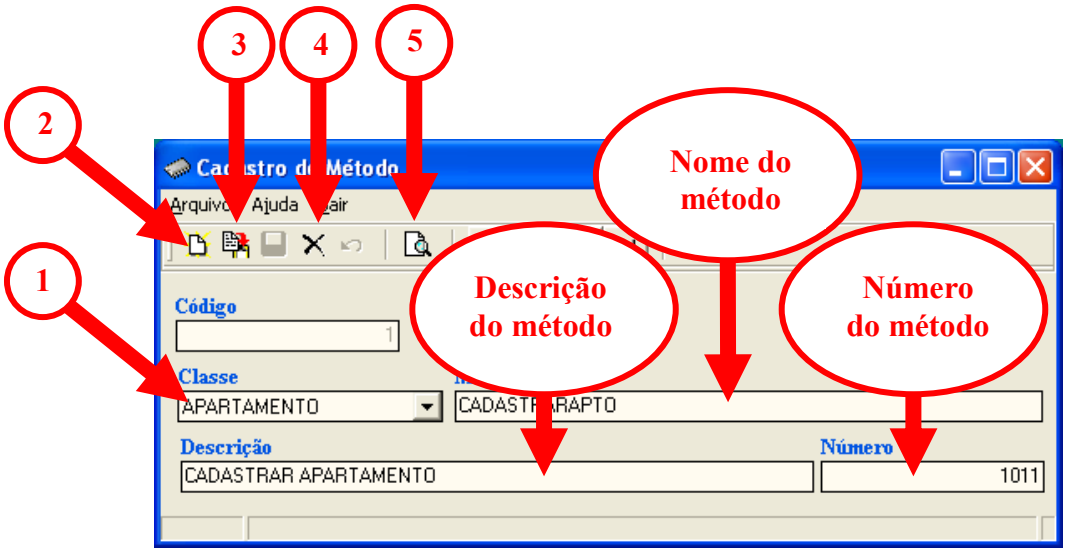
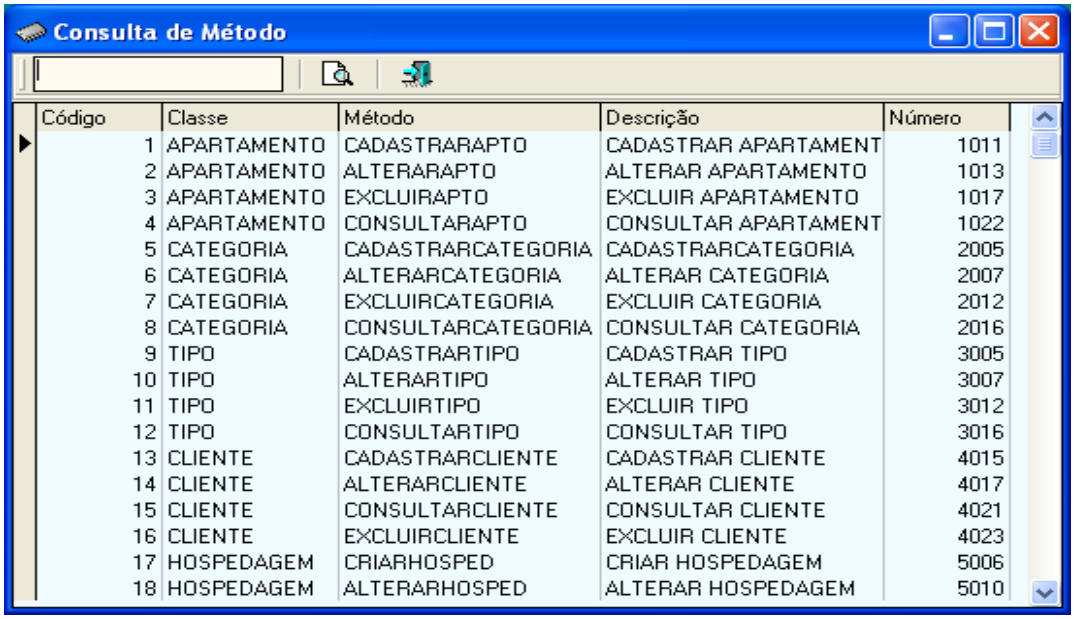


Figura 3.29 – Tela de Cadastro de Métodos



Código	Classe	Método	Descrição	Número
1	APARTAMENTO	CADASTRARAPTO	CADASTRAR APARTAMENT	1011
2	APARTAMENTO	ALTERARAPTO	ALTERAR APARTAMENTO	1013
3	APARTAMENTO	EXCLUIRAPTO	EXCLUIR APARTAMENTO	1017
4	APARTAMENTO	CONSULTARAPTO	CONSULTAR APARTAMENT	1022
5	CATEGORIA	CADASTRARCATEGORIA	CADASTRARCATEGORIA	2005
6	CATEGORIA	ALTERARCATEGORIA	ALTERAR CATEGORIA	2007
7	CATEGORIA	EXCLUIRCATEGORIA	EXCLUIR CATEGORIA	2012
8	CATEGORIA	CONSULTARCATEGORIA	CONSULTAR CATEGORIA	2016
9	TIPO	CADASTRARTIPO	CADASTRAR TIPO	3005
10	TIPO	ALTERARTIPO	ALTERAR TIPO	3007
11	TIPO	EXCLUIRTIPO	EXCLUIR TIPO	3012
12	TIPO	CONSULTARTIPO	CONSULTAR TIPO	3016
13	CLIENTE	CADASTRARCLIENTE	CADASTRAR CLIENTE	4015
14	CLIENTE	ALTERARCLIENTE	ALTERAR CLIENTE	4017
15	CLIENTE	CONSULTARCLIENTE	CONSULTAR CLIENTE	4021
16	CLIENTE	EXCLUIRCLIENTE	EXCLUIR CLIENTE	4023
17	HOSPEDAGEM	CRIARHOSPED	CRIAR HOSPEDAGEM	5006
18	HOSPEDAGEM	ALTERARHOSPED	ALTERAR HOSPEDAGEM	5010

Figura 3.30 – Tela de Consulta de Métodos

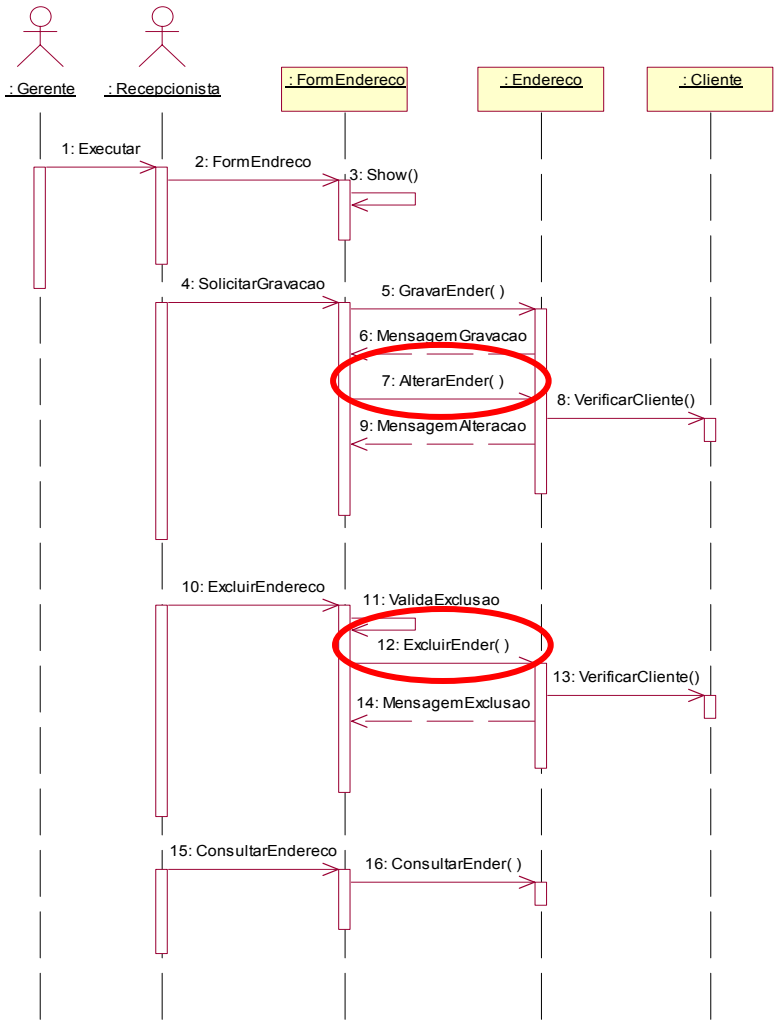


Figura 3.31 – Número de método gerado pela Rational Rose.

- Passo 4 – Cadastro de Sentido: Esta funcionalidade é necessária para que os sentidos das associações entre os métodos de cada classe possam ser cadastrados, com isso é possível saber de que forma um método depende do outro, se é Verificação, Informação, Execução, Execução-Informação ou Verificação-Informação. No cadastro de sentidos, é possível incluir (1), alterar (2), excluir (3) e consultar (4) sentidos, conforme mostrado na Figura 3.32. O cadastro do sentido é feito a partir dos Diagramas de Seqüência. Após o cadastro dos sentidos, é possível verificar quais cadastrados, conforme pode ser verificado na Figura 3.33.

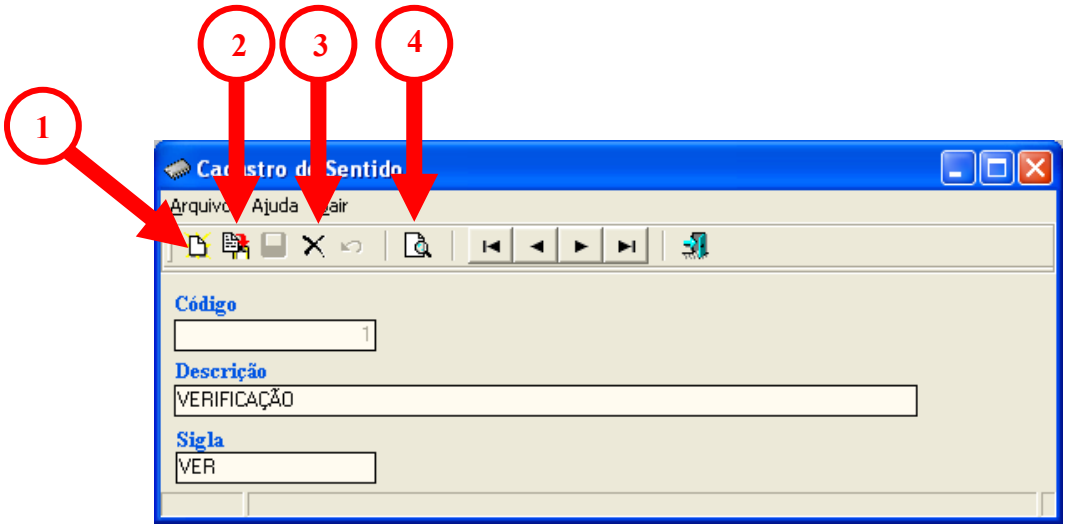


Figura 3.32 – Tela de Cadastro de Sentidos

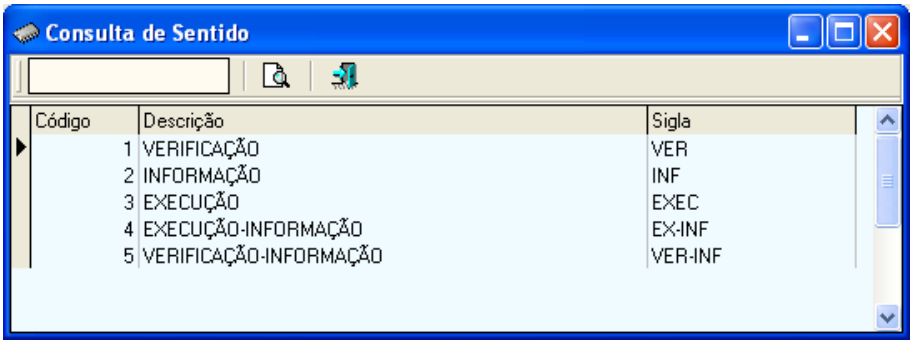


Figura 3.33 – Tela de Consulta de Sentidos

- Passo 5 – Cadastro de Elementos Requeridos: Dentro do cadastro de Elementos Requeridos é possível incluir (1), alterar (2) e excluir (3) Elementos Requeridos, conforme mostrado na Figura 3.34. Na Figura 3.35 é apresentada a interface da Ferramenta, na qual é possível gerar os Elementos Requeridos de Teste Funcional por meio das classes. Quando selecionada uma determinada classe (1), aparecem somente os métodos da mesma (2); o mesmo acontece com a classe dependência (3) e seus métodos (4). O campo sentido também é selecionado de uma lista pré-existente (5). Existe um campo observação (6), onde é descrito o Elemento Requerido para que o mesmo seja auto-explicativo. Existe o campo Resultado Esperado(7) onde o usuário informa o que se espera do elemento requerido. Também existem

os campos Pré e Pós–condições (8), que definem as condições de ocorrência de cada elemento requerido. A extração das informações necessárias para a criação dos Elementos Requeridos é feita por meio da análise dos Diagramas de Seqüência. Após carregar as informações na ferramenta, ela gera os elementos requeridos com base nos dados catalogados e nas ações válidas (marcadas em azul) e ações inválidas,conforme mostrado na Figura 3.36. Também é possível exportar todos os Elementos Requeridos em um arquivo com extensão “txt”, para isso basta clicar no botão exportar, conforme é mostrado na Figura 3.34. É possível ainda, imprimir todos os Elementos Requeridos criados, para tanto basta clicar no botão IMPRIMIR, conforme é mostrado na Figura 3.34.

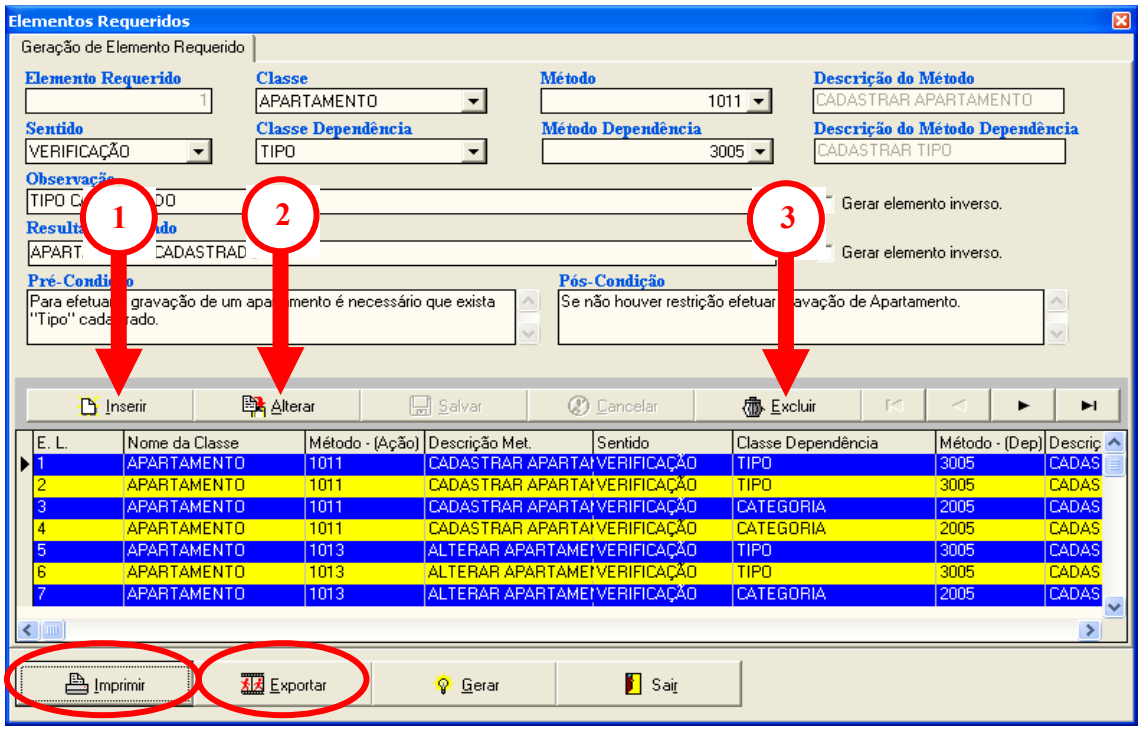


Figura 3.34 – Tela de Cadastro de Elementos Requeridos.

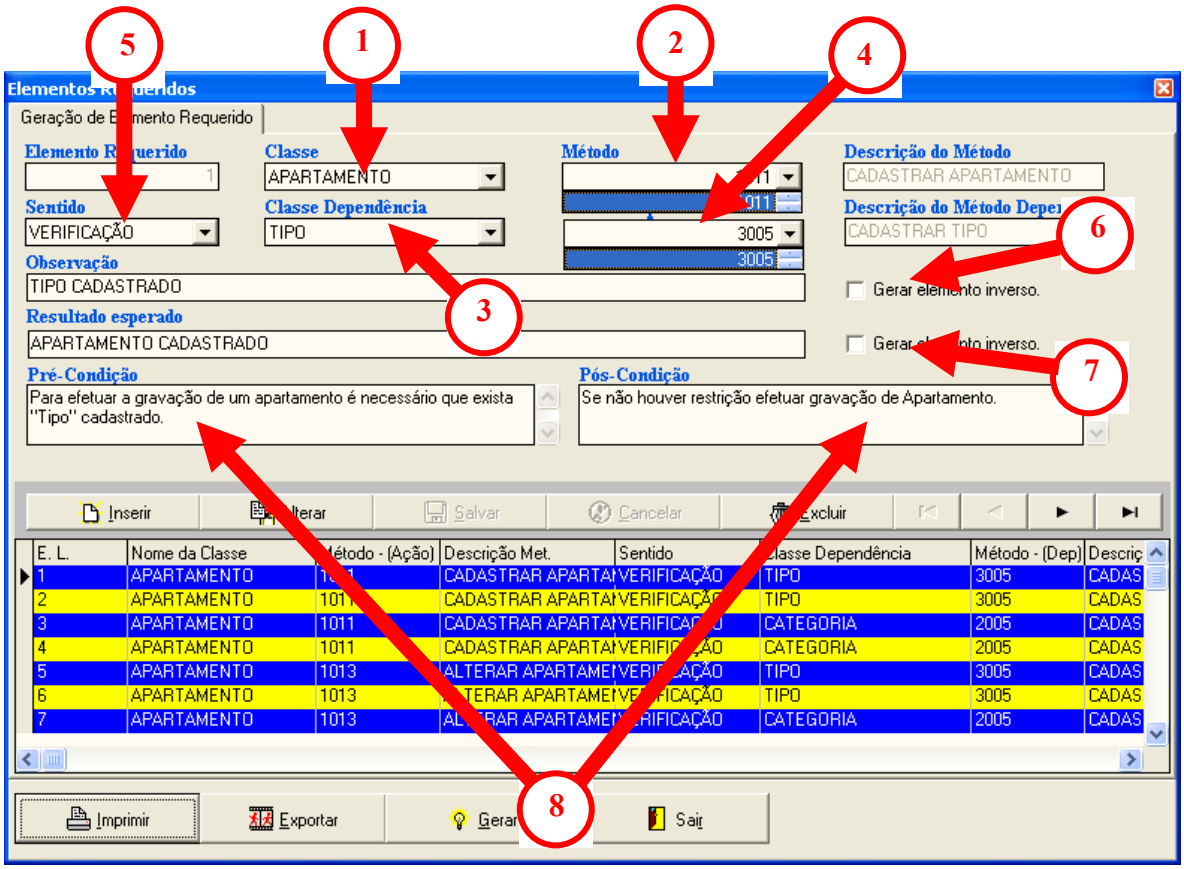


Figura 3.35 – Tela da Ferramenta FuTeBOOL.

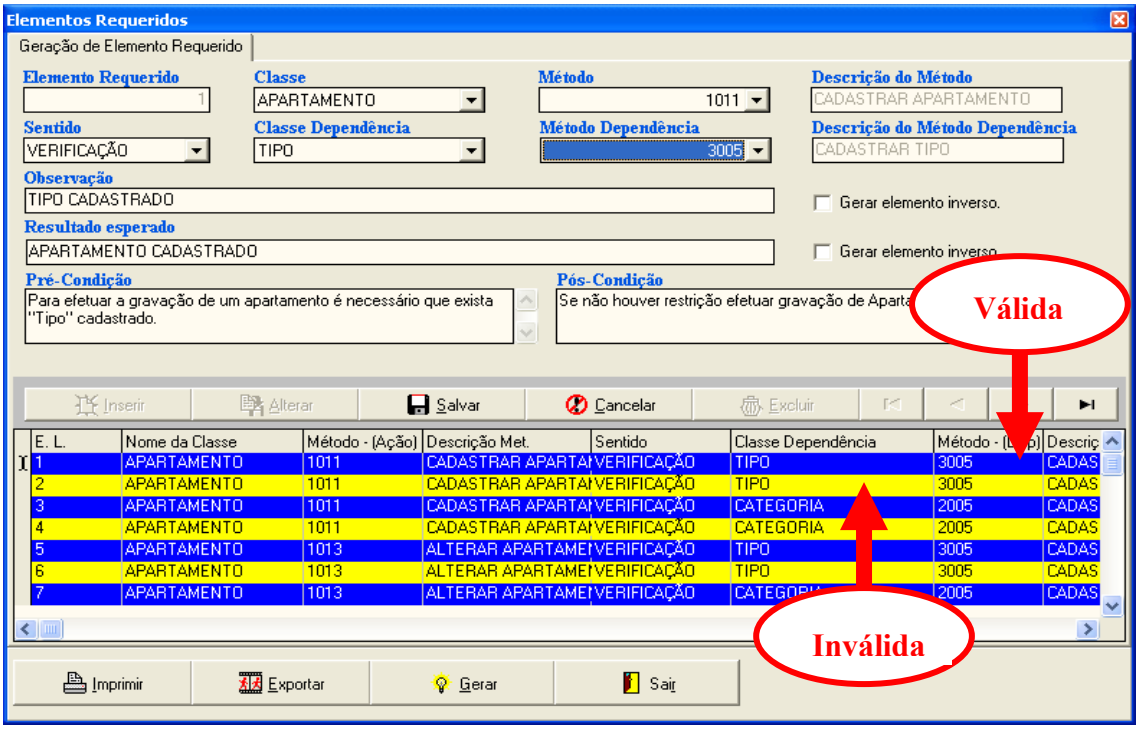


Figura 3.36 – Elementos Requeridos Criados.

- Passo 6 – Geração de *Checklist*: Depois que todos os Elementos Requeridos são criados, basta clicar em GERAR (conforme mostrado na Figura 3.37) para que seja gerada uma lista com todos os Elementos Requeridos criados, possibilitando a marcação dos que foram exercitados, por meio de duplo *click* em cada elemento requerido. Enquanto é efetuada a marcação dos Elementos Requeridos Exercitados, é indicado se um elemento requerido foi ou não satisfeito; e a porcentagem da cobertura de teste é atualizada automaticamente, conforme mostrado na Figura 3.37.

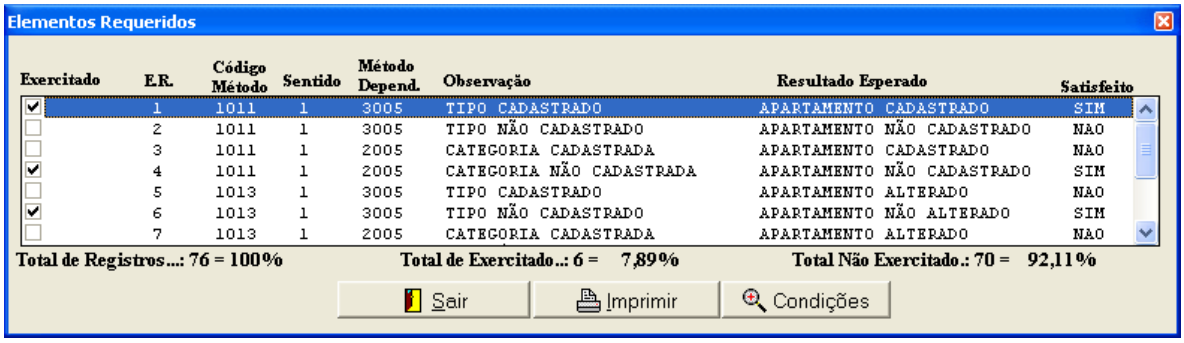


Figura 3.37 – Lista de Elementos Requeridos e Estatística.

3.4 Análises Finais

Com o estudo de caso é possível ter uma visão detalhada de todas as especificações do software, incluindo seus requisitos funcionais. Com os diagramas apresentados neste capítulo foi possível mostrar o uso da Ferramenta FuTeBOOL que auxilia na geração e documentação dos Elementos Requeridos de teste funcional e acompanhar a execução do programa, a partir da geração dos casos de testes procedendo a uma avaliação de resultados obtidos e esperados em cada elemento requerido gerado.

A partir dos resultados esperados e obtidos, é possível assim obter uma avaliação de

cobertura dos elementos requeridos exercitados. Essa técnica de teste funcional baseado em diagramas da UML (Casos de Uso, Seqüência ou Colaboração e Classes), abre um estudo de aplicação de novos elementos de teste com base nas funcionalidades e relacionamentos entre as principais classes sem a necessidade de conhecer o código do programa. Tais conceitos podem ser expandidos para camadas mais internas das funcionalidades, bem como regras do negócio que poderiam estar envolvidas, porém essas características são mais detalhadas e em uma mesma funcionalidade pode existir mais que uma ação distinta deste tipo e resultar numa explosão ainda maior dos elementos requeridos.

A idéia, porém, deste trabalho é exercitar apenas as dependências funcionais, integração entre as classes e outros tipos de ações isoladas, mas existentes nos diagramas de Seqüência e de Classe.

4 ELEMENTOS FUNCIONAIS REQUERIDOS BASEADOS EM DIAGRAMAS DA UML

Neste capítulo, são apresentados os resultados obtidos com o Estudo da Técnica de Teste Funcional Baseado em Diagramas da UML, com o uso da Ferramenta FuTeBOOL – Tool.

O projeto escolhido para o teste foi o Sistema de Gerenciamento de Hotéis – SOFTHOTEL, desenvolvido utilizando como ambiente de desenvolvimento o Delphi 7 e como banco de dados o Interbase 6.5, conforme visto no Capítulo 3.

4.1 Teste Funcional Baseado em Diagramas da UML.

Com o estudo desenvolvido nesta dissertação, foi possível chegar aos Elementos Requeridos para teste Funcional baseados em Diagramas da UML. Os números descritos como Elementos Requeridos Satisfeitos são os números fornecidos pela Ferramenta FuTeBOOL (E.R.) conforme pode ser visto na Figura 4.1. A seguir, estão listados alguns casos de teste utilizados para este trabalho:

- **Caso de Teste 1: Criação de hospedagem**

  - Avaliação: Para efetuar uma reserva, é necessário verificar se existe Reserva para um determinado apartamento para o mesmo período.

  - Resultado da Avaliação: Não é possível a criação de uma hospedagem se houver uma reserva para um determinado apartamento no período que se pretende efetuar esta hospedagem.

- Elementos Requeridos satisfeitos: 37 e 38.

- **Caso de Teste 2: Cancelar hospedagem**

  - Avaliação: Para cancelar uma Hospedagem, é necessário verificar se há Pendências em fechamento.

  - Resultado da Avaliação: Se houver despesas para uma hospedagem, não é possível cancelá-la até que essas despesas sejam pagas.

  - Elementos Requeridos satisfeitos: 45 e 46.

- **Caso de Teste 3: Excluir hospedagem**

  - Avaliação: Para excluir uma Hospedagem, é necessário verificar se há Pendências em fechamento.

  - Resultado da Avaliação: Se houver despesas para uma hospedagem, não é possível excluí-la até que essas despesas sejam pagas.

  - Elementos Requeridos satisfeitos: 47 e 48.

- **Caso de Teste 4: Criar reserva**

  - Avaliação: Para efetuar uma Reserva, é necessário verificar se existe reserva para um determinado Apartamento para o mesmo período.

  - Resultado da Avaliação: Se o apartamento em questão estiver reservado para o mesmo período, não é possível efetuar a reserva.

  - Elementos Requeridos satisfeitos: 49 e 50.

- **Caso de Teste 5: Alterar Preço**

  - Avaliação: Para alterar Lista de Preços é necessário verificar se existe Hospedagem para algum quarto da categoria em questão..

  - Resultado da Avaliação: Só é possível alterar Lista de Preços se não existir Hospedagem para algum quarto da categoria em questão

  - Elementos Requeridos satisfeitos: 59 e 60.

- **Caso de Teste 6: Gravar fechamento**

  - Avaliação: Para efetuar Fechamento de Hospedagem, é necessário verificar despesas extras.

  - Resultado da Avaliação: Só é possível gravar fechamento depois de verificar todas as despesas extras de determinada hospedagem.

  - Elementos Requeridos satisfeitos: 63, 64, 65 e 66.

- **Caso de Teste 7: Alterar fechamento**

  - Avaliação: Para alterar Fechamento de Hospedagem, é necessário verificar despesas extras.

  - Resultado da Avaliação: Só é possível alterar fechamento depois de verificar todas as despesas extras de determinada hospedagem.

  - Elementos Requeridos satisfeitos: 67, 68, 69 e 70.

- **Caso de Teste 8: Cadastrar cliente**

  - Avaliação: Para cadastrar um Cliente, é necessário que se tenha Endereço cadastrado.

  - Resultado da Avaliação: Se não houver endereço cadastrado, não é possível cadastrar cliente.

  - Elementos Requeridos satisfeitos: 71 e 72.

- **Caso de Teste 9: Alterar endereço**

  - Avaliação: Para alterar um Endereço, é necessário verificar se não existe Cliente cadastrado com o Endereço em questão.

  - Resultado da Avaliação: Se houver cliente cadastrado com o endereço em questão, não é possível alterá-lo.

  - Elementos Requeridos satisfeitos: 73 e 74.

- **Caso de Teste 10: Excluir endereço**

  - Avaliação: Para excluir um Endereço, é necessário verificar se não existe Cliente cadastrado com o Endereço em questão.

  - Resultado da Avaliação: Se houver cliente cadastrado com o endereço em questão, não é possível excluí-lo.

  - Elementos Requeridos satisfeitos: 75 e 76.

4.2 Resultados gerados pela FuTeBOOL

A Ferramenta FuTeBOOL gera uma listagem dos Elementos Requeridos para o teste Funcional Baseado em Diagramas da UML apresentando os mesmos em uma interface que possibilita a marcação dos elementos que foram exercitados para o teste.

Na Figura 4.1, é apresentada essa interface, mostrando também a estatística gerada pela ferramenta, o que possibilita uma análise mais precisa dos resultados na etapa de teste, mais especificamente no Teste Funcional.

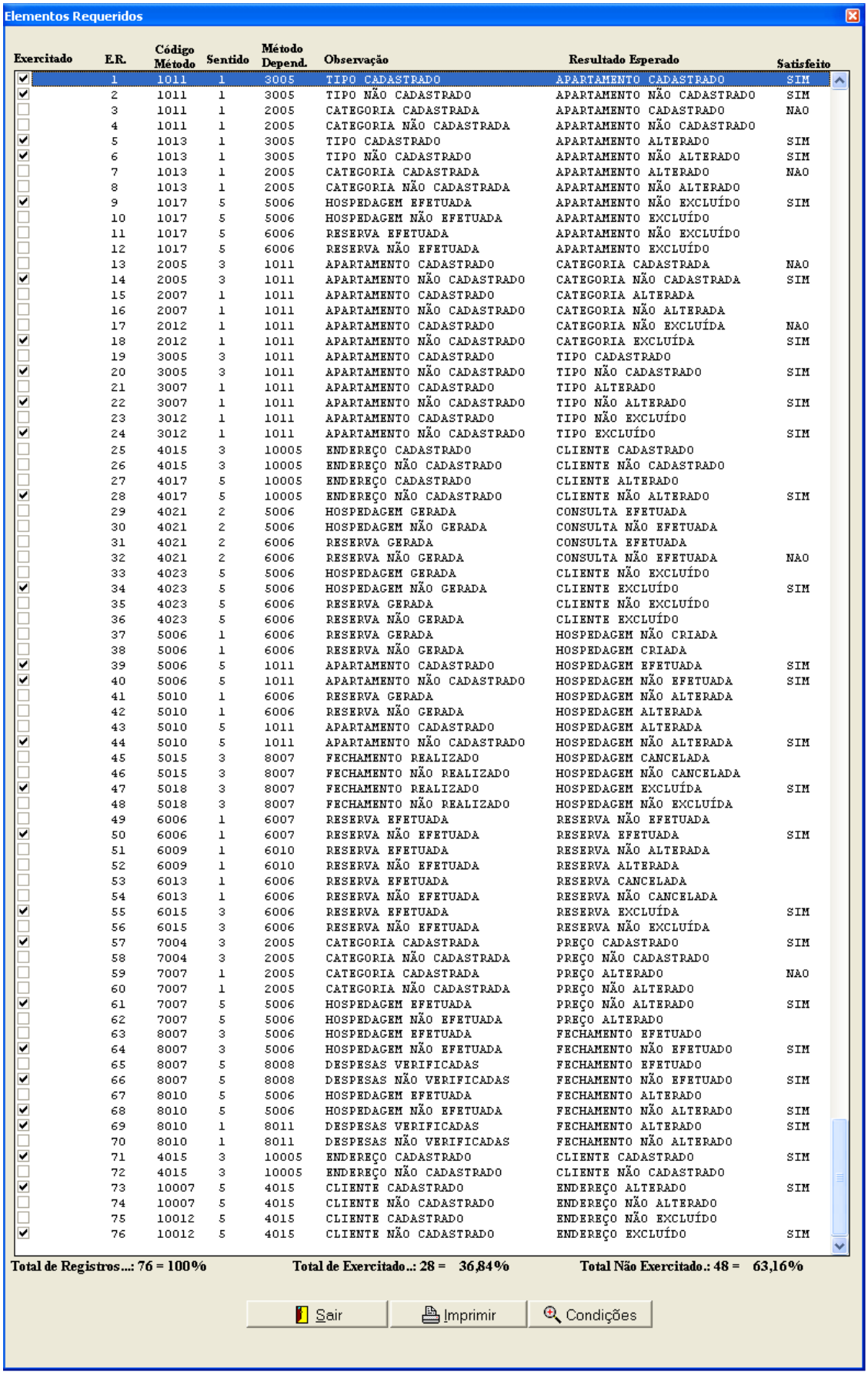


Figura 4.1 – Estatística gerada pela Ferramenta FuTeBOOL.

4.3 Análise dos Resultados

A maior vantagem obtida no Teste Funcional Baseado em Diagramas da UML com o uso da Ferramenta FuTeBOOL é que a mesma apóia a etapa de teste, uma vez que após feita análise do Diagrama de Classes, é possível identificar todas as classe e respectivos métodos do projeto. Outra vantagem é a possibilidade de poder analisar os Diagramas de Seqüência e Colaboração com a finalidade de identificar todas as dependências existentes entre as classes e métodos deste projeto.

A ferramenta auxilia na elaboração dos Elementos Requeridos para o teste e possibilita a geração de um *check-list* onde são marcados os elementos que foram exercitados. Ao final deste processo, é possível gerar as estatísticas do teste em questão, conforme mostrado na Figura 4.1. Nesta estatística, é mostrado o Total de Elementos Requeridos, Total de Elementos Requeridos Exercitados e Total de Elementos Requeridos Não – Exercitados.

A grande dificuldade encontrada nesse estudo foi o fato de a Ferramenta Rational Rose 2002 não possibilitar a exportação das classes, métodos e dependências entre esses métodos. Sendo assim, existe uma etapa manual que é a alimentação do FuTeBOOL – Tool, o que acaba tornando o teste um pouco mais demorado, porém atinge as expectativas do testador no que diz respeito à técnica apresentada que é um Teste Funcional Baseado em Diagramas da UML.

4.4 Considerações Finais

Os estudos apresentados nesta dissertação vêm colaborar com a técnica de teste

funcional tradicional, em que se pode, por meio dos Diagramas da UML, fundamentar de forma mais contundente a etapa de teste.

Com a geração de uma ferramenta de apoio ao teste funcional, pode-se realizar um teste mais aprimorado, uma vez que os Elementos Requeridos são gerados a partir de uma análise cuidadosa e rigorosa das dependências existentes entre as diferentes classes e métodos de um projeto.

Isso faz com que a etapa de teste, fundamentada na técnica proposta nesta dissertação, seja mais objetiva, mostrando de forma muito clara o que deve ser testado e podendo marcar o elemento que foi exercitado ou não, obtendo assim uma visão estratégica com relação à técnica de teste funcional tradicional.

CONCLUSÃO

A proposta deste trabalho é a criação de elementos requeridos de teste que auxiliarão na derivação dos dados de teste, já que essa é a atividade desempenhada por um critério de teste, não é necessário ter o dado para gerar, mas um ponto de análise para que o testador consiga derivar o dado e, em seguida, verificar se o que o dado escolhido exercitou, alcançou ou não o objetivo do critério. O teste de software orientado a objetos apresenta algumas características novas, mas muitas técnicas convencionais ainda são apropriadas.

Por exemplo, o teste funcional do software orientado a objetos não é diferente do teste funcional do software convencional, casos de teste são desenvolvidos baseados em requisitos funcionais que são descritos na especificação.

Ao se desenvolver um trabalho dessa natureza, pode-se chegar à conclusão de que quanto mais cedo se inicia com o teste, mais cedo são diagnosticados os problemas e, com certeza, o produto final do desenvolvimento de software terá mais indicadores de qualidade com menos custos e menor esforço.

O desenvolvimento deste trabalho possibilitou um resultado de grande importância, uma vez que foi possível apresentar novos critérios para Teste Funcional baseados em Diagramas da UML e, com a execução desse tipo de teste, pode-se atestar maior qualidade ao software desenvolvido.

Com a geração da Ferramenta FuTeBOOL – Tool, foi possível tornar a etapa de teste mais segura e mais eficiente pois por meio dela é possível gerar estatísticas que colaboram de maneira quantitativa com a etapa de teste e não mais apenas qualitativa.

Trabalhos desenvolvidos anteriormente também tinham a intenção de desenvolver o teste funcional baseado em alguns diagramas da UML, como, por exemplo, o trabalho que efetua o teste funcional baseado em Casos de Usos, desenvolvido por Chaim *et al* (2003).

Porém tais trabalhos tinham o foco no teste de variáveis de domínio.

O Teste Funcional Baseado em Diagramas da UML, além de desenvolver teste baseado em variáveis de domínio, também se baseia em Dependências Funcionais, podendo se testar uma ou mais classes, possibilitando o teste intra-classe (métodos que agem numa mesma classe) e inter-classe (métodos que interagem em duas ou mais classes). Neste trabalho não foram separados os tipos de testes que atendam os níveis intra-classe dos testes inter-classes, uma vez que o foco é testar as ações do programa baseado nos diagramas da UML.

Uma grande contribuição deste trabalho é a possibilidade de geração dos Elementos Requeridos para teste de maneira semi-automática, pois, depois de alimentar a ferramenta FuTeBOOL com as informações necessárias, é possível gerar tais elementos, o que facilita a atividade de teste funcional e isso poderá ser abordado em trabalhos futuros.

A FuTeBOOL auxilia o testador na geração dos Elementos Requeridos baseando-se nas ações válidas e inválidas e ao final, após verificar quais Elementos Requeridos foram exercitados e quais não foram exercitados, a ferramenta apresenta uma porcentagem de Elementos Requeridos cobertos pelo teste.

Portanto, sem esta ferramenta pouco se teria evoluído na técnica de Teste Funcional, pois além da criação desta nova técnica denominada de Teste Funcional Baseado em Diagramas da UML, foi criada a Ferramenta FuTeBOOL que agrega valores no que diz respeito à automatização desta técnica.

Trabalhos Futuros

Como trabalhos futuros, pretende-se melhorar as atividades de geração de elementos

de testes da ferramenta proposta neste trabalho, de forma a complementar os estudos iniciados na FuTeBOOL com linguagens puramente OO, como é o caso de JAVA. Para isso a Ferramenta FuTeBOOL terá de ser novamente desenvolvida em uma linguagem que possibilite maior mobilidade à ferramenta como é o caso de JAVA ou C. Também será analisada a possibilidade de uso da ferramenta de modelagem Argo UML que poderá fazer a exportação automática de informações para a Ferramenta FuTeBOOL, numa versão mais automatizada, uma vez que a mesma possui repositório em SQL e usa formatos baseados em XML.

Outras modificações na geração dos elementos requeridos podem ser ampliados, considerando as ações existentes em cada funcionalidade, em específico aplicações de regras de negócio ou ações necessárias para validar uma dada tarefa das funcionalidades e não deixar apenas uma ação isolada como é apresentado no diagrama de Seqüência ou Colaboração.

O Teste Funcional baseado em diagramas da UML pode ser ampliado futuramente visando exercitar elementos funcionais sob o aspecto da OO (polimorfismo, heranças, sobrecarga, etc) entre outros aspectos que norteiam as técnicas funcionais da OO.

REFERÊNCIAS

BEIZER, B. **Software testing techniques**. 2nd ed. New York: Van Nostrand Reinhold Company, 1990.

BINDER, Robert V. **Testing object-oriented systems:** models, patterns and tools. Addison-Wesley, 2000.

BINDER, Robert. V. **Testing Object-Oriented Systems, Models, Patterns and Tools**. Addison-Wesley, 1999.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML – Guia do Usuário**. Rio de Janeiro: Campus, 2000.

CHAIM, Marcos Lordello; CARNIELLO, Adriana; Jino, Mário. **Teste baseado em casos de uso.** Campinas: Embrapa Informática Agropecuária, 2003.

DATE, C. J. **Introdução a Sistemas de Banco de Dados**. 7. ed. Rio de Janeiro: Campus LTDA, 2000.

HARROLD, M. J.; ROTHERMEL, G. **Performing data flow testing on claases**. In: Second ACM SIGSOFT Symposium on Foundations os Software Engineering, New York: ACM Press, 1994, p. 154-163.

HEUSER, Carlos Alberto. **Projeto de Banco de Dados.** 5 ed. Porto Alegre: Instituto de Informática da UFRGS: Editora Sagra Luzzatto, 2004.

HOFFMAN, D.; STROOPER, P. **A methodology and framework for automated class testing**. Software Practice and Experience, 1997.

HOWDEN, W. E. **Software Engineering and Technology: Functional Program Testing**

and Analysis. New York: McGrall-Hill Book Co, 1987.

JACOBSON, Ivar. **Object Oriented Software Engineering**, Addison-Wesley, 1992.

LARMAN, Craig. **Utilizando UML e Padrões**. 2 ed. Porto Alegre: Bookman, 2004.

MALDONADO, J. C. **Critérios de Teste de Software: Aspectos teóricos, empíricos e de automatização.** Concurso de Livre-Docência – ICMC-USP, 1997.

MALDONADO, J. C. **Critérios Potenciais Usos: Uma contribuição ao teste estrutural de software.** Tese de Doutoramento, DCA/FEE/UNICAMP, Campinas, SP, 1991

MALDONADO, J. C.; Vincenzi, A. M. R.; Barbosa, E. F.; Souza, S. R. S.; Delamaro, M. E. **Aspectos teóricos e empíricos de teste de cobertura de software**. Relatório Técnico 31, Instituto de Ciências Matemática e de Computação – ICMC-USP, 1998.

MYERS, J. G. **The art of Software Testing**. Wiley, New York, 1979.

PRESSMAN, Roger. **Software engineering – a practitioner's approach**. 5 ed. McGraw-Hill, 2000.

PRESSMAN, Roger. **Engenharia de Software**. 5 ed. Rio de Janeiro: McGraw-Hill, 2002.

PRESSMAN, Roger. **Engenharia de Software**. 6 ed. São Paulo: McGraw-Hill, 2006.

ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, K. C. **Qualidade de Software – Teoria e Prática**. São Paulo: Prentice Hall, 2001.

SOUZA, Rogéria Cristiane Gratão de. **Características de Testabilidade nos Diagramas UML (Unified Modeling Language): Apoio aos testes de sistemas de software orientados a objetos.** São Paulo, 2003.

VINCENZI, A. M. R. **Subsídios para o estabelecimento de estratégias de teste baseadas na técnica de mutação**. Dissertação de Mestrado, ICMC-USP, São Carlos – SP, 1998.

ANEXO A – Especificação de Caso de Uso

Nesta seção, apresenta-se a Especificação de cada Caso de Uso do Sistema de Gerenciamento de Hotéis, de acordo com os seguintes tópicos:

- Breve Descrição;
- Fluxo de Eventos: Básico e Alternativo;
- Requisitos Especiais;
- Pré-condições;
- Pós-Condições.

Especificação do Caso de Uso Cadastrar Tipos

- Breve Descrição:

  Esse caso de uso retrata como seria o cadastro dos tipos de apartamento.

- Fluxo de Eventos:

  - Fluxo Básico:

  1. Usuário seleciona a opção para cadastrar um tipo:

  Este caso de uso inicia-se quando um usuário (gerente) deseja fazer um cadastro de tipo. Para o usuário fazer um cadastro de tipo de apartamento, é necessário que fiquem cadastrados no sistema alguns dados pertinentes ao cadastro de tipos. Para isso o usuário deverá escolher a opção ‘Tipos’ e, então, o sistema irá exibir a tela de Cadastro de Tipos.

2. O Usuário informa dado de identificação.

O Usuário deverá informar o dado de identificação do Tipo. Com este dado será feita a verificação se já existem cadastrados, na base de dados, os dados do referido tipo. Caso não exista, serão solicitados outros dados.

3. Usuário informa outros dados referentes ao tipo.

O usuário deverá informar a descrição deste tipo.

4. O Usuário clica no botão salvar.

Quando o usuário clicar no botão salvar o sistema deverá salvar os dados informados do tipo, na base de dados.

5. Usuário deseja definir outro tipo.

Se o usuário desejar definir outro tipo, o caso de uso retorna para o passo 2.

6. Usuário finaliza cadastros.

Se o usuário desejar finalizar o cadastro de tipo, deverá clicar no botão sair. Este caso de uso é finalizado aqui.

- Fluxos Alternativos:

A1. Sistema verifica dados do tipo.

No passo 2, é verificado se existe um tipo cadastrado na base de dados com a mesma identificação informada; caso já exista, o sistema irá exibir todos os dados do tipo previamente cadastrados na tela. Caso o usuário desejar informar uma nova identificação, o caso de uso continua no início do passo 2.

- Requisitos Especiais:

  O sistema irá garantir a não duplicidade dos tipos.

- Pré-Condições:

  Para a realização deste caso de uso, o caso de uso Cadastrar Tipos deverá estar definido.

- Pós-Condições:

  Após a definição desse caso de uso, será possível cadastrar a lista de preço.

Especificação do Caso de Uso Cadastrar Categorias

- Breve Descrição:

  Esse caso de uso retrata como seria o cadastro das categorias de apartamento.

- Fluxo de Eventos:

  - Fluxo Básico:

  1. Usuário seleciona a opção para cadastrar uma categoria.

  Este caso de uso inicia-se quando um usuário (gerente) deseja fazer um cadastro de categoria. Para o usuário fazer uma cadastro de categoria de apartamento, é necessário que fiquem cadastrados no sistema alguns dados pertinentes ao cadastro de categorias. Para isso, o usuário deverá escolher a opção 'Categorias' e, então, o sistema irá exibir a tela de Cadastro de Categorias.

  2. O Usuário informa dado de identificação.

  O Usuário deverá informar o dado de identificação da categoria, com este dado, será feita a verificação se já existem cadastrados os dados da referida categoria, na base de dados. Caso não exista, serão solicitados outros dados.

3. Usuário informa outros dados referente à categoria.

O usuário deverá informar alguns dados pertinentes ao cadastro de categoria, como por exemplo: descrição e quantidade de pessoas.

4. O Usuário clica no botão salvar.

Quando o usuário clicar no botão salvar o sistema deverá salvar os dados informados da categoria, na base de dados.

5. Usuário deseja definir outra categoria.

Se o usuário desejar definir outra categoria, o caso de uso retorna para o passo 2.

6. Usuário finaliza cadastros.

Se o usuário desejar finalizar o cadastro de categoria, deverá clicar no botão sair. Este caso de uso é finalizado aqui.

- Fluxos Alternativos:

A1. Sistema verifica dados da categoria.

No passo 2, é verificado se existe uma categoria cadastrada na base de dados com a mesma identificação informada; caso já exista, o sistema irá exibir todos os dados da categoria previamente cadastrados na tela. Caso o usuário desejar informar uma nova identificação, o caso de uso continua no início do passo 2.

- Requisitos Especiais

  O sistema irá garantir a não duplicidade das categorias.

- Pré-Condições

  Para a realização deste caso de uso, o caso de uso Cadastrar Categorias deverá estar definido.

- Pós-Condições

  Após a definição desse caso de uso, será possível cadastrar a lista de preço.

Especificação do Caso de Uso Elaborar Lista de Preços

- Breve Descrição:

  Esse caso de uso retrata como seria a elaboração de lista de preço.

- Fluxo de Eventos:

  - Fluxo Básico:

  1. Usuário seleciona a opção para elaborar uma lista de preço.

  Este caso de uso inicia-se quando um usuário (gerente) deseja fazer um cadastro de lista de preço. Para o usuário fazer uma cadastro, é necessário que fiquem cadastrados no sistema alguns dados pertinentes ao cadastro de lista de preço. Para isso, o usuário deverá escolher a opção 'Lista de Preços' e, então, o sistema irá exibir a tela de Cadastro de Lista de Preços.

  2. O Usuário informa dado de identificação.

  O Usuário deverá informar o dado de identificação do tipo do apartamento, com este dado, serão listadas todas as categorias ; se já tiver algum preço informado na categoria, será mostrado o dado conforme o cadastro. Caso não exista, será solicitado o preço.

  3. O Usuário clica no botão salvar.

  Quando o usuário clicar no botão salvar, o sistema deverá salvar os dados

informados da lista de preço, na base de dados.

4. Usuário deseja definir outra lista.

Se o usuário desejar definir outra lista , o caso de uso retorna para o passo 2.

5. Usuário finaliza cadastros.

Se o usuário desejar finalizar o cadastro de lista de preço, deverá clicar no botão sair. Este caso de uso é finalizado aqui.

- Fluxos Alternativos:

A1. Sistema verifica dados da lista preço.

No passo 2, é verificado se existe um preço informado para a categoria. Caso o usuário desejar informar uma nova identificação, o caso de uso continua no início do passo 2.

- Requisitos Especiais

  O sistema irá garantir a não duplicidade das listas.

- Pré-Condições

  Para a realização deste caso de uso, o caso de uso Elaborar Lista de Preços deverá estar definido.

- Pós-Condições

  Após a definição desse caso de uso, será possível cadastrar o apartamento e realizar reservas e hospedagens.

Especificação do Caso de Uso Cadastrar Apartamentos

- Breve Descrição:

  Esse caso de uso retrata como seria o cadastro de apartamento.

- Fluxo de Eventos:

  - Fluxo Básico:

  1. Usuário seleciona a opção para cadastrar um apartamento.

  Este caso de uso inicia-se quando um usuário (gerente ou recepção) deseja fazer um cadastro de apartamentos. Para o usuário fazer uma cadastro de apartamento, é necessário que fiquem cadastrados no sistema alguns dados pertinentes ao cadastro de apartamentos. Para isso, o usuário deverá escolher a opção 'Apartamentos' e, então, o sistema irá exibir a tela de Cadastro de Apartamentos.

  2. O Usuário informa dado de identificação.

  O Usuário deverá informar o dado de identificação do apartamento, com este dado será feita a verificação se já existem cadastrados os dados do referido apartamento, na base de dados. Caso não exista serão solicitados outros dados.

  3. Usuário informa outros dados referentes ao apartamento.

  O usuário deverá informar alguns dados pertinentes ao cadastro de apartamento, como por exemplo : tipo, categoria, status, ramal e composição do apartamento.

  4. O Usuário clica no botão salvar.

  Quando o usuário clicar no botão salvar o sistema deverá salvar os dados informados do apartamento, na base de dados.

  5. Usuário deseja definir outro apartamento.

  Se o usuário desejar definir outro apartamento, o caso de uso retorna para o

passo 2.

6. Usuário finaliza cadastros.

Se o usuário desejar finalizar o cadastro de apartamento, deverá clicar no botão sair. Este caso de uso é finalizado aqui.

- Fluxos Alternativos:

A1. Sistema verifica dados do apartamento.

No passo 2, é verificado se existe um apartamento cadastrado na base de dados com a mesma identificação informada; caso já exista, o sistema irá exibir todos os dados do apartamento previamente cadastrado na tela. Caso o usuário desejar informar uma nova identificação, o caso de uso continua no início do passo 2.

- Requisitos Especiais

  O sistema irá garantir a não duplicidade dos apartamentos.

- Pré-Condições

  Para a realização deste caso de uso, o caso de uso Cadastrar Apartamentos deverá estar definido.

- Pós-Condições

  Após a definição desse caso de uso, será possível efetuar reservas e hospedagens.

Especificação do Caso de Uso Cadastrar Clientes

- Breve Descrição:

  Esse caso de uso retrata como seria o cadastro de Clientes.

- Fluxo de Eventos:

  - Fluxo Básico:

  1. Usuário seleciona a opção para cadastrar um cliente.

  Este caso de uso inicia-se quando um usuário (gerente ou atendente) deseja fazer um cadastro de clientes. Para o usuário fazer uma cadastro de cliente, é necessário que fiquem cadastrados no sistema alguns dados pertinentes ao cadastro de clientes. Para isso o usuário deverá escolher a opção 'Clientes' e, então, o sistema irá exibir a tela de Cadastro de Clientes.

  2. O Usuário informa dado de identificação.

  O Usuário deverá informar o dado de identificação do cliente, com este dado será feita a verificação se já existem cadastrados os dados do referido cliente, na base de dados. Caso não exista, serão solicitados outros dados.

  3. Usuário informa outros dados referentes ao clientes.

  O usuário deverá informar alguns dados pertinentes ao cadastro de cliente, como por exemplo : Nome, CPF, RG etc.

  4. O Usuário clica no botão salvar.

  Quando o usuário clicar no botão salvar o sistema deverá salvar os dados informados do cliente, na base de dados.

  5. Usuário deseja definir outro cliente.

  Se o usuário desejar definir outro cliente, o caso de uso retorna para o passo 2.

6. Usuário finaliza cadastros.

Se o usuário desejar finalizar o cadastro de cliente, deverá clicar no botão sair. Este caso de uso é finalizado aqui.

- Fluxos Alternativos:

A1. Sistema verifica dados do cliente.

No passo 2, é verificado se existe um cliente cadastrado na base de dados com a mesma identificação informada; caso já exista, o sistema irá exibir todos os dados do cliente previamente cadastrados na tela. Caso o usuário desejar informar uma nova identificação, o caso de uso continua no início do passo 2.

- Requisitos Especiais

  O sistema irá garantir a não duplicidade dos clientes.

- Pré-Condições

  Para a realização deste caso de uso, o caso de uso Cadastrar Clientes deverá estar definido.

- Pós-Condições

  Após a definição desse caso de uso, será possível efetuar reservas e hospedagens.

Especificação do Caso de Uso Cadastrar Reservas

- Breve Descrição:

Esse caso de uso retrata como seria efetuada uma reserva.

- Fluxo de Eventos:

  - Fluxo Básico:

  1. Usuário seleciona a opção para efetuar reservas.

  Este caso de uso inicia-se quando um usuário (gerente ou atendente) deseja efetuar uma reserva. Para o usuário efetuar uma reserva, é necessário que o usuário faça uma pesquisa de disponibilidade de quartos no período informado e também que fiquem cadastrados no sistema alguns dados pertinentes ao cadastro das reservas. Para isso, o usuário deverá escolher a opção 'Reservas' e então o sistema irá exibir a tela de Reservas.

  2. O Usuário deverá selecionar um apartamento.

  O Usuário deverá selecionar o apartamento pesquisado no passo 1

  3. Usuário informa outros dados referente à reserva.

  O usuário deverá informar alguns dados pertinentes à reserva, como por exemplo : tipo do apartamento, categoria do apartamento, data do contato, hora do contato, datas prevista para hospedagens etc.

  4. O Usuário clica no botão salvar.

  Quando o usuário clicar no botão salvar, o sistema deverá salvar na base de dados os dados informados da reserva.

  5. Usuário deseja definir outra reserva.

  Se o usuário desejar definir outra reserva, o caso de uso retorna para o passo 2.

  6. Usuário finaliza cadastros.

  Se o usuário desejar finalizar a reserva, deverá clicar no botão sair. Este caso de uso é finalizado aqui.

- Fluxos Alternativos:

A1. Sistema verifica dados da reserva.

No passo 2, é verificado se existe uma disponibilidade de apartamentos, caso exista disponibilidade, é cadastrada a reserva. Caso o usuário desejar informar uma nova identificação, o caso de uso continua no início do passo 2.

- Requisitos Especiais

  O sistema irá garantir a não duplicidade das reservas.

- Pré-Condições

  Para a realização deste caso de uso, o caso de uso Efetuar reservas deverá estar definido.

- Pós-Condições

  Após a definição desse caso de uso, será possível efetuar as hospedagens.

Especificação do Caso de Uso Efetuar Hospedagens

- Breve Descrição:

  Esse caso de uso retrata como seria efetuada uma hospedagem.

- Fluxo de Eventos:

  - Fluxo Básico:

  1. Usuário seleciona a opção para efetuar hospedagem.

Este caso de uso inicia-se quando um usuário (gerente ou recepcionista) deseja efetuar uma hospedagem. Para o usuário efetuar uma hospedagem, é necessário que o usuário faça uma pesquisa de disponibilidade de quartos no período informado e também que fiquem cadastrados no sistema alguns dados pertinentes ao cadastro das hospedagens. Para isso o usuário deverá escolher a opção ‘Hospedagens’ e, então, o sistema irá exibir a tela de Hospedagens.

2. O Usuário deverá selecionar um apartamento.

O Usuário deverá selecionar o apartamento pesquisado no passo 1.

3. Usuário informa outros dados referentes a hospedagens.

O usuário deverá informar alguns dados pertinentes à hospedagem, como por exemplo : tipo do apartamento, categoria do apartamento, data do contato, hora do contato, datas prevista para saída etc.

4. O Usuário clica no botão salvar.

Quando o usuário clicar no botão salvar o sistema deverá salvar os dados informados da hospedagem, na base de dados.

5. Usuário deseja definir outra hospedagem.

Se o usuário desejar definir outra hospedagem, o caso de uso retorna para o passo 2.

6. Usuário finaliza cadastros.

Se o usuário desejar finalizar a hospedagem, deverá clicar no botão sair. Este caso de uso é finalizado aqui.

- Fluxos Alternativos:

A1. Sistema verifica dados da hospedagem.

No passo 2, é verificado se existe uma disponibilidade de apartamentos; caso

exista disponibilidade, é cadastrada a hospedagem. Caso o usuário desejar informar uma nova identificação, o caso de uso continua no início do passo 2.

- Requisitos Especiais

  O sistema irá garantir a não duplicidade das hospedagens.

- Pré-Condições

  Para a realização deste caso de uso, o caso de uso Efetuar hospedagens deverá estar definido.

- Pós-Condições

  Após a definição desse caso de uso, será possível efetuar fechamento das hospedagens.

Especificação do Caso de Uso Efetuar Fechamento de Hospedagens

- Breve Descrição:

  Esse caso de uso retrata como seria efetuado um fechamento de hospedagem.

- Fluxo de Eventos:

  - Fluxo Básico:

  1. Usuário seleciona a opção para efetuar fechamento de hospedagem.

  Este caso de uso inicia-se quando um usuário (gerente ou recepcionista) deseja efetuar um fechamento de hospedagem. Para o usuário efetuar um fechamento, é

necessário que o usuário faça uma pesquisa das hospedagens em aberto. Para isso, o usuário deverá escolher a opção ‘Fecha Hospedagens’ e, então, o sistema irá exibir a tela de Fechamento de Hospedagens.

2. O Usuário deverá informar um apartamento.

O Usuário deverá selecionar o apartamento pesquisado no passo 1.

3. Usuário informa outros dados referentes à realização do fechamento de hospedagens.

O usuário deverá informar alguns dados pertinentes ao fechamento da hospedagem, como por exemplo : Data do fechamento e as despesas efetuadas na aba de despesas etc.

4. O Usuário clica no botão salvar.

Quando o usuário clicar no botão salvar o sistema deverá salvar os dados informados no fechamento, na base de dados.

5. Usuário finaliza cadastros.

6. Se o usuário desejar finalizar o fechamento de hospedagem, deverá clicar no botão sair. Este caso de uso é finalizado aqui.

- Fluxos Alternativos:

A1. Sistema verifica dados da hospedagem.

No passo 2, é verificado se existe a hospedagem e apartamento informado; caso exista a hospedagem, é possível efetuar o fechamento. Caso o usuário desejar informar uma nova identificação, o caso de uso continua no início do passo 2.

- Requisitos Especiais

  O sistema irá garantir a não duplicidade dos fechamentos.

- Pré-Condições

  Para a realização deste caso de uso, o caso de uso Efetuar Fechamento de hospedagens deverá estar definido.

- Pós-Condições

  Após a definição desse caso de uso, será possível fazer retiradas de dados tomando como base os relatórios.

Especificação do Caso de Uso Cadastrar Usuários

- Breve Descrição:

  Esse caso de uso retrata como seria o cadastro de Usuários.

- Fluxo de Eventos:

  - Fluxo Básico:

  1. Gerente seleciona a opção para cadastrar um usuário.

  Este caso de uso inicia-se quando um gerente deseja fazer um cadastro de usuários. Para o gerente fazer um cadastro de usuário, é necessário que fiquem cadastrados no sistema alguns dados pertinentes ao cadastro de usuários. Para isso, o gerente deverá escolher a opção 'Usuário' e então o sistema irá exibir a tela de Cadastro de Usuários.

  2. O Gerente informa dado de identificação.

  O gerente deverá informar o dado de identificação do usuário, com este dado será feita a verificação se já existem cadastrados os dados do referido ao usuário,

na base de dados. Caso não exista, serão solicitados outros dados.

3. Gerente informa outros dados referente ao usuário.

O gerente deverá informar alguns dados pertinentes ao cadastro de usuário, como por exemplo: Nome e Senha.

4. O Gerente clica no botão salvar.

Quando o gerente clicar no botão salvar o sistema deverá salvar os dados informados do usuário, na base de dados.

5. Gerente deseja definir outro usuário.

Se o gerente desejar definir outro usuário, o caso de uso retorna para o passo 2.

6. Gerente finaliza cadastros.

Se o gerente desejar finalizar o cadastro de usuário, deverá clicar no botão sair. Este caso de uso é finalizado aqui.

- Fluxos Alternativos:

A1. Sistema verifica dados do usuário.

No passo 2, é verificado se existe um usuário cadastrado na base de dados com a mesma identificação informada; caso já exista, o sistema somente irá exibir todos os dados do usuário previamente cadastrados na tela. Caso o usuário desejar informar uma nova identificação, o caso de uso continua no início do passo 2.

- Requisitos Especiais

  O sistema irá garantir a não duplicidade dos usuários.

- Pré-Condições

Para a realização deste caso de uso, o caso de uso Cadastrar Usuários deverá estar definido.

- Pós-Condições

  Após a definição desse caso de uso, será possível efetuar as movimentações no sistema.

Especificação do Caso de Uso Cadastrar Categorias

- Breve Descrição:

  Esse caso de uso retrata como seria o cadastro de Endereços.

- Fluxo de Eventos:

  - Fluxo Básico:

  1. Usuário seleciona a opção para cadastrar um endereço.

  Este caso de uso inicia-se quando um usuário (gerente, atendente ou recepcionista) deseja fazer um cadastro de endereços. Para o usuário fazer um cadastro de endereço, é necessário que fiquem cadastrados no sistema alguns dados pertinentes ao cadastro de endereços. Para isso o usuário deverá escolher a opção ‘Endereços’ e, então, o sistema irá exibir a tela de Cadastro de Endereços.

  2. O Usuário informa dado de identificação.

  O Usuário deverá informar o dado de identificação do endereço, com este dado, será feita a verificação se já existem cadastrados os dados do referido ao endereço, na base de dados. Caso não exista, será solicitado outros dados.

3. Usuário informa outros dados referentes ao endereços.

O usuário deverá informar alguns dados pertinentes ao cadastro de endereço, como por exemplo : CEP, Endereço, Bairro, Complemento, Cidade e UF.

4. O Usuário clica no botão salvar.

Quando o usuário clicar no botão salvar, o sistema deverá salvar na base de dados os dados informados do endereço.

5. Usuário deseja definir outro endereço.

Se o usuário desejar definir outro endereço, o caso de uso retorna para o passo 2.

6. Usuário finaliza cadastros.

Se o usuário desejar finalizar o cadastro de endereço, deverá clicar no botão sair. Este caso de uso é finalizado aqui.

- Fluxos Alternativos:

A1. Sistema verifica dados do endereço.

No passo 2, é verificado se existe um endereço cadastrado na base de dados com a mesma identificação informada; caso já exista, o sistema irá exibir todos os dados do endereço previamente cadastrados na tela. Caso o usuário desejar informar uma nova identificação, o caso de uso continua no início do passo 2.

- Requisitos Especiais

  O sistema irá garantir a não duplicidade dos endereços.

- Pré-Condições

  Para a realização deste caso de uso, o caso de uso Cadastrar Endereços deverá estar definido.

- Pós-Condições

  Após a definição desse caso de uso, será possível efetuar Cadastro de Clientes.