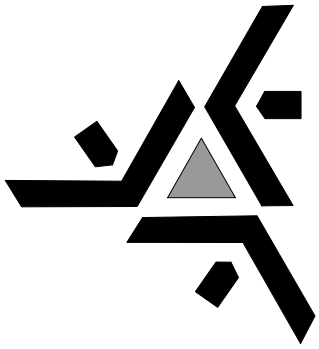

Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Informática

A Utilização do Método Catalysis no Desenvolvimento de Software Baseado em Componentes

Flávio Rogério Uber

TG-22-98

Maringá - Paraná
Brasil

Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Informática

A Utilização do Método Catalysis no Desenvolvimento de Software Baseado em Componentes

Flávio Rogério Uber

Trabalho de Graduação apresentado ao Curso de Ciência da Computação, do Centro de Tecnologia, da Universidade Estadual de Maringá.
Orientadora*: Profa. Dra. Elisa Hatsue Moriya Huzita*

Maringá - Paraná

1998

Flávio Rogério Uber

A Utilização do Método Catalysis no Desenvolvimento de Software Baseado em Componentes

Este exemplar corresponde à redação final da monografia aprovada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Universidade Estadual de Maringá, pela comissão formada pelos professores:

_____

Orientadora: Profa. Dra. Elisa Hatsue Moriya Huzita
Departamento de Informática, CTC, DIN

_____

Prof. Dr. Márcio Eduardo Delamaro
Departamento de Informática, CTC, DIN

_____

Prof. Ademir Carniel
Departamento de Informática, CTC, DIN

Maringá, Dezembro de 1998

Agradecimentos

Agradeço em primeiro lugar a Deus, que me deu saúde e capacidade para superar todos os obstáculos, não apenas durante esses quatro anos, mas em todos os momentos de minha vida.

Não apenas agradeço, mas dedico todos esses anos aos meus pais, Alcides e Sueli, que me deram todo amor e todo apoio para prosseguir nos momentos mais difíceis. Sei que, sem eles, nada em minha vida teria sido possível. A esses seres humanos fantásticos, minha eterna gratidão. Que Deus os ilumine sempre.

Aos meus irmãos, Ana Paula e Renato que, da mesma forma, deram todo o apoio familiar para que esse fim de curso se tornasse realidade. São meus anjos da guarda em todos os momentos .

Ao meu amigo Danillo, com quem ri e chorei durante esses quatro anos, e quem eu aprendi a respeitar e admirar, que Deus o ilumine e o conduza pelas mãos nos caminhos que ele escolher.

Aos não menos amigos, Luciano, Gerson e Alexandre (Doná) que também enfrentaram juntos todas as batalhas e tornaram-se praticamente uma família para mim. Que a vida traga-lhes tudo de melhor, eles merecem.

Aos demais amigos, Fábio (Honda), Junior, Fábio (Além), Fábio Chavenco, Alisson (Jordy), Eduardo, Kazumi com quem tive a alegria de conviver durante esse curso. São amizades que eu espero que perdurem por muitos anos

Aos demais colegas de curso, eles também fizeram parte dessa caminhada, e estarão sempre na minha lembrança e no meu coração.

Aos professores, que durante todo o curso transmitiram seus conhecimentos e portanto tiveram papel fundamental em minha vida acadêmica, que Deus lhes dê saúde para que continuem sua missão educadora por muitos anos.

Um agradecimento especial a minha orientadora, profa. Elisa Hatsue Moriya Huzita, que sempre apoiou e incentivou para que esse trabalho se tornasse uma realidade. Obrigado pela compreensão, pela paciência e por tudo que aprendi e pude melhorar, tanto em minha vida pessoal, quanto profissional.

Resumo

Ao longo dos anos, tem-se presenciado o surgimento de vários métodos e abordagens para o desenvolvimento de software. Este dinamismo é motivado pela crescente complexidade dos software e pelos cuidados a serem observados para a obtenção de produtos de qualidade. Neste contexto, surgiu o desenvolvimento baseado em componentes (CBD), uma abordagem que visa a rápida construção de uma aplicação a partir de componentes de software pré-fabricados [Bro97].

Neste trabalho foi realizado um estudo do método Catalysis, tema central do trabalho, que se trata de um método para desenvolvimento de software baseado em componentes. Em seguida efetuou-se a especificação de um sistema para a realização de uma análise do método e avaliar os pontos fortes e fracos do mesmo com base na experiência obtida com essa especificação.

Abstract

Several methods and approach for the software development have been studied and adopted by developers. This dynamism is motivated by the growing of software complexity and for the requirements to be observed to obtain software with quality. In this context, appeared the component-based development (CBD), an approach that seeks the fast construction of an application starting from on the shelf software components [Bro97].

A study of the Catalysis method was accomplished. This is a method for component-based development. This is the central issue of the work. The specification of a system, using the Catalysis method has been done aiming at the evaluation of the method.

Índice

Índice de Ilustrações

Capítulo 1

Introdução

1.1 Motivação

Desde a construção dos primeiros software, sob encomenda e de pequeno porte, até os dias atuais, com sistemas distribuídos, técnicas avançadas e usuários exigentes, ocorreram muitas mudanças de filosofia na especificação de um software, devido à crescente preocupação com a qualidade do software, e portanto, com o processo de desenvolvimento destes. Pressman [Pre95] define a qualidade de software como a "conformidade a requisitos funcionais e de desempenho explicitamente declarados, a padrões de desenvolvimento claramente documentados e a características implícitas que são esperadas de todo software profissionalmente desenvolvido".

Durante os anos 50 e 60 as aplicações eram construídas de acordo com critérios pessoais de desenvolvedores. Isso criou um problema quando tornou-se necessária a manutenção desses software, que não possuíam qualquer documentação e estruturação.

Na década de 60 assistiu-se a uma evolução no sentido de definir processos para desenvolvimento de software aliado à adoção de linguagens de programação de mais alto nível. Nesta época surgiram métodos como a análise estruturada, com ênfase sobre os processos e que procurava uma maneira de se documentar o processo de software, para que seu desenvolvimento seguisse um padrão que tornasse a construção de aplicativos não mais uma tarefa para poucos intelectuais capacitados.

Nos anos 70 e 80, com o aparecimento do Banco de Dados relacional e a modelagem de entidade-relacionamento passaram a ser enfatizados os dados.

Em meados da década de 80, começaram a surgir deficiências na análise estruturada, quando tentou-se aplicar o método em aplicações orientadas a controle. Surgiram então, extensões que consideravam aspectos de tempo real, introduzidas por Ward e Mellor [War85] e Hatley e Pirbhai [Hat87]. No final da década de 80, o paradigma orientado a

objeto começava a amadurecer numa abordagem poderosa [Pre95], com a proposta de resolução dos problemas da análise estruturada e de preocupação com manipulação de dados. Trata-se da aplicação de conceitos que são aprendidos na infância, – objetos, atributos ... – e que demorou tanto tempo para se notar sua aplicação na análise e especificação de um sistema. Por que? Segundo [Coa90] "porque talvez estivéssemos preocupados demais 'seguindo o fluxo', durante o apogeu da análise estruturada, para considerarmos alternativas".

Atualmente o desenvolvimento de software encontra-se em meio a duas tendências significativas de mudança. Um diz respeito aos tipos de aplicações existentes, e a outra de como estas aplicações são construídas. A primeira mudança, enfatiza a alta dos negócios usando Internet, e o aparecimento da *network computing* como um novo modelo computacional. A segunda, diz respeito ao aparecimento da abordagem de Desenvolvimento Baseado em Componentes *(CBD)* [Ste97b].

1.2 Objetivos

Neste sentido, o desenvolvimento baseado em componentes norteia a realização deste trabalho que tem ainda o intuito de realizar o estudo do método Catalysis e efetuar uma análise com base na especificação de um sistema exemplo .

A teoria de desenvolvimento baseado em componentes, como será mostrado no capítulo 2, é centrada no reuso de componentes pré-fabricados e na melhora da qualidade do produto obtido, já que utilizando-se componentes testados no mercado tem-se maior garantia quanto à confiabilidade deste.

1.3 Estrutura da Monografia

O Capítulo 2 mostra toda a teoria e aplicabilidade do desenvolvimento baseado em componente. O Capítulo 3 expõe os princípios de Catalysis, bem como os diagramas que a compõem. O Capítulo 4 descreve o sistema exemplo ilustrado com alguns exemplos dos diagramas desenvolvidos. O Capítulo 5 trata da análise do método, quais seus pontos fracos e fortes com relação aos recursos oferecidos, tendo por base a especificação do

sistema exemplo. No Capítulo 6 tem-se a conclusão obtida com a realização do trabalho e o Anexo A apresenta os demais diagramas, não demonstrados no Capítulo 4.

Capítulo 2

Desenvolvimento Baseado em Componente

2.1 Introdução

Com base nas tendências atualmente presentes no processo de desenvolvimento de software, o objeto de estudo deste trabalho se refere ao Desenvolvimento Baseado em Componente e a aplicação do método Catalysis para desenvolvimento de sistemas utilizando esta abordagem. Este capítulo propõe uma série de conceitos referentes à esta abordagem bem como justificativas para a sua aplicabilidade. Com estas informações é possível se entender o porquê de várias pesquisas relevantes que são desenvolvidas na área, e consequentemente sua importância na Engenharia de Software e na informática de modo geral. Num primeiro momento tem-se algumas definições pertinentes à abordagem baseada em componentes, na sequência algumas aplicações possíveis para a tecnologia e alguns benefícios que se pode atingir com o seu uso. Por fim algumas considerações acerca do que pode ser um componente.

2.2 Definição

Desenvolvimento Baseado em Componentes (CBD) propõe-se a ser uma maneira prática e radicalmente diferente de construir aplicações oferecendo técnicas imensamente melhores de desenvolvê-las, que seria mais barata, mais rápida e se adaptaria naturalmente com a emergente tecnologia de objetos distribuídos.

O conceito de componentes implica na separação da especificação da implementação, garantindo a propriedade da reusabilidade, podendo-se, desta forma, utilizar um componente para diversas aplicações e substituir um componente por outro em um sistema a qualquer momento. Como definiu [DSo97], “Um componente é uma unidade de software, desenvolvida independentemente, que encapsula seu projeto e sua implementação e oferece interfaces ao mundo externo, pela qual ele pode ser unido a outros componentes e formar um grande sistema”.

Alguns componentes podem ser vistos, como descritos acima, como unidades que são unidas a outros componentes, enquanto outros componentes funcionarão mais como um *framework*, ou seja, uma unidade de implementação projetada para um domínio genérico, onde outros componentes poderão ser associados e desta forma constituir um sistema funcional ao usuário.

2.3 Aplicações e Benefícios

A idéia de CBD é centralizada em dois pontos[Ste97c]:

- O desenvolvimento de aplicações pode ser melhorado significativamente se elas puderem ser reunidas rapidamente a partir de componentes de software pré-fabricados.
- Uma coleção cada vez maior de componentes de software interoperáveis estará disponível para desenvolvedores em catálogos gerais e especializados.

Estes pontos do Desenvolvimento Baseado em Componentes podem proporcionar aos seus desenvolvedores e usuários vantagens que justificarão seu uso bem como o investimento na mudança de tecnologia [Ste97c]. Algumas dessas vantagens seriam:

- Redução do tempo de desenvolvimento: através de consultas a catálogos de componentes.
- Redução dos custos de desenvolvimento: os custos de construção de um componente são maiores do que a aquisição de um componente já pronto.
- Aumento de produtividade: a preocupação dos desenvolvedores seria com a reunião de componentes e não com o desenvolvimento dos mesmos.
- Redução nos riscos de problemas: a garantia é maior quando se pode obter componentes devidamente testados e certificados.
- Maior consistência: componentes utilizam arquiteturas padrões para desenvolver aplicações.

Pode-se observar que estas vantagens estão centradas basicamente no reuso de componentes. No entanto, CBD possui três princípios que se constituem em desafios para se obter esse reuso [Bro97]. Estes princípios são:

1. **Separação da especificação da implementação**: Um dos maiores prejuízos no reuso é a necessidade de se conhecer detalhadamente a implementação de um componente para se procurar, integrar ou atualizar uma peça. No desenvolvimento baseado em componente a especificação é separada de como ele é implementado, permitindo que implementações sejam usadas ou trocadas sem impacto para o sistema. Logo, um componente pode se adaptar prontamente a novos requisitos de operação.

2. **Encapsulamento de dados e processos**: Através do encapsulamento, o acesso às funcionalidades de um componente está limitado às operações fornecidas pela sua interface diminuindo o impacto das mudanças em um componente.

3. **Independência da tecnologia de componentes**: Um dos desafios de muitas organizações é reduzir os esforços para reimplementar a funcionalidade de um componente quando há uma mudança de plataforma. Para isso, o modelo de implementação deve estar suficientemente detalhado para que o código possa ser gerado para uma variedade de plataformas. Quando há uma mudança de plataforma, essa descrição abstrata da implementação de um componente é usada para gerar automaticamente o código para esta nova plataforma.

Contudo, o reuso não é o único desafio no Desenvolvimento Baseado em Componentes. Outros pontos merecem atenção, como por exemplo a sua aplicação. Neste sentido, podem ser verificadas algumas perspectivas e realidades envolvendo CBD.

Com relação às perspectivas, nota-se como é impressionante a rapidez com que os negócios têm explorado a Web como uma tecnologia, expandindo os limites dos negócios. A Web pode ser vista como uma nova plataforma de implementação para aplicações cliente/servidor. Oferecendo uma localização independente e protocolos consistentes para acesso à informação e serviço de software, a Web tem iniciado simultaneamente a distribuição computacional e um novo comportamento de aplicações.

O uso crescente de redes e da Web torna cada vez mais necessário o desenvolvimento de aplicações que funcionem em uma rede corporativa ou na Internet, e o desenvolvimento baseado em componente visa facilitar e tornar realidade esta tendência.

Num futuro próximo [DSo97], se eventualmente um componente de um sistema encontrar-se na rede, poderá ser feito um *download* automático deste conforme seja necessário, da mesma forma que muitos *applets* Java fazem hoje em dia.

Mas uma mudança radical como esta pode levar bastante tempo, afinal é necessário que os desenvolvedores de software mudem suas atitudes, estando constantemente alerta para o reuso; a tecnologia de desenvolvimento mudará e será melhorada. O desenvolvimento de novas metodologias será necessário e padrões para integração de componentes precisa ser definido.

Além disso, outras aplicações já se tornaram realidade. Algumas organizações já fazem uso dos primeiros estágios de desenvolvimento de componentes[Ste97d], como VBX da Microsoft para aplicações de *Visual Basic* e controles *ActiveX*.

Um padrão para descrição de interfaces de componentes é a IDL (*Interface Definition Language*), que permite aos componentes de software executar e se intercomunicar através de redes, em diferentes plataformas.

O maior desafio, continua sendo o fato de sistemas serem desenvolvidos de forma "artística", muito mais pela criatividade do desenvolvedor do que pela adoção de padrões, métricas e regras de modo geral. Estes padrões serão possíveis quando as organizações atingirem uma maturidade da construção de software, seguindo, por exemplo, o modelo CMM (*Capability Maturity Model*) [Pau95].

A abordagem de componentes separa o desenvolvimento de componentes de sua reunião final dentro de sistemas, ou seja, menos atenção é dada à construção interna de componentes e mais atenção é dada às necessidades dos clientes. O importante é O QUE o componente faz e não COMO ele faz.

2.4 Tipos de Componentes

Na prática os diferentes tipos de componentes possuem potencial para reuso em níveis diferentes, e também melhoram a produtividade do desenvolvedor em escalas variáveis. O potencial para o reuso é maior quanto menor for a granularidade do componente, ou seja, quanto menos funcionalidades ele reunir, enquanto a produtividade do desenvolvedor é maior quanto maior for a granularidade do componente. Os tipos de componente são: bibliotecas de classes, – grande potencial de reuso e baixa produtividade do desenvolvedor - componentes encapsulados, frameworks e aplicações pré-construídas – baixo potencial de reuso e alta produtividade do desenvolvedor.

As bibliotecas de classes correspondem ao nível mais baixo de abstração de componentes e possuem um grande potencial para reuso. O problema na sua utilização é a necessidade de um conhecimento maior por parte do programador já que muitas vezes as bibliotecas de classes são *black box*[1].

Os componentes encapsulados são na verdade o mesmo que bibliotecas de classes porém com a definição de uma interface que separará a especificação da implementação. Esta separação faz com que componentes encapsulados elevem a produtividade do programador em relação às bibliotecas de classes já que não se exige mais tanto conhecimento do profissional.

Um *framework* é uma reunião pré-construída de componentes junto com uma ligação que os une. Sua arquitetura inicial não pode ser modificada, mas a possibilidade de se expandir com a agregação de novos componentes faz com que seu objetivo se volte para atender necessidades individuais. Muitos *frameworks* que aparecem no mercado – como por exemplo o CIM (*Computer-Integrated Manufacturing*) da SEMANTECH em um projeto conjunto com a *Texas Instruments* - são construídos usando bibliotecas de classes orientadas a objeto.

[1] Um componente é *black box* quando o usuário tem direito apenas de utilização, e *white box* quando tem acesso a seu código fonte.

As aplicações pré-construídas proporcionam grande produtividade para o desenvolvedor, já que na teoria, com elas, nenhum código é requerido e o domínio é totalmente coberto pela funcionalidade do pacote pré-construído. Hoje em dia um exemplo de aplicações pré-construídas é o COM da Microsoft que permite que o *Word* acesse o *Excel*.

2.5 Considerações Finais

Este capítulo apresentou os conceitos envolvidos no Desenvolvimento Baseado em Componente. CBD se propõe a garantir a construção de software de forma mais rápida e com maior qualidade, a partir de componentes pré-fabricados e reusáveis.

A reusabilidade de um componente, apesar de ser sua característica principal, visa sempre a qualidade do processo de software e do produto final. Este é o princípio de CBD: obter produtos mais confiáveis e garantidos através da reutilização de componentes já testados no mercado.

Capítulo 3

Catalysis

3.1 Introdução

Diversas metodologias têm a proposta de suporte ao desenvolvimento baseado em componentes. Nesse trabalho utilizou-se, a fim de realizar sua avaliação, o método Catalysis, uma metodologia centrada na UML (*unified modeling language*) e que adota muitos conceitos da OMT (*object modeling technique*). Este capítulo apresenta a fundamentação teórica inerente ao método bem como os diagramas que fazem parte da mesma. É apresentado também o processo de desenvolvimento, conforme a proposta de autores, em três níveis: nível de domínio do problema, nível de especificação do componente e nível de projeto do componente.

3.2 Conceitos Básicos

O método Catalysis de desenvolvimento de software é baseado [Ico96] nos princípios de abstração, precisão e associação de componentes. Estes princípios podem ser aplicados em todos os níveis de desenvolvimento de software a partir do domínio de um problema.

Segundo Georg Pigel da Sterling Software [Ico96], "Catalysis não pretende ser a resolução de todos os problemas, ela define uma maneira prática de construir software baseada em componentes de forma sistemática. Fornece regras práticas ao invés de conceitos ... uma abordagem passo a passo para implementação ..." .

Segundo os conceitos[Ste97b] de Catalysis um componente de software possui:

- Uma especificação, que descreve o que o componente faz e como ele se comporta quando os seus serviços são utilizados.
- Uma implementação, expressa em termos de código e dado que será a garantia de atendimento da especificação.
- E o encapsulamento, a divisão entre a especificação e a implementação. Este encapsulamento se dá através de uma interface, que dará ao desenvolvedor, acesso à especificação do componente, mas o isolará da implementação.

As interfaces de um componente resumem como o cliente deveria interagir com ele, mas escondem detalhes que fundamentam a implementação. Clientes de aplicações e componentes trabalham com a interface, não com as implementações.

Os principais elementos de uma interface são:

Lista de operações: cada operação possui um nome e sua assinatura (os parâmetros esperados do cliente e o valor retornado para ele); também precisa de uma semântica precisa e previsível para obter a confiança do cliente e poder ser implementada pelos programadores.

Modelo de especificação: é a descrição do vocabulário ou termos para explicar detalhadamente os efeitos da operação. Este modelo deve ser claro, preciso e ajudar a descrever o estado de algum tipo de objeto.

Um conceito importante seria o de invariantes, ou seja, regras, ou condições estabelecidas para o comportamento de um componente. Por exemplo, supondo que o salário máximo de um empresa não poderia ultrapassar dez mil reais.

Por sua própria natureza, herdada da orientação à objeto, Catalysis suporta polimorfismo, agregação, associação e os demais princípios básicos da OO.

O método Catalysis é baseado em três conceitos principais [DSo97]:
1. Tipo: Um tipo não é o mesmo que classe. Ele define o que um objeto faz, pela especificação de seu comportamento externamente visível; não descreve sua implementação, como faz a classe. Por exemplo, um calendário pode ser implementado de várias maneiras mas todas as maneiras exibem o mesmo comportamento, que é descrito no TIPO calendário.

2. Colaboração: Como um grupo de objetos interage. Uma colaboração define um conjunto de ações entre objetos tipados desempenhando certos papéis em relação a outros objetos na colaboração.

3. Refinamento: Relacionamento entre duas descrições da mesma coisa (a realização e a abstração). Um modelo abstrato de uma pessoa usa um atributo dinheiro, modelos mais detalhados são baseados, por exemplo, em contas bancárias, cartão de crédito...

3.3 Processo de Desenvolvimento

Quanto ao processo desenvolvimento de software, Catalysis na verdade não propõe uma ordem para o desenvolvimento dos diagramas. Alguns autores consideram como a melhor forma de se executar o desenvolvimento pela divisão em três níveis que seriam [DSo97]: nível de domínio do problema, nível de nível de especificação do componente e nível de projeto do componente.

3.3.1 Nível de domínio do problema

Neste nível, a ênfase está no entendimento do problema. Envolve o desenvolvimento de *mind-maps* (figura 3.1), diagrama de contexto (figura 3.2) e cenários (figura 3.3).

Mind-maps é a representação estruturada de termos que são relacionados. Um *mind-map* é representado simplesmente por conceitos e/ou frases conectadas através de linhas. Os conceitos são representados por verbos ou substantivos extraídas da idéia do problema. Esta técnica ajuda o desenvolvedor a obter uma visão geral do comportamento do sistema e suas funcionalidades. O desenvolvedor passa a ter uma visão mais clara do escopo que será considerado. Além de obter uma primeira impressão acerca das classes que farão parte do sistema.

Exemplo de notação:

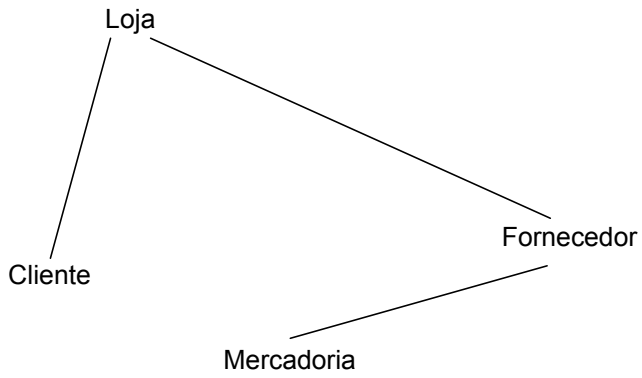

Figura 3.1 - ***Mind Map***

Outro diagrama dentro deste nível seria o diagrama de contexto do sistema (figura 2). Ele representa os atores e suas interações. Por atores entende-se que sejam todos aqueles que interagem com o sistema ou o próprio sistema, isto é, todos que agem diretamente através de operações. Interações são as ações que são praticadas pelos atores e que alteram estados e/ou atributos de uma classe. Quando o ator é um sistema ele é representado por um retângulo, e quando trata-se de um usuário é representado pela ilustração de um homem. Isto ajuda a definir mais precisamente os limites do sistema com relação àqueles limites definidos nos *mind-maps*. Com o diagrama de contexto o desenvolvedor tem a idéia das operações do sistema e quais são os atores que interagem a fim de executar a operação. As ações nas quais o sistema em questão é um ator são aquelas que devem ser desenvolvidas como parte do sistema.

Notação:

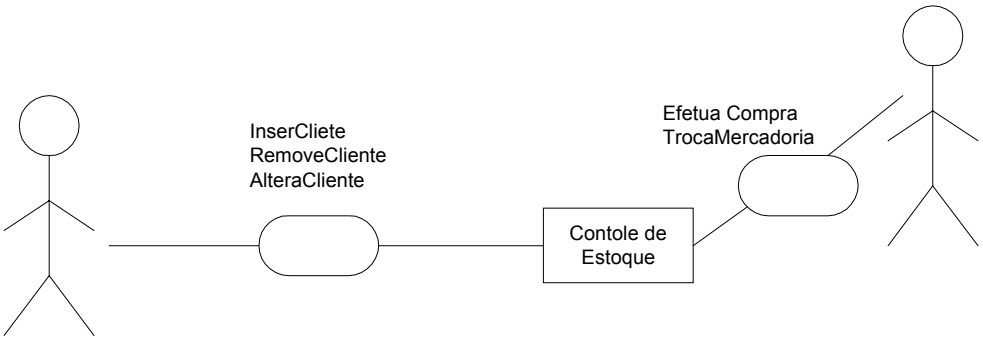

Figura 3.2 - **Diagrama de Contexto**

Finalmente, neste nível seriam incluídos os cenários (figura 3). Cada cenário ilustra uma sequência na qual as ações acontecem. Geralmente, o desenvolvimento de cenários ajuda na identificação de ações que ainda não foram descritas no diagrama de contexto. Eles passam ao desenvolvedor quais são as ações e interações que são necessárias para que ocorra determinada operação. Os atores que interagem nas ações são representados como setas verticais e setas horizontais representam as ações que são realizadas e unem os atores que interagem na ocorrência desta ação.

Notação:

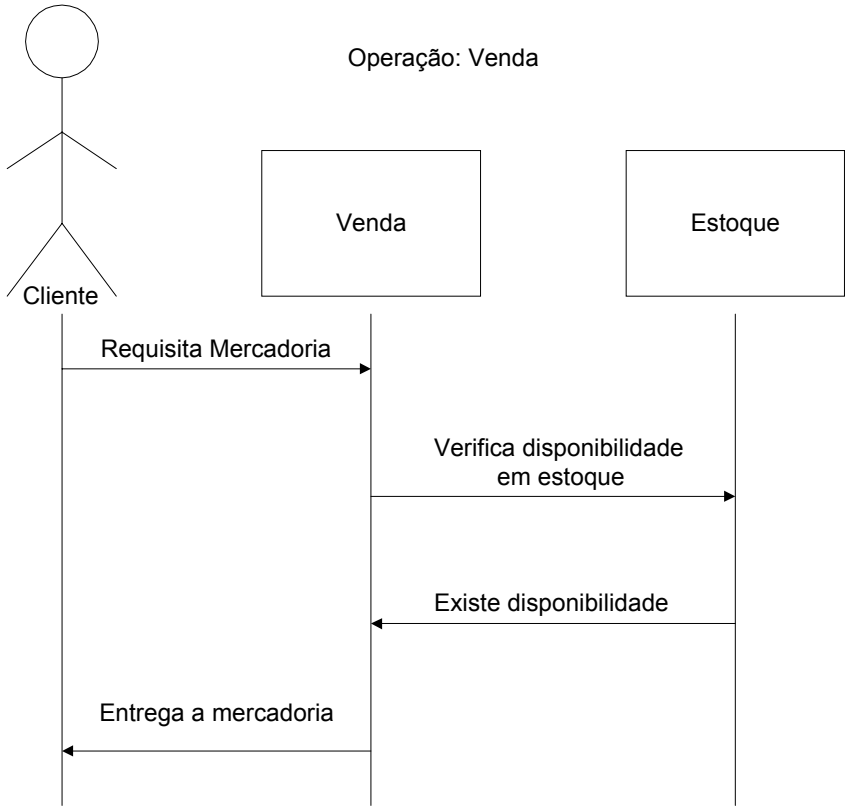

Figura 3.3 - **Cenário**

3.3.2 Nível de Especificação do Componente

Neste nível deseja-se descrever o comportamento do sistema, visível externamente, de forma não ambígua. O ponto de partida, neste nível, é o diagrama de contexto do nível anterior. Nesse nível são desenvolvidos os *snapshots* (figura 3.4), a especificação das operações e o modelo de tipos (figura 3.5).

O primeiro ponto a ser considerado neste nível é a especificação das operações. Inicialmente as operações são especificadas informalmente indicando as entradas requeridas pela operação, as mudanças no objeto e as saídas retornadas, e ainda as condições sob as quais o comportamento é garantido. Isso é feito através da definição de pré e pós condições.

Em seguida será necessária uma especificação formal das operações. Para isto é necessário: identificar entradas e saídas (valores retornados); identificar atributos (termos derivados das entradas das operações); identificar tipos para parâmetros de entrada/saída; formalizar a especificação da operação de modo a especificar saídas e mudanças de estados nos atributos, estritamente em termos de entradas e atributos do objeto; revisar e melhorar a especificação informal.

Os atributos do tipo são descritos como tipos do modelo, com atributos e associações entre eles, e invariantes que relacionam os valores válidos dos atributos. As associações têm uma cardinalidade para indicar a multiplicidade de um tipo em relação a outro.

Para esclarecer como os atributos de um sistema são afetados por uma operação descrita no cenário, podemos criar os *snapshots* (figura 3.4). Eles mostram o subconjunto de valores de atributos que existem antes da execução de uma operação e depois desta. Olhando para um *snapshot*, temos na parte superior a representação dos atributos antes da operação, e na parte inferior, a representação dos atributos após a operação, podendo-se assim visualizar e especificar precisamente os efeitos de uma operação. Os *snapshots* têm o intuito de fornecer uma representação visual dos efeitos da operação, no entanto, eles podem ser expressos em forma de texto, indicando o valor de um atributo (classe.atributo) antes e depois da operação.
Representação gráfica:

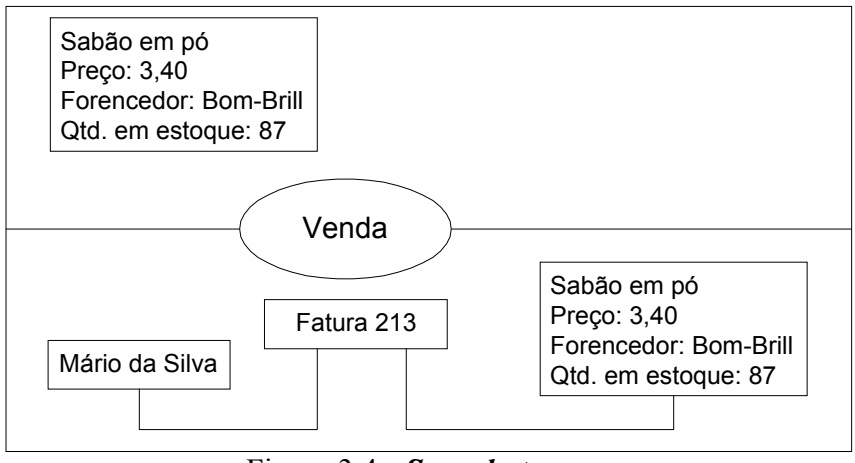

Figura 3.4 - ***Snapshot***

Finalmente, neste nível, o sistema que está sendo desenvolvido é representado como um tipo, que consiste de um modelo e um conjunto de operações sobre esse modelo. A definição inicial envolve a identificação de cada ação na qual o sistema a ser desenvolvido

participa, especificando-a como uma operação sobre o tipo. É representado por um retângulo, onde a parte superior identifica o nome do tipo, a parte central a sua interação com outros tipos e finalmente na parte inferior as operações na qual o tipo interage (figura 3.5).

Representação:

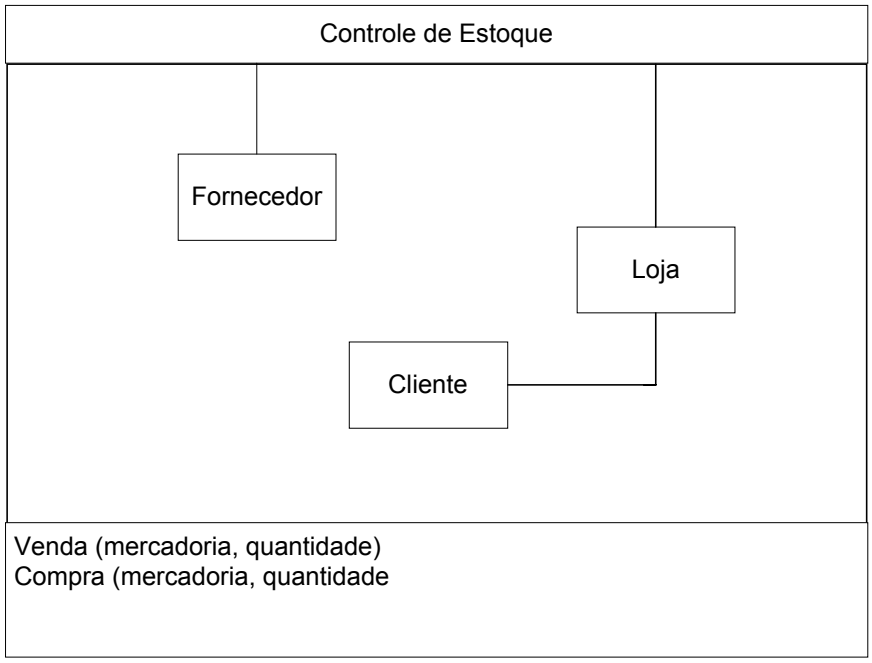

Figura 3.5 - **Modelo de Tipos**

3.3.3 Nível de projeto do componente

A ênfase aqui é definir uma estrutura interna e interações que satisfaçam os requisitos comportamentais, tecnológicos e de engenharia de software de um sistema. São desenvolvidos os diagramas de interação (figura 3.6), diagrama de classes (figura 3.7) e os diagramas de estados (figura 3.8).

Normalmente, cada tipo identificado na especificação do modelo de tipo do componente será implementado como uma classe separada.

Para cada operação especificada, identifica-se um objeto para "receber" a operação requisitada. Identificado o receptor da operação, constrói-se um diagrama de interação (figura 3.6) para especificar como os objetos interagem com outros para atingir o resultado da operação.

Para construir este diagrama de interação:

- Use os *snapshots* criados na análise para definir a configuração inicial do objeto.
- Identifique as requisições recebidas por cada objeto como uma operação sobre esta classe.
- Identifique os atributos de classe para indicar o que cada classe precisa considerar antes e depois de cada requisição recebida.

Notação:

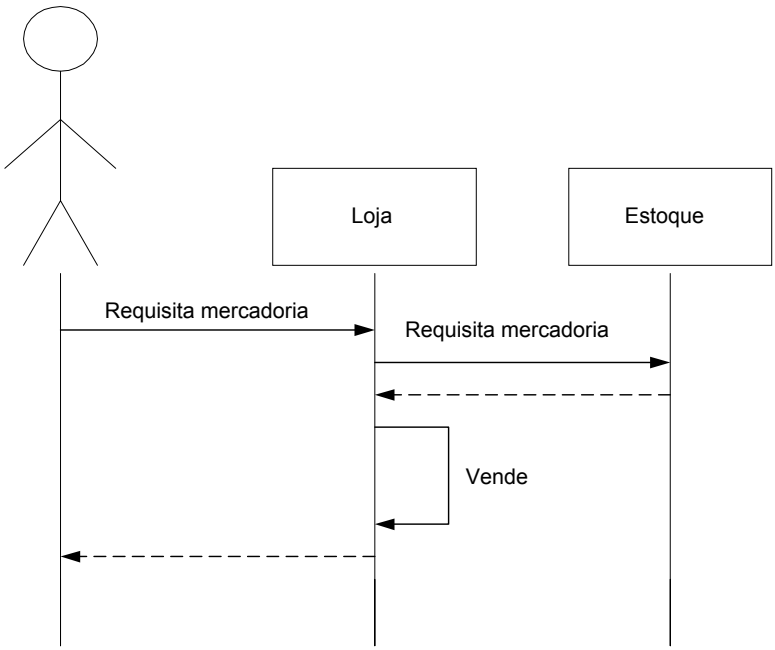

Figura 3.6 - **Diagrama de Interação**

Pode-se agora começar a definir o modelo de classes (figura 3.7). Este consiste das classes que compõe o sistema, seus atributos e operações e as associações entre eles. Na parte superior da representação de uma classe descreve-se o nome da classe, enquanto na parte intermediária são descritos os atributos e na parte inferior as operações.

Notação:

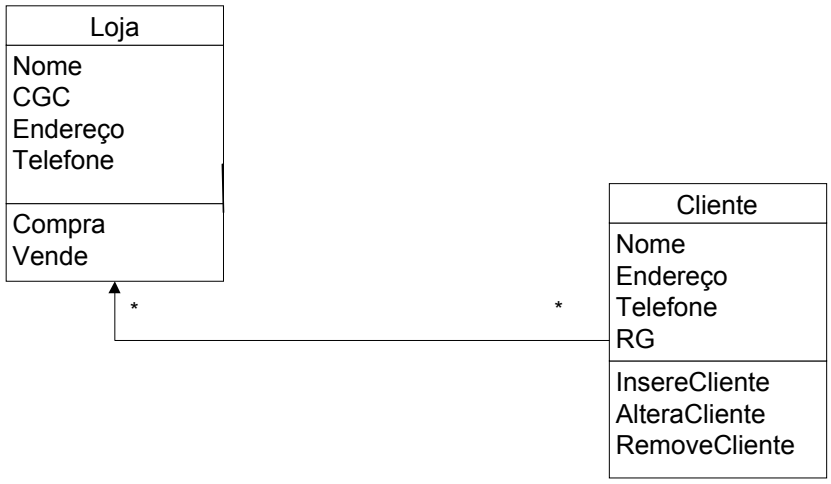

Figura 3.7 - **Diagrama de Classes**

Associado a isso existiria o diagrama de estados representado por *statecharts* [Har96] que representariam os estados de uma classe (Figura 3.8), ou seja, quais as condições de uma classe após a realização de transição – que em Catalysis correspondem a ocorrência de uma operação.

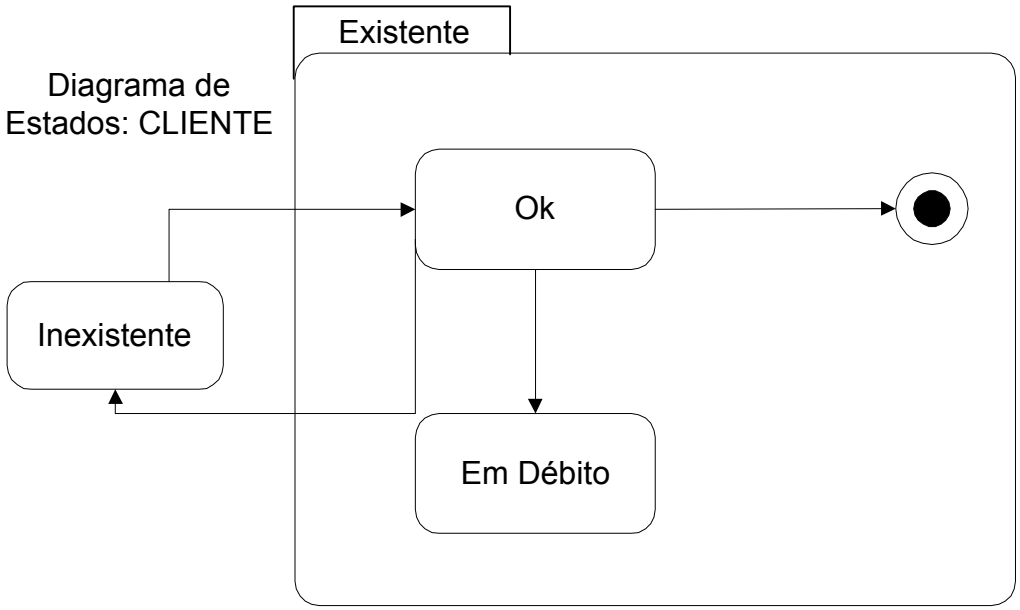

Figura 3.8 - **Diagrama de Estados (*Statechart)***

3.4 Pacotes

Realizada a especificação de um sistema, pode-se dar a construção de pacotes (figura 3.9). O pacote é a unidade básica de desenvolvimento de um produto.

Ele tem como finalidade a reunião de informações que estejam associadas para que um usuário possa utilizar-se de apenas parte de um componente, ou seja, ele faria uso apenas do pacote. Outro aspecto que justificaria seu uso é a facilidade de controle no desenvolvimento a partir de unidades básicas que possuem uma certa independência de outros pacotes.

O pacote é a coleção de todas as classes definidas e das operações sobre elas, seja em forma de diagramas ou de texto. Toda modelagem é feita dentro de algum pacote. Ele poderia conter tipos e classes, código fonte ou compilado, grupos de classes que não fariam sentido uma sem a outra, diagramas, outros pacotes, *frameworks* e etc...
Exemplo da notação de um pacote:

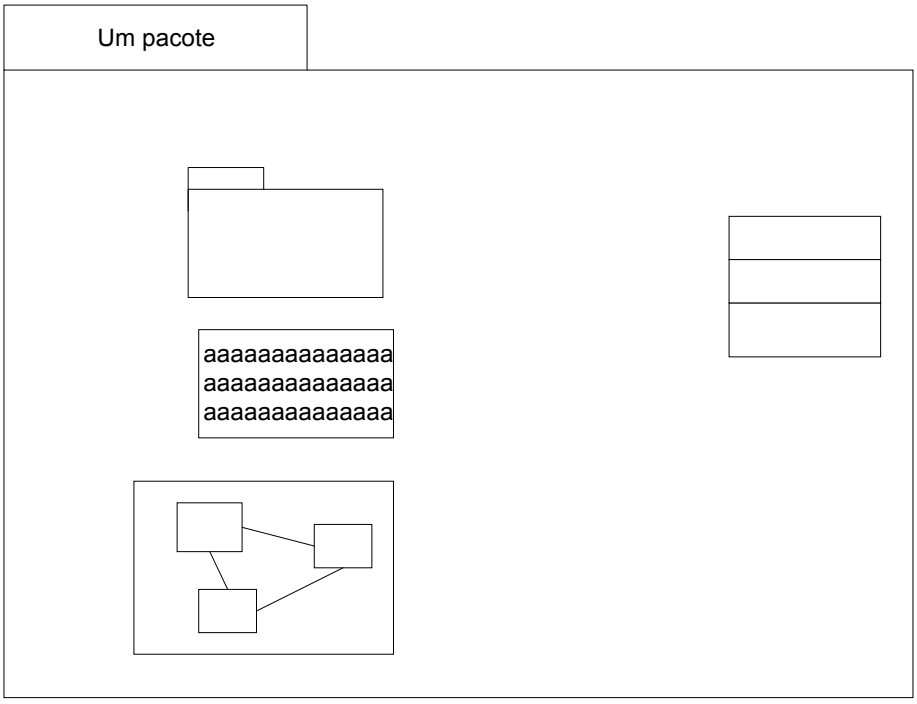

Figura 3.9 - **Pacote**

3.5 Considerações Finais

Neste capítulo foram apresentados os conceitos de Catalysis, fazendo seus recursos irem ao encontro da proposta do desenvolvimento baseado em componentes. Foram demonstrados os diagramas que compõem a metodologia e qual a função de cada um deles na especificação de um sistema.

Notou-se o suporte ao desenvolvimento baseado em componentes através da construção do modelo de tipos – e eventualmente, de pacotes – que serão analisados com mais detalhes no capítulo 5.

Capítulo 4

O Sistema Exemplo

4.1 Introdução

Para que se pudesse aplicar a teoria referente ao método Catalysis realizou-se a especificação de um sistema exemplo. O sistema escolhido foi de reserva de passagens aéreas. Este sistema, mesmo com as simplificações propostas para sua modelagem, mostrou-se adequado para o desenvolvimento baseado em componente, já que ele pode ser usado também por uma empresa de turismo, por exemplo, que utilize esse sistema associado a um sistema de reserva em hotéis ou qualquer outra funcionalidade pertinente a ela.

Como descrito no Capítulo 3, foram desenvolvidos: diagrama de contexto, cenários, *snapshots*, especificação das transações, modelo de tipos, diagrama de interação, diagrama de classe e diagrama de estados. A seguir será apresentado um exemplo de cada diagrama desenvolvido. No anexo A serão apresentados os demais diagramas.

4.2 O Sistema Exemplo

O sistema exemplo modelado é hipotético e tratará de reservas de passagens para um determinado aeroporto X. Como tal, este sistema atenderá a reserva e venda para todos os vôos e empresas disponíveis em X.

O passageiro, quando desejar efetuar uma reserva, deve informar ao sistema o destino e a data desejada para a viagem. O sistema automaticamente apresentará opções para que o passageiro escolha o horário, a classe e a empresa de sua preferência. No caso de não existirem vagas o passageiro pode optar por ser incluído na lista de espera e desta forma obter uma reserva tão logo se torne disponível uma vaga em um vôo. Em ambos os casos o passageiro será cadastrado na lista de passageiros.

Se a reserva for concretizada o passageiro deve fornecer ao sistema, dados básicos a seu respeito como nome, RG, endereço e telefone.

Encerrada a reserva, o sistema fornece um número que poderá ser utilizado para, eventualmente, ter acesso ao sistema e modificar os itens desejados da reserva. Esse número será usado também para a obtenção do PTA tão logo seja confirmada a reserva por parte do passageiro. Essa confirmação se dará através do pagamento da passagem.

O sistema manterá também uma base de dados com os vôos disponíveis em X. Esta base será utilizada para consulta dos horários de vôos disponíveis, e também para atualização dos lugares disponíveis em cada vôo, conforme forem acontecendo as reservas. As empresas terão acesso a esta base de dados para cadastro, ou seja, inserção, remoção e alteração de seus vôos, e o sistema deve dar suporte para isso.

4.3 Diagramas

A seguir são apresentados um exemplo de cada um dos diagramas desenvolvidos.

4.3.1 Diagrama de Contexto

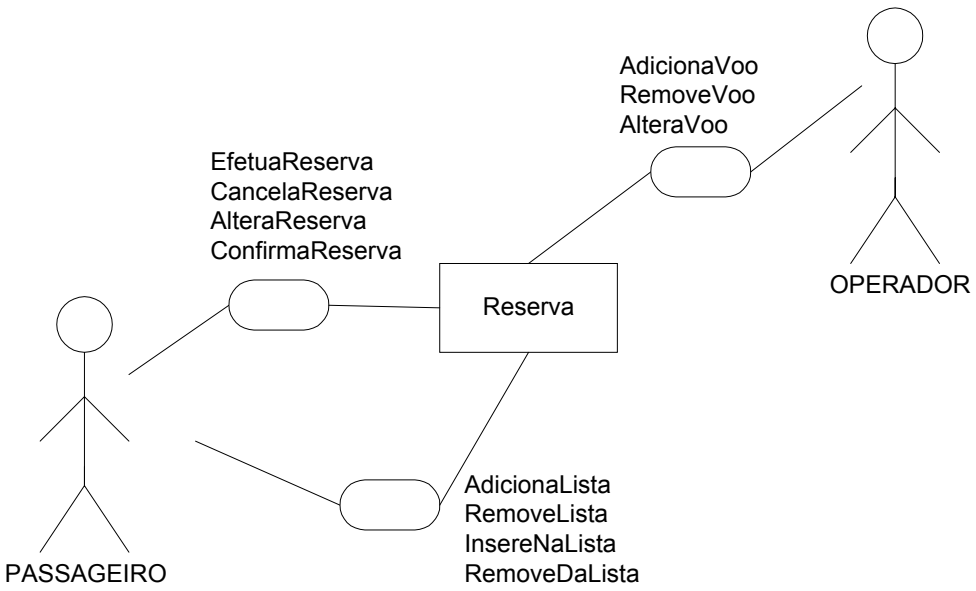

Figura 4.1 - **Diagrama de Contexto**

Através desse diagrama (Figura 4.1) pode-se notar que, em contextos diferentes, a interação entre o passageiro e o sistema pode gerar as operações referentes à reserva ou à lista de espera, e por esse motivo são representados separadamente.

4.3.2 Cenários

EfetuaReserva / InserePassageiro

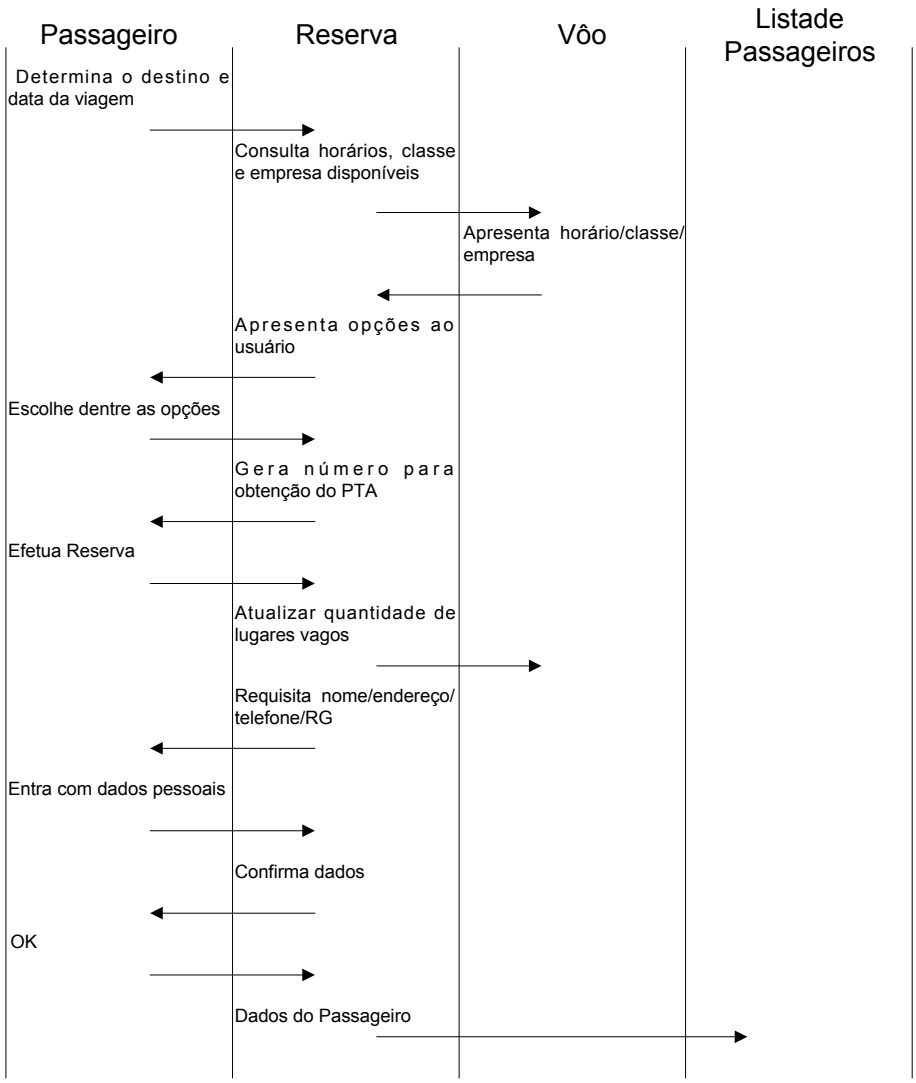

Figura 4.2 - **Cenário (EfetuaReserva)**

Através deste diagrama (Figura 4.2) podemos observar todos os passos, ou seja, todas as **ações** realizadas entre os atores para desempenhar a operação de "EfetuaReserva", bem

como **quais** são os atores que participam desta operação. Neste diagrama, todas as interações passam pelo sistema de reserva.

Note que a operação "InserePassageiro" é representada no mesmo cenário, por essa ser uma consequência da operação de "EfetuaReserva". Isso ocorrerá em outros diagramas pelo mesmo motivo.

4.3.3 Snapshots */ Especificação das transações*

EfetuaReserva

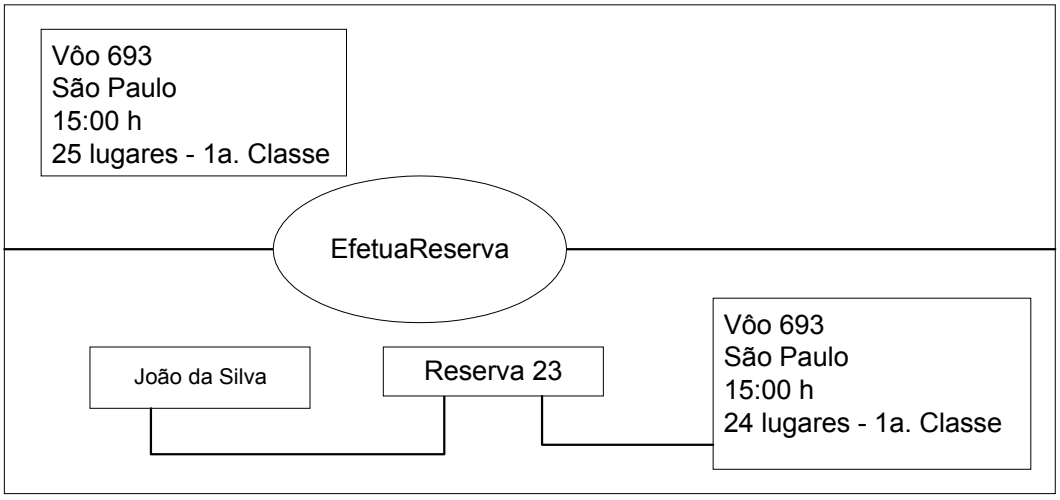

Figura 4.3 - ***Snapshot*** **(EfetuaReserva)**

Operação EfetuaReserva (num_reserva, num_voo, nome_passageiro)
Pré: O vôo desejado está cadastrado e com disponibilidade de lugares (lugares>0)
Pós: Passa a existir um registro com a reserva do passageiro requisitante para o vôo requisitado, que passa a ter um lugar disponível a menos (lugares=lugares-1)

Especificadas as operações foi construído o dicionário de dados, uma tabela que descreve como os atributos são criados e modificados.

Dicionário:

Tipo	Descrição		Ações que o criam
	Atributos	Descrição	Ações que o escrevem
Reserva	Registro que estabelece uma reserva para determinado vôo.		EfetuaReserva
	Nome_passageiro	Atributo: Nome do passageiro para o qual a reserva foi efetuada	InserePassageiro
	num_voo	Atributo: Vôo para o qual a reserva foi feita	AdicionaVôo
	Data	Atributo: Data para a qual a reserva foi feita	EfetuaReserva
	Classe	Atributo: Classe para a qual a reserva foi feita	EfetuaReserva
	Número_reserva	Atributo: Número da reserva	EfetuaReserva
	PTA	Atributo: Identificação do passageiro para o vôo	ConfirmaReserva
Passageiro	Registro que cadastra um passageiro que esteja efetuando uma reserva.		InserePassageiro
	Nome_passageiro	Atributo: Nome do passageiro que está efetuando a reserva	InserePassageiro
	Rg_passageiro	Atributo: RG do passageiro que está efetuando a reserva	InserePassageiro
	End_passageiro	Atributo: Endereço do passageiro que está efetuando a reserva	InserePassageiro
	Tel_passageiro	Atributo: Telefone do passageiro que está efetuando a reserva	InserePassageiro
Vôo	Registro que cadastra um vôo de forma a disponibilizá-lo para reservas		AdicionaVôo
	NomeEmpresa	Atributo: Nome da empresa que fornece o vôo	AdicionaVôo
	Num_vôo	Atributo: Número do vôo cadastrado	AdicionaVôo
	Destino_vôo	Atributo: Destino do vôo cadastrado	AdicionaVôo
	Horário_vôo	Atributo: Horário de saída do vôo	AdicionaVôo
	Lugares	Atributo: Quantidade de lugares vagos no vôo	AdicionaVôo/ EfetuaReserva

ListaEspera	Base de dados que armazena todos os passageiros que estão aguardando liberação de lugar em algum vôo		AdicionaLista
	Destino	Atributo: Destino para o qual se deseja uma reserva	AlteraLista
	Data_lista	Atributo: Data para a qual se deseja uma reserva	AlteraLista
	Horário_lista	Atributo: Horário para o qual se deseja uma reserva	AlteraLista
	Nome_passageiro	Atributo: Nome do passageiro que deseja uma reserva	InserePassageiro

4.3.4 Modelo de Tipos

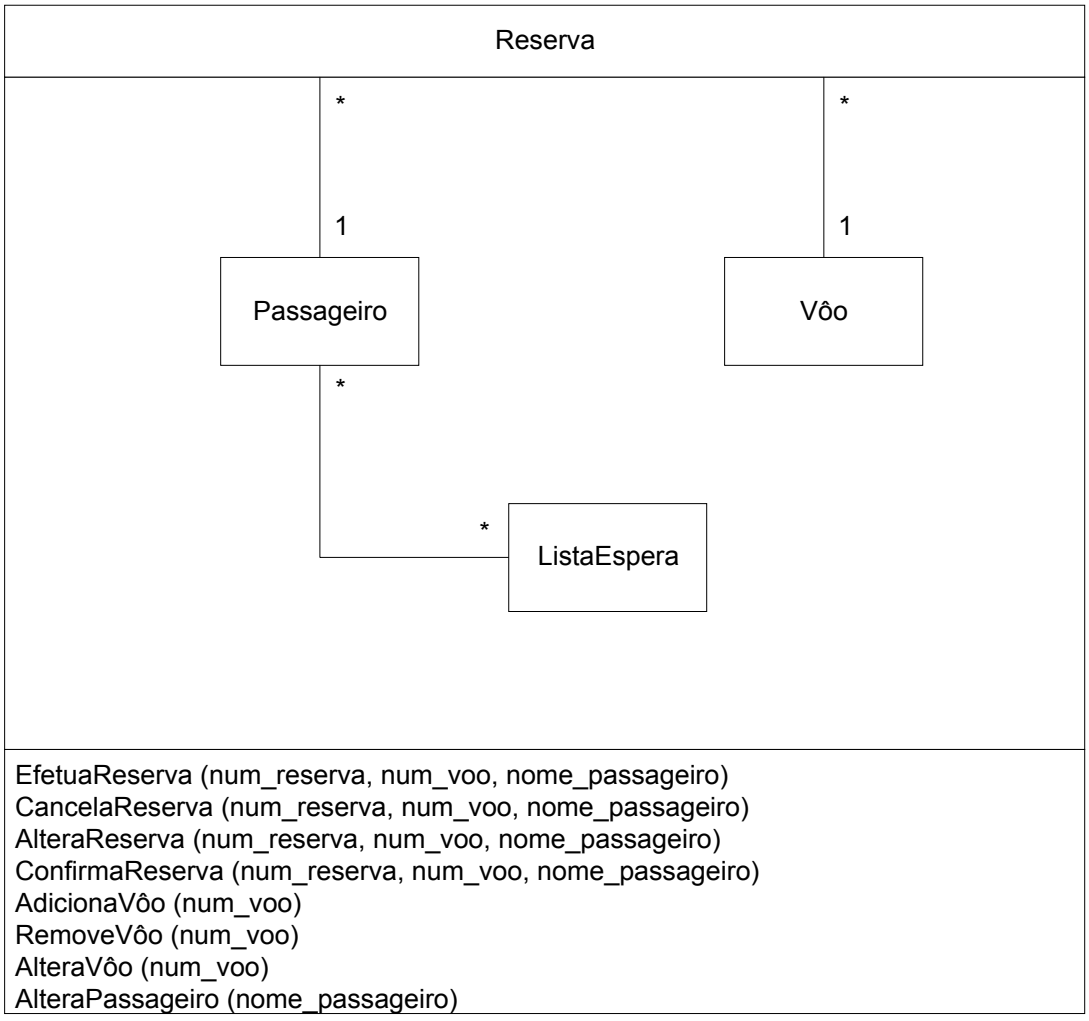

Figura 4.4 - **Modelo de Tipos**

4.3.5 Diagrama de interação

Nesses diagramas, a notação é muito semelhante à de cenários, no entanto semanticamente possuem diferenças. No diagrama de interação representamos as operações e respostas - setas tracejadas - a essas operações por parte dos atores do sistema. Em termos práticos, uma representação de operação feita por um diagrama de interação pode ser detalhada em termos de ações que compõem esta operação (cenários).

As setas que saem de um ator e retornam a ele mesmo, formando um ciclo, representam ações deste ator que afeta a ele mesmo, em seus atributos e/ou estados.

De modo a facilitar a compreensão, ao longo do desenvolvimento dos diagramas foram representados não somente as requisições de operações mas também as requisições de dados e etc...

EfetuaReserva

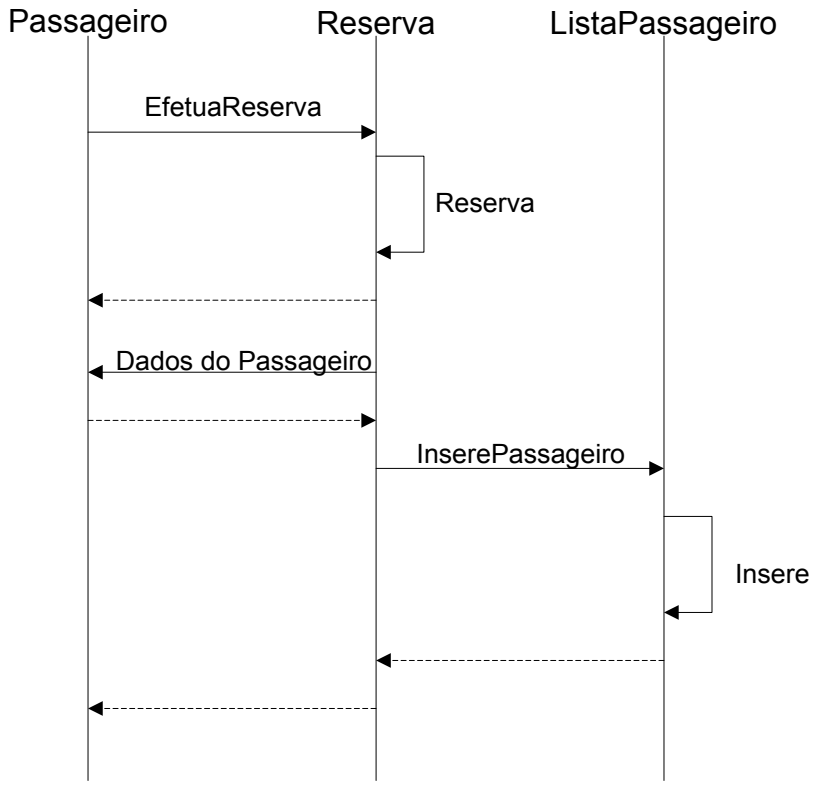

Figura 4.5 - **Diagrama de interação (EfetuaReserva)**

Nesse diagrama (Figura 4.5) ListaPassageiro é uma base de dados e a operação InserePassageiro, sobre ele identifica uma entrada de dados que é executada e retorna uma resposta que, no caso, simboliza a concretização da inserção. Os ciclos Reserva e Insere podem ser visualizados detalhadamente em uma consulta ao cenário correspondente a esse diagrama de interação.

4.3.6 Diagrama de estados

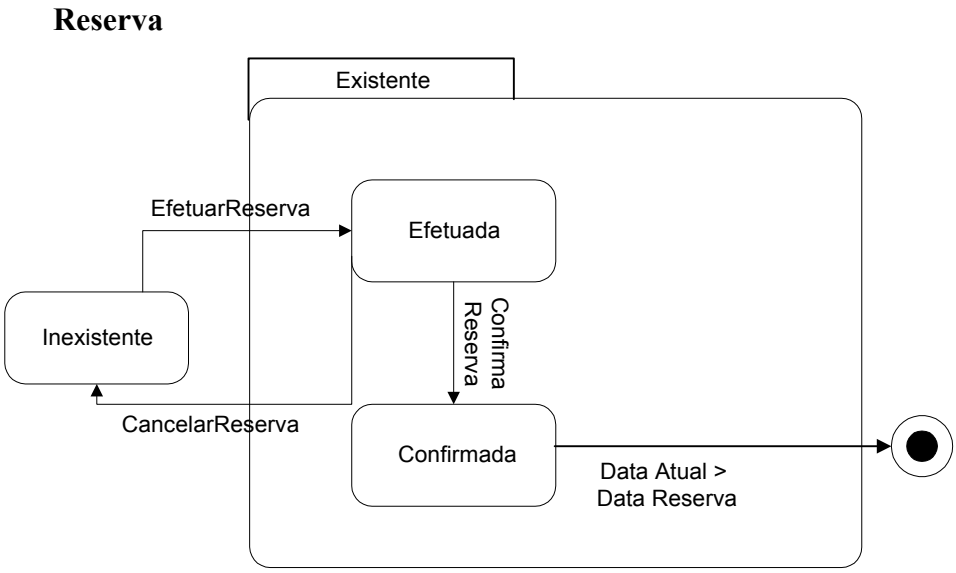

Figura 4.6 - **Diagrama de Estados (Reserva)**

Este diagrama de estados da classe Reserva (Figura 4.6) representa, a princípio dois estados: existente ou inexistente. No entanto, quando a reserva é existente, ela pode ser ainda, efetuada ou confirmada. Com a realização das transições – operações – a classe pode mudar de estado, conforme representado. Quando a data atual ultrapassa a data programada para a reserva, a instância da classe é destruída.

4.3.7 Diagrama de classes

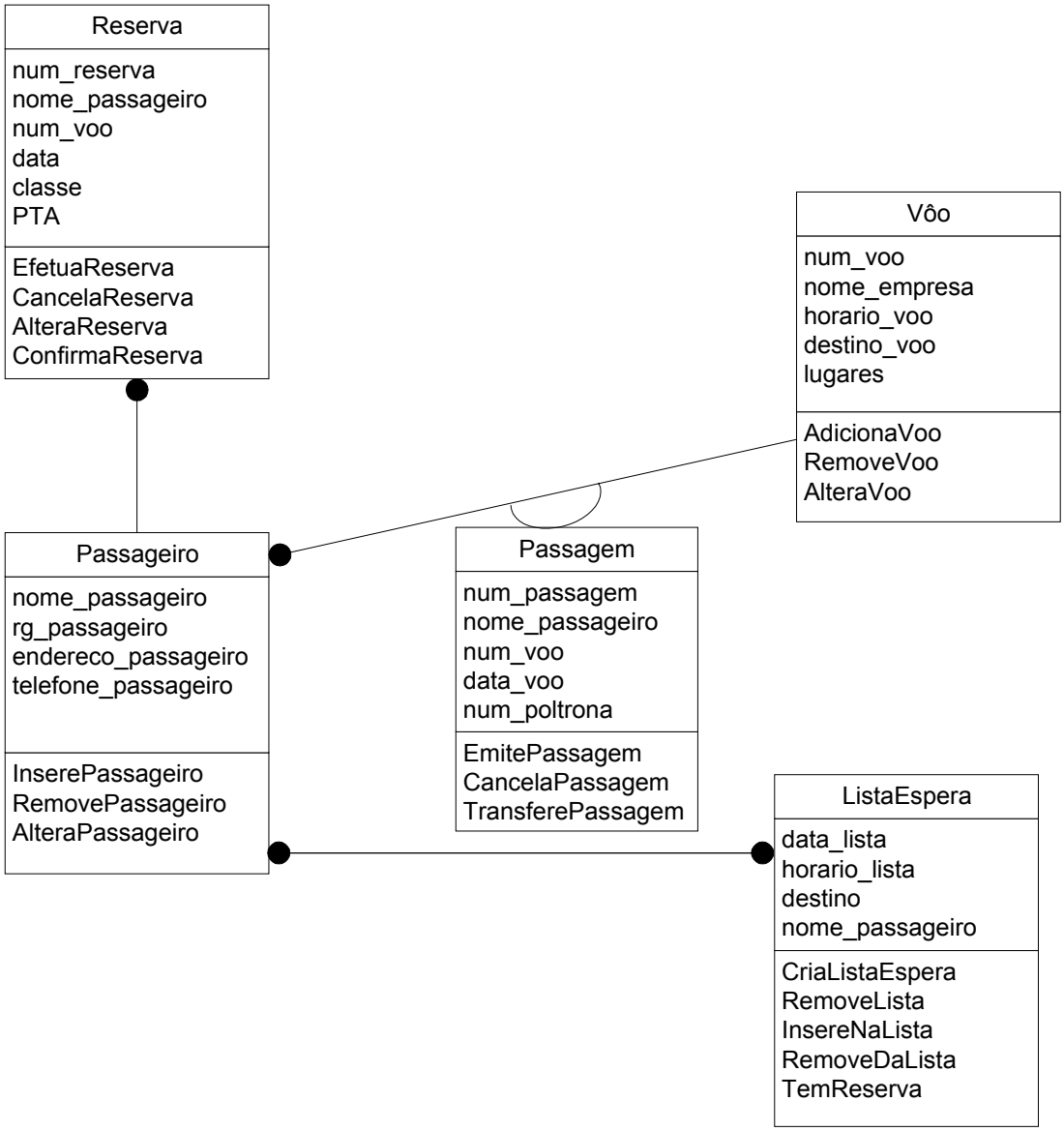

Figura 4.7 - **Diagrama de classes**

4.4 Considerações Finais

O sistema de reserva de passagens aéreas modelado, foi simplificado quanto às suas atribuições, no entanto mostrou-se satisfatório para a aplicação dos recursos do método Catalysis, e desta forma, prover o material necessário para sua análise, que será apresentada no capítulo 5.

De acordo com os conceitos de componente apresentados no capítulo 3, este sistema pode funcionar como um sistema independente, ou como um componente associado a outros componentes que teriam a funcionalidade de reserva em hotéis para os passageiros, ou de aluguel de carros quando o passageiro chegasse à cidade destino e uma série de outros componentes que expandiriam a utilidade deste sistema.

Capítulo 5

Experiência na Utilização do Método

Como definido anteriormente o objetivo primordial deste trabalho é realizar uma análise da metodologia Catalysis a fim de tentar traçar um perfil de seus pontos fracos e fortes. Os aspectos aqui abordados não a tornam melhor ou pior que outra metodologia, ou seja, a análise foi feita em termos absolutos para evitar uma comparação em relação a outras que não foram objetos de estudo. A análise aqui tratada foi realizada com base na experiência de utilização do método. Desta forma foram avaliados aspectos acerca da aprendizagem do método, dos recursos proporcionados para a modelagem, do suporte dado pelo método à reusabilidade e por fim, algumas considerações à respeito do processo de desenvolvimento e restrições impostas quanto aos recursos do método.

5.1 Aprendizado do método

O primeiro aspecto considerado foi com relação à aprendizagem da metodologia, ou seja, se este se constituiria em um ponto negativo para os desenvolvedores ou não. Para o desenvolvimento deste trabalho existiu este problema devido à pouca disponibilidade de material e a falta de didática do material existente. No entanto, é fácil concluir, pela descrição aqui apresentada, que a partir de um material bem elaborado o aprendizado pode tornar-se uma tarefa além de fácil, agradável, principalmente para aqueles que já tem conhecimento de UML, ou já tem contato direto com outras metodologias orientadas a objeto.

O conhecimento de outras metodologias orientadas a objeto facilita o aprendizado já que Catalysis adota muitos aspectos conceituais e práticos dessas metodologias, como por exemplo a existência do diagrama de classes – desenvolvido da mesma forma que nos métodos OO – e outros diagramas, como os cenários e o diagrama de estados.

Em relação ao diagrama de estados, apesar deste estar presente em outras metodologias, é um ponto desfavorável ao aprendizado de Catalysis, já que o conhecimento de *statecharts* é desejável para a confecção destes diagramas. Este conhecimento é desejável na medida

em que facilita o trabalho do desenvolvedor, que atravessaria essa fase da especificação sem maiores obstáculos.

5.2 Recursos para modelagem

Analisando os recursos proporcionados por uma modelagem utilizando Catalysis o primeiro aspecto considerado foi a visão do sistema. Essa visão refere-se a quanto pode-se abstrair sobre o funcionamento do sistema a partir de seus diagramas.

Pode-se notar que uma visão detalhada é favorecida à medida que temos diagramas que descrevem com bastante precisão detalhes do sistema, como por exemplo os requisitos e consequências de operações – através da especificação de operações em termos de pré e pós condições - ou mesmo as classes que interagem e qual a sequência de ações em uma operação – através dos cenários.

No entanto, essa visão muito detalhada acaba por prejudicar uma visão geral de como o sistema interage já que não se tem nenhum diagrama que represente todas as interações das classes simultaneamente. Exceção feita ao diagrama de classes que, no entanto, apenas descreve quais classes que estão relacionadas, independente de quais operações as relacionam. Uma tentativa de se obter uma visão geral é através do diagrama de interação. No entanto, ele mostra o comportamento global para cada operação e não o comportamento de todas as operações.

Os demais recursos proporcionados pela modelagem referem-se aos recursos individuais de cada diagrama, que podem ser sumarizados da seguinte maneira:

Diagrama de contexto – Este diagrama relaciona todos os atores envolvidos nas operações que o sistema se dispõe a realizar. Assim o desenvolvedor passa a ter a visão do escopo do sistema, isto é, do problema a ser resolvido.

Cenários – Determinado o escopo, através dos cenários, como o próprio nome sugere, o desenvolvedor terá a descrição de quais passos compõem cada operação, ou seja, quais as

ações que serão praticadas pelos integrantes do sistema (atores, bases de dados e o próprio sistema) para atingir o resultado desejado.

Especificação das operações – Determina uma operação em termos de pré e pós condições. Desta forma, o usuário terá uma descrição dos requisitos de uma operação e quais as consequências desta. Agem juntamente com os *snapshots* para proporcionar esta visão.

Snapshots – Como dito anteriormente, tem a função de proporcionar, em uma representação gráfica, os efeitos de uma operação e analisados em conjunto, proporciona a visão dos requisitos da operação. Isso pode ser visto, no caso do sistema modelado, para a efetivação de uma reserva. A visualização do estado das classes e atributos antes da operação de EfetuaReserva e CriaListaEspera é a mesma, com exceção da quantidade de lugares vagos que é igual a 0 no segundo caso. Assim, analisados em conjunto, esses *snapshots* provêem informações sobre quais são os requisitos para ambas as operações.

Modelo de Tipos – Este diagrama praticamente define o sistema como um componente, ou seja, representa o sistema como um tipo e quais operações são disponibilizadas para o usuário externo, definindo em grande parte qual a interface do componente.

Diagrama de interações – Este diagrama representa a interação entre as classes para atingir o resultado de uma operação. Pode ser definido como uma generalização dos cenários. Desta forma, um diagrama de interações representa a comunicação entre classes e as respostas geradas desta comunicação, enquanto, para um detalhamento de quais ações são realizadas para se ter esta comunicação, consulta-se os cenários.

Diagrama de classes – Representa as classes com todos os seus atributos e operações, e ainda, a interação entre as classes e a cardinalidade desse relacionamento. Assemelha-se à representação interna do modelo de tipos, onde também estão representados todos os relacionamentos entre as classes do sistema.

Diagrama de estados – Representa os estados de uma classe. Este estado é alterado em função da mudança de atributos e/ou na situação da classe. O diagrama representa também

quais são as operações que causam a mudança no estado. Desta forma tem-se uma representação da dinâmica do funcionamento do sistema com a ocorrência das operações.

5.3 Suporte à reusabilidade

A grande questão que analisa-se quanto ao desenvolvimento baseado em componentes é com relação à reusabilidade. Desta forma, Catalysis deve fornecer mecanismos que dêem suporte à reusabilidade do componente especificado.

Este suporte é fornecido principalmente através do diagrama de tipos. Este diagrama define o sistema (componente) modelado como um tipo. Representa-se quais as classes que estão a ele relacionadas a fim de representar um aspecto interno do sistema e que é irrelevante para seus objetivos.

São especificados também quais as operações que este componente provê externamente, isto é, através de quais operações ele fará sua comunicação com componentes externos. Pode-se notar que esta é justamente a característica da interface de um componente.

5.4 Suporte ao processo de desenvolvimento

Os autores de Catalysis propõem como um processo de desenvolvimento, uma divisão em três níveis – como descrito no Capítulo 3. No entanto, verificou-se que algumas alterações nessa divisão melhoram a compreensão do desenvolvedor para a especificação de um sistema. Estas alterações são descritas a seguir.

No item 5.2 foi colocada a relação existente entre os cenários e os diagramas de interação. Portanto, o desenvolvimento em conjunto, ou em sequência, desses dois diagramas facilita o entendimento do desenvolvedor com relação às interações entre as classes.

Outra alteração é na construção do diagrama de tipos, que deveria ser efetuada tão logo sejam identificadas as operações – através do diagrama de contexto e dos cenários – para que o desenvolvedor já saiba qual o perfil do componente que está sendo modelado.

5.5 Mapeamento para a implementação

A passagem da especificação para a implementação tende a ser uma facilitada com o uso de Catalysis. Várias visualizações do sistema facilitam essa visão em termos de implementação. Entre as quais pode-se destacar: a especificação das transações, na qual se tem os requisitos para se realizar uma operação e quais os efeitos desta nos atributos e/ou estados de uma classe. E ainda os diversos diagramas - cenários, diagrama de interação – que representam quais as entidades que se relacionam para a realização das operações.

A seguir podemos verificar a declaração das classes do sistema onde podemos verificar a influência do diagrama de classes:

```
package sistema;

public class reserva{

      int numero;
      int num_voo;
      String data;
      String classe;
      String PTA;
      passageiro pr;

      public void EfetuaReserva(){}
      public void CancelaReserva(){}
      public void AlteraReserva(){}
      public void ConfirmaReserva(){}
}



package sistema;

public class passageiro {

      String nome;
      String rg;
      String endereco;
      String telefone;

      public void InserePassageiro(){}
      public void RemovePassageiro(){}
      public void AlteraPassageiro(){}
}

package sistema;
```

```
public class voo{

      String numero;
      String nome_empresa;
      String horario;
      String destino;
      String lugares;

      public void AdicionaVoo(){}
      public void RemoveVoo(){}
      public void AlteraVoo(){}
}


package sistema;

public class ListaEspera{

      String data;
      String horario;
      String destino;
      passageiro ple;

      public void CriaListaEspera(){}
      public void RemoveLista(){}
      public void InsereNaLista(){}
      public void RemoveDaLista(){}
      public void TemReserva(){}
}
```

Nessas declarações pode-se verificar que a classe “passageiro” será instanciada toda vez que “reserva” ou uma “ListaEspera” for instanciada – destacada em negrito - , o que foi identificado como requisito do sistema e desenvolvido ao longo de toda especificação.

Outro ponto da implementação que é facilitada com a especificação é a instanciação de classes no que se refere às operações. Percebe-se que não basta apenas a instanciação da classe a qual ela pertence e sim de todas as classes que participam da operação o que é visualizado no diagrama de interação.

5.6 Restrições

Se considerarmos que uma metodologia deveria prover suporte a todo o ciclo de vida de um software, Catalysis deixa a desejar, já que não proporciona qualquer recurso, para

explicitamente, fazer a execução de testes. Seria necessário a elaboração de um plano de testes independentes para se verificar a validade de um sistema.

Uma outra restrição refere-se à especificação da distribuição de um software. Visto que o desenvolvimento baseado em componentes prevê a integração de um software distribuído, a metodologia deveria prever essa distribuição de forma mais transparente, já que a única maneira de se prever a distribuição é através de pacotes, cujo desenvolvimento é opcional, não se tratando de um diagrama integrante de Catalysis.

Uma solução possível para isso seria que os usuários futuros considerassem a construção de pacotes como parte integrantes do processo de desenvolvimento.

Capítulo 6

Conclusões

A crescente busca pela qualidade no software por parte do usuário final, faz com que os desenvolvedores busquem constantemente novas tecnologias para a construção mais rápida de aplicações que resultem em produtos com maior qualidade.

Uma destas tecnologias é o desenvolvimento baseado em componentes, que propõe a construção de componentes reusáveis – através da especificação de uma interface que faz com que o usuário se preocupe com o que o componente faz e não como faz – e de maior qualidade – utilizando-se componentes que já são utilizados no mercado e portanto devidamente testados.

A propósito, a reusabilidade e a qualidade são características muito em voga na produção de software hoje em dia, o que torna o desenvolvimento baseado em componentes uma tecnologia promissora no que se refere aos requisitos impostos pela sociedade à indústria do software.

O método Catalysis provê os recursos necessários para o suporte ao desenvolvimento baseado em componentes através de uma semântica clara e sem ambiguidades. O suporte é dado através, principalmente, do modelo de tipos, que define a interface do sistema modelado, isolando-o da implementação. Esse suporte também é possível através da construção de pacotes – capítulo 3 – onde pode-se definir partes do sistema modelado como um componente, característica que não se aplicou ao sistema representado nesse trabalho em virtude da simplificação do sistema. No caso de um sistema mais completo, como sugerido anteriormente, com a agregação de reserva de hotéis, esta característica poderia ser visualisada.

O método apresentou-se apropriado para o objetivo a que se dispõe, sendo o aprendizado, como descrito no capítulo 5, relativamente fácil. O processo de desenvolvimento, com as mudanças propostas neste capítulo, torna a tarefa de especificação agradável e compreensível para o desenvolvedor.

O que destaca Catalysis como metodologia é o rigor de seus conceitos e da sua notação, o que dificulta uma especificação com ambiguidades na interpretação de seus resultados. Esse rigor vai desde a definição clara e precisa do problema a ser resolvido até a utilização de métodos formais nos diagramas de classe e na especificação de operações.

Um trabalho futuro seria a utilização deste sistema em conjunto com outros componentes, como a integração com sistema de reserva de hotéis ou aluguel de carros para os passageiros e desta forma analisar um comportamento mais complexo e visualizar a construção e a utilidade dos pacotes.

A conclusão principal obtida com esse trabalho é que a utilização de metodologias adequadas para especificação facilita o desenvolvimento e permite zelar por características que conduzirão a produtos de melhor qualidade, o que significa dizer que serão cumpridas as maiores exigências de desenvolvedores e clientes no que se refere à produção de software.

Referências Bibliográficas

[Bro97] Brown, Alan W. & Morrow, Brian. *Component Based Development for the Enterprise* , http://www.jorvik.com/alan/oo-cbd.html. 1997.

[Coa90] Coad, P. e E. Yourdon. *Object-Oriented Analysis*. Prentice-Hall. 1990.

[DSo97] D'Souza Desmond, Wills Alan C. *Objects, Components, and Frameworks With UML – The Catalysis Approach.* Addison-Wesley Publishing Company. 1997.

[Har96] Harel, David. *Statecharts: a visual formalism for complex systems*. Science of Computer Programming. 1987.

[Hat87] Hatley, D. J. e I. A. Pirbhai, *Strategies for Real-Time System Specification*. Dorset House. 1987.

[Ico94] Icon Computing, Inc. *Catalysis* , http://www.iconcomp.com/papers/catalysis-flyer.html. 1994.

[Ico96] Icon Computing, Inc. *Catalysis* , http://www.iconcomp.com/catalysis/europe-tour.html. 1996.

[Ste97a] Sterling Software. *Compontent Based Development and Object Modeling* , http://www.cool.sterling.com/cbd/whitepaper/4.html. 1997.

[Ste97b] Sterling Software. *Component Based Development and Object Modeling* , http://www.cool.sterling.com/cbd/whitepaper/1.html. 1997.

[Ste97c] Sterling Software. *Component Based Development and Object Modeling* , http://www.cool.sterling.com/cbd/whitepaper/2.html. 1997.

[Ste97d] Sterling Software. *Component Based Development and Object Modeling* , http://www.cool.sterling.com/cbd/whitepaper/3.html. 1997.

[Pau95] Paulk Mark C., Weber Charles V., Curtis Bill, Chrissis Mary Beth. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley Publishing Company. 1995.

[Pre95] Pressman, Roger S. *Engenharia de Software* . Makron Books. 1995.

[War85] Ward, P. T. e S. J. Mellor, *Structured Development for Real-Time Systems*. Yourdon Press. 1985.

Bibliografia

[Bar97] Barn, Balbir & Brown, Alan W. Improving Software Reuse Through Component-Based Development Tools . http://jorvik.com/alan/icrs5/index.html. 1997.

[Sys95] Systems Techniques, Inc. *OO and Component Based Development: Where Do We Go From Here ?* . 1995.

Anexo A

Aqui estão representados os demais diagramas desenvolvidos, ou seja, o restante dos cenários, dos diagramas de interação e dos diagramas de estados. Estes diagramas em conjunto com aqueles apresentados no capítulo 4 constituem-se na documentação deste software exemplo.

CENÁRIOS

CancelaReserva / RemovePassageiro

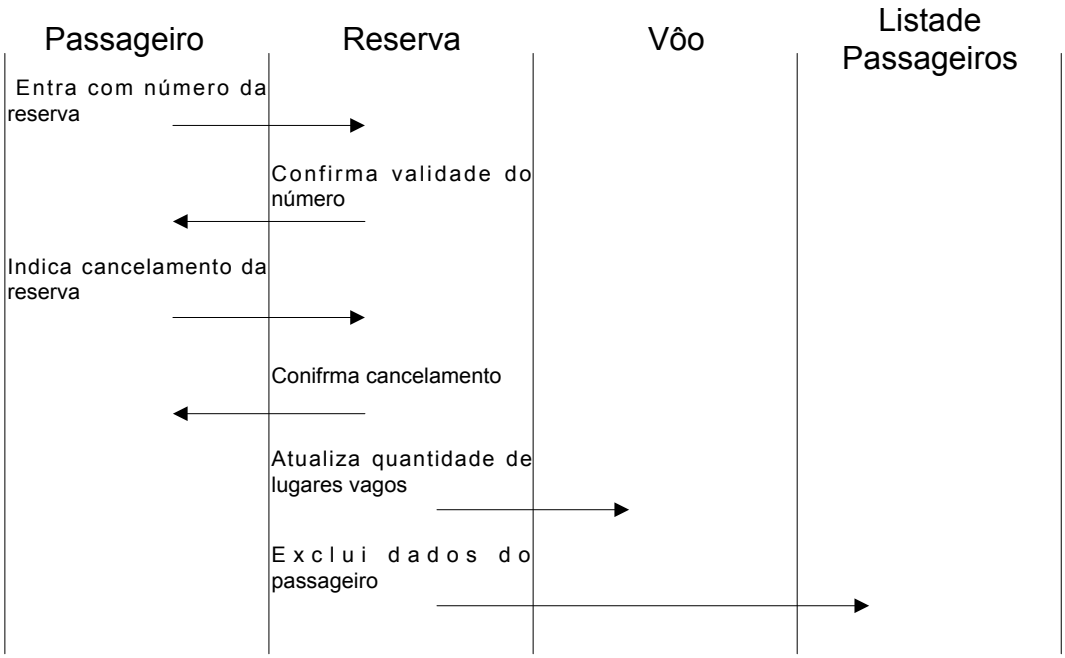

Figura A.1 – **Cenário (CancelaReserva)**

Através deste diagrama podemos observar todos os passos, ou seja, todas as **ações** realizadas entre os atores para desempenhar a operação de cancelar uma reserva, bem como **quais** são os atores que participam desta operação.

AlteraReserva

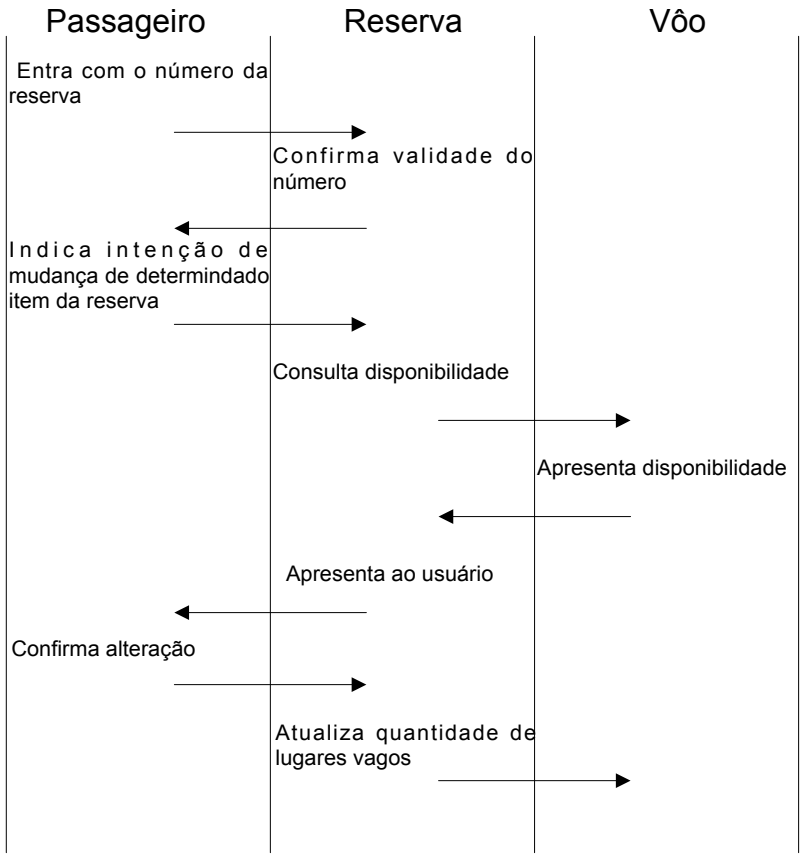

Figura A.2 – **Cenário (AlteraReserva)**

Neste diagrama, a alteração de uma reserva pode ser feita com relação a vários atributos, mas todas as possibilidades implicam em um outro vôo, logo, torna-se necessária uma consulta com relação à disponibilidade de lugares nesse vôo. Representou-se aqui apenas o caso em que esta consulta retorna a existência de lugares disponíveis. O caso em que não existem lugares, pode ser associado com o cenário CriaListaEspera ou InsereNaLista, ou seja, será efetuada a inserção do passageiro na lista de espera.

ConfirmaReserva

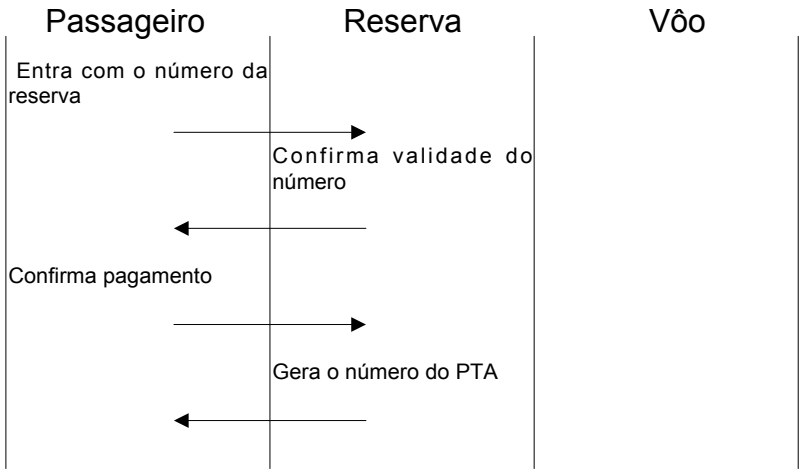

Figura A.3 – **Cenário (ConfirmaReserva)**

Nesse diagrama a confirmação da reserva, ou seja, o pagamento da passagem, resulta na geração do número do PTA.

AdicionaVôo

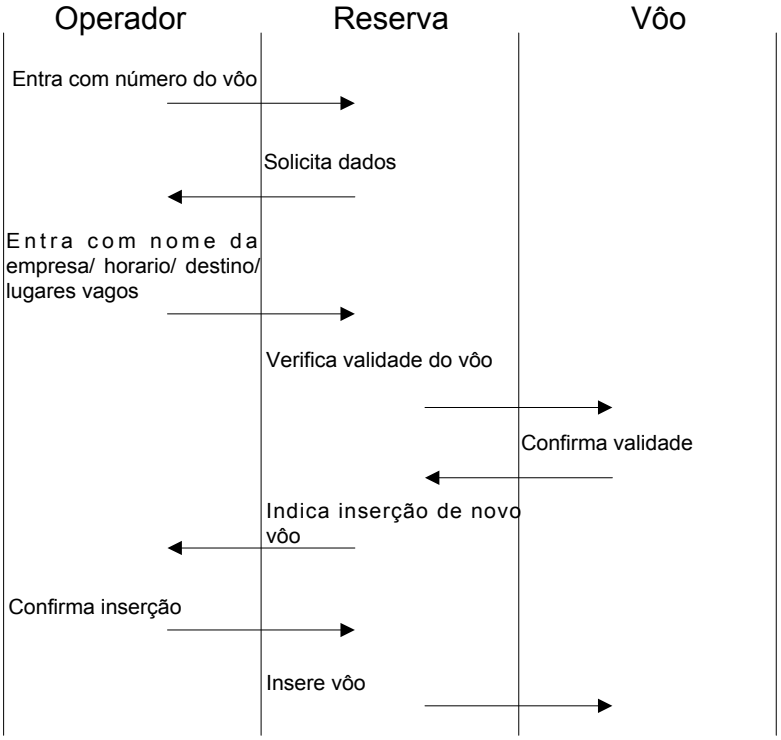

Figura A.4 – **Cenário (AdicionaVôo)**

Neste cenário, especificou-se a necessidade de se verificar a validade de um vôo, ou seja, se não existe um outro vôo no mesmo horário para o mesmo destino e outras inconsistências possíveis nesses casos.

RemoveVôo

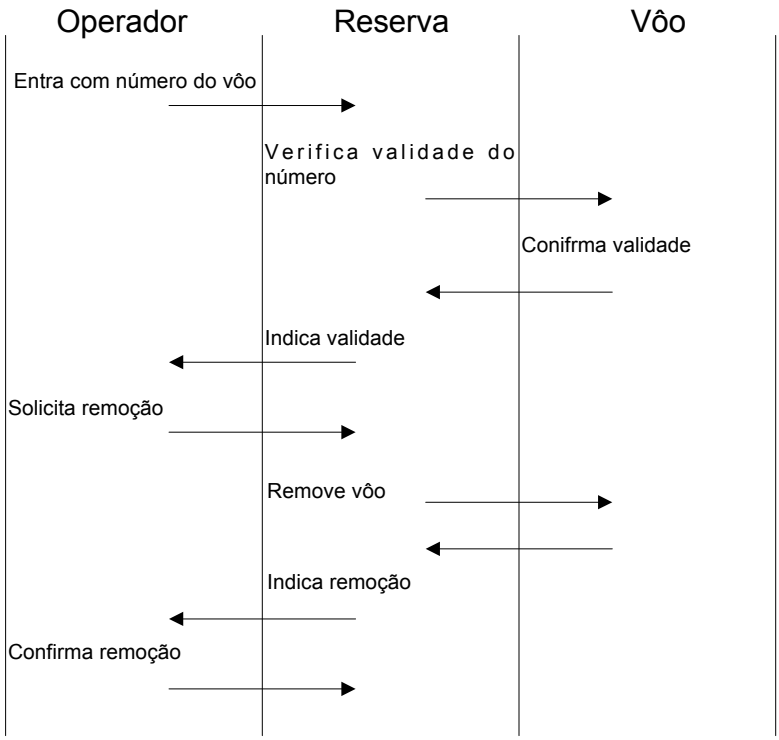

Figura A.5 – **Cenário (RemoveVôo)**

AlteraVôo

Figura A.6 – **Cenário (AlteraVôo)**

AlteraPassageiro

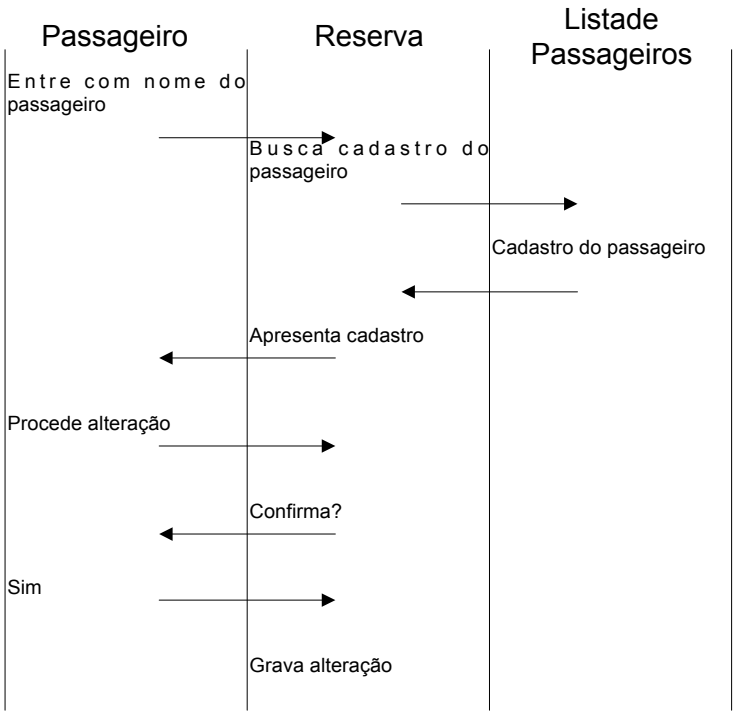

Figura A.7 – **Cenário (AlteraPassageiro)**

CriaListaEspera

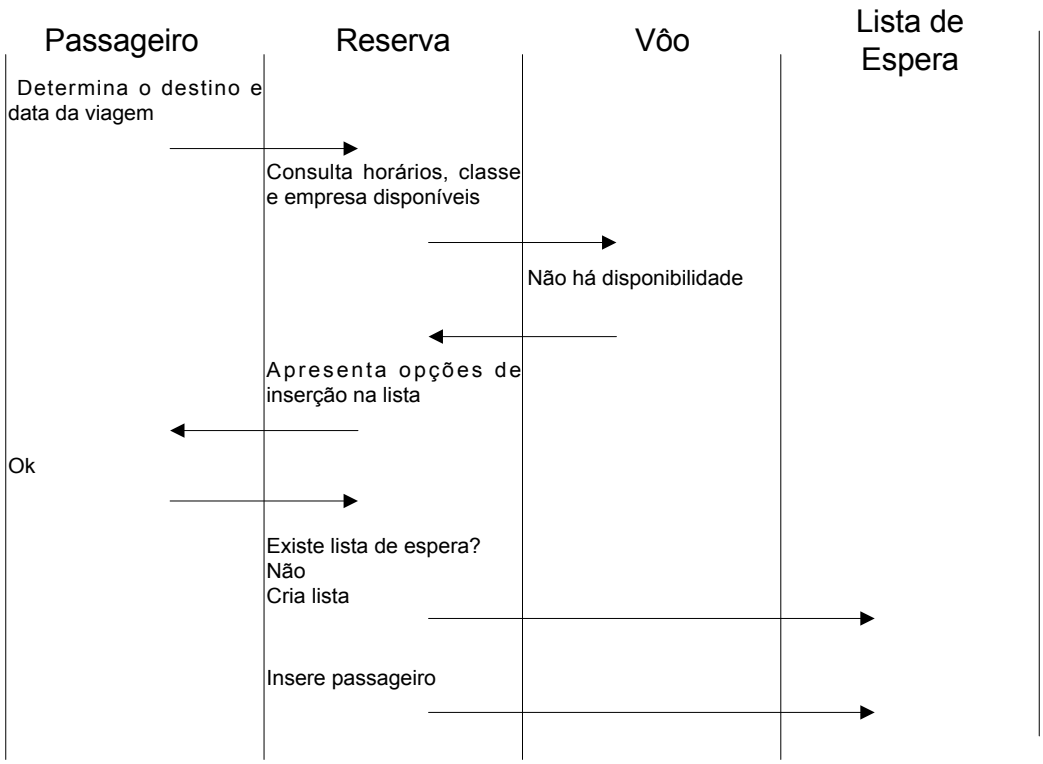

Figura A.8 – **Cenário (CriaListaEspera)**

Note que o início das ações nesse cenário são iguais àquelas do cenário de "EfetuaReserva" a diferença se dá na consulta da disponibilidade de lugares, sendo que aqui essa disponibilidade não existe.

RemoveLista

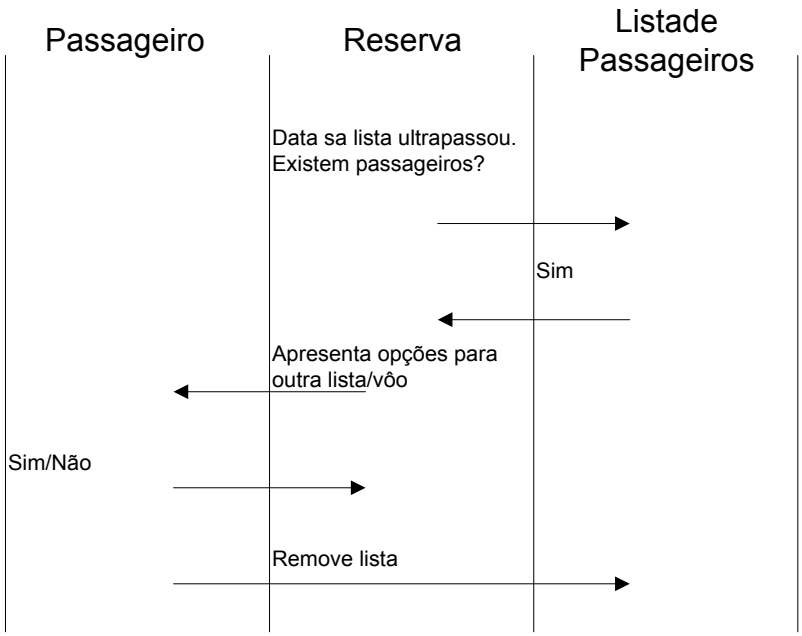

Figura A.9 – **Cenário (RemoveLista)**

Esse cenário representa a remoção da lista de espera. Isso ocorre sempre que a data para qual a lista de espera foi formada é ultrapassada. Nesse caso, representou-se algo que se aproximaria mais da realidade, a nível de implementação: a opção do passageiro ser incluído em outra lista com a data e horários mais próximos possíveis.

InsereNaLista

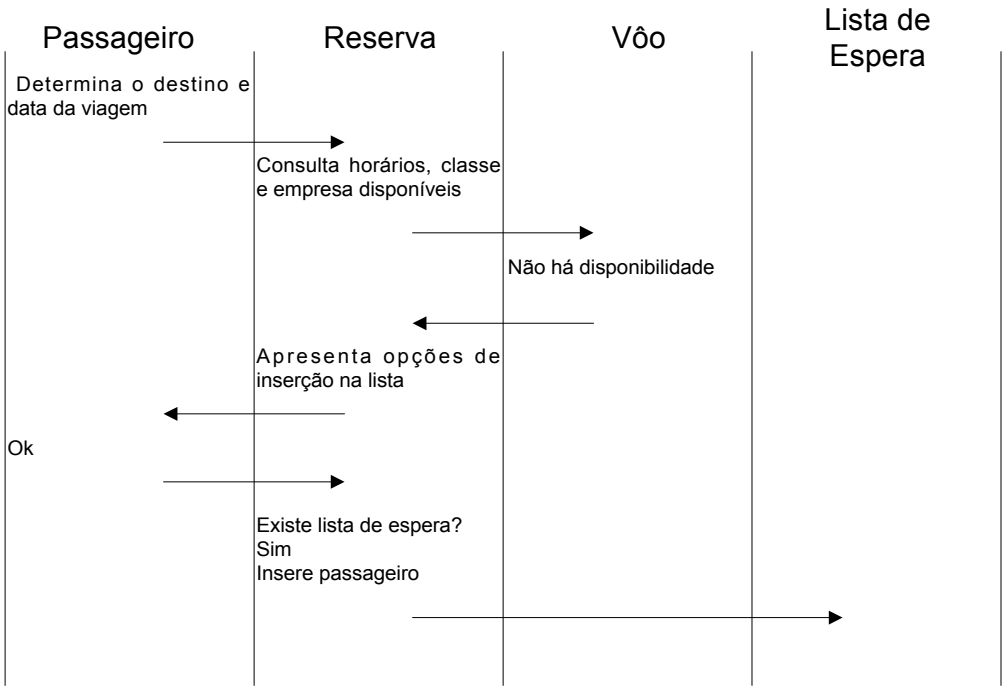

Figura A.10 – **Cenário (InsereNaLista)**

Esse cenário assemelha-se muito ao cenário de "CriaListaEspera" porém diferencia-se no fato de que aqui a lista de espera já existe, bastando a inserção do passageiro.

RemoveDaLista

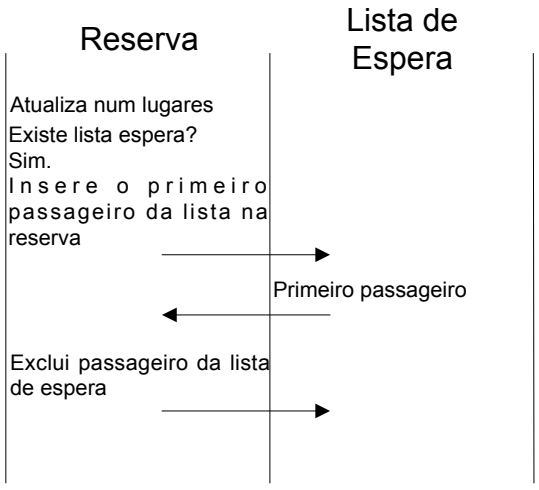

Figura A.11 – **Cenário (RemoveDaLista)**

A operação “RemoveDaLista” acontece quando um passageiro desiste de uma reserva e o primeiro passageiro da lista será inserido em seu lugar, sendo necessário para isso, que ele seja excluído da lista de espera. Outro caso de remoção não representado aqui por se tratar de um caso mais raro, é quando um passageiro simplesmente desiste de esperar um lugar em um vôo e solicita sua remoção da lista.

TemReserva

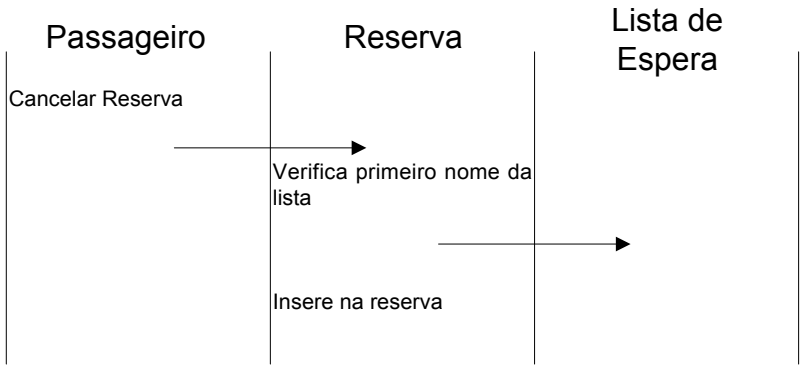

Figura A.12 – **Cenário (TemReserva)**

Essa operação é realizada em conjunto com a operação de “RemoveDaLista”, já que ocorre quando um passageiro é transferido da lista de espera para a reserva. Seria um passo posterior à remoção do passageiro da lista.

SNAPSHOTS / ESPECIFICAÇÃO DAS TRANSAÇÕES

InserePassageiro

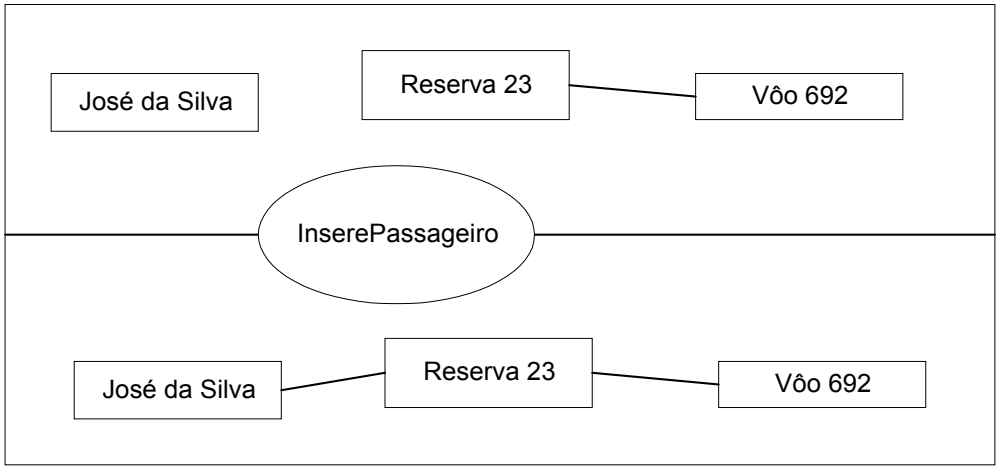

Figura A.13 – ***Snapshot*** **(InserePassageiro)**

Operação InserePassageiro (nome_passageiro, num_reserva)

Pré: A reserva está efetuada para o vôo requerido, restando apenas a inserção dos dados do passageiro requisitante. A base de dados "Lista de Passageiros" tem n passageiros cadastrados.

Pós: A base de dados "Lista de Passageiros" passa a ter n+1 passageiros cadastrados e a reserva é finalizada.

AdicionaVôo

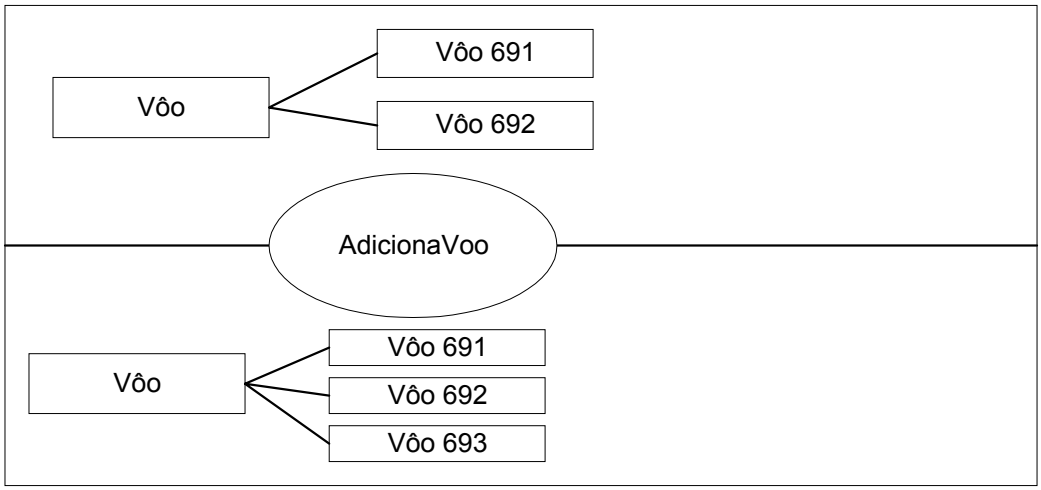

Figura A.14 – ***Snapshot*** **(AdicionaVôo)**

Operação AdicionaVôo (num_voo)

Pré: A base de dados possui n vôos cadastrados.

Pós: A base de dados passa a Ter n+1 vôos cadastrados.

CancelaReserva / RemovePassageiro

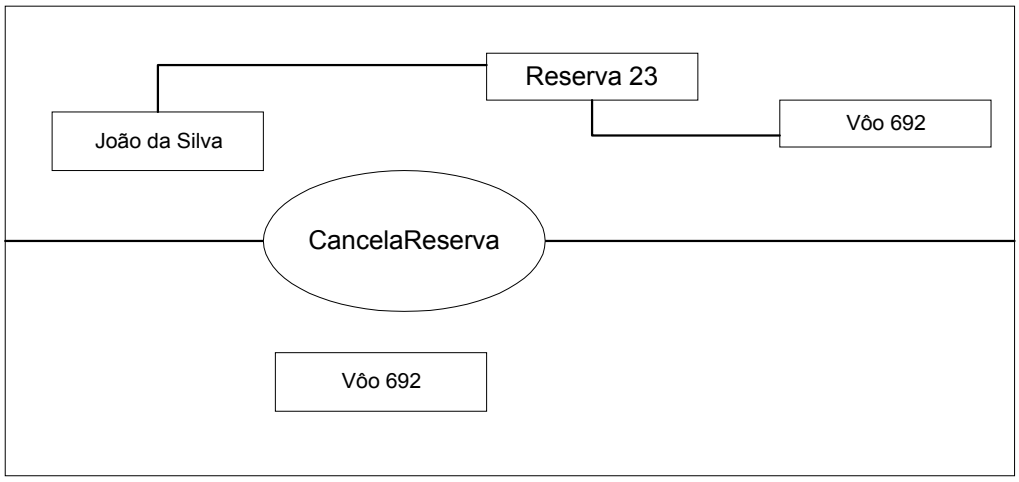

Figura A.15 – ***Snapshot*** **(CancelaReserva)**

Operação CancelaReserva (num_reserva, num_voo, nome_passageiro)

Pré: Existe um registro confirmando a existência de uma reserva para o vôo.

Pós: O registro e dados do passageiro são excluídos e o vôo passa a ter um lugar disponível a mais.

AlteraReserva

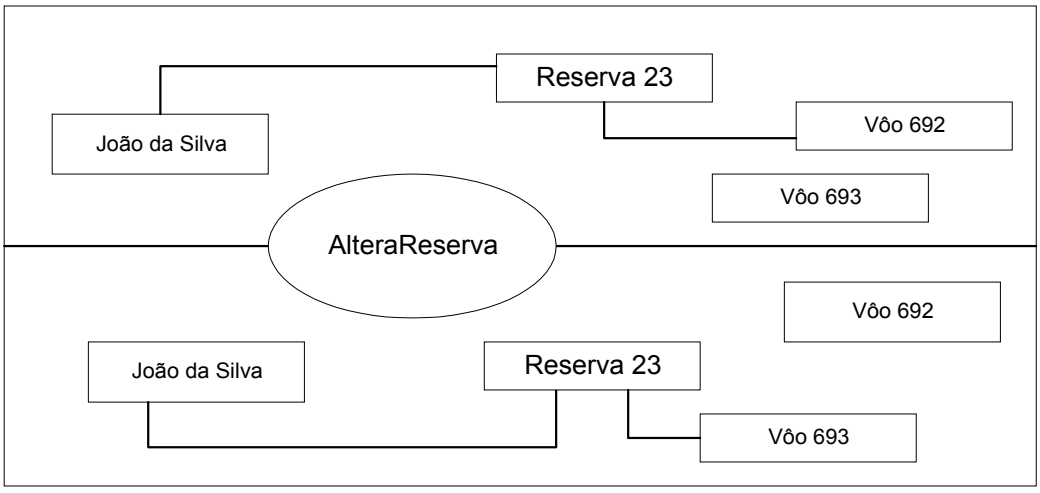

Figura A.16 – ***Snapshot*** **(AlteraReserva)**

Operação AlteraReserva (num_reserva, num_voo, nome_passageiro)

Pré: Existe uma reserva para um determinado vôo (x) em nome do passageiro e existe disponibilidade de lugares para um outro vôo requisitado(y).

Pós: A reserva passa a ser para o vôo y, que passa a ter um lugar disponível a menos, enquanto o vôo x passa a ter um lugar disponível a mais.

ConfirmaReserva

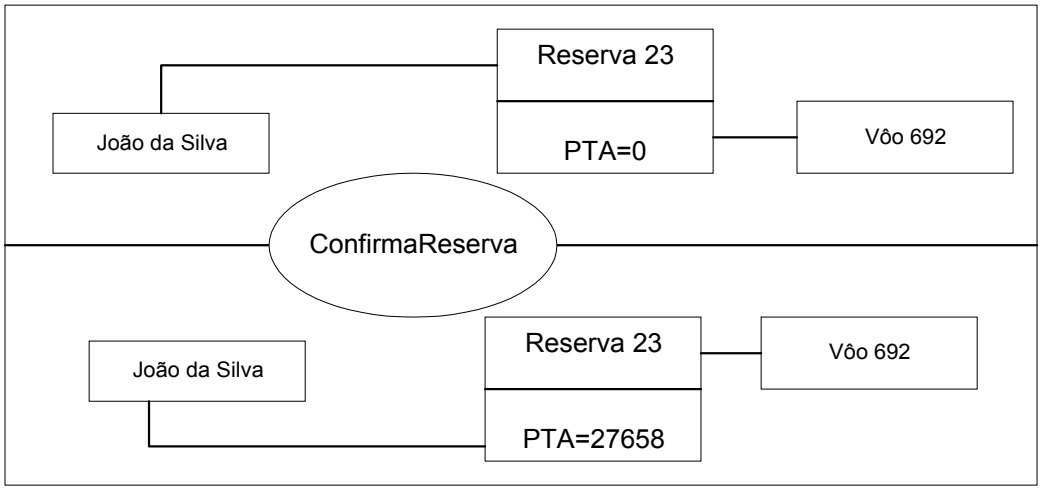

Figura A.17 – ***Snapshot*** **(ConfirmaReserva)**

Operação ConfirmaReserva (num_reserva, num_voo, nome_passageiro)

Pré: Existe uma reserva efetuada e o pagamento foi efetuado.

Pós: O sistema gera o número do PTA como identificação do passageiro.

CriaListaEspera

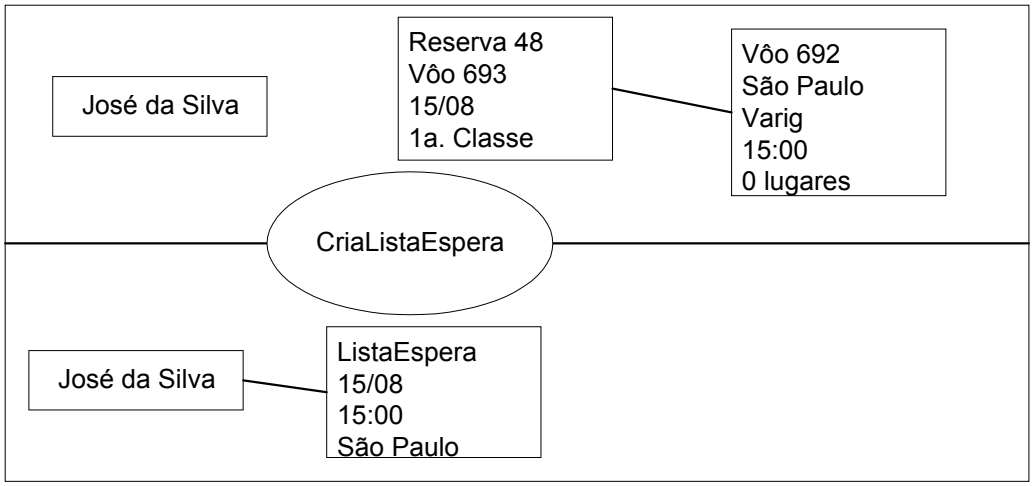

Figura A.18 – ***Snapshot*** **(CriaListaEspera)**

Operação CriaListaEspera (nome_passageiro, data_lista, horario_lista, destino)

Pré: Não existem mais lugares disponíveis em nenhum vôo para o destino, a data e o horário desejados.

Pós: Criada uma lista de espera da qual passará a fazer parte o passageiro requisitante.

RemoveLista

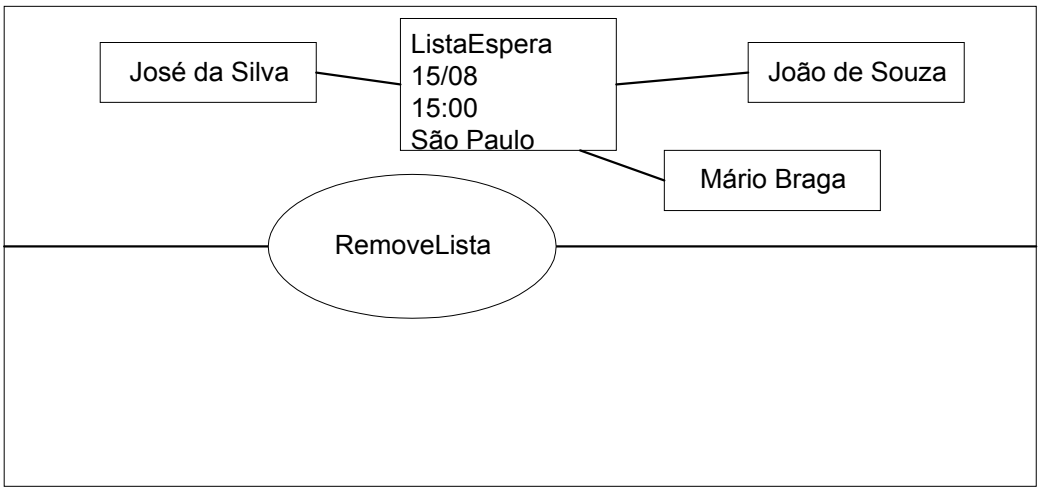

Figura A.19 – ***Snapshot*** **(RemoveLista)**

Operação RemoveLista (data_lista, horario_lista, destino)

Pré: Existe uma lista de espera para uma determinada data e horário passados.

Pós: A lista e os passageiros a ela associados são removidos.

InsereNaLista

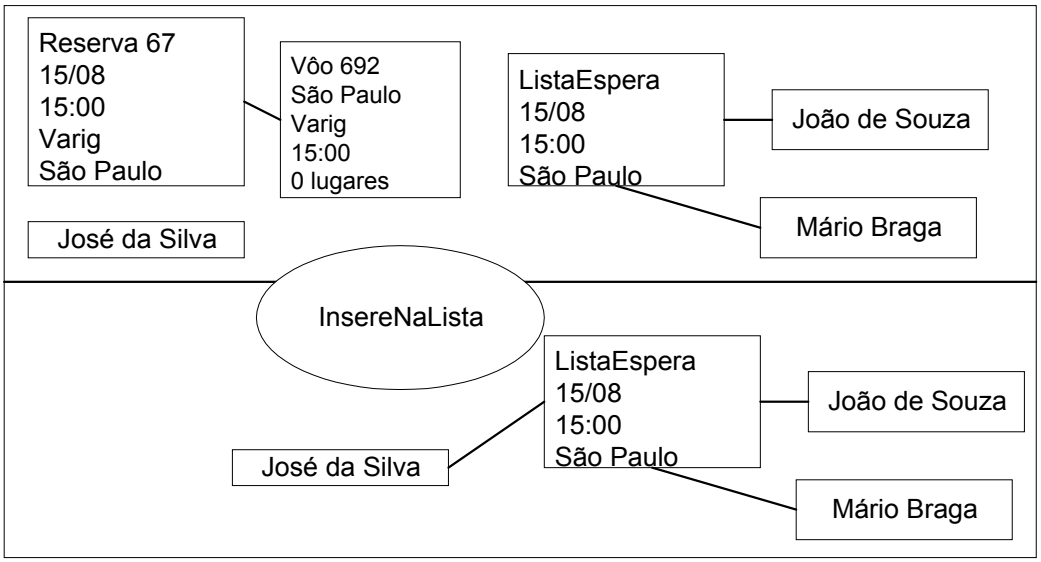

Figura A.20 – ***Snapshot*** **(InsereNaLista)**

Operação InsereNaLista (nome_passageiro, data_lista, horario_lista, destino)

Pré: Não existem mais lugares disponíveis em nenhum vôo para o destino, horário e data requisitados e já foi criada uma lista de espera com o mesmo destino, horário e data.

Pós: O passageiro requisitante passa a fazer parte da lista de espera.

RemoveDaLista

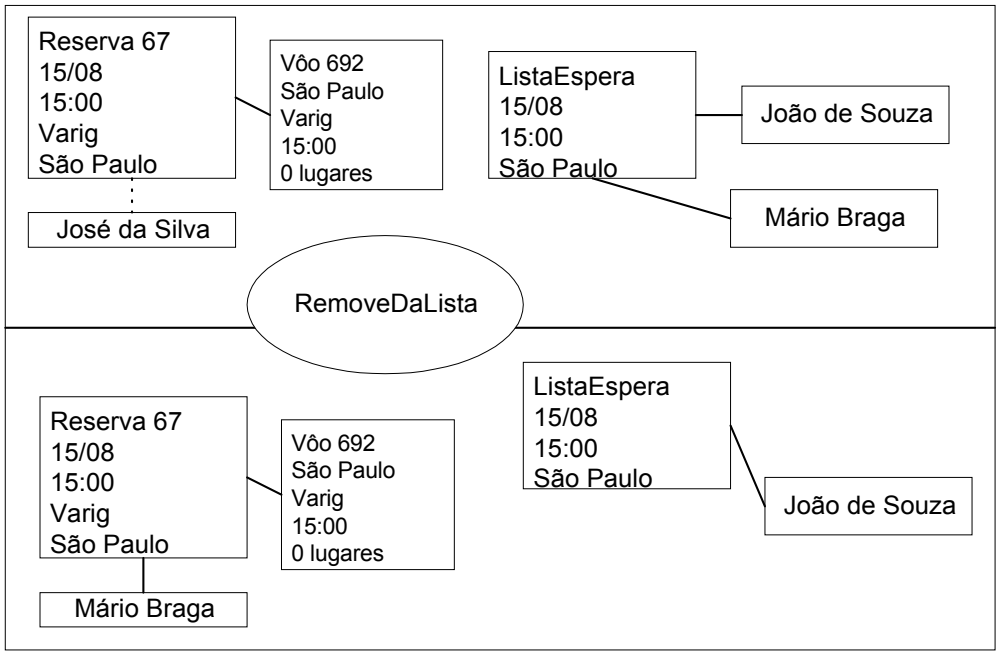

Figura A.21 – ***Snapshot*** **(RemoveDaLista)**

Operação RemoveDaLista (nome_passageiro, data_lista, horario_lista, destino)

Pré: Foi cancelada uma reserva para o vôo desejado.

Pós: Insere o primeiro passageiro da lista de espera na reserva e remove-o da lista de espera.

TemReserva

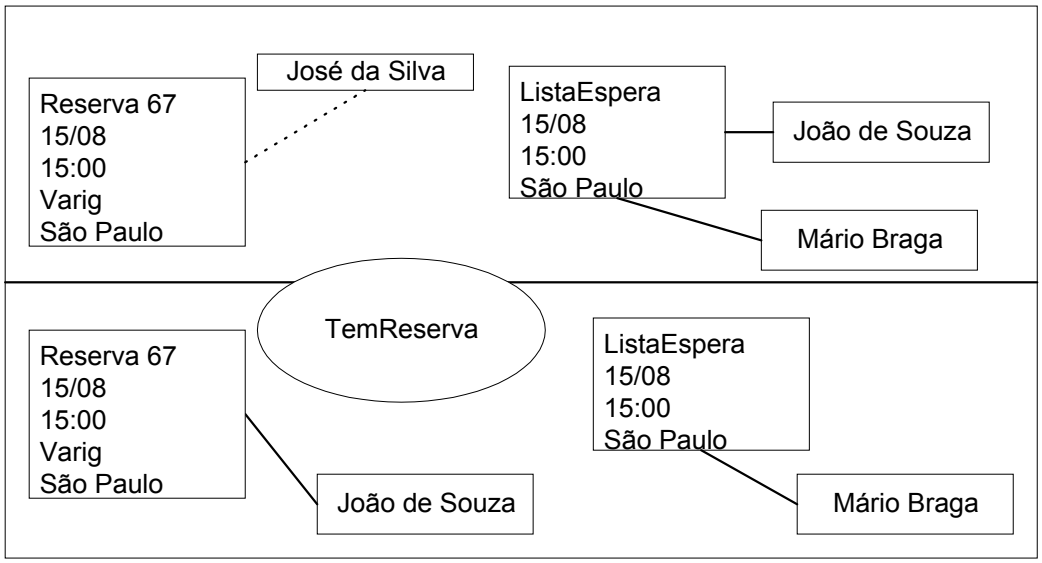

Figura A.22 – ***Snapshot*** **(TemReserva)**

Operação TemReserva (nome_passageiro, num_voo)

Pré: Foi cancelada uma reserva que abriu uma vaga no vôo x.

Pós: O primeiro passageiro que estava na lista de espera para o destino de x, num horário próximo ao do vôo x e na mesma data, é inserido na reserva do vôo e notificado.

DIAGRAMA DE INTERAÇÃO

CancelaReserva

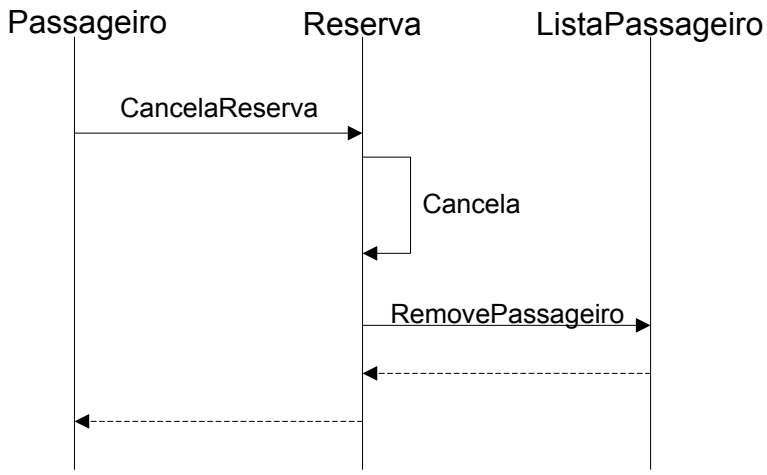

Figura A.23 – **Diagrama de Interação (CancelaReserva)**

ConfirmaReserva

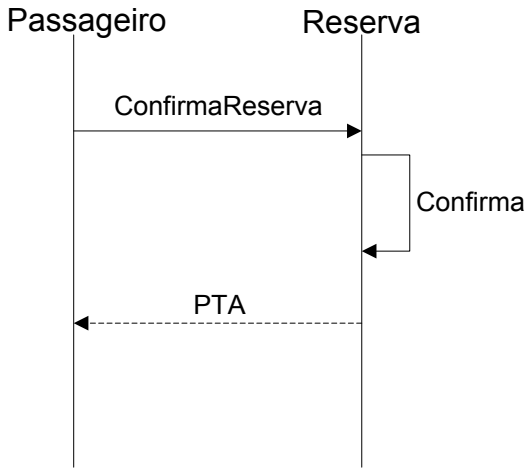

Figura A.24 – **Diagrama de Interação (ConfirmaReserva)**

AdicionaVôo

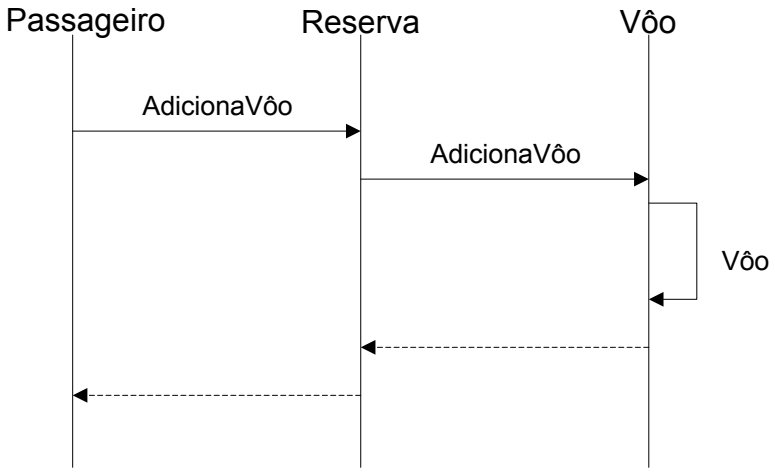

Figura A.25 – **Diagrama de Interação (AdicionaVôo)**

CriaListaEspera

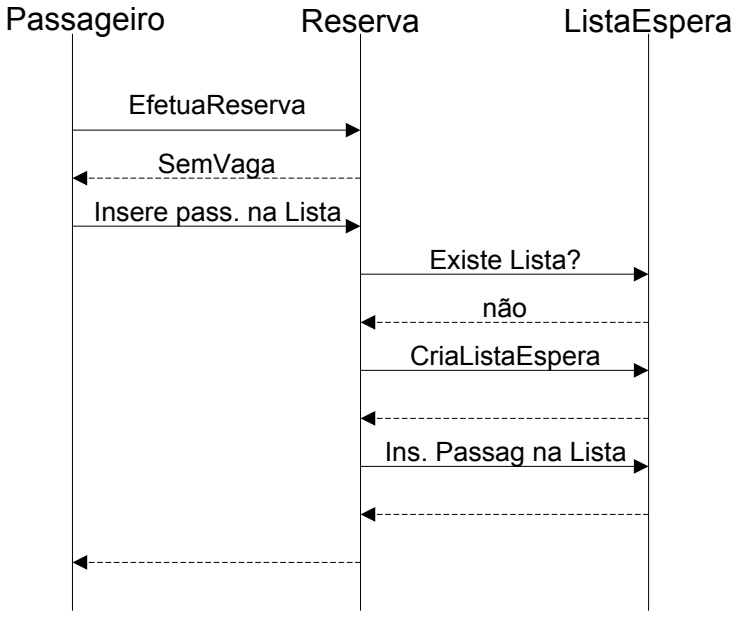

Figura A.26 – **Diagrama de Interação (CriaListaEspera)**

InsereNaLista

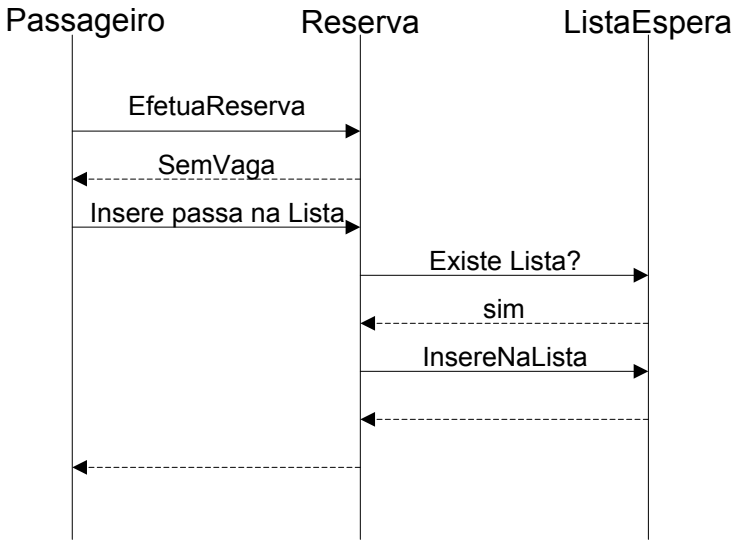

Figura A.27 – **Diagrama de Interação (InsereNaLista)**

RemoveDaLista

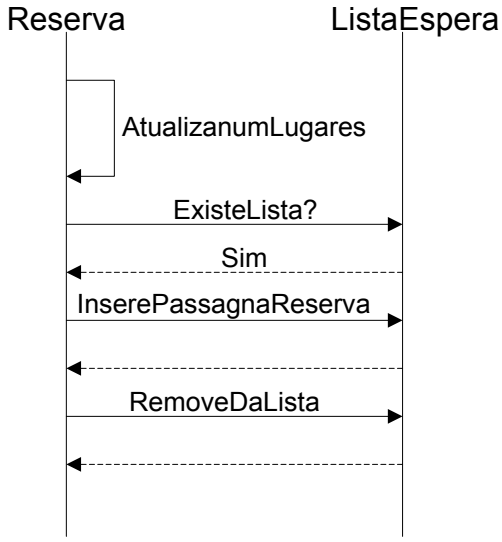

Figura A.28 – **Diagrama de Interação (RemoveDaLista)**

TemReserva

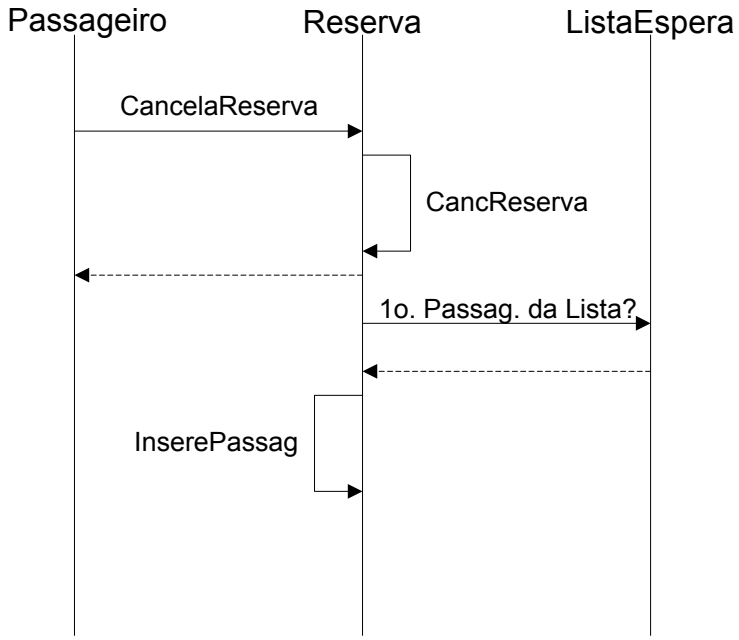

Figura A.29 – **Diagrama de Interação (TemReserva)**

DIAGRAMA DE ESTADOS

Passageiro

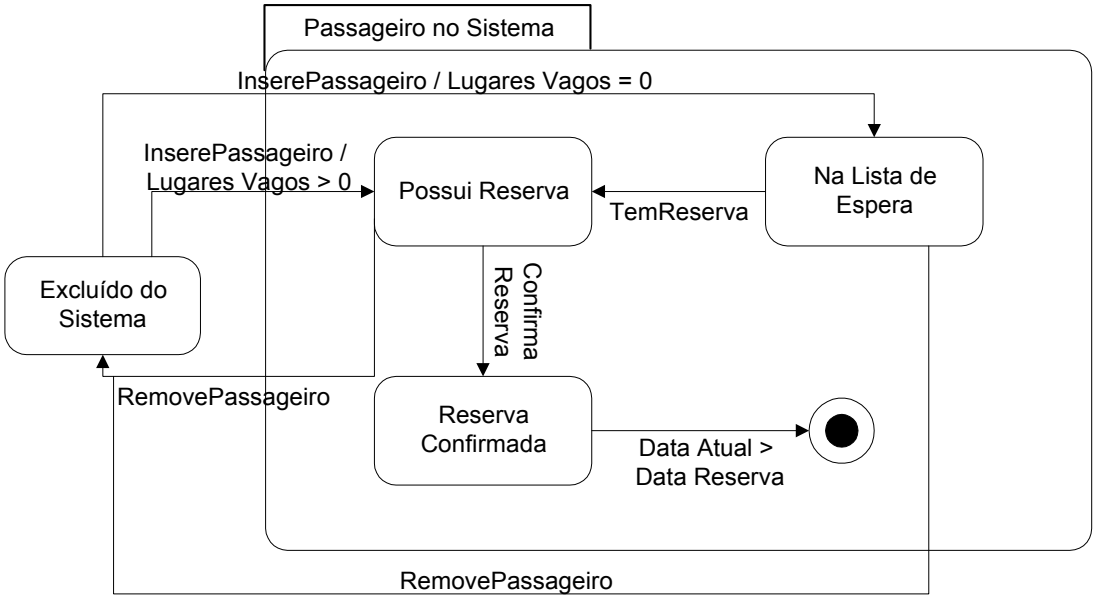

Figura A.30 – **Diagrama de Estados (Passageiro)**

ListaEspera

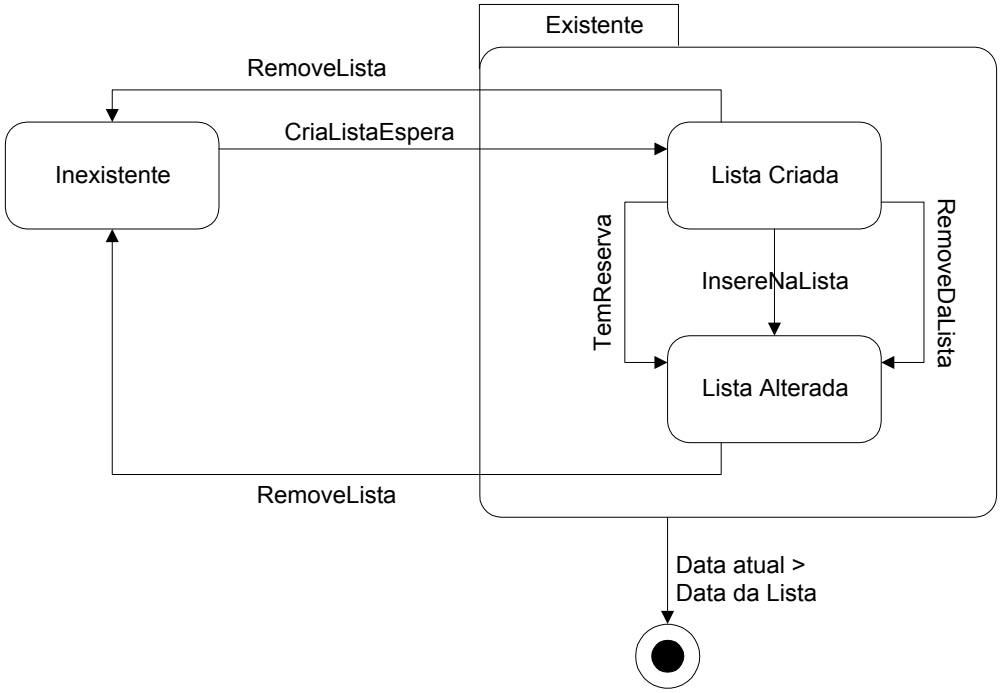

Figura A.31 – **Diagrama de Estados (ListaEspera)**

Vôo

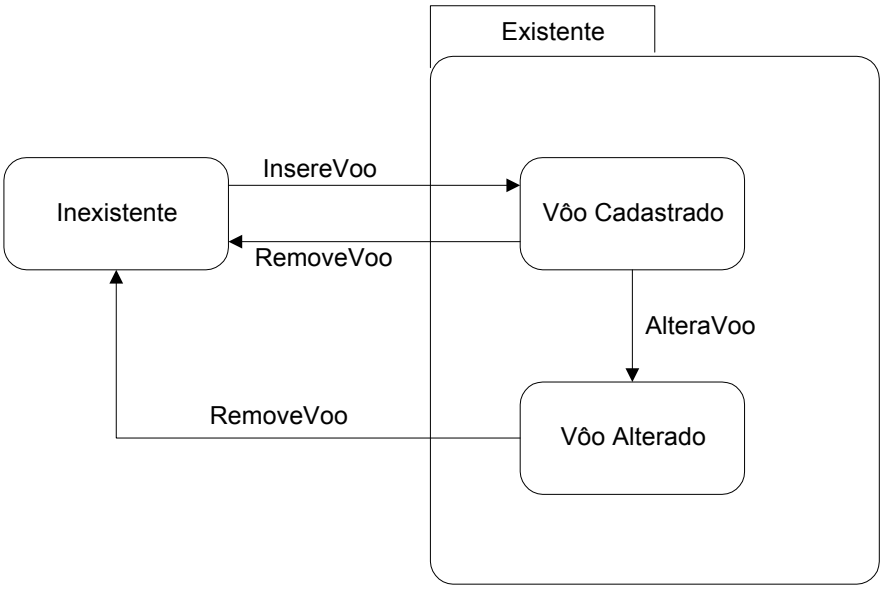

Figura A.32 – **Diagrama de Estados (Vôo)**