Uma comparação de dois métodos de desenvolvimento de software baseado em componentes: *Catalysis* e *UML Components*

por
Bernardo Faria de Miranda e Diego Delgado Lages

UFRJ

Projeto de final de curso submetido para a obtenção do título de
Bacharel em Informática
ao Departamento de Ciência da Computação do
Instituto de Matemática
da Universidade Federal do Rio de Janeiro
por
Bernardo Faria de Miranda e Diego Delgado Lages
Outubro de 2003

Uma comparação de dois métodos de desenvolvimento de software baseado em componentes: *Catalysis* e *UML Components*

Bernardo Faria de Miranda e Diego Delgado Lages

Uma comparação de dois métodos de desenvolvimento de software baseado em componentes: *Catalysis* e *UML Components*

Bernardo Faria de Miranda e Diego Delgado Lages

PROJETO DE FINAL DE CURSO SUBMETIDO AO CORPO DOCENTE DO DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO, DO INSTITUTO DE MATEMÁTICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM INFORMÁTICA.

Aprovada por:

Profª Cláudia Maria Lima Werner, D.Sc.

Prof. Marco Aurélio Souza Mangan, M.Sc.

Prof. Jano Moreira de Souza, Ph.D.

Prof. Márcio de Oliveira Barros, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

OUTUBRO DE 2003

Resumo do Projeto apresentado ao DCC/IM - UFRJ como parte dos requisitos necessários para a obtenção do grau de Bacharel em Informática

Uma comparação de dois métodos de desenvolvimento de software baseado em componentes: *Catalysis* e *UML Components*

Bernardo Faria de Miranda e Diego Delgado Lages
Outubro/2003

Orientadores: Cláudia Maria Lima Werner e Marco Aurélio Souza Mangan
Departamento: Ciência da Computação

Atualmente, os sistemas estão cada vez mais complexos e os métodos de desenvolvimento baseado em componentes (DBC) surgem como uma forma de reduzir a complexidade das atividades de desenvolvimento e de manutenção. Entretanto, não existem métodos formais e amplamente utilizados que possam servir de guia para o desenvolvedor e para o analista. Neste trabalho, comparamos duas propostas para o DBC, por meio do uso efetivo dos métodos em dois estudos de caso. Como resultado, oferecemos ao leitor uma visão geral do que estes métodos apresentam em consenso e suas diferenças.

Resumo do Projeto apresentado ao DCC/IM - UFRJ como parte dos requisitos necessários para a obtenção do grau de Bacharel em Informática

Uma comparação de dois métodos de desenvolvimento de software baseado em componentes: *Catalysis* e *UML Components*

Bernardo Faria de Miranda e Diego Delgado Lages

Outubro/2003

Orientadores: Cláudia Maria Lima Werner e Marco Aurélio Souza Mangan
Departamento: Ciência da Computação

Software is becoming extremely complex and component-based development (CBD) emerges as one of the ways to reduce the complexity of development and maintenance activities. But, there are no formal and broadly adopted methods that can be used as a guide to the developer and to the analyst. Is this work, we compare two proposals for CBD, by applying these two methods in two case studies. As a result, we show what these two methods have in common and their differences.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

Atualmente, nenhum negócio do mundo corporativo funciona sem o auxílio do *software*. Esta dependência faz do *software* um fator crítico para o sucesso dessas organizações. A criação ou mudança de um negócio tem efeito direto em algum sistema. Porém, a produtividade da indústria de *software* não tem acompanhado esta demanda crescente, ou pelo menos não tem acompanhado da forma esperada. Os métodos tradicionais de desenvolvimento de *software* ainda não permitem atingir os ganhos de produtividade e qualidade esperados [7]. Portanto, novos métodos devem ser criados a fim de suprir esses ganhos.

A busca por novos métodos tem apontado para o `desenvolvimento baseado em componentes (DBC)` como solução para esta crise. O uso de componentes é a fonte primária de produtividade e qualidade. Esta é a lei natural em qualquer disciplina de engenharia madura [16][7]. Componentes proporcionam o re-uso de "pedaços" de *software* em larga escala, ou seja, componentes já produzidos em outros sistemas de *software*.

O conceito de construir *software* utilizando componentes não é novo [10]. A criação de sistemas a partir de componentes está baseada no princípio de dividir para conquistar, com o objetivo de gerenciar a complexidade. Ou seja, quebrar

um problema em partes menores e, então, solucionar estas partes menores, para depois construir uma solução mais elaborada através destas bases mais simples [9]. A evolução dos métodos de desenvolvimento de *software*, tradicionalmente, tem caminhado na direção da diminuição do esforço de produção e manutenção através da reutilização. No entanto, apenas o aparecimento recente de novas tecnologias permitiu o aumento significativo das possibilidades de construção de sistemas e aplicações por meio de componente reutilizáveis [10].

Desde o seu surgimento, há aproximadamente uma década, o DBC ainda permanece imaturo perante as necessidades dos desenvolvedores de *software* [18]. Estas dificuldades existem porque as atuais abordagens de DBC procuram enfatizar muito mais suas notações e diagramas do que como, efetivamente, podemos especificar e montar os componentes, a partir dos elementos sintáticos modelados na análise do problema [18].

1.2 Objetivos

Visto que os métodos de DBC ainda estão em uma etapa de evolução e afirmação, tanto no meio acadêmico quanto no meio corporativo, temos como objetivo esclarecer algumas questões iniciais relativas a esta forma de desenvolvimento.

A criação de sistemas baseados em componentes impõe como principal desafio a seleção e utilização de componentes já existentes. Apesar disso, os métodos ainda não apresentam um consenso quanto às etapas mais básicas. São elas:

- **identificação** dos componentes do sistema;
- **especificação** dos serviços oferecidos por cada um desse componentes e suas restrições de contexto;
- **representação** do componente e sua especificação na forma de documentos.

Desta forma, **o nosso objetivo é comparar como os métodos atuais de DBC tratam a identificação e representação dos componentes que estão**

sendo criados.

Apesar de muito ter sido dito sobre componentes no mundo do desenvolvimento de *software*, a carência de métodos baseados em componentes ainda é evidente. Para o nosso trabalho de comparação, estudamos dois métodos que foram publicados recentemente em livros:

- *Catalysis* (1999)[20];
- *UML Components* (2000)[9].

Escopo

Não faz parte do nosso trabalho justificar a utilização de métodos de DBC ou mostrar como estes abordam o processo de reutilização de componentes. Por se tratar de um projeto final de curso e não dispormos de uma equipe de desenvolvimento, são impostas algumas limitações. Os relatos deste trabalho não estão relacionados a nenhum projeto de *software* real. Os requisitos dos casos utilizados para aplicação das técnicas estão descritos nos livros de nossa bibliografia. A modelagem usando o *Catalysis* foi baseada nos requisitos do exemplo utilizado no livro do *UML Components* e vice-versa. Cada um dos dois alunos que compõe este projeto teve a responsabilidade de experimentar um dos métodos. Ao final da experimentação seus resultados e impressões foram comparados. Desta forma, o trabalho limitou-se a comparar como estes métodos guiam o desenvolvedor para identificação, especificação e representação de componentes.

1.3 Organização

Este documento está divido em seis capítulos. Este **primeiro** capítulo é uma pequena introdução ao nosso trabalho, além de servir para definir seu objetivo e escopo.

O **segundo** capítulo apresenta uma visão geral de DBC, ou seja, conceitos relacionados e o que se espera de um método de DBC.

Nos próximos dois capítulos são descritos os métodos estudados. No **terceiro** capítulo, o *Catalysis*; e no **quarto** capítulo, o *UML Components*.

No **quinto** capítulo, é apresentada a modelagem de um estudo de caso usando *Catalysis*. Fazemos o mesmo no **sexto** capítulo utilizando o *UML Components*.

No **sétimo** capítulo, é feita a comparação dos métodos baseado nos estudos de caso.

Finalmente, no **oitavo** capítulo, são apresentadas as considerações finais deste trabalho.

Capítulo 2

Visão geral sobre DBC

Neste capítulo, apresentamos inicialmente uma visão geral sobre os conceitos relacionados com componentes. Em seguida, na segunda e na terceira seções, definimos o que seriam os processos de DBC.

2.1 Conceitos

A definição de componente merece um certa atenção, pois diversos autores possuem visões diversas do que seria um componente. **Entendemos um componente de *software* como uma unidade de composição com interfaces contratualmente especificadas e dependências de contexto explícitas. Um componente de *software* pode ser instalado independentemente e está sujeito a composição por terceiros** [17].

Os componentes são baseados na tecnologia de objetos. Desta forma, também estão calcados em alguns princípios da orientação a objetos:

- Unificação de dados e funções: um objeto consiste de dados (ou estados) e funções que processam estes dados. Esta dependência entre função e dados aumenta a coesão [9].
- Encapsulamento: um objeto não mostra para seu cliente como são implemen-

tadas as funções ou como armazena os dados [9].

- Identidade: Cada objeto tem identidade única, independentemente do seu estado [9].

A especificação de um componente está nas dependências que possuem com o meio. Assim como os objetos, as dependências do componente são expressas por meio de suas interfaces. Veja a Figura 2.1:

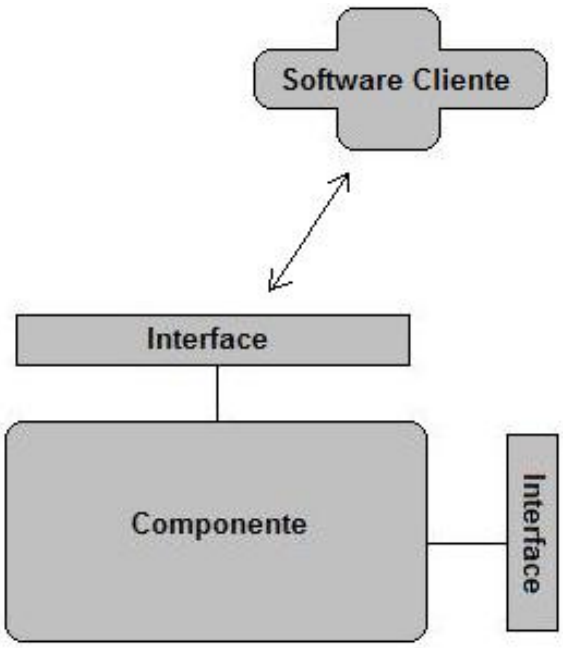


Figura 2.1: Um componente, suas interfaces e seu *software* cliente.

2.1.1 Interfaces

O projeto interno e a implementação de um componente estão fortemente encapsulados e sua comunicação com outros componentes se dá exclusivamente por meio de suas interfaces [13]. A interface define tudo que um cliente precisa saber, e mais nada além disso [9]. As referências são feitas para a interface e não ao componente como um todo. Um componente pode implementar mais de uma interface. Sendo a interface um protocolo, concordamos que a substituição de um componente por outro é completamente factível desde que este novo componente implemente as

mesmas interfaces. Ressalta-se, entretanto, que apenas a definição deste protocolo não informa nada quanto à sua implementação. Por esta razão, as interfaces de componentes são contratualmente especificadas de maneira a informar o que se deve esperar de um componente.

2.1.2 Contratos

Para assegurar que os módulos de um sistema irão manter a forma como interagem ao longo do processo de desenvolvimento, foi proposta a abordagem chamada de `Design By Contract` [14]. Funciona de maneira análoga aos contratos que conhecemos de nosso cotidiano. Contrato nada mais é do que o acordo formal entre duas ou mais partes. Da mesma forma, o principal objetivo é criar contratos formais e sem ambigüidades entre uma interface e seus clientes. Os contratos podem ser de três tipos:

- pré-condição;
- pós-condição;
- invariante.

Para ter sucesso na chamada de um método de uma interface, o cliente deve garantir alguma **pré-condição**. E, então, ao final da chamada do método, por meio de uma **pós-condição**, são asseguras algumas propriedades do componente que implementa a interface.

Ainda existem componentes que possuem um estado que deverá ser mantido ao longo de toda sua existência. Estes estados são assegurados através de **invariantes**.

Os contratos, assim como outras restrições dos modelos, podem ser escritos em **linguagem natural** ou ***Object Constraint Language(OCL)*** [19].

2.2 Processo

Admitindo-se que todo o desenvolvimento do sistema está centrado nos componentes envolvidos e na sua arquitetura, a grosso modo, podemos considerar dois processos distintos [7]:

- composição de componentes;
- desenvolvimento de componentes.

Por serem processos distintos, podem estar sob a responsabilidade de diferentes equipes de desenvolvimento de *software*. Mesmo a composição de componentes pode ocorrer muito depois do desenvolvimento dos componentes envolvidos. O passo inicial de construção do sistema é a análise de quais componentes podem ser utilizados para satisfazer os requisitos levantados. Esta busca por componentes não está limitada a uma biblioteca de componentes da equipe ou da empresa, pois eles podem ser comprados. Teoricamente, esta fase pode proporcionar um ganho de tempo de projeto, considerando que não teríamos que desenvolver todas as peças que compõem o sistema. Ainda existe a possibilidade de adaptação de componentes para que satisfaçam os requisitos.

A construção de componentes é o processo menos atraente, pois requer mais esforço e tempo [10]. Por outro lado, oferece a oportunidade de desenvolver funcionalidades do sistema que vão ser, provavelmente, as de maior valor para a empresa.

Nos dois métodos estudados, observamos um foco maior na construção de componentes do que na seleção e utilização de componentes existentes. John Cheesman e John Daniels, autores do *UML Components*, afirmam que, ao contrário do que muitos pensam, o objetivo principal dos componentes não é a re-utilização. Obviamente esta idéia projeta muitos benefícios como ganho de produtividade e qualidade, mas ainda temos outro objetivo que deve ser alcançado antes. A manutenção dos sistemas em constantes mudanças acompanhando a dinâmica dos negócios seria o principal objetivo. É mais importante controlar as mudanças de requisitos e os componentes envolvidos do que assegurar que um componente seja utilizado em

contextos variados.

Como os métodos estudados não abordam questões ligadas à seleção e à reutilização de componentes existentes e se mantêm na orientação de como criar aplicações a partir da construção de componentes, nosso trabalho de comparação seguiu nesta linha. Na próxima seção, expomos o que se espera que um método de DBC ofereça como ferramentas para o desenvolvedor.

2.3 Desenvolvimento de componentes

Um processo para o desenvolvimento de componentes deve conter algumas etapas básicas. Primeiro, este processo deve orientar o que de importante seria registrado na fase de análise de requisitos e como proceder. Este primeiro passo vai possibilitar a identificação de componentes e seus serviços oferecidos. E, logicamente, todas estas informações devem ser representadas formalmente em modelos.

2.3.1 Especificação de requisitos

Esta etapa corresponde ao entendimento do negócio em questão e à definição do que se espera do sistema que deverá ser construído. As técnicas não difererem de qualquer método tradicional. E sendo assim, para elicitação dos requisitos do sistema, tradicionalmente são criados dois artefatos:

- modelo de casos de uso;
- modelo conceitual.

O *UML Components* [9] ainda sugere a criação de um modelo de processos para auxiliar a identificação dos casos de uso.

Como qualquer outro método, existem variações de estilo nos modelos gerados. Porém, estas variações não têm grande impacto nos resultados do processo.

2.3.2 Identificação de componentes

As técnicas para identificação de componentes de negócio consiste, basicamente, em encontrar os tipos da aplicação em questão e traçar os limites dos componentes entre estes tipos.

No *UML Components*, o *diagrama de tipos de negócio* (ou *business type diagram* – BTD) é fundamental para qualquer aplicação baseada em componentes, porque representa como os dados serão gerenciados persistentemente pelo sistema, além de ser o último diagrama elaborado antes da montagem e definição da arquitetura de componentes [18]. Representado através do diagrama de classes da *Unified Modeling Language (UML)*[5], os tipos, assim como as classes, possuem atributos. Os tipos têm correspondência com o modelo conceitual, mas apenas os tipos estão dentro do escopo do sistema que será construído.

Os tipos deverão ser agrupados dentro de um mesmo componente. O método de DBC deve oferecer alguma heurística para definir os agrupamentos.

2.3.3 Identificação de serviços e restrições de cada componente

Após a definição da arquitetura e a identificação dos componentes, deve-se explorar os serviços oferecidos pelos componentes. Os serviços são investigados durante a descoberta dos métodos do componente. E para prover tais serviços, os componentes precisam ter algumas restrições asseguradas dentro do contexto de aplicação. Os serviços são especificados por meio de contratos.

2.3.4 Representação de componentes

O método deve informar ao desenvolvedor como representar o componente que foi construído, usando as notações estabelecidas pela *UML*. Trata-se de um desafio, pois a *UML* foi criada para representar os conceitos de análise e projeto orientado

a objetos. Para contornar este problema, os autores normalmente utilizam a característica extensível da linguagem. Não é raro encontrar propostas para novas notações.

Capítulo 3

Catalysis

O *Catalysis* é um método de modelagem que surgiu em 1998, por meio da publicação do livro *Object, Components and Frameworks with UML: The Catalysis Approach* [20] dos autores Desmond Francis D'Souza e Alan Cameron Wills. A motivação do livro está relacionada com a larga experiência dos autores no desenvolvimento baseado em componentes.

O *Catalysis* define um conjunto de técnicas e métodos para a análise de negócios e desenvolvimento de sistemas usando *UML*. O nível de detalhe no qual estas técnicas são aplicadas é variável. Logo, é possível que elas possam ser aplicadas tanto em grandes, quanto em pequenos projetos.

O *Catalysis* é, especialmente, focado no desenvolvimento de sistemas baseados em componentes, através do qual sistemas são criados a partir de componentes mais básicos. Além de permitir a reutilização de componentes, ele permite a reutilização de outros artefatos de modelagem como *frameworks* [1], padrões [2] e especificações.

Uma característica do *Catalysis* é a capacidade de eliminação de ambigüidades

[1] *Frameworks* são coleções flexíveis de classes abstratas e concretas projetadas para serem estendidas e refinadas para a reutilização. [15]

[2] Padrões ajudam a construir sistemas com base na experiência coletiva de engenheiros de softwares capacitados. Os padrões capturam a experiência existente e provada do desenvolvimento de software e ajudam a promover uma boa prática de modelagem. Cada padrão trata de um problema específico e recorrente na modelagem ou na implementação de um sistema. [8]

aos longo da modelagem. Particularmente, isso é importante para a modelagem baseada em componentes, onde leitores e escritores de uma especificação de interface podem não ter contato entre si.

O *Catalysis* surgiu como uma nova forma de modelagem aproveitando a já existente *UML* e estendendo-a para incluir novos conceitos. O *Catalysis* também serviu de base para o surgimento de outros métodos, como o próprio *UML Components*.

3.1 Especificação de Requisitos

Para fazer a especificação de requisitos em *Catalysis*, são utilizados casos de uso, diagrama de estados e *snapshots*. A partir de cada um desses, é possível retirar invariantes e ações do sistema, necessários para realizar a modelagem deste.

3.1.1 Casos de uso

O *Catalysis* define um *caso de uso* como "a especificação de uma seqüência de ações, incluindo variantes, que um sistema (ou outra entidade) pode realizar, interagindo com os atores do sistema". Mas o *Catalysis* não só define um caso de uso desta forma, mas também considera que um caso de uso é uma ação conjunta.

Assim como numa ação conjunta, um caso de uso reflete a interação entre vários objetos e também qual será o efeito em cascata nesses objetos. Como é um ação conjunta, ele tem pré e pós-condições, invariantes e também pode ser refinado, como será visto posteriormente.

3.1.2 Estados

Assim como observamos na figura 3.1, o diagrama de estados do *Catalysis* serve para ilustrar os estados possíveis de um objeto e suas transições. Este modelo está de acordo com as notações da *UML*. Um objeto pode ter estados sob perspectivas

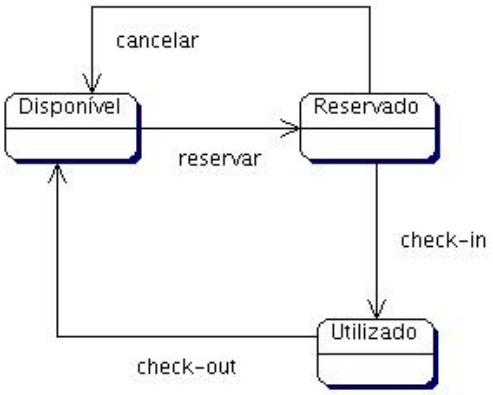


Figura 3.1: Diagrama de estados.

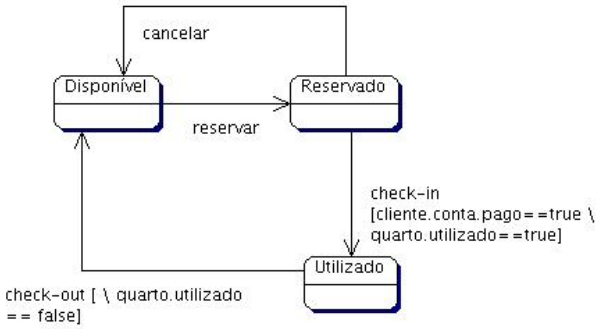


Figura 3.2: Diagrama de estados com pré e pós-condições.

distintas, de forma a criarmos diagramas distintos para cada uma destas perspectivas.

Um diagrama de estados de um objeto, implicitamente, define invariantes sobre o objeto. Diante de uma perspectiva, só é possível ao objeto estar somente em um estado em qualquer momento.

Os diagramas de estados também definem ações. As ações definidas são justamente as transições dos estados, já que para um objeto ir de um estado para outro é necessária uma ação. Observe na figura 3.2 que podemos definir dentro do próprio diagrama de estados as pré e pós-condições das transições.

3.1.3 *Snapshots*

Uma forma que o *Catalysis* define para a modelagem de comportamento é através de *snapshots*. Um *snapshot* é como uma "foto" dos objetos de um componente em um determinado momento.

Para a modelagem de uma ação, podemos usar um *snapshot* antes e depois da execução da ação, destacando as mudanças que essa ação proporcionará. A partir destes *snapshots* é possível esclarecer melhor uma ação, assim como mostrado na figura 3.3.

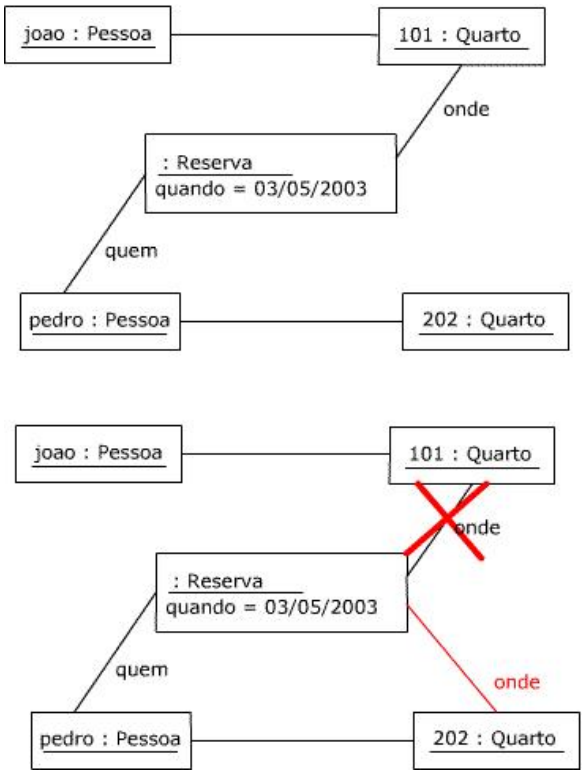


Figura 3.3: *Snapshots*.

3.1.4 Ações

A limitação dos *snapshots* é que eles só permitem a visualização em situações particulares. Mas é necessário modelar a ação para todas as situações possíveis. Então, para isso, modela-se as ações do sistema, que representam estas para todas

as situações possíveis. Para modelar uma ação, precisamos do nome da ação e dos tipos que serão afetados por esta ação, representando também as invariantes, as pré e pós-condições, como mostrado na Figura 3.4.

```
Action agendarCurso(cursoReq : Curso, iniciaReq : Data)
pre:   A partir do fato que existe um instrutor qualificado para
       este curso que está livre nesta data, durante toda a duração
       curso.
post:  Uma nova sessão confirmada está criada, com curso = cursoReq,
       dataInicio = iniciaReq, e
       dataFinal - dataInicio = cursoReq.duracao
```

Figura 3.4: Exemplo de especificação de uma ação.

Uma ação pode ter suas pré e pós-condições definidas de forma textual ou de forma mais precisa, usando a *OCL*. Como será visto mais a frente, por meio do refinamento, as pré e pós-condições definidas inicialmente na forma de texto podem passar a uma linguagem mais precisa. Um método comum para a formalização de especificações é criar especificações textuais, depois criar *snapshots* ilustrando a ação e, finalmente, criar a especificação formal a partir destes *snapshots*.

Existem dois tipos de ações no *Catalysis* que correspondem aos comportamentos individuais e coletivos dos objetos. As ações podem ser localizadas ou conjuntas.

Uma ação localizada, ou operação, é uma ação na qual somente um objeto é requisitado para a ação, sem necessidade de saber quem inicia a ação. Uma ação localizada é reconhecida pelo tipo precedendo a especificação dessa ação, como pode ser observado na Figura 3.5.

```
action Tipo::nomeAcao
```

Figura 3.5: Exemplo de uma ação localizada.

Uma ação conjunta descreve o comportamento e as interações entre grupos de objetos. O foco é no efeito dessas interações entre múltiplos objetos. Podemos

reconhecer uma ação conjunta através da Figura 3.6.

```
action (parte1: Tipo1, parte2 : Tipo2, ...)::nomeAcao()
```

Figura 3.6: Exemplo de uma ação conjunta.

3.2 Identificação de componentes

Os modelos no *Catalysis* podem ser estáticos, dinâmicos ou de interações. Os modelos estáticos modelam o conhecimento de um objeto em qualquer momento. Em geral, os modelos estáticos são criados a partir de diagramas de tipos, os modelos dinâmicos são representados por ações e por diagrama de estados e os modelos de interação são representados por diagramas de ações conjuntas e colaborações.

Todos estes modelos precisam de alguma forma representar invariantes, pré e pós-condições, e para realizar isso utilizamos texto livre em um nível mais abstrato e a linguagem *OCL* nos refinamentos e/ou em níveis menos abstratos.

3.2.1 Tipos

Um tipo define um conjunto de objetos pelo seu comportamento visível externamente, sem se preocupar com sua implementação. Os tipos modelam o estado abstrato de qualquer implementação daquele tipo.

Tipos são diferentes de classes, pois um tipo pode ser implementado de diversas formas, desde que respeite as definições abstratas. Um tipo pode até mesmo ser implementado através de mais de uma classe.

Um tipo, assim como uma classe, possui atributos e métodos. Mas os atributos e métodos de um tipo são sempre aqueles visíveis externamente, de forma que possam existir diferentes implementações para um mesmo tipo, mas sempre obedecendo a especificação deste tipo.

Os atributos de um tipo podem conter invariantes, ou seja, condições que devem ser sempre obedecidas a qualquer momento. Métodos ou ações de tipos podem ter pré e pós-condições, que são, respectivamente, condições que devem ser obedecidas antes da execução de uma ação e condições que devem ser obedecidas depois da execução de uma ação.

3.3 Identificação de Serviços e Restrições de cada Componente

O *Catalysis*, ao contrário do *UML Components*, não tem um método sistemático para a identificação de componentes. Ele sugere que esta identificação seja feita com base nas diferentes necessidades que o componente deverá suportar. No entanto, é dito que, em geral, cada colaboração deverá representar um método na interface do componente. Isso se deve ao fato de, na maioria dos casos, uma colaboração representar uma ação em vários tipos e o interesse deste tipo de ação se dar por entidades externas, ou seja, os clientes do componente.

3.4 Representação de Componentes

No DBC, devemos separar o comportamento externo de sua implementação interna. Através do diagrama de tipos, ou seja, da modelagem estática, temos a representação dos tipos de um componente. Por outro lado, através dos modelos de comportamento, estamos interessados em obter a modelagem das ações aplicadas ao componente, ou seja, o seu modelo de comportamento.

Por meio de modelos estáticos e dinâmicos, é possível determinar as mudanças em objetos particulares. Com os modelos de interação, é possível mostrar o comportamento dinâmico de todo o componente, englobando quantos objetos forem necessários.

3.4.1 Colaborações

Uma colaboração é um conjunto de ações envolvendo vários tipos, e tendo como resultado uma alteração nos estados destes tipos. Em geral, cada colaboração será representada por um método na interface do componente.

Quando existem ações com os mesmos participantes, de forma que essas ações sejam relacionadas, existe uma colaboração. Uma colaboração pode, por exemplo, refletir um caso de uso inteiro, quando este necessita que várias ações sejam executadas.

3.5 Fatoração de Modelos

Ao projetar um grande sistema, nós devemos dividir este sistema em partes menores. Mas, para conseguir montar o sistema, será necessário juntar todas as partes, através da composição destes modelos. Para qualquer divisão em partes menores, será necessário a abstração e refinamento destes modelos.

3.5.1 Abstração e Refinamento

Abstrair numa modelagem é mostrar somente os aspectos que são relevantes para um propósito particular. Se o interesse é mostrar somente um aspecto geral do sistema, temos que abstrair o máximo possível para mostrar somente, no caso do *Catalysis*, os tipos, as ações e as colaborações da maneira mais genérica possível. A modelagem no *Catalysis* não se preocupa com a implementação, de forma que a própria modelagem no *Catalysis*, no nível mais detalhado, é uma abstração da implementação.

No *Catalysis*, é possível abstrair basicamente tipos, ações e colaborações. É possível também criar "árvores de abstração", onde existe uma modelagem no mais alto nível e faz-se um refinamento (abstração) sucessivo de pequenas partes de cada modelo.

Para cada refinamento e/ou abstração, é necessário ilustrar como isso foi feito, de forma a termos uma espécie de rastreabilidade (*traceability*). Um modelo refinado deve sempre manter a conformidade (*conformance*) com o modelo superior e inferior numa escala de abstração.

3.5.2 Composição e Modelos

Para o desenvolvimento baseado em componentes, pode ser necessário que um componente satisfaça diferentes necessidades. O componente que satisfaz estas diferentes necessidades é um componente reutilizável.

Mas um sistema não é, em geral, feito de um só componente, e sim da composição de vários componentes, sejam eles reutilizados ou não. A composição destes componentes não é uma tarefa simples, e o *Catalysis* define métodos sistemáticos para a realização destas composições.

Capítulo 4

UML Components

O *UML Components* [9] descreve um processo prático para o desenvolvimento de sistemas que fazem uso de tecnologias de componentes, tais como *Enterprise JavaBeans (EJB)* [2] e *Component Object Model (COM+)* [3]. John Cheesman e John Daniels criaram este processo de DBC que guia o desenvolvedor desde a descrição dos requisitos até à especificação detalhada dos componentes que serão necessários. A notação gráfica dos modelos deste método utilizam a UML [5]. As atividades do *UML Components* estão separadas em dois *workflows*:

- *Workflow* de requisitos;
- *Workflow* de especificação.

4.1 Especificação de requisitos

No *UML Components*, a modelagem de domínio corresponde ao *workflow* de requisitos. É a etapa em que se define tudo o que será feito. Os autores não descrevem como devem ser feitas as entrevistas com os clientes, apenas orientam a produção de uma quantidade mínima de documentos que possibilita o início das atividades de análise. Deve-se assumir que é necessário entender o negócio que será suportado. Desta forma, faz sentido documentar o processo de negócio com

um diagrama de atividades da *UML*. Após representar o processo de negócio, são criados dois diagramas principais : Modelo Conceitual de Negócio e Diagrama de Casos de Uso.

4.1.1 Modelo conceitual de negócio

Uma vez entendido qual o processo de negócio, deve-se mapear o entendimento dos conceitos que estão envolvidos e seus relacionamentos. Este mapeamento será chamado de **Modelo Conceitual de Negócio**. Utilizando o diagrama de classes, é feita uma descrição destes conceitos sem a preocupação com escopo ou detalhamento. São adicionados ao modelo apenas os atributos e a cardinalidade dos relacionamentos julgados necessários.

4.1.2 Modelo de casos de uso

Modelo de Casos de Uso tem como objetivo identificar e especificar alguns dos requisitos funcionais do sistema. Este modelo é construído com o Diagrama de Casos de Uso e Descrição de Casos de Uso da *UML*. Utilizando uma linguagem semi-formal, a descrição é feita indicando os passos da interação entre um usuário e o sistema, e, conseqüentemente, este fato ajuda a definir os limites.

Os casos de uso começaram a ser investigados já no diagrama de atividades. Traçam-se raias de responsabilidade sobre aquele diagrama de atividades que descreve o processo para encontrar os participantes dos casos de uso. Estes participantes são atores ou sistemas. O ator é uma entidade que interage com o sistema, tipicamente uma pessoa que tem algum papel dentro do processo.

4.2 Identificação de componente

De forma geral, podemos dividir a arquitetura da aplicação em quatro camadas:

- interface do usuário – apresentação para o usuário e captura de entradas;

- diálogo do usuário – gerência do diálogo com o usuário dentro de uma sessão;
- serviços de sistema – apresentação externa dos serviços do sistema;
- serviços de negócio – implementação central das informações e regras do negócio.

Componentes podem ser encontrados em todas estas quatro camadas da arquitetura do sistema. No entanto, o *UML Components* [9] está concentrado nos componentes que estão do lado do servidor da aplicação, ou seja, nas duas últimas camadas. Portanto, trataremos de dois tipos de componentes;

- componentes de sistema;
- componentes de negócio.

A identificação de componentes é a primeira fase do *workflow* de especificação. O objetivo é identificar um conjunto inicial de componentes de sistema e um conjunto inicial de componentes de negócio, e então colocá-los numa arquitetura inicial do sistema. Isto cria um contexto para os próximos passos deste *workflow*.

4.2.1 Identificação de componente de sistema

Na maioria dos casos, uma especificação de componente é criada separadamente para cada especificação de interface que foi identificada. Contudo, múltiplas interfaces de sistema são freqüentemente suportadas por uma única especificação de componente, pois em sua maioria as interfaces envolvidas possuem grande similaridade nos dados que manipulam. Sendo assim, deve-se identificar as interfaces de sistema primeiramente.

Identificação de interface de sistema

Em um primeiro momento, uma interface de sistema é definida para cada caso de uso. Então, ao analisar cada caso de uso, é considerado a cada passo o que deve ou não ser responsabilidade do sistema que está sendo modelado. Desta forma, o

que for responsabilidade do sistema tem uma ou mais operações correspondentes dentro da interface de sistema apropriada. Esta estratégia fornecerá um conjunto inicial de interfaces e operações para o início do trabalho.

4.2.2 Identificação de componente de negócio

O objetivo do **Modelo de Tipos de Negócio** é a descoberta de quais informações devem ser gerenciadas e quais componentes são necessários para prover uma funcionalidade. Os componentes de negócio também são descobertos por meio de suas interfaces. Na maior parte do casos, cada componente de negócio corresponde a uma inteface encontrada.

Identificação de interface de negócio

Interfaces de negócio são abstrações das informações que precisam ser gerenciadas pelo sistema. Segundos os autores, o método para identificar é o seguinte :

1. Produza uma cópia do modelo conceitual de negócio para formar o modelo inicial de tipo de negócio;
2. Coloque este modelo dentro do escopo do sistema, removendo todos os elementos irrelevantes;
3. Refine o modelo de tipo de negócio e especifique algumas regras de negócio utilizando restrições;
4. Identifique os **tipos-núcleo do negócio**;
5. Crie interfaces de negócio para os tipos-núcleo e adicione estes ao modelo de tipo de negócio;
6. Refine o modelo de tipo de negócio para indicar as responsabilidades das interfaces de negócio.

Tipo-núcleo

Para decidir quais os tipos no diagrama de tipos de negócio que serão considerados como núcleo, deve-se começar a pensar qual informação é dependente de outras e qual informação não é.

O tipo-núcleo é um tipo do negócio que tem existência independente dentro do negócio, caracterizada por:

- Identificador único no negócio, normalmente independente de outros identificadores;
- Existência independente - nenhuma **associação mandatória**, com exceção de tipos categorizantes.

Tipo categorizante é um tipo cuja instância categoriza ou classifica a instância de outro tipo. Isto não é associação de agregação nem composição, simplesmente uma associação de classificação. Todos os outros tipos provêem detalhes dos tipos-núcleo.

Criando interfaces do negócio e associando responsabilidades
A regra principal nesta etapa é criar uma interface de negócio para cada tipo-núcleo no modelo de tipo de negócio. Cada interface de negócio gerencia a informação representada pelo tipo núcleo e seus tipos detalhantes. Por freqüentemente gerenciar uma coleção de instâncias de tipos-núcleo, este tipo de interfaces de negócio é designado como interface de gerência e, portanto, são nomeadas como IxxxMgt. Ou seja, se o tipo núcleo fosse Cliente, sua interface seria IClienteMgt.

Agora que algumas interfaces de negócio iniciais foram adicionadas ao modelo de tipos de negócio, qualquer diagrama que descreve estas interfaces são referidas como **Diagramas de Responsabilidade da Interface**.

O propósito destes diagramas é esclarecer quais informações serão gerenciadas por quais interfaces e começar a pensar em dependências. Para fazer isso, cada interface deve possuir seus tipos detalhantes apropriados. Cada tipo deve ser possuído por exatamente uma interface. Alocação de tipo para interfaces é mostrado por uma associação de composição, representado pelo símbolo do diamante sólido.

Se o tipo detalhante apenas provê detalhe para apenas um tipo, este é alocado para a mesma interface deste tipo. Se o tipo detalhante detalha mais de um tipo e estes estão todos alocados na mesma interface, aquele tipo pertence a esta interface também. O problema ocorre quando um tipo detalha outros tipos que estão alocados em diferentes interfaces. A solução é escolher a posse do tipo detalhante para quem estiver mais acoplado. A referência entre o tipo detalhante e o tipo menos acoplado é feita utilizando o símbolo de associação com a direção de navegação apenas daquele para este.

Segundo os autores, criar bons sistemas usando componentes é primeiramente um problema de gerência de dependências. Por esta razão, realmente é desejado evitar, quando possível, referências de dupla orientação entre interfaces. Com isso, sugerem que seja determinada apenas navegação em um único sentido para todas as associações entre interfaces.

4.3 Identificação de serviços e restrições de cada componente

Neste ponto do método, já é sabido quais são os componentes de sistema e de negócios e suas interfaces. As assinaturas das interfaces de sistema ainda não são conhecidas. E quanto às interfaces de negócio, seus métodos são desconhecidos. Deve-se descobrir quais são os serviços providos, analisando quais são e o que fazem os métodos dos componentes identificados. Isto é feito em duas etapas: interação de componente e especificação de componente.

4.3.1 Interação de componente

O estágio de interação dos componentes vem justamente examinar como cada operação da interface de sistema vai ser atingida, usando a arquitetura do componente. O **diagrama de colaboração** da *UML* é utilizado para modelar tais interações. Assim como é descrito na Figura 4.1, interfaces, especificações de com-

ponentes e arquitetura serão os insumos para esta etapa.

Ao final, as assinaturas dos métodos das interfaces de negócio e sistemas serão descobertas. Escolhas de responsabilidade tornam-se mais claras e operações são movidas de uma interface para outra. Agrupamentos alternativos de interfaces em componentes podem ser investigados. É também o momento para pensar como gerenciar as referências entre objetos de componentes, para que dependências sejam minimizadas e a política de integridade referencial seja acomodada.

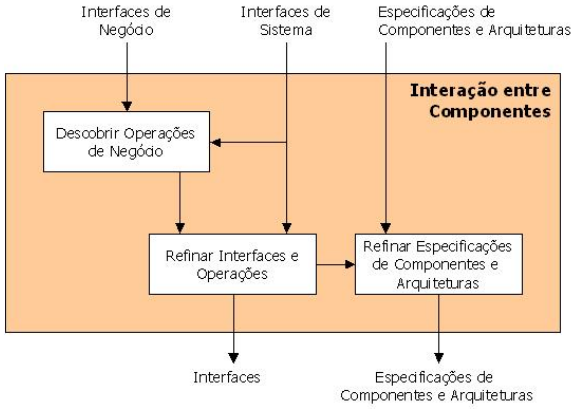


Figura 4.1: Interação de componente.

4.3.2 Especificação de componentes

Uma grande motivação para realizar este tipo de abordagem é montar sistemas com componentes desacoplados. São componentes que podem ser re-utilizados em outros sistemas, em outros contextos e com confiabilidade. Desta forma, deve-se especificar cada componente para garantir precisamente o que faz ou o que deixa de fazer e quais são suas restrições com as outras partes do sistema.

Nesta etapa, será criada uma especificação completa e sem ambigüidades das interfaces que descrevem precisamente o comportamento das operações. Verifica-se na Figura 4.2 que artefatos gerados nesta etapa são: interfaces de negócio e sistema, e especificações de componentes e arquitetura.

Para fazer esta especificação, utiliza-se a abordagem `Design by Contract`. De-

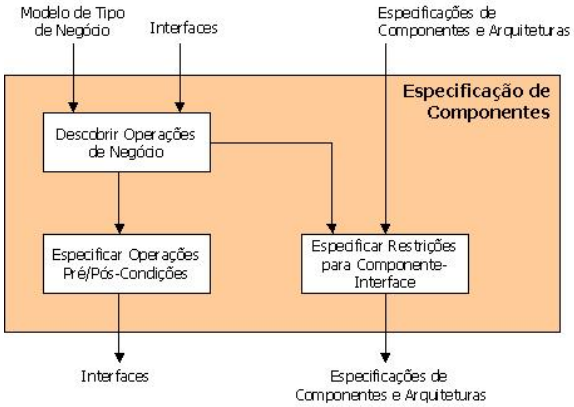


Figura 4.2: Especificação de componente.

senvolvida por Meyer [14], esta abordagem já se mostrou bastante útil para fazer esta especificação e funciona como os contratos vistos no capítulo de visão geral.

Existem dois tipos distintos de contratos :

- **Contrato de uso:** o contrato entre uma interface de objeto de componente e seu cliente;
- **Contratos de realização:** o contrato entre uma especificação de componente e sua implementação.

Contrato de uso

Os contratos de uso são definidos para cada método da interface. Suas assertivas estão baseadas no **Modelo de Informação de Interface**.

Este modelo é utilizado para representar os estados que podem ser assumidos por objeto de componente através das operações definidas na interface deste. Este modelo de tipo deve conter informação suficiente para permitir que o contrato da operação da interface seja definido. É especificado apenas o conjunto de estados que o objeto de componente pode assumir e não descreve o modo pelo qual o estado é implementado ou persiste.

Contrato de realização

Neste momento, é provida a informação de especificação adicional que deve estar clara para o montador e implementador do componente, especialmente a que relata as dependências entre componentes e outras interfaces.

Para cada especificação de componente, é preciso dizer quais interfaces estas realizações devem suportar. Já foi feito isto no diagrama de arquitetura de componente, mas este diagrama deve ser dissecado em pedaços específicos para cada componente.

4.4 Representação de componentes

O *UML Components* difere do *Catalysis* quanto a distinção clara entre especificação de componente e interface de componente. Uma especificação de componente pode estar relacionada a várias especificações de interfaces. Esta especificação de componente deve informar as interfaces que o componente oferece para o sistema e as interfaces usadas durante a realização de seus serviços. Os relacionamentos existentes entre as interfaces oferecidas devem estar explicitados. Cada especificação de interface deve conter o seu modelo de informação e seus métodos contratualmente especificados.

Capítulo 5

Estudo de caso 1: *Catalysis*

5.1 Processo Utilizado para Modelagem

O *Catalysis* não define um método formal de processo para a modelagem e desenvolvimento. Ele permite uma flexibilidade na ordem de geração dos artefatos. Mas, no livro texto, ele cita como exemplo uma ordem possível a ser seguida.

Processo Sugerido pelo *Catalysis*

No processo citado como exemplo em [20] (pág. 559-660), os artefatos devem ser gerados na seguinte ordem:

1. *Business Case*: Fornece os requisitos iniciais, definindo o problema de negócio ou a oportunidade que o projeto vai tratar.
2. Modelo de Domínio ou de Negócio: Descreve o domínio ou o negócio em mãos independente de soluções particulares de software.
3. Sessões JAD: Workshop estruturado para chegar a um entendimento comum dos requisitos do sistema.
4. Glossário: Um conjunto inicial de definições de termos utilizados para definir o problema ou os requisitos.
5. *UI Sketches*: Rascunhos de interface gráfica, com suas ligações e seus fluxos.

6. Modelo de Tipos + Especificações de Sistema: O contexto de sistema é definido, tendo com o artefato final as primeiras ações do sistema especificadas abstratamente.

7. *Subject Areas*: Divisão do sistema em áreas de interesse com o objetivo de particionar o sistema em várias partes.

8. *Frameworks*: O projeto pode apresentar estruturas genéricas (*Frameworks* de problemas) e que podem ser reutilizadas. Estes *Frameworks* são descobertos, listados e modelados.

9. Refatoração para re-uso: Divisão do sistema em especificação do sistema e modelagem interna, onde a modelagem interna será adaptada (ou refatorada) para reutilizações futuras.

10. Regras de modelagem para a arquitetura técnica: Criação de padrões para lidar com a infraestrutura técnica do sistema.

11. Modelo de arquitetura técnica: Descreve a colaboração entre os componentes de infraestrutura no nível arquitetural técnico.

12. Pedaços horizontais: Divisão do modelo arquitetural em pedaços horizontais, com o objetivo de poder adicionar novas funcionalidades para o usuário final.

13. Arquitetura da Aplicação: Arquitetura final da aplicação, mostrando todos seus componentes (externos ou não) e suas iterações.

14. Padrões de Desenvolvimento para o Projeto: Definição dos artefatos para cada fase do projeto.

15. Plano de Projeto: Plano geral do projeto, listando todas as atividades para todo o projeto.

16. Instalação: Instalação do sistema e a preparação para a instalação, listando requisitos de hardware e software para tal.

5.1.1 Processo Utilizado

Como o *Catalysis* permite uma flexibilização no processo de modelagem, e não define uma ordem de artefatos de modelagem a serem gerados, foi definida uma ordem de artefatos para a preparação do estudo de caso apresentado neste trabalho. Esta ordem foi gerada independente da ordem do exemplo do *Catalysis*, apesar de apresentar semelhanças.

Foram levados em conta o fato da modelagem estar completa, o fato de não haver desenvolvimento de código e a simplicidade do estudo de caso.

A ordem utilizada foi:

1. Casos de uso: A partir da modelagem, foram criados os casos de uso. Tentou-se utilizar a notação e a forma de casos de uso que o *Catalysis* adota, mas em função da simplicidade, isso não foi possível. Portanto, os casos de uso em *Catalysis* estão iguais aos casos de uso apresentados para o *UML Components*.

2. Glossário: Foi criado um glossário com termos básicos, assim como o *Catalysis* sugere.

3. Ações e Cenários: Assim como sugerido pelo *Catalysis*, foram criados as ações e cenários a partir da modelagem dada. Este método mostrou-se bastante eficiente para diminuir ambigüidades e serviu para gerar um modelo de tipos inicial. Os cenários também ajudam na criação da especificação OCL das ações.

4. UI Sketches: Assim como sugerido como parte do processo pelo *Catalysis*, mas sem entrar em detalhes mais profundos, foram criados rascunhos de interfaces gráficas para o sistema proposto. Mostrou-se bastante eficiente na eliminação de dúvidas e ambigüidades, além de ajudar a montar o modelo de tipos final.

5. Modelo de Tipos: Através da ajuda dos métodos anteriores (Ações, Cenários e UI Sketches) foi criado um modelo de tipos sem ambigüidades, através da ajuda dos métodos anteriores.

6. Refinamento da Modelagem: Poderia ser feito, caso necessário, um refinamento em preparação para a implementação.

5.2 Descrição e Requisitos do Sistema

5.2.1 Introdução

A modelagem, contida em [9], não contém referência para a descrição e/ou requisitos do problema. A modelagem assume que a análise feita em cima do problema é de tal maneira perfeita, de modo que não há inconsistências e não foi necessário nenhum passo adicional para completá-la.

Na maior parte dos casos, isso não é possível. É muito difícil ter-se uma análise completa e sem defeitos. Além disso, a modelagem apresentada no livro do *UML Components* parece não supor mudanças nos requisitos do sistema.

O *Catalysis* assume que a análise não é perfeita nem completa de antemão, e por isso, supõe que a modelagem irá cobrir algumas inconsistências e ambigüidades da análise e da própria modelagem em si. Segundo D'Souza e Wills [20], através da escrita de diagramas mais abstratos e de seus refinamentos, é possível eliminar grande parte das ambigüidades e inconsistências dos modelos.

A descrição e os requisitos aqui apresentados foram feitos pelo autor, levando em consideração a modelagem proposta em [9]. O modelo dado não inclui regras de negócios mais complexas, e ao mesmo tempo comuns, como por exemplo: Até quanto tempo antes um cliente pode cancelar uma reserva? Qual é o intervalo mínimo entre a reserva e o início de uma estadia? Existem políticas de preços sazonais?

5.2.2 Descrição e Requisitos

O sistema serve para o controle de vários hotéis. Um cliente pode fazer uma requisição de reserva e iniciar a sua estadia. Na listagem de casos de uso, exis-

tem outras funcionalidades do sistema, como cancelar reserva, administrar quartos, etc., mas não há nenhuma modelagem específica, de modo que possa ser feita uma descrição.

Ao fazer uma reserva, o cliente indica seus dados pessoas (nome, cep e e-mail) e também qual hotel ficará hospedado, quais serão as datas de sua estadia e qual é o tipo de quarto desejado pelo cliente (duas camas, três camas, presidencial, etc.). Ao terminar de fazer uma reserva, o sistema gera um identificador desta reserva e envia um e-mail para o cliente com este.

Caso não haja o tipo quarto no hotel desejado pelo cliente, o sistema permite que o cliente faça uma nova seleção.

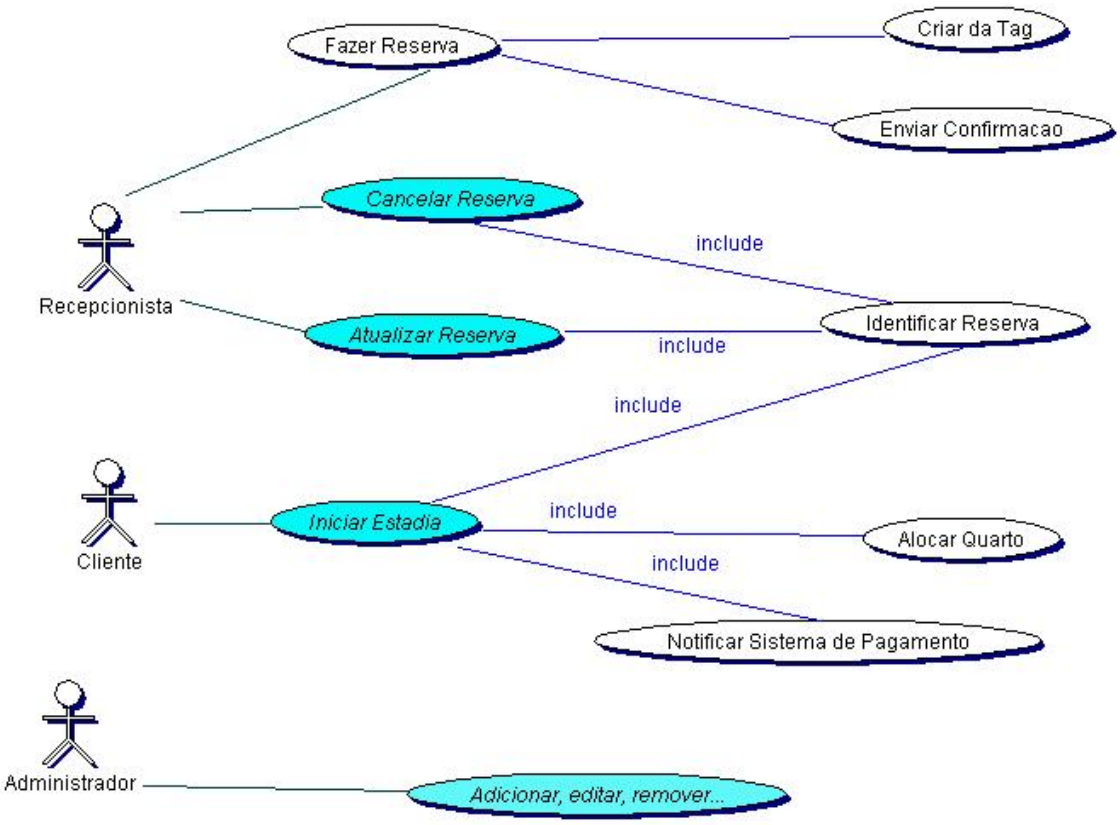
A qualquer momento, o cliente pode solicitar a sua estadia no hotel. Para tanto, ele deve informar o código da sua reserva. Caso o cliente não saiba o código, ele pode informar o seu nome, cep e email, que o sistema listará as reservas desse cliente. Ao terminar de solicitar a sua estadia, o sistema aloca um quarto para o cliente. Em [9], não foi informado o método para alocação de um quarto para um cliente.

5.3 Modelagem

Nesta seção é apresentada a modelagem, seguindo o processo descrito na primeira seção deste capítulo. Dentro de cada sub-seção a seguir, mostramos a modelagem de um artefato, como descrito no processo, e também a explicação sobre como ela foi feita.

5.3.1 Casos de uso

Os casos de uso do sistema foram os mesmos da modelagem apresentada no exemplo do livro do *UML Components*. No *Catalysis*, utiliza-se muito o conceito de casos de uso abstratos, quando ainda não é possível determinar com exatidão o caso de uso e postegar decisões.


Fazer Reserva
Criar da Tag
Enviar Confirmacao
Cancelar Reserva
include
Recepcionista
Atualizar Reserva
Identificar Reserva
include
include
Cliente
Iniciar Estadia
include
Alocar Quarto
include
Notificar Sistema de Pagamento
Administrador
Adicionar, editar, remover...

Nome Fazer Reserva
Iniciador Atendente
Objetivo Reservar quartos no hotel.
Cenário Principal
1 O atendente pede para fazer uma reserva.
2 O atendente seleciona em qualquer ordem o hote, as datas e o tipo de quarto.
3 O sistema mostra os preços para o atendente
4 O atendente pede uma reserva.
5 O atendente prove o nome, o cep e o e-mail.
6 O sistema faz a reserva e aloca uma *tag* para a reserva.
7 O sistema mostra a tag para o atendente.
8 O sistema cria e envia a confirmação por e-mail.
Extensões
3.a Quarto não disponível.
3.a.1 Sistema oferece alternativas.
3.a.2 Atendente seleciona uma das alternativas.
3.a.2.i O atendente rejeita alternativas: **Falha**
4 Atendente rejeita a oferta: **Falha**
6 O cliente já está no arquivo (baseado no nome e no cep)
6.a Continua no passo 7.

Nome Iniciar Estadia
Iniciador Cliente
Objetivo Reinvidicar uma reserva e fazer *check-in* no hotel.
Cenário Principal
1 O cliente chega ao hotel e reinvidica uma reserva.
2 Inclui **Identificar Reserva**.
3 Cliente confirma detalhes da estadia e tipo de quarto.
4 Sistema aloca um quarto.
5 Sistema notifica o sistema de pagamento que uma estadia está sendo iniciada.
Extensões
3 Reserva não identificada.
3.a **Falha**.

Nome Identificar Reserva
Iniciador Somente incluso
Objetivo Identificar uma reserva existente.
Cenário Principal
1 O ator prove a *tag* de reserva.
2 O sistema localiza a reserva.
Extensões
2 Sistema não pode localizar uma reserva com a *tag* data.
2.a Ator prove nome e cep.
2.b Sistema mostra reserva ativas para este cliente.
2.c Ator seleciona uma reserva.
2 A Tag de reserva refere para uma reserva em um hotel diferente.
2.a *Falha.*
2 Sem reservas ativas neste hotel para este cliente.
2.a *Falha.*

5.3.2 Glossário

Como sugerido no *Catalysis* foi feito um glossário com os termos comuns no domínio do sistema. Este artefato pode ser de muita importância quando se desenvolve sistemas para domínios não conhecidos.

Mas mesmo no caso de um domínio de um sistema de hotel, onde, em geral, já se conhece os termos utilizados, este artefato foi de fundamental importância para a eliminação de ambigüidades e fixação dos conceitos. Este artefato permitiu perceber uma ambigüidade na modelagem em *UML Components* com relação ao número de hotéis que o sistema irá gerenciar. Neste estudo de caso, estamos assumindo que o sistema irá gerenciar vários hotéis.

Hotel: Cada hotel no sistema tem seus clientes e suas reservas. Cada hotel tem seus quartos, sendo que cada quarto é de um Tipo de Quarto específico.

Cliente: Cliente de um hotel pode conter mais de uma reserva em hotéis distintos. Vale lembrar que se o cliente se hospeda no hotel A e no hotel B, ele terá dois cadastros.

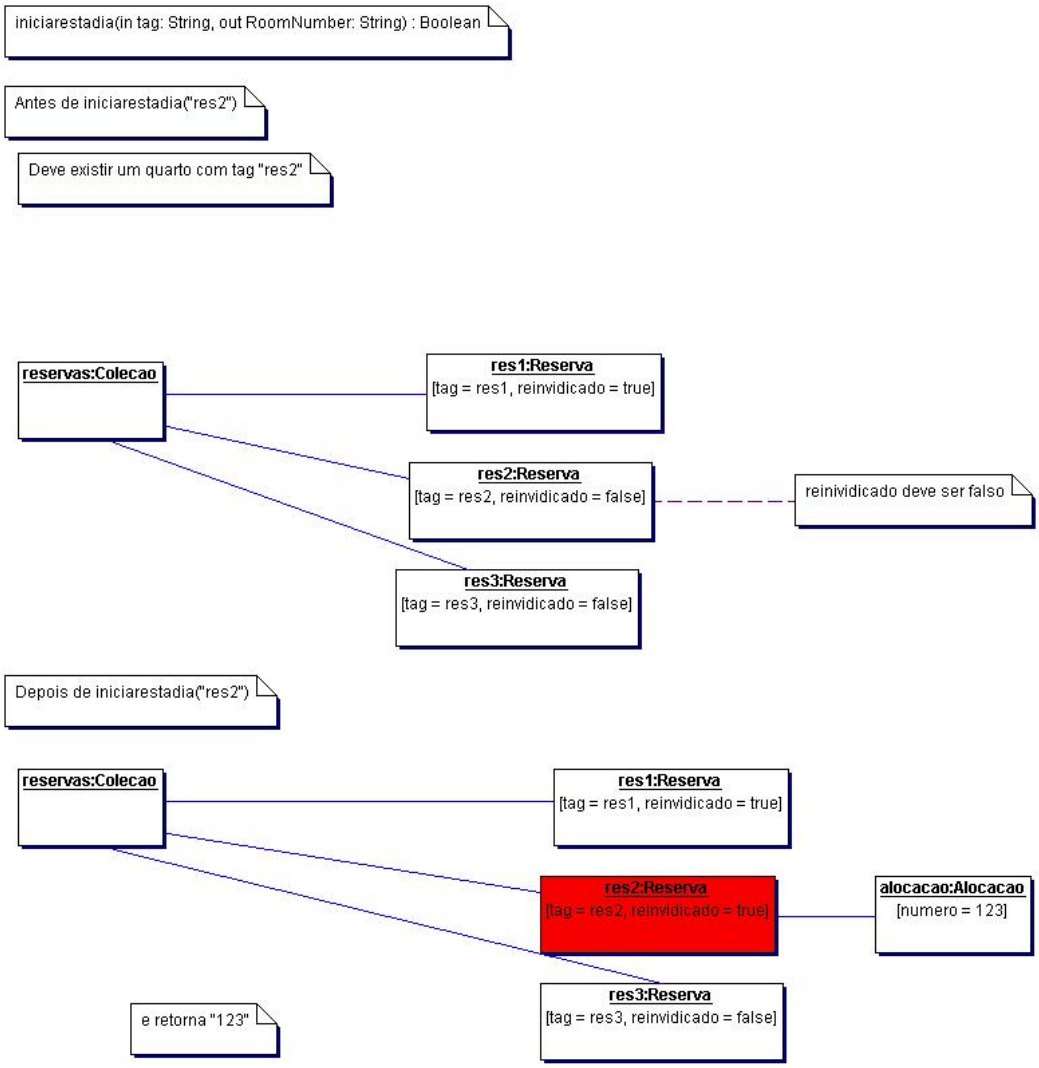
Quarto: Quarto de um hotel. Cada quarto é de um tipo.

Tipo de Quarto: Descreve o tipo de quarto. Pode descrever quantos banheiros tem, se têm TV, DVD, etc.

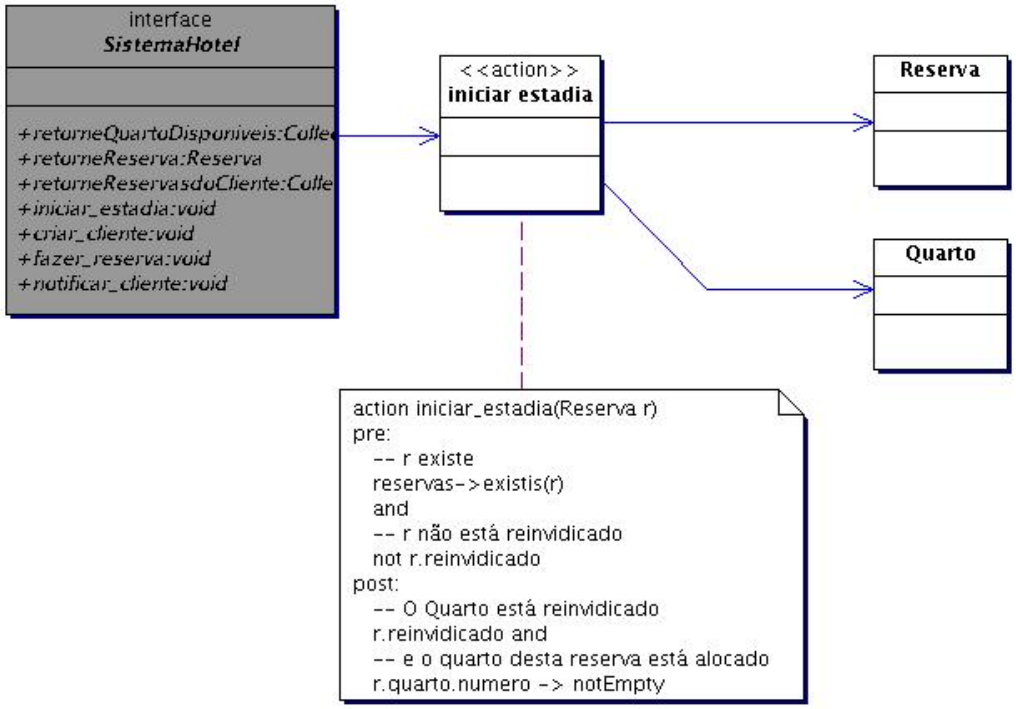
5.3.3 Ações e Cenários

Já com os casos de uso e o glossário, já é possível modelar algumas ações e cenários. Como ainda não existe conhecimento sobre o sistema, é recomendado montar cenários inicialmente para casos particulares, e depois generalizar estes cenários através da modelagem de ações.

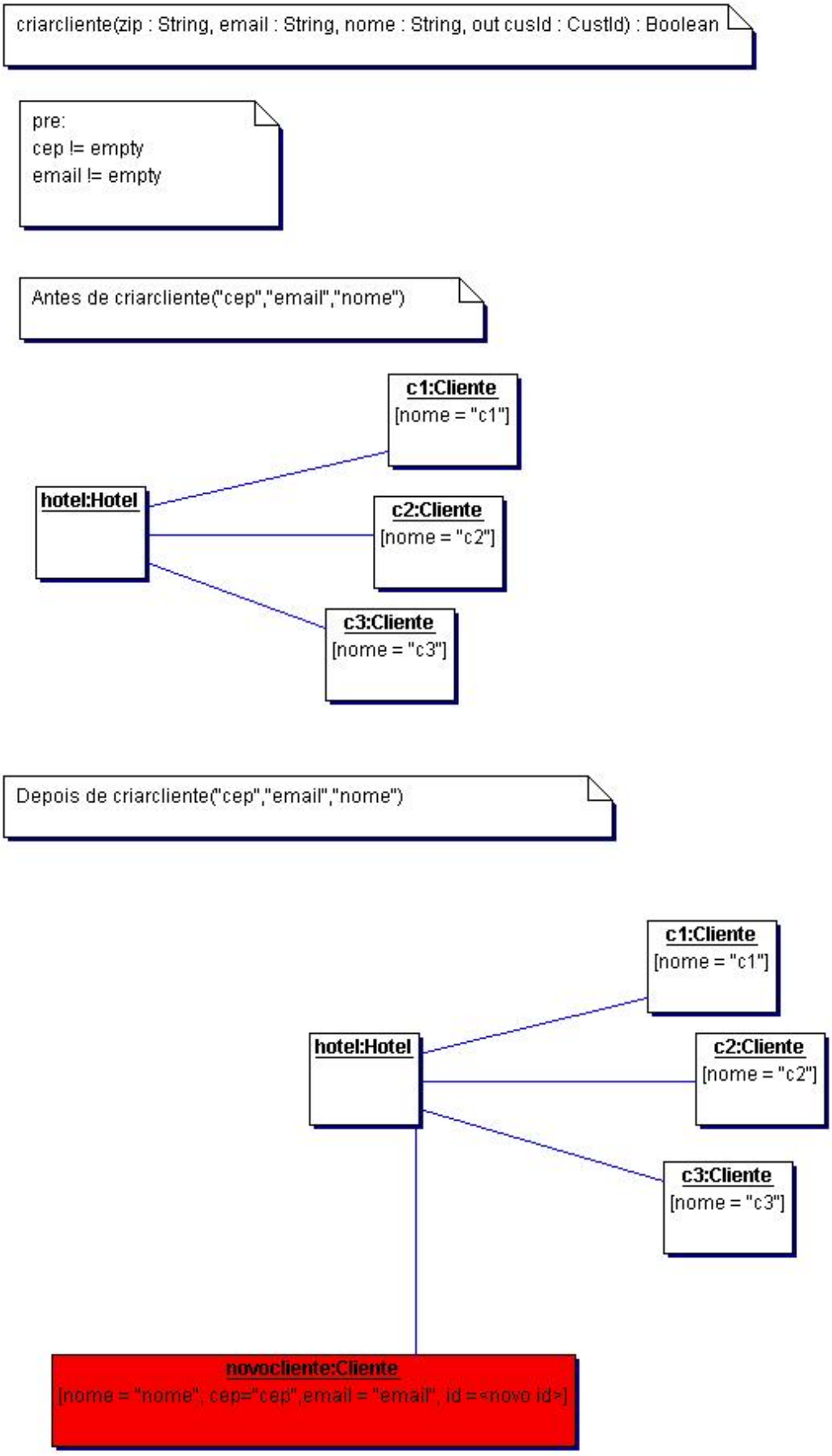
Cenário iniciarestadia



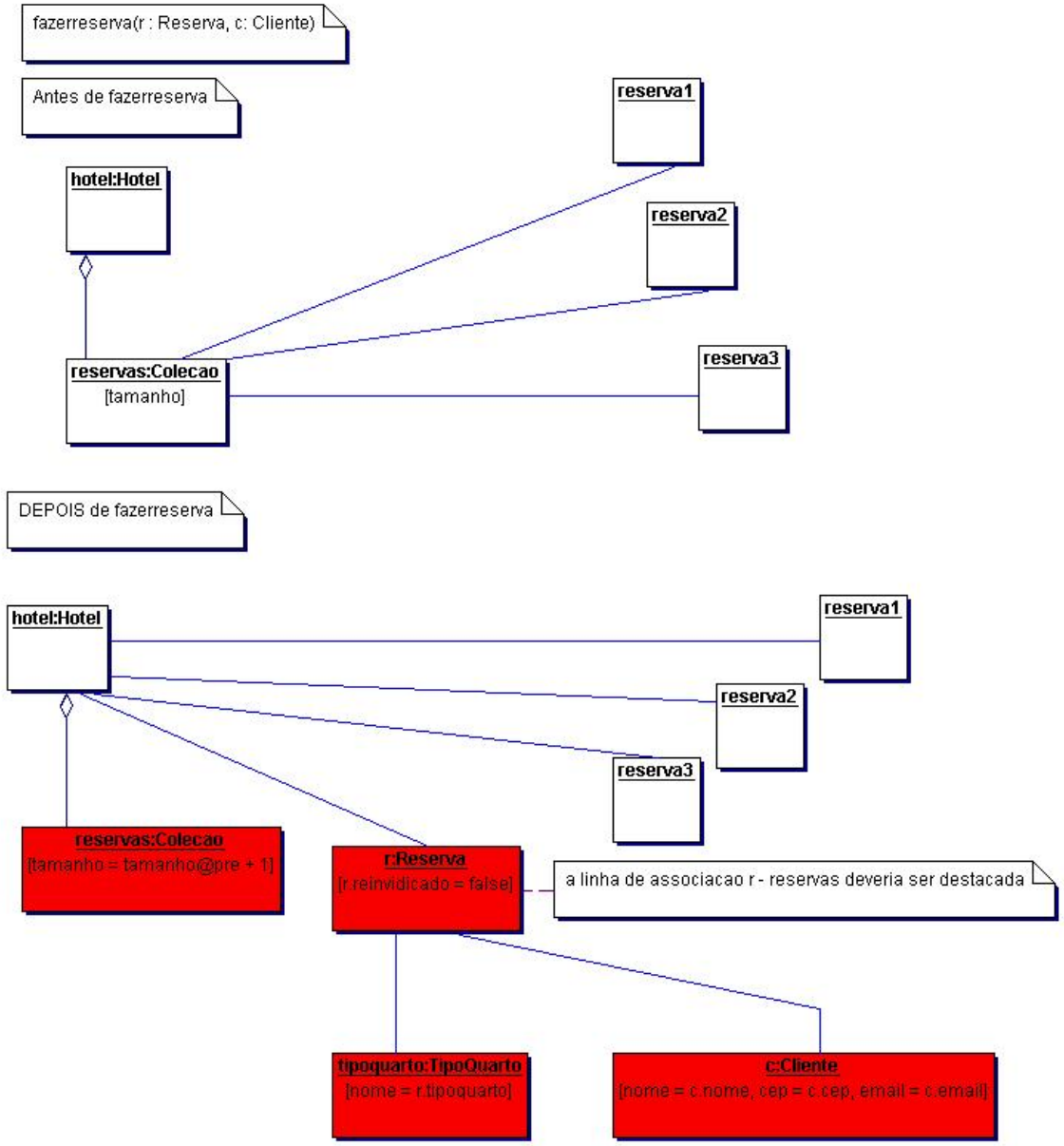
Ação iniciarestadia



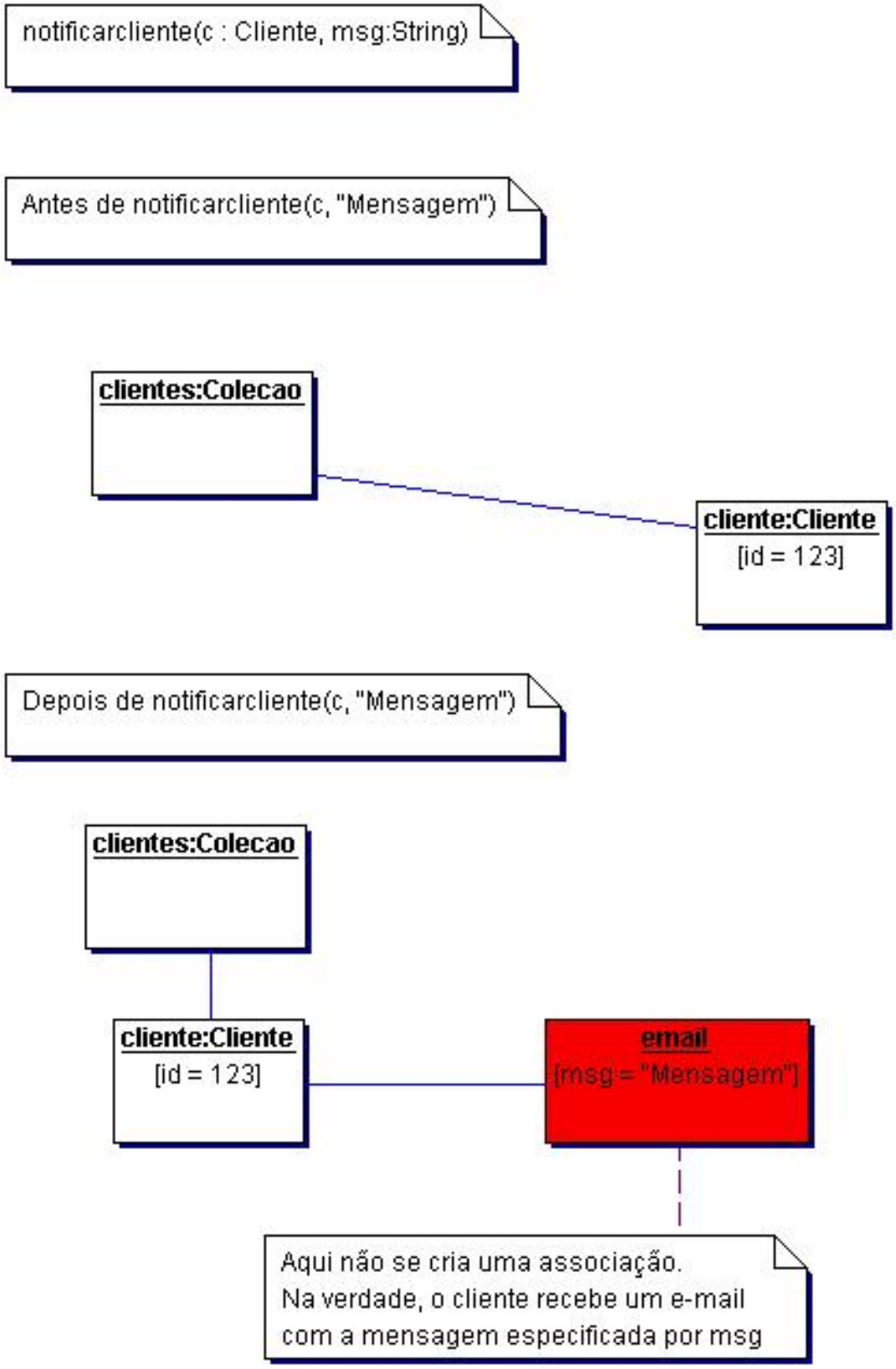
criarcliente



fazerreserva

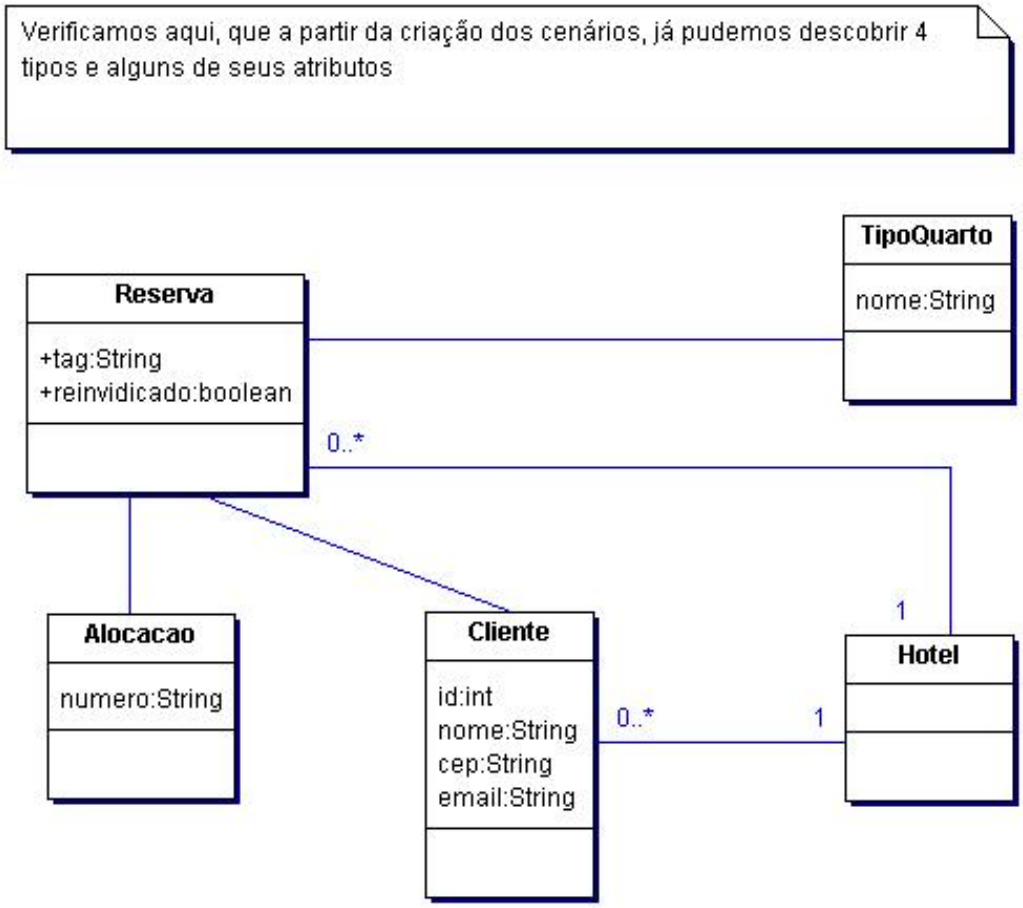


notificarcliente



5.3.4 Modelo Inicial de Tipos

A partir dos cenários criados, já é possível determinar um modelo de tipos básico que o sistema terá. Este modelo está ilustrado a seguir:



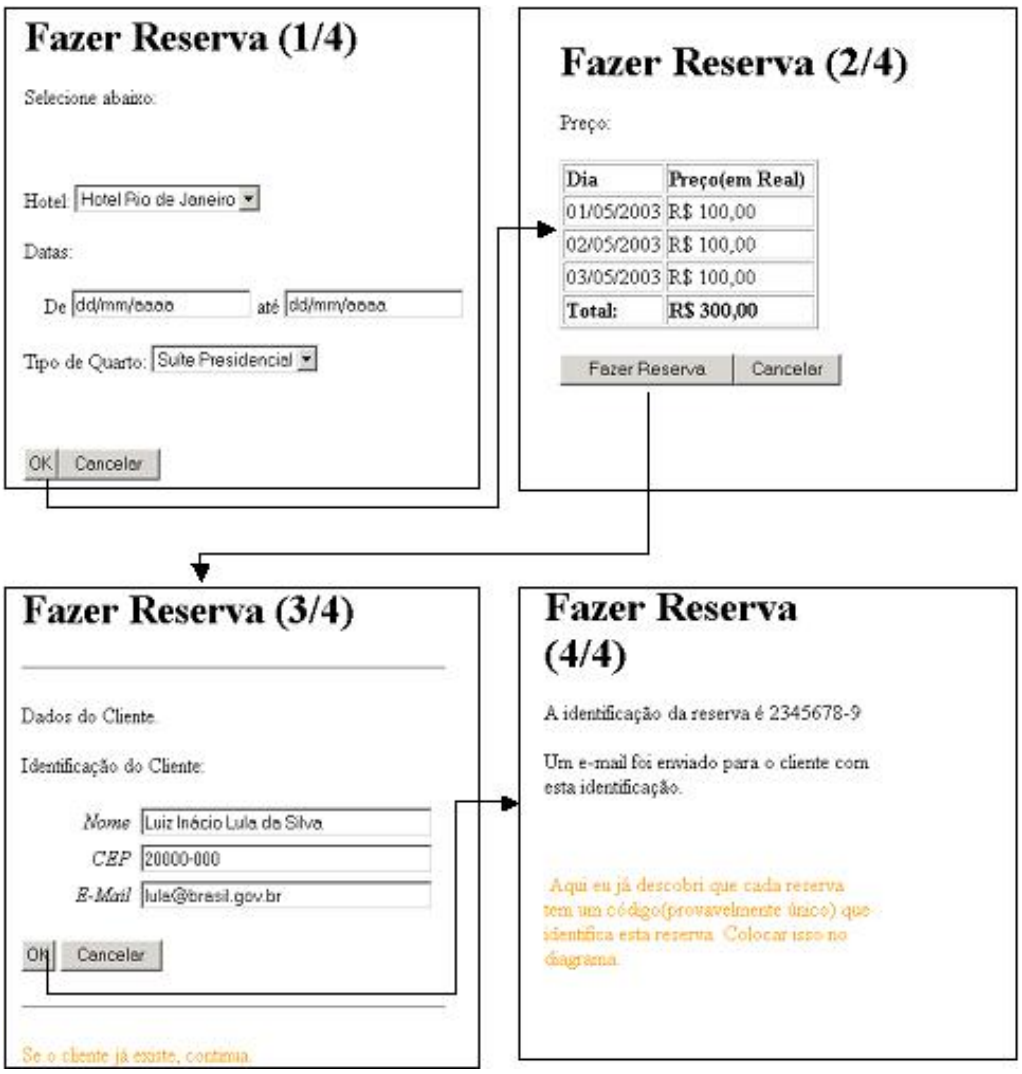
5.3.5 UI Sketch

No estudo de caso de hotel apresentado no livro *UML Components*, nada é dito em relação à interface de utilização do sistema. Então, foi construído um rascunho de interface gráfica (*UI Sketch*) para este sistema, baseado na descrição do problema e também na modelagem em *UML Components*.

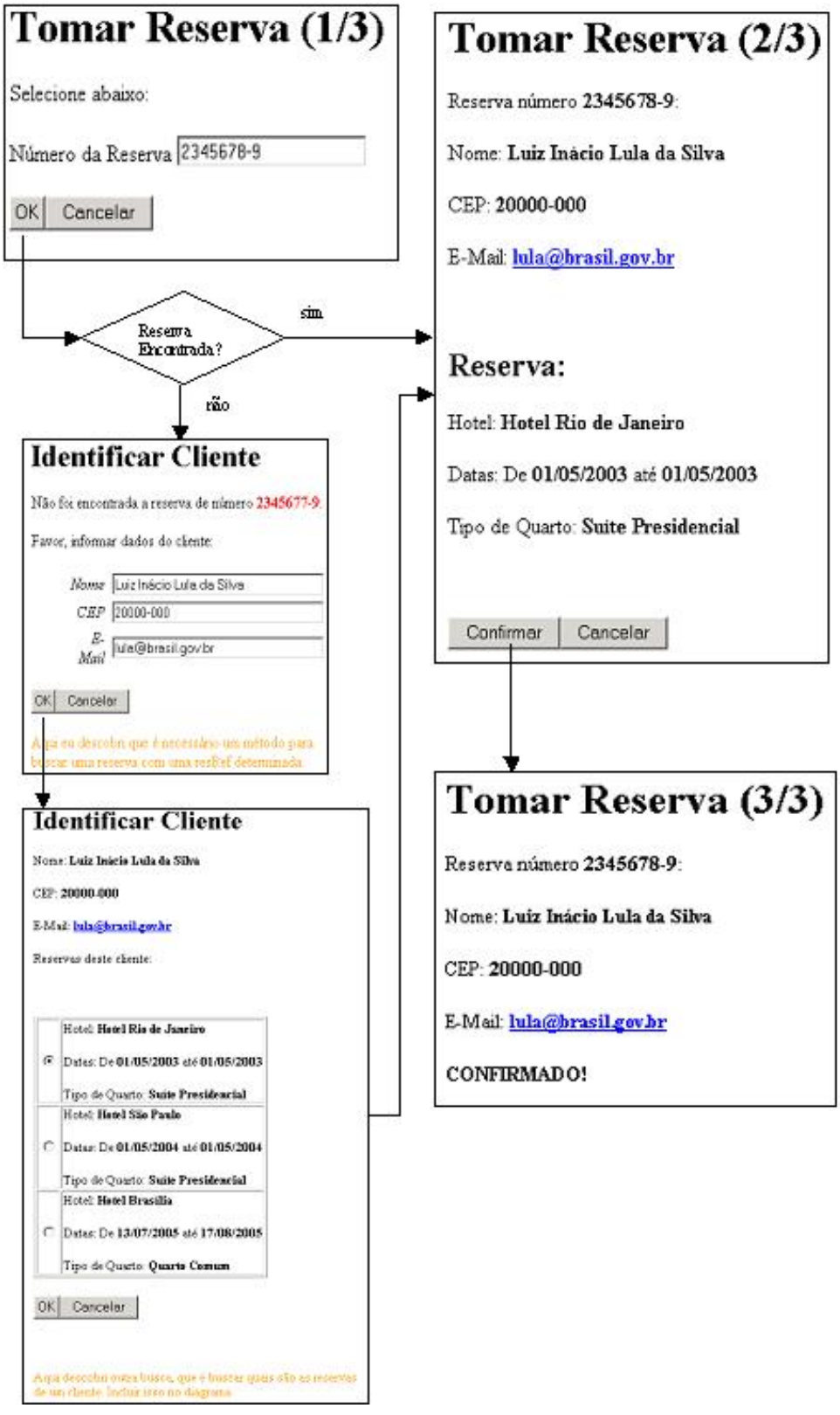
Este artefato se mostrou bastante útil na determinação do relacionamento entre tipos e também na descoberta de atributos para estes tipos. Neste *UI Sketch*, são mostrados algumas funcionalidades do sistema que não estão sendo modeladas, mas

servem de ilustração.

Fazer Reserva



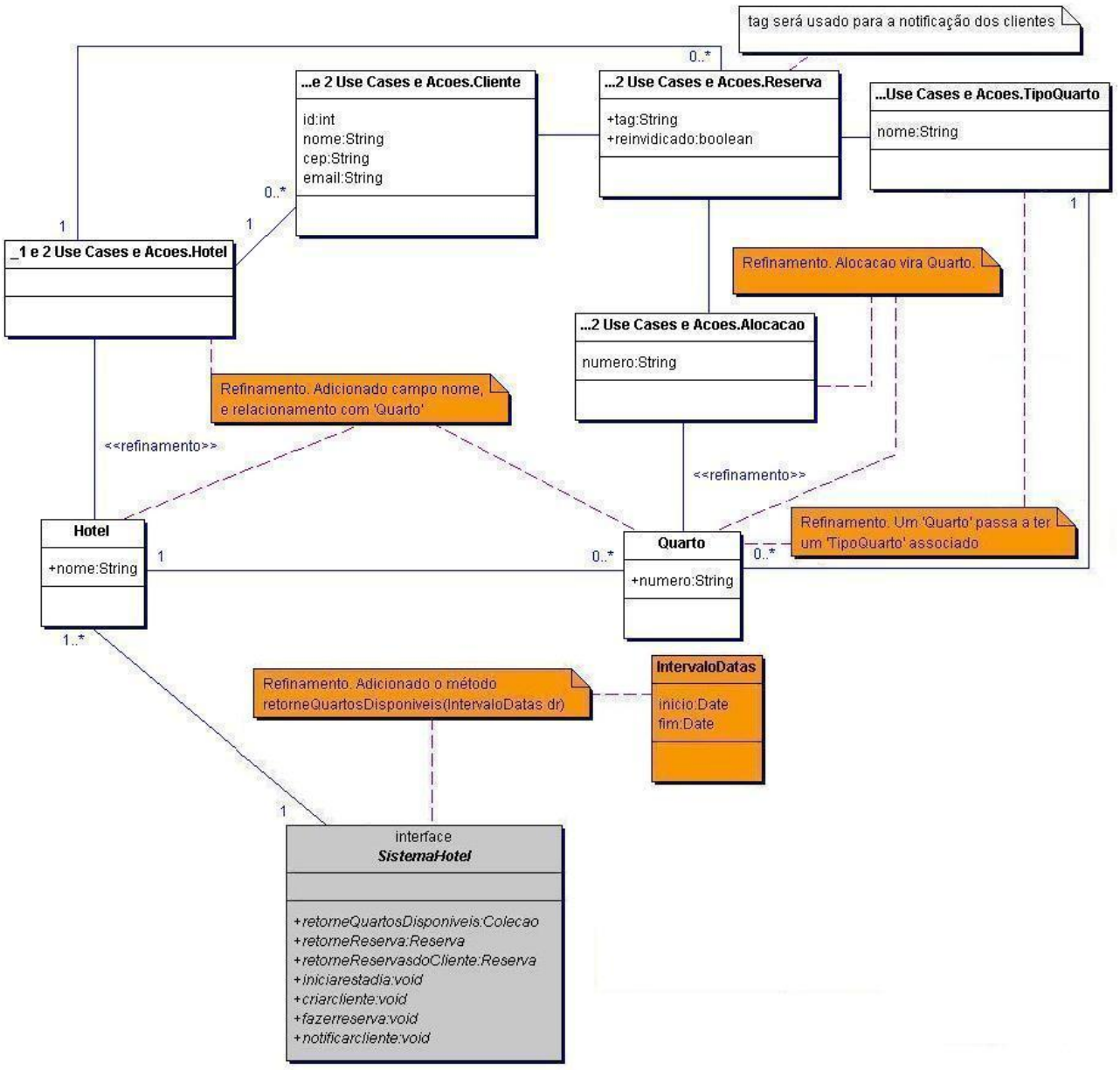
Tomar Reserva



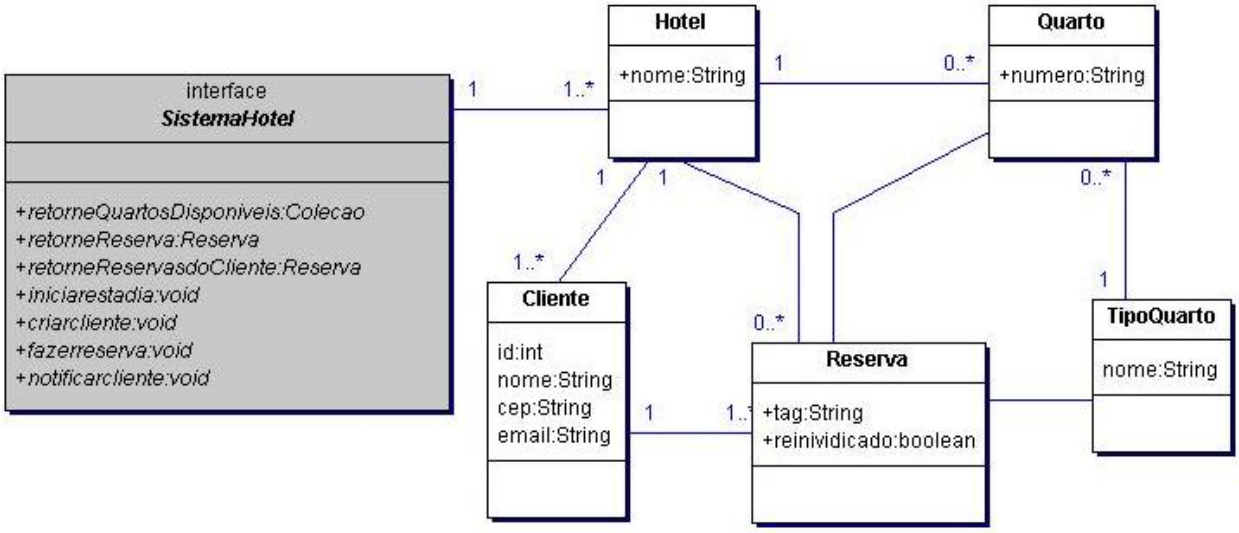
5.3.6 Modelo de Tipos

A partir do modelo inicial de tipos, feito por meio dos cenários e casos de uso, foi possível saber quais são os tipos básicos.

Depois de feitos os rascunhos da interface gráfica, foi possível descobrir algumas cardinalidades que estavam faltando no modelo inicial, assim como novos campos e até mesmo uma refatoração do modelo. O refinamento do modelo inicial para o modelo final está a seguir:



E o modelo final ficou:



Capítulo 6

Estudo de caso 2: *UML Components*

6.1 *UML Components*

UML Components é basicamente um processo de desenvolvimento. Como foi dito anteriormente, não trata o processo de gerência. É um método bastante simples. Os seus passos são bem definidos, assim como o fluxo de seus artefatos. Seus *workflows* estão organizados da seguinte forma e na seguinte ordem:

1. Requisitos
2. Especificação
   (a) Identificação de componentes
   (b) Interação de componentes
   (c) Especificação de componentes
3. Provisionamento
4. Montagem

O foco maior do método está no *workflow* de especificação. Como toda especificação depende dos requisitos gerados, o método também oferece uma guia para

criar o modelo conceitual e os casos de uso. Mas como todo guia, o desenvolvedor tem alguma liberdade para encontrar a melhor forma de completar a atividade.

6.2 Requisitos

Nesta seção definimos quais são os requisitos de sistema e apresentamos os modelos gerados.

6.2.1 Descrição do negócio

Uma vídeo locadora vende e aluga vídeos para pessoas. Uma pessoa necessita ser um membro desta loja para fazer aluguéis. Qualquer um pode comprar uma cópia de vídeo.

Membros podem reservar vídeos para alugar apenas se todas as cópias deste vídeo estão alugadas. Quando um vídeo retorna, o membro com a reserva será contactado e o vídeo será retido durante 3 dias. Se o membro não retirar a cópia até o prazo, a reserva será cancelada. Taxas de aluguel (R$ 2 por 2 dias) são pagas no momento da retirada. Devoluções atrasadas são taxadas (em R$ 1 por dia), e taxas devem ser pagas antes que outro vídeo seja alugado.

Um estoque limitado de vídeos são destinados à venda, mas um membro pode fazer um pedido de vídeo para compra.

Contas fechadas são apagadas após 3 anos.

Um membro pode cadastrar-se para ser informado a respeito de novos títulos.

6.2.2 Escopo do sistema

O sistema deve suportar as operações deste negócio. Diferentes atores irão usar este sistema: o atendente e o cliente.

- O atendente adiciona novos membro;
- O atendente aluga e faz devolução de vídeos em nome dos clientes;
- O atendente cobra o pagamento de taxas;
- O cliente pode procurar por título (por nomes) e pode fazer reservas por título;
- Membros podem se cadastrar para serem informados a respeito de novos títulos.

6.2.3 Observações

Este método assume que se tem o conhecimento geral do sistema no momento inicial. Indica um processo de especificação de início, meio e fim. Não mostra como fazer adição ou alteração de requisitos no meio do processo. As estatísticas armazenadas serviriam apenas para indicar regras de negócio adicionais e conseqüentemente novos atributos, não sendo necessária a criação de classes especializadas para o armazenamento de tais estatísticas.

6.2.4 Processos do negócio

Foram identificados os seguintes processos do negócio: alugar vídeo e comprar vídeo. Como a compra é um processo simples, apenas o diagrama de atividades do aluguel foi trabalhado. Veja a Figura 6.1.

6.2.5 Modelagem conceitual de negócio

Fazendo uma nova leitura da descrição do problema, foram identificados 7 conceitos envolvidos. Estes conceitos e suas associações estão representados na Figura 6.2.

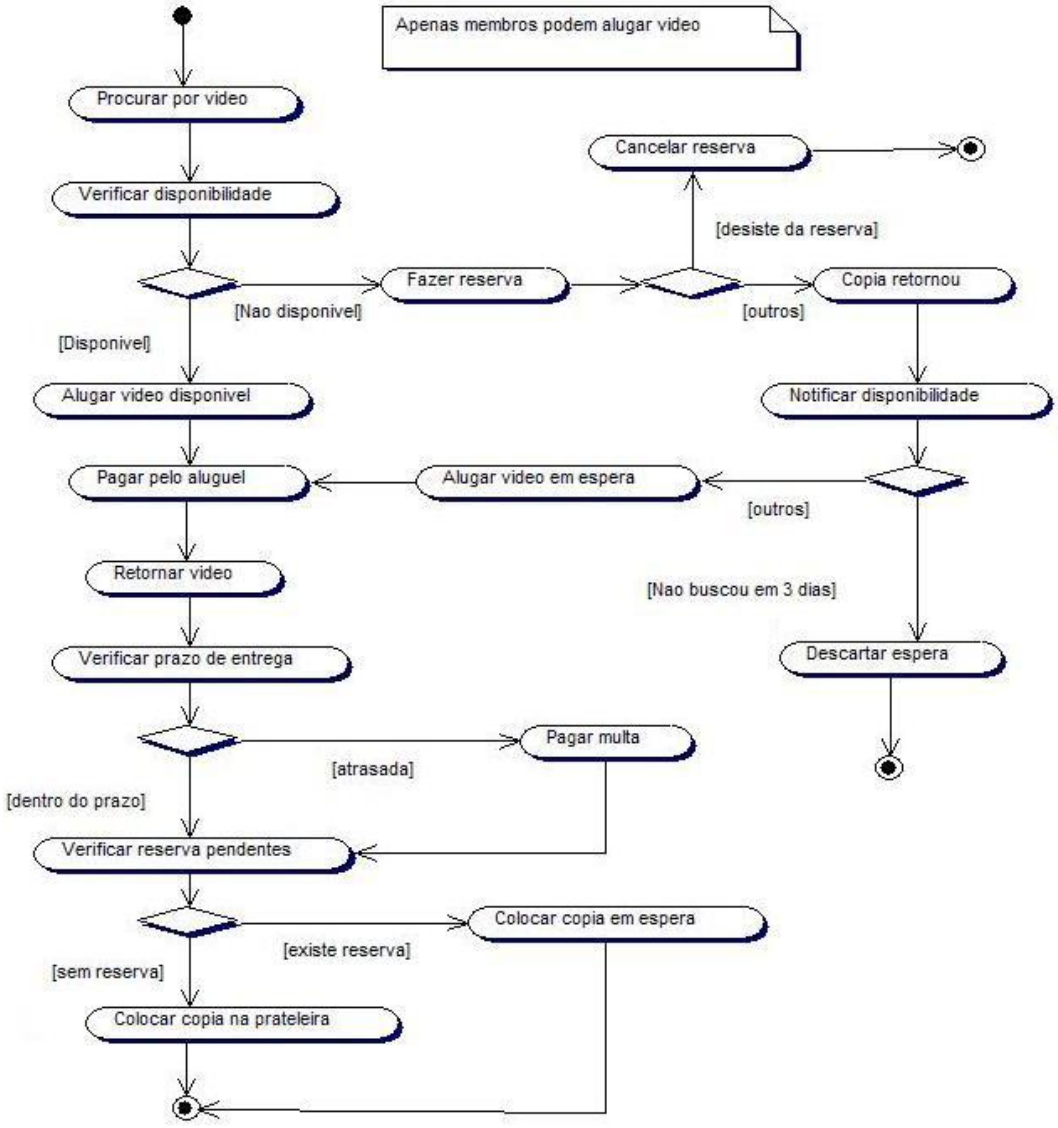


Figura 6.1: Processo de negócio para o aluguel.

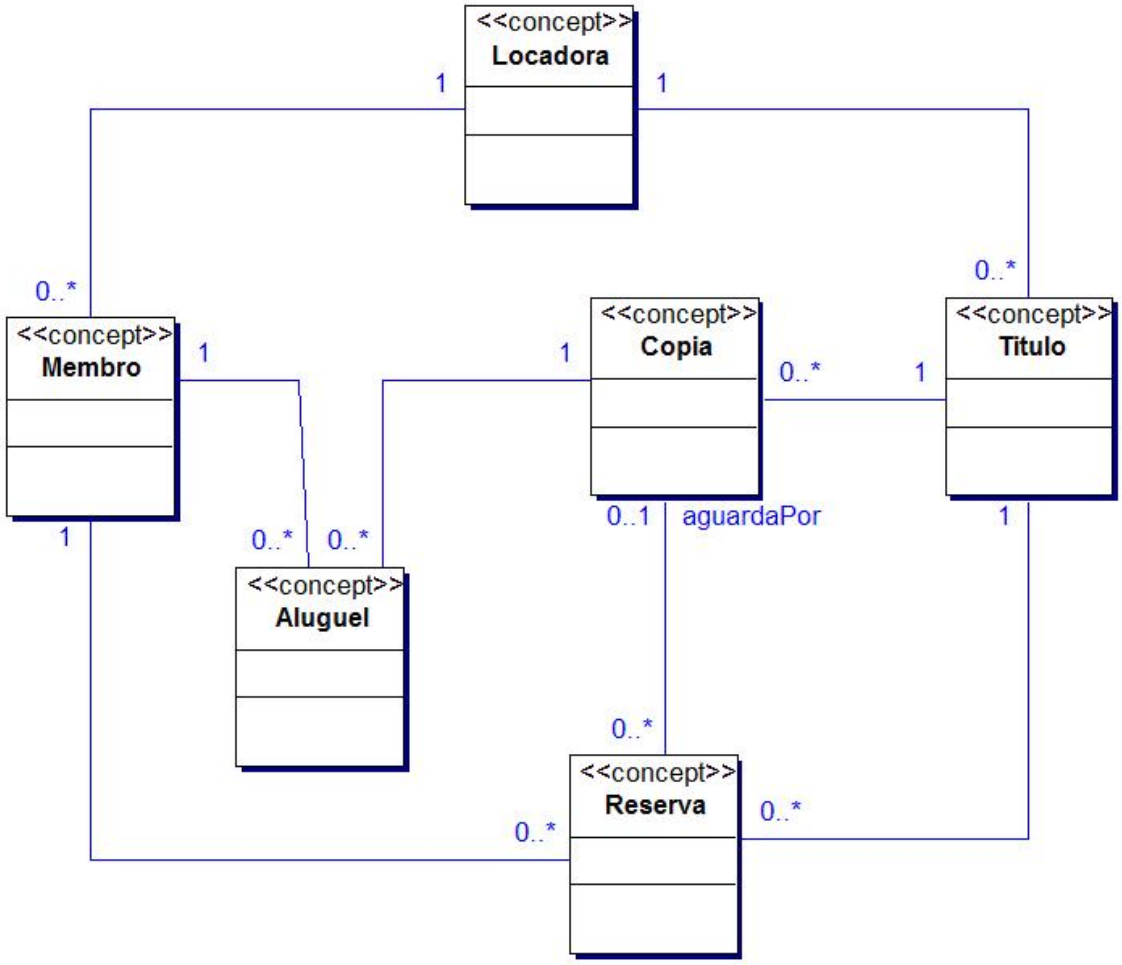


Figura 6.2: Modelagem conceitual.

6.2.6 Diagrama de casos de uso

Entendidos o processo e o conceito do negócio que está sendo tratado, já pode ser criado um diagrama de casos de uso, como na 6.3, enumerando os casos de uso identificados. Iniciando pelos casos de uso envolvidos no processo: alugar, devolver cópia, pagar multa, cadastrar membro, cancelar reserva, cancelar aluguel, comprar cópia, procurar vídeo, cadastrar em mala direta e reservar. O único caso de uso que envolve relatórios é gerar relatórios. Para finalizar, são verificados se os casos de uso já citados englobam todas as atividades de manutenção dos conceitos envolvidos, ou seja, inserir, alterar e remover. Portanto, ainda faltam os casos de manter título, manter cópia e manter membro.

O próximo passo é descrever cada caso de uso. Veja a Figura 6.3.

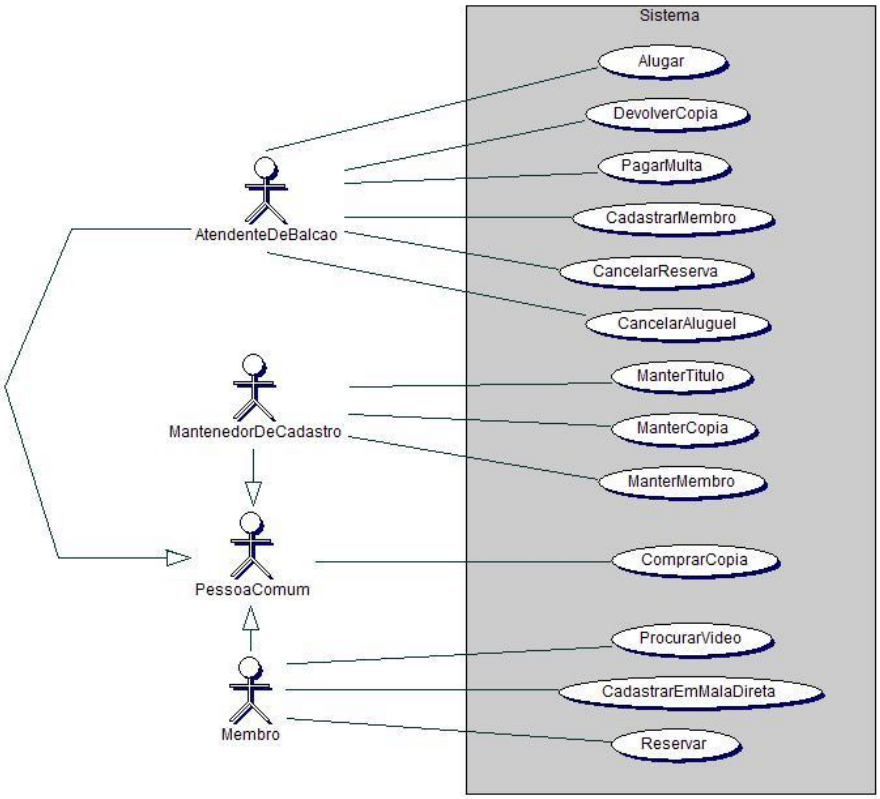


Figura 6.3: Diagrama de casos de uso.

Nome Alugar
Iniciador Atendente de balcão
Objetivo Membro alugar cópia de vídeo na locadora.
Cenário Principal
1 O atendente pede o cartão.
2 O sistema verifica a validade do cartão.
3 O sistema verifica o membro
4 Caso a cópia desejada esteja em espera, vamos para o passo 4a.
Caso a cópia desejada esteja na prateleira, vamos para o passo 4b.
5 O atendente recebe o pagamento.
6 A atendente sacramenta o aluguel.
7 O balanço de caixa do sistema é atualizado.
Cenário Complementar
4.a.1 O atendente identifica as cópias em espera que estão reservadas.
4.a.2 O membro diz qual cópia ele deseja.
4.a.3 O atendente busca a cópia que estava reservada. Continue no passo 5.
4.b.1 O atendente identifica a cópia. Continue no passo 5.

Nome Devolver cópia
Iniciador Atendente de balcão
Objetivo Membro devolver a cópia alugada para a locadora.
Cenário Principal
1 O atendente dá baixa no sistema.
2 O sistema verifica se o título da cópia tem alguma reserva em espera.
3 Caso a resposta seja positiva, continue para o passo 4.
Caso contrário, continue no passo 4b
Cenário Complementar
1.a. Caso a devolução tenha excedido o prazo, uma multa é gerada e o membro é alertado.
3.a.1 A cópia vai para o estado de "em espera".
3.a.2 O atendente entra em contato com o membro.
3.b.1 A cópia vai para o estado de "disponível".

Nome Pagar multa
Iniciador Atendente de balcão
Objetivo O membro pagar a multa relacionada a uma entrega atrasada.
Cenário Principal
1 O atendente identifica o membro.
2 O atendente identifica multas pendentes daquele membro.
3 O atendente recebe o pagamento.
4 A multa vai para o estado de "paga".
5 O balanço do caixa é atualizado.

Nome Cadastrar membro
Iniciador Atendente de balcão
Objetivo Tornar o cliente comum em membro.
Cenário Principal
1 O cliente manifesta o desejo de se tornar membro.
2 O atendente criar um novo registro de membro.
3 O atendente cadastra os dados do novo membro (nome, endereço, telefone e correio eletrônico).
4 O cliente confirma o dados.

Nome Cancelar reserva
Iniciador Membro
Objetivo Cancelar a reserva de um título.
Cenário Principal
1 O membro identifica quais de suas reserva ele deseja editar.
2 O membro cancela a reserva selecionada.

Nome Cancelar aluguel
Iniciador Atendente de balcão
Objetivo Cancelar o aluguel da cópia de um membro.
Cenário Principal
1 O atendente recebe o pedido do cliente de cancelamento do aluguel.
2 O atendente identifica o cliente.
3 O atendente seleciona um aluguel dentro da lista de aluguéis ativos do cliente.
4 O atendente cancela o aluguel.

Nome Manter título
Iniciador Mantenedor de cadastro
Objetivo Manter (incluir, alterar, remover) o cadastro de título da locadora.
Cenário Principal
1 Se o mantenedor deseja incluir, continue no passo 1a.
Se o mantenedor deseja alterar, continue no passo 1b.
Se o mantenedor deseja remover, continue no passo 1c.
Cenário Complementar
1.a.1 O mantenedor cria um novo registro de título.
1.a.2 O mantenedor cadastra os dados do título (código, título, gênero).
1.b.1 O mantenedor seleciona um título dentre os cadastrados.
1.b.2 O mantenedor altera algum dos dados (título, gênero).
1.c.1 O mantenedor seleciona um título dentre os cadastrados.
1.c.2 O mantenedor exclui o registro.

Nome Manter cópia
Iniciador Mantenedor de cadastro
Objetivo Manter (incluir, remover) o cadastro de cópia da locadora.
Cenário Principal
1 Se o mantenedor deseja incluir, continue no passo 1a.
Se o mantenedor deseja remover, continue no passo 1b.
Cenário Complementar
1.a.1 O mantenedor cria um novo registro de cópia.
1.a.2 O mantenedor cadastra os dados da cópia (título relacionado e código da cópia).
1.b.1 O mantenedor seleciona uma cópia dentre as cadastradas.
1.b.2 O mantenedor exclui o registro.

Nome Manter membro
Iniciador Mantenedor de cadastro
Objetivo Manter (incluir, remover) o cadastro de título da locadora.
Cenário Principal
1 Se o mantenedor deseja alterar, continue no passo 1a.
Se o mantenedor deseja remover, continue no passo 1b
Cenário Complementar
1.a.1 O mantenedor seleciona um membro dentre os cadastrados.
1.a.2 O mantenedor altera os dados do membro (nome, endereço, telefone, correio eletrônico).
1.b.1 O mantenedor seleciona um membro dentre os cadastrados.
1.b.2 O mantenedor exclui o registro.

Nome Comprar cópia
Iniciador Pessoa comum
Objetivo Comprar cópia da loja.
Cenário Principal
1 A pessoa comum entrega ao atendente a cópia que deseja comprar.
2 O atendente informa o preço da cópia.
3 O atendente registra o pagamento.
4 O balanço de caixa é atualizado.
5 A cópia é retirada do acervo do sistema.

Nome Procurar vídeo
Iniciador Membro
Objetivo Procurar título.
Cenário Principal
1 O membro entra no sistema.
2 O membro procura o título por nome.

Nome Cadastrar em mala direta
Iniciador Membro
Objetivo O membro se cadastra para ser informado a respeito de novos títulos.
Cenário Principal
1 O membro entra no sistema.
2 O membro pede para receber mala direta.

Nome Reservar
Iniciador Membro
Objetivo O membro deseja reservar um título.
(O membro não achou nenhuma cópia disponível do título que deseja)
Cenário Principal
1 O membro entra no sistema.
2 O membro seleciona o título que deseja.
3 O membro confirma a reserva.

6.3 Especificação

Ao final da modelagem dos requisitos, iniciamos a fase de especificação. Como foi dito anteriormente, esta fase é composta pela identificação de componentes, interação de componentes e especificação de componentes.

6.3.1 Identificação de componentes

Para identificar os componentes, devemos antes identificar quais são as interfaces de negócio e de sistema. As interfaces e seus relacionamentos guiam a identificação dos componentes.

Identificar interfaces de sistema e suas operações

Para cada caso de uso temos uma interface de sistema. No modelo da Figura 6.4, podemos observar os passos para descobrir os métodos de uma interface de sistema. Completamos o cenário da identificação de interfaces de sistema na Figura 6.5:

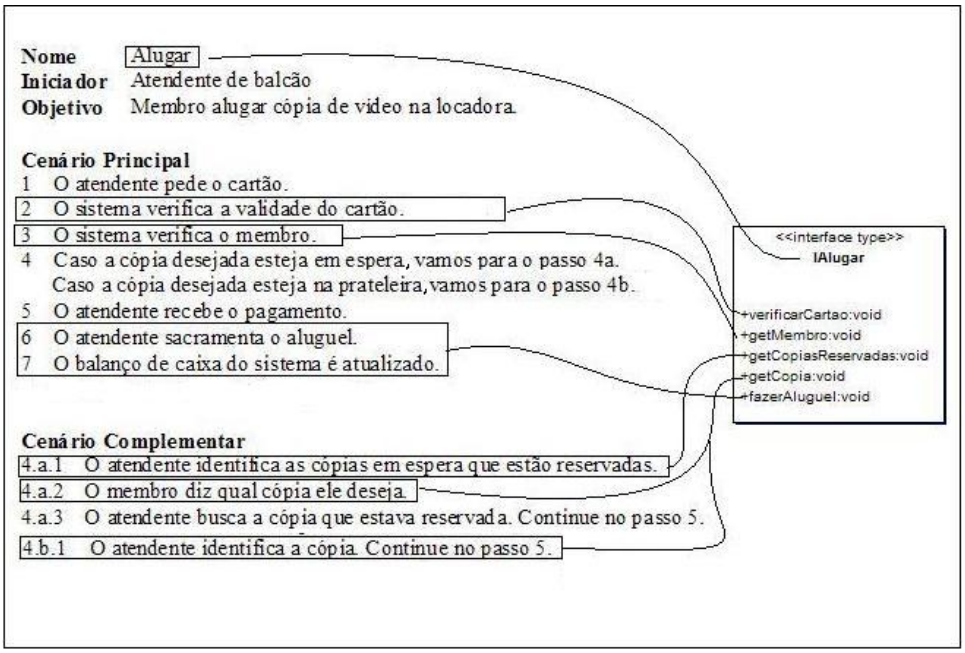


Figura 6.4: Modelagem conceitual.

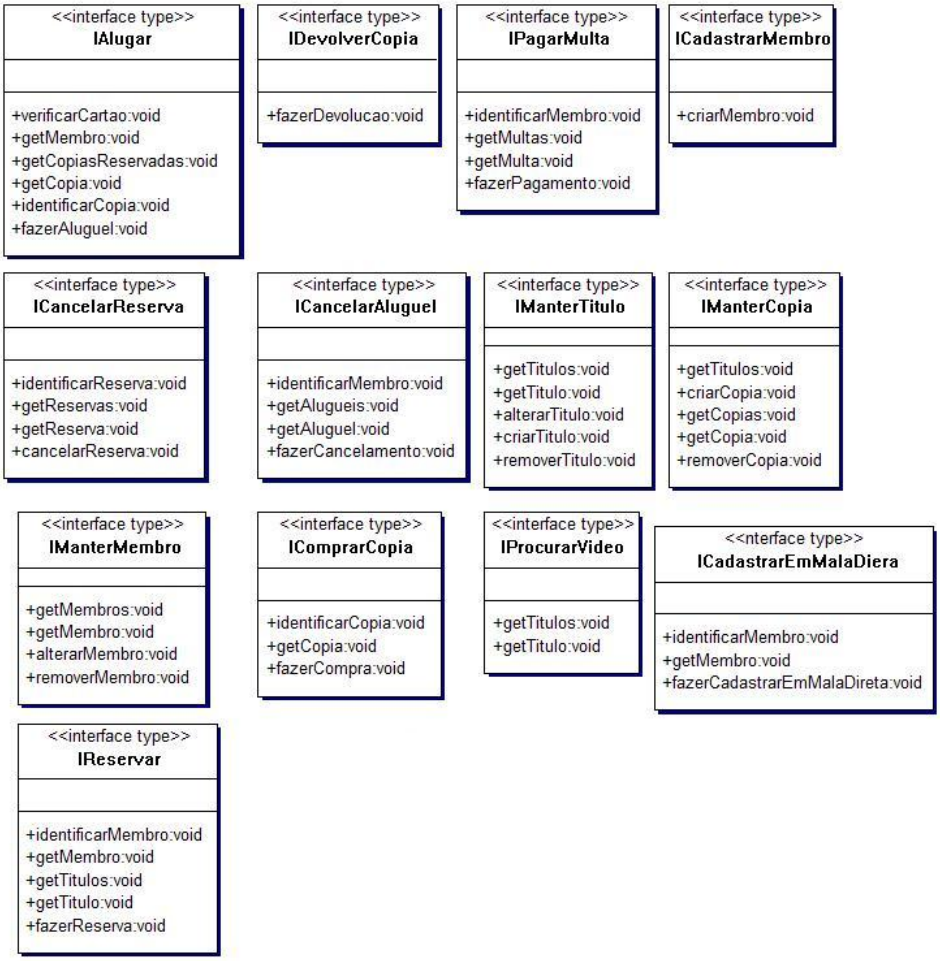


Figura 6.5: Conjunto inicial de interfaces de sistema.

Desenvolver modelo de tipos do negócio

O modelo de tipos de negócio é criado de forma sistemática. Em um primeiro momento, o modelo conceitual que é produzido é idêntico ao modelo conceitual de negócio.

Em seguida, deve-se colocar o modelo dentro do escopo do projeto, como é mostrado na Figura 6.6. Como o sistema diz respeito a apenas uma locadora, não faz sentido o tipo locadora no modelo de tipos.

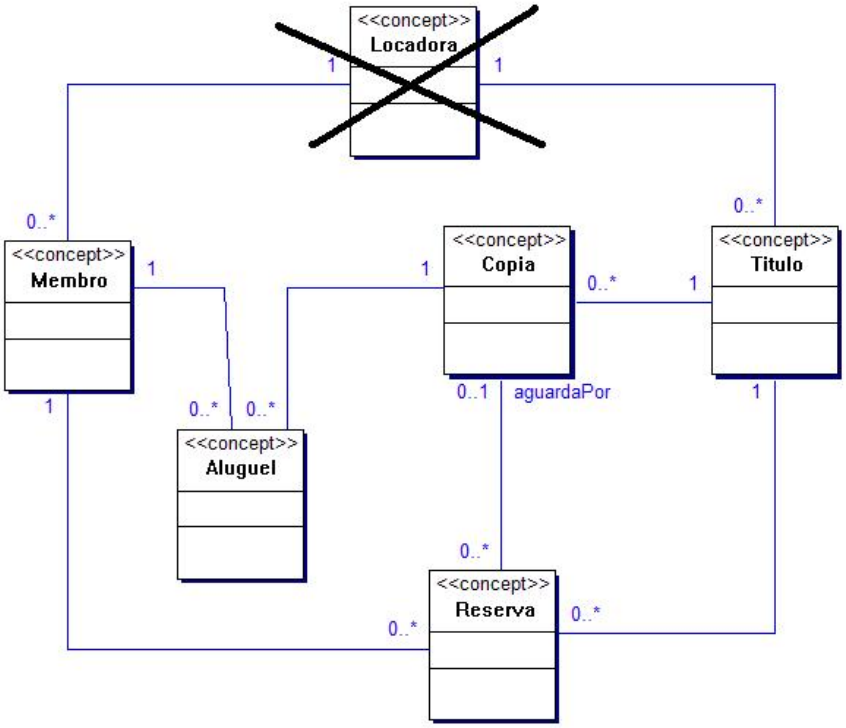


Figura 6.6: Colocando o modelo no escopo correto.

Segundo o livro *UML Components* [9], "o tipo de negócio define dados ou estados que a empresa precisa manter e monitorar, e é conhecido como um conceito de negócio ou processo por usuários e gerentes. É importante reconhecer que os tipos de negócio podem corresponder a aspectos não físicos do negócio".

Adicionando atributos

No passo seguinte, são adicionados as restrições e atributos necessários a cada tipo. Lendo novamente os casos de uso, devem ser buscados os candidatos a atributos nas

descrições que foram escritas.

Identificando tipos núcleo

Como pode ser observado na Figura 6.7, foram identificados três tipos núcleo: Membro, Cópia e Título. Membro não possui associação mandatória com nenhum outro tipo. Logo, na Figura, é identificado como um tipo núcleo. Cópia tem relacionamento mandatório com título, mas como título é um tipo categorizante, ainda considera-se cópia como um tipo de existência independente. Segundo *UML Components*, "O tipo categorizante é aquele que suas instâncias categorizam ou classificam as instâncias de outro tipo". Então, cópia é um tipo núcleo. Título não possui associação mandatória com nenhum outro tipo, tornando-se também um tipo-núcleo.

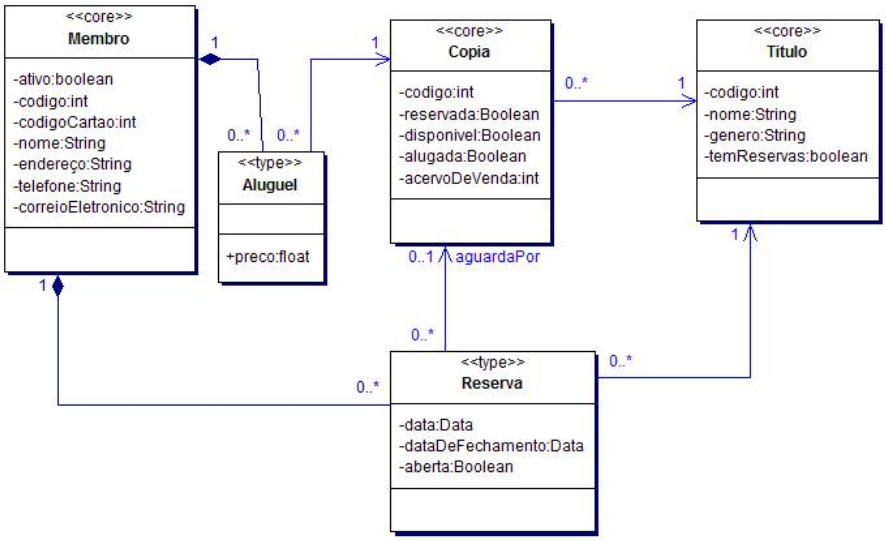


Figura 6.7: Modelo de tipos do negócio.

Identificar interfaces de negócio

Criando interfaces de negócio e definindo responsabilidades "A regra geral é que criamos uma interface de negócio para cada tipo núcleo no modelo de tipos de negócio. Cada interface de negócio gerencia a informação representada por cada tipo núcleo e seus tipos detalhantes". Sem demora, foram criadas as interfaces IMembroMgt, ICopiaMgt e ITituloMgt correspondentes aos seus respectivos tipos

núcleo.

Quanto à definição de responsabilidade, basicamente, aloca-se os tipos detalhantes, ou seja, os não tipos núcleo, a cada tipo núcleo que possui associação mandatória. Esta alocação é representada pela associação de composição (símbolo do diamante sólido). O problema ganha complexidade quando existe um tipo que detalha mais de um tipo núcleo diferente. Em que interface este tipo será alocado?

Este problema é encontrado nos tipos Reserva e Aluguel. Reserva detalha Membro e Cópia, e Reserva detalha Membro, Cópia e Título. *UML Components* apenas orienta o desenvolvedor dizendo para alocar o tipo à interface em que suas informações estão mais fortemente acopladas. O conceito de "fortemente acoplado" é vago, deixando a cargo do analista que está trabalhando no modelo fazer esta escolha, baseando-se na sua interpretação pessoal. Sendo assim, como um cliente se torna membro com a exclusiva intenção de fazer aluguel, enquanto a cópia pode estar envolvida com vendas, foi decidido que aluguel estaria alocado a interface IMembroMgt. Da mesma forma, reserva estaria mais acoplado a membro do que cópia ou título.

Associações inter-interface

São assim chamadas as associações existentes entre tipos de diferentes interfaces. O objetivo, neste momento, é fazer tais associações, reduzindo as dependências entre os componentes. Logo, estas associações serão feitas de forma indireta e em apenas uma direção. Isto significa que se cópia é independente de aluguel, aluguel terá um atributo que será o identificador que referencia cópia. A interface IMembroMgt será responsável por gerenciar a associação entre IMembroMgt e ICópiaMgt. Na Figura 6.8, são definidas as outras associações inter-interface.

Criar especificação inicial de componentes e arquitetura

Para cada interface de negócio, é criada uma especificação de componente, como mostra a Figura 6.9. Faz sentido criar uma especificação de componente por interface, porque cada interface gerencia informações diferentes e independentes.

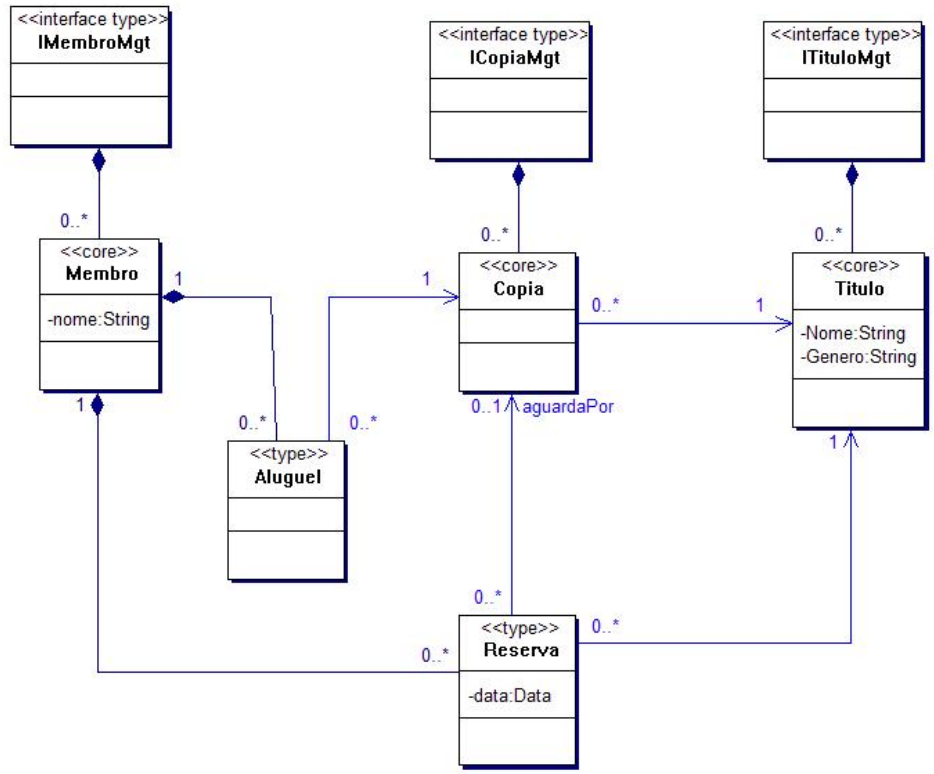


Figura 6.8: Diagrama de responsabilidade de interface.

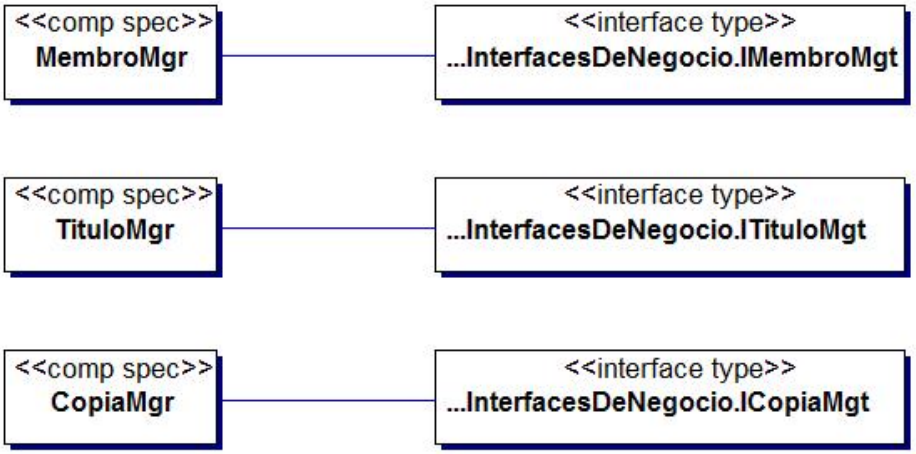


Figura 6.9: Especificação inicial de componentes de negócio.

Para simplificar o estudo de caso, a quantidade de interfaces de sistemas será reduzida nas análises posteriores. Apenas será considerada a interface IAlugar. Muitas das interfaces de sistema possuem métodos que se assemelham em seus objetivos. Por este motivo, não existe razão para criar uma especificação de componente por interface. Seguindo pela mesma linha de raciocínio, pode-se dizer que cada especificação de componentes de sistema agrupa interfaces que possuem alguma similaridade na sua função e nas informações que tratam. Desta forma, foi criado apenas um componente de sistema que reúne apenas a interface IAlugar, como pode ser observado na Figura 6.10.

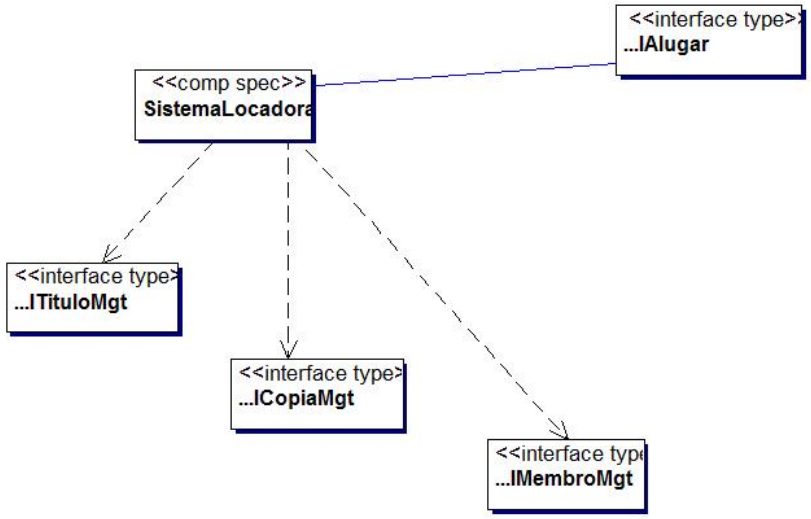


Figura 6.10: Especificação inicial de componentes de sistema.

Verificando os dois últimos modelos apresentados, facilmente é criado o diagrama inicial de arquitetura de componentes. Veja a Figura 6.11.

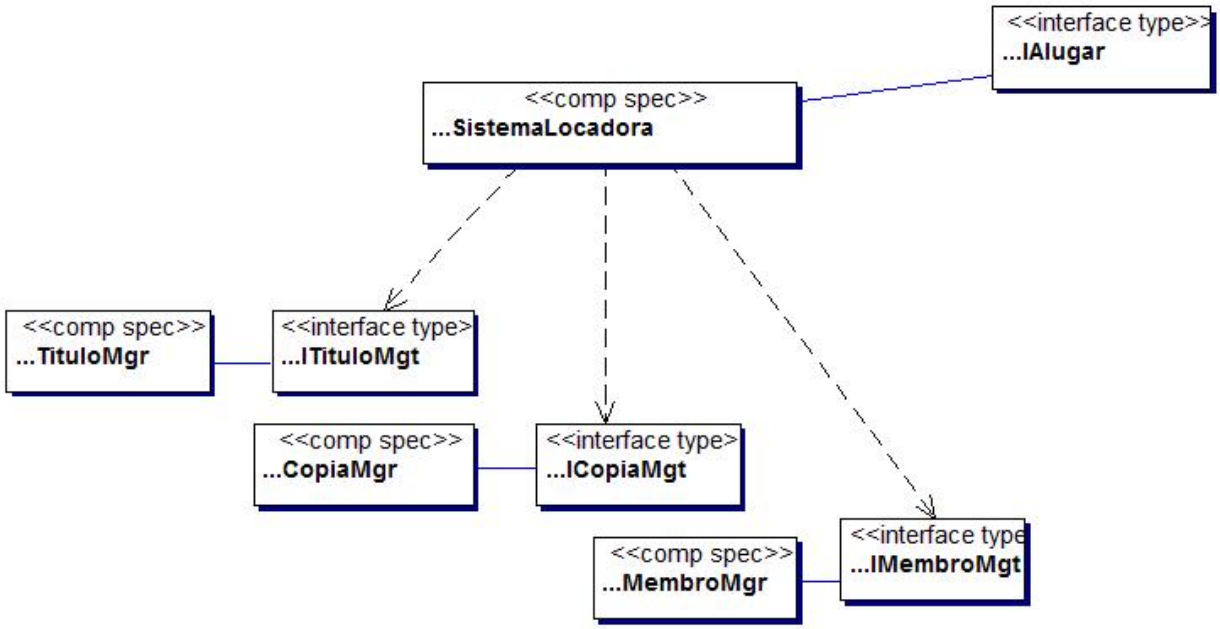


Figura 6.11: Arquitetura inicial.

6.3.2 Interação de componentes

Descobrindo operações de negócio

Neste momento do processo, inicia-se a busca pelas assinaturas dos métodos de interfaces de sistema e as operações das interfaces de negócio. Segundo *UML Components*, seria suficiente apenas conhecer as interfaces de sistema e de negócio para fazer a descoberta dos métodos. Porém, sem o auxílio dos casos de uso para recuperar qual é o objetivo daquele método, a tarefa fica sem informações suficiente.

Os próximos modelos mostrarão para a interface de sistema da Figura 6.12 abaixo, quais são as interações necessárias para a conclusão das operações, utilizando as interfaces de negócio ICopiarMgt, IMembroMgt e ITituloMgt.

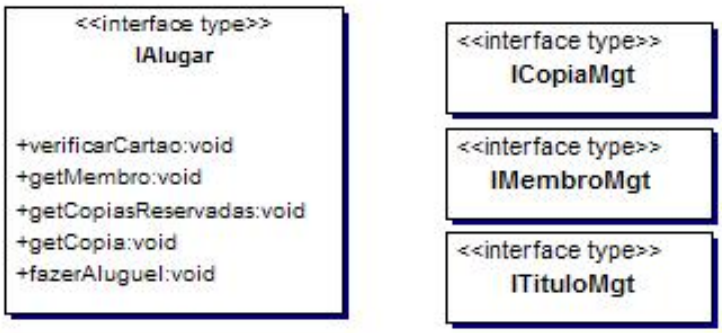


Figura 6.12: A interface de sistema IAlugar, e as interfaces de negócio ICópiaMgt, IMembroMgt e ITituloMgt.

verificarCartao

O método recebe o número identificador do cartão e retorna o número identificador membro.

getMembro

O método recebe o identificador membro e retorna os seus detalhes. Um tipo de dado estruturado foi criado para guardar as informações que serão retornadas.

getCopia

O método deve receber o identificador da cópia e retornar os seus detalhes.

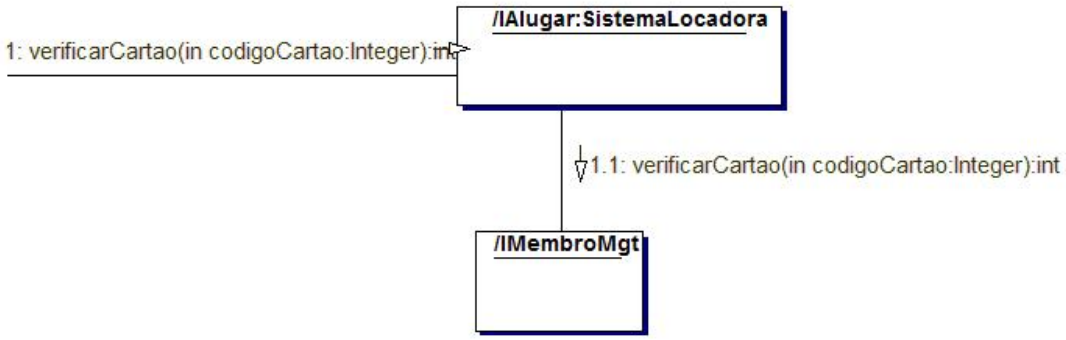


Figura 6.13: Interação verificarCartao()

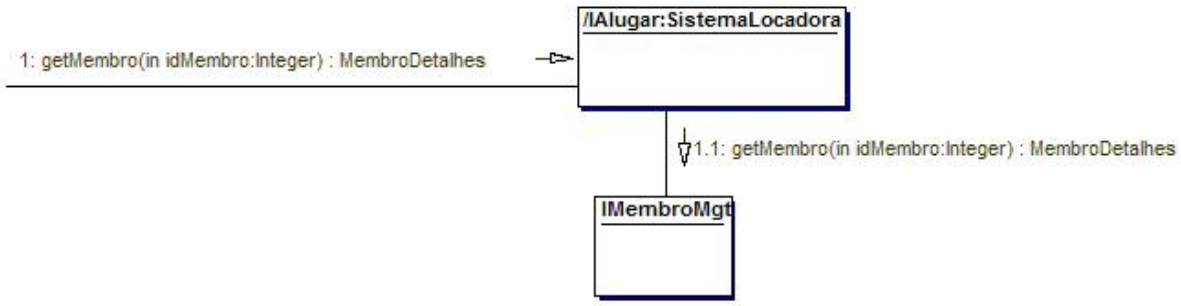


Figura 6.14: Interação getMembro()

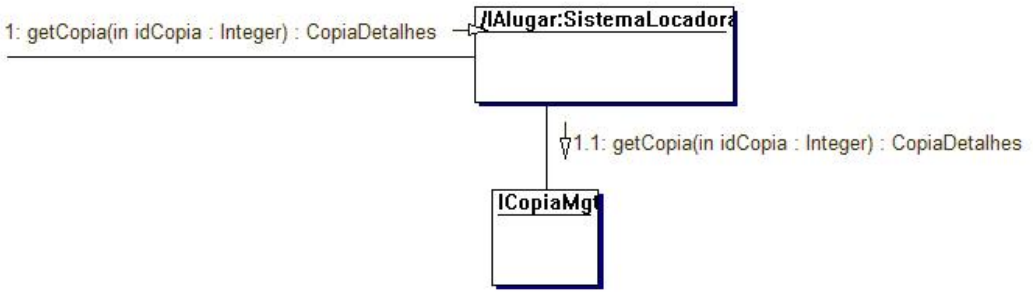


Figura 6.15: Interação getCopia()

getCopiasReservadas

O método deve receber o identificador membro e retornar os detalhes das reservadas das cópias para aquele membro. Para exibir o detalhe daquelas cópias, foi criado outro tipo de dado estruturado chamado ReservaDetalhes.

Figura 6.16: Tipo de dado estruturado ReservaDetalhes()

Para preencher as informações de identificador da reserva, identificador da cópia, identificador do título, foi chamado um método na interface IMembroMgt, que é responsável pelo tipo Reserva. A única informação faltante é o nome do título da cópia reservada. Logo, foi criado um método na interface ITítuloMgt para responder o nome do título mediante o fornecimento do seu identificador.

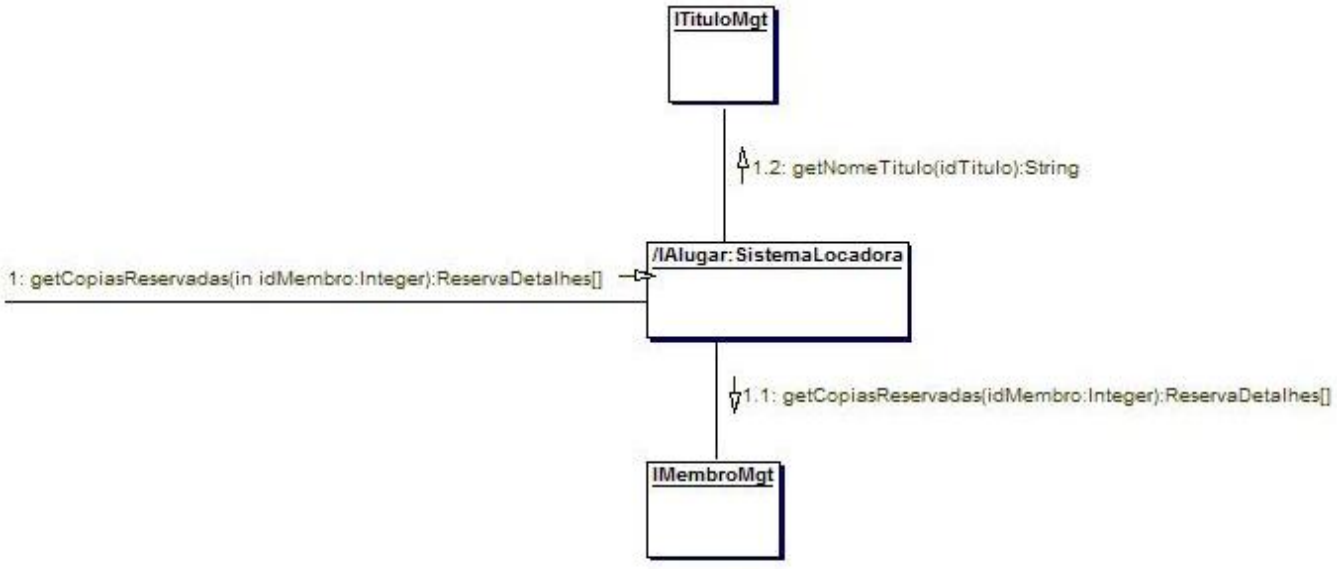


Figura 6.17: Interação getCopiasReservadas()

fazerAluguel

O método fazerAluguel é o que deveria receber mais atenção durante a análise da interface IAlugar. Mas como a interface IMembroMgt gerencia os tipos Membro, Reserva e Aluguel, a maior parte da interações ocorreram dentro do componente Membro.

Neste método deve ser sacramentado o aluguel da cópia, ou seja, a criação de um tipo Aluguel que relaciona Membro e Cópia. Caso exista alguma reserva deste membro para esta cópia, tal reserva deve ser considerada como concluída.

Como pode ser observado na Figura 6.18, este método gerou uma simples interação.

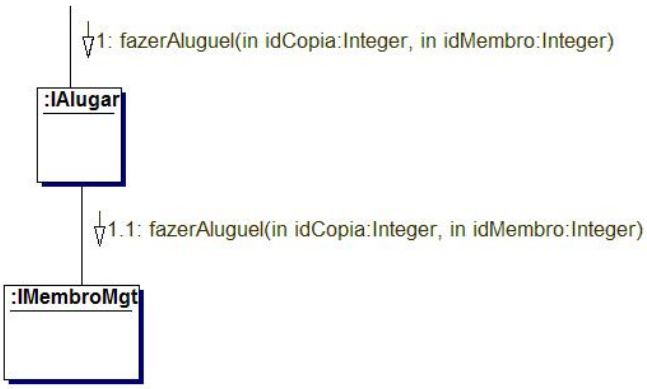


Figura 6.18: Interação fazerAluguel()

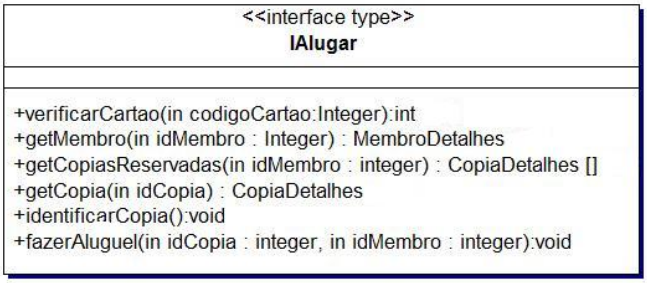


Figura 6.19: Interface de sistema IAlugar com suas assinaturas.

6.3.3 Especificação de componentes

Cada especificação de interface terá seu modelo de informação de interface. São criados a partir do modelo de responsabilidade de interface. Basta criar uma cópia deste diagrama e remover as informações não relevantes. Vale a pena ressaltar que o modelo de informação não tem relação direta com os dados que serão persistentes ou até mesmo com a implementação dos métodos. O modelo de informação de interface deve ser apenas uma abstração que apoia a escrita dos contratos. O procedimento para as interfaces de negócio é semelhante ao utilizado nas interfaces de sistema, já que estas últimas, basicamente, apenas repassam as chamadas para outras interfaces. Observe as Figuras 6.20 e 6.21.

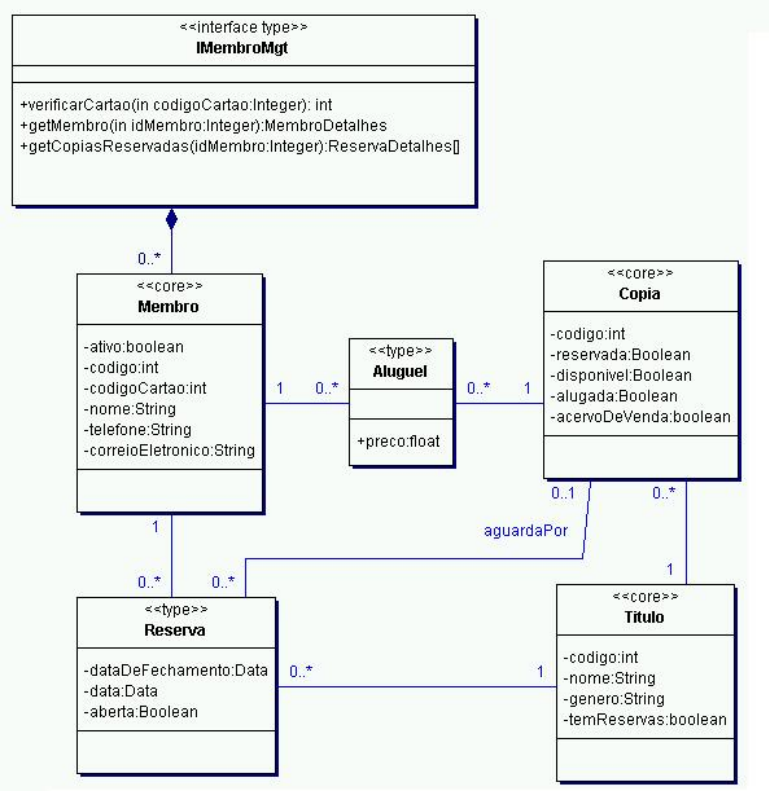


Figura 6.20: Modelo de informação de interface do IMembroMgt.

O contrato de cada método deve ser escrito em linguagem natural ou *OCL*. Porém, a complexidade do *OCL* é um limitador para a sua utilização em projetos. Não é um trabalho simples para o desenvolvedor que não conhece a linguagem compreender o que significam os contratos escritos em *OCL*. Devido a esta complexidade,

decidimos por não utilizar *OCL* neste trabalho.

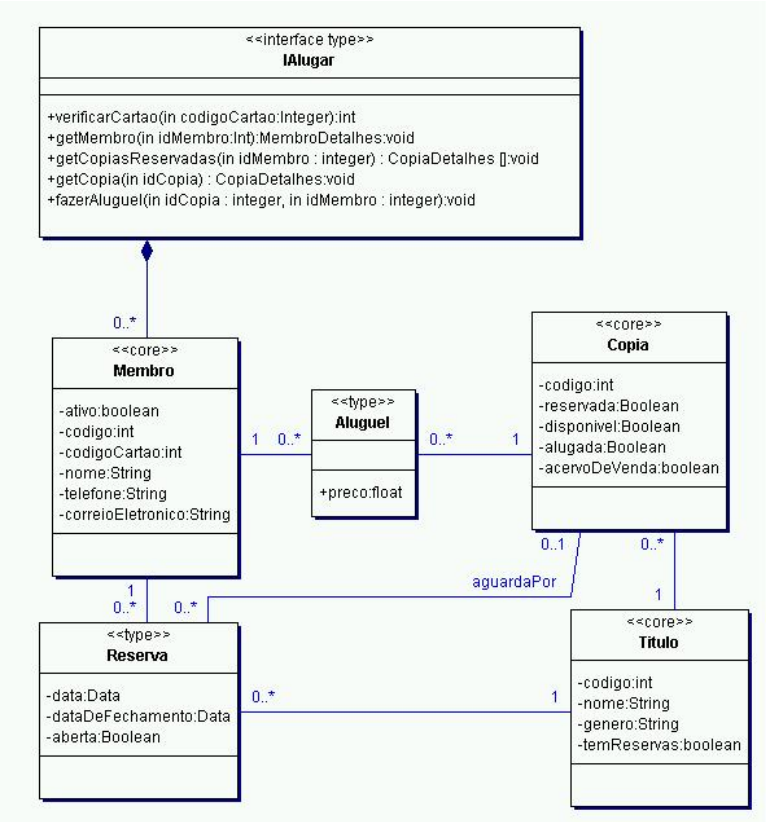


Figura 6.21: Modelo de informação de interface do IAlugar.

Capítulo 7

Comparação

O principal objetivo deste trabalho é apresentar a comparação entre o *Catalysis* e o *UML Components*. A comparação apresentada neste capítulo foi baseada na modelagem dos estudos de caso descritos nos dois últimos capítulos. Durante o trabalho de modelagem, obtivemos várias observações e conclusões que permitiram fazer a comparação destes dois métodos.

Iniciamos o capítulo com comentários gerais sobre a modelagem dos estudos de caso. Depois, apresentamos os itens de comparação, e logo após colocamos mais alguns pontos comparando os dois métodos. Finalmente, encerramos o capítulo com uma tabela de comparação que resume os principais pontos de comparação dos métodos estudados.

7.1 Comentários Gerais

O comentário principal é com relação aos exemplos dados. Os exemplos mostraram ser muito simples, e poderiam ser muito mais complexos, levando-se em conta que são a única fonte de referência dos leitores dos livros em questão.

A notação de casos de uso e ações no *Catalysis*, apesar de não utilizada em nosso estudo de caso devido a simplicidade do exemplo, parece oferecer muito mais expressão do que a notação utilizada pelo *UML Components*, que é a mesma espe-

cificada pela *UML*. Os casos de uso abstratos, tanto citados no *Catalysis*, permitem uma boa abstração entre as camadas de refinamento, podendo ajudar a eliminar inconsistências e dúvidas, assim como citado em [20].

O *UML Components* parece supor que a análise do problema está completa e perfeita, fato que na maioria dos casos é quase impossível. Através da ordem utilizada no *Catalysis*, foi possível descobrir os tipos, seus atributos e operações incrementalmente, principalmente através do uso de cenários e *UI Sketches*. No *UML Components*, a definição dos tipos parece empírica e totalmente retirada da análise, o que a primeira vista pode permitir a inserção de inconsistências.

Uma dessas inconsistências foi percebida através da modelagem em *Catalysis*. Na modelagem dada, não é claro quantos hotéis o sistema gerencia e também não é claro se os clientes são compartilhados entre os hotéis ou não.

No entendimento inicial, através da modelagem dada, pode-se concluir que o sistema gerencia somente um hotel. Mas olhando, cuidadosamente, a especificação do caso de uso "Identificar Reserva", Chessman diz que uma *tag* de reserva pode ser referente a outro hotel, indicando que o sistema deve gerenciar mais de um hotel.

Através da modelagem em *Catalysis*, na elaboração do glossário, foi possível perceber essa dúvida e, portanto, tomar uma atitude quanto a isso. Neste trabalho, considerou-se, para não ficar inconsistente com o caso de uso "Identificar Reserva", que o sistema gerencia vários hotéis e que os clientes não são compartilhados entre os hotéis.

Outra inconsistência descoberta ao longo do processo de modelagem usando *Catalysis* foi a cardinalidade da relação Cliente-Reserva, no modelo inicial de tipos. A inconsistência com relação a essa cardinalidade só foi descoberta através do *UI Sketch*. O método empírico do *UML Components* de descoberta de tipos poderia dar margem a essa inconsistência.

O *Catalysis* não propõe uma forma clara para a descoberta dos componentes. Portanto, foi utilizada a forma do exemplo da locadora de vídeo em *Catalysis*, onde ele define um *façade* [11] para todo o sistema.

7.2 Itens de comparação

Selecionamos quatro itens para a comparação dos métodos. Estes itens foram baseados nos itens comumente sugeridos pela literatura [12], levando em conta o escopo da comparação deste trabalho. O itens são:

- especificação de requisitos;
- identificação de componentes;
- identificação de serviços e restrições de cada componente;
- representação de componentes.

7.2.1 Especificação de requisitos

Neste item comparamos como cada método aborda a especificação de requisitos. Nos estudos de caso, não fizemos a especificação dos requisitos, visto que os requisitos foram retirados da bibliografia. Entretanto, fizemos uma comparação da especificação com base no que se encontra na literatura de referência sobre os métodos.

7.2.2 Identificação de componentes

Comparamos neste item como cada método identifica seus componentes. Em função do tamanho reduzido dos estudos de caso, não foi possível usar os métodos para a identificação de uma grande quantidade de componentes.

7.2.3 Identificação de serviços e restrições de cada componente

A partir da identificação dos componentes, este item compara como cada método trata da identificação dos serviços e restrições de cada componente.

7.2.4 Representação de componentes

Após os passos anteriores, é comparado como cada método trata da representação dos componentes. Como ambos os métodos ampliam a *UML*, a comparação foi feita levando mais em consideração que tipo de informação e qual a relevância da informação que o método trata, ao invés de fazer uma comparação da representação gráfica de cada método.

7.3 Comparação dos métodos

Com relação aos quatro itens selecionados (especificação de requisitos, identificação de componentes, serviços e restrições de cada componente e representação de componentes), os dois métodos apresentam diferenças significativas somente na identificação de componentes. A identificação de componentes do *UML Components* é sistemática, enquanto o *Catalysis* não apresenta tal forma. Nos outros três itens, as diferenças são, em geral, relativas a representação gráfica dos diagramas e também eventuais extensões do padrão *UML*.

Um comentário pertinente, mas não abordado pelos itens identificados anteriormente, é com relação a ordem de produção dos artefatos. O *UML Components* contém uma ordem de produção de artefatos, enquanto o *Catalysis* faz uma sugestão de ordem, e que neste trabalho mostrou-se satisfatória.

7.3.1 Especificação de requisitos

Os métodos acreditam que a forma tradicional de análise por meio de casos de uso e modelo conceitual é um bom início para especificação de requisitos. Na verdade, são diagramas que podem ser utilizados em qualquer projeto, independentemente do método. Permitem criar, de forma simples, um consenso entre o cliente e a equipe desenvolvedora.

Catalysis

O *Catalysis* também usa o modelo de casos de uso. Entretanto, para chegar ao modelo conceitual, é necessário alguns passos anteriores. Os passos sugeridos são a modelagem de estados e também de ações (algumas construídas a partir dos cenários ou *Snapshots*) em um nível mais abstrato. A partir destes modelos, é criado então um modelo de tipos inicial para o sistema, já com algumas invariantes, que corresponde a um modelo conceitual.

Na medida que a modelagem vai avançando, os modelos anteriores, incluindo o modelo de tipos, vão sendo refinados para acomodar mudanças e novas restrições não percebidas anteriormente.

UML Components

O *UML Components* trata com simplicidade o modelo conceitual e os casos de uso. Todo o desenvolvimento vai se apoiar nestes dois modelos. Os autores deste método proporcionaram uma forma sistemática para ir adiante na especificação. Não existe o conceito de ações e não é pensado quais objetos estariam envolvidos na realização dos casos de uso.

7.3.2 Identificação de componentes

Como foi dito anteriormente, os métodos de DBC acreditam que os componentes são formados por meio do agrupamento dos elementos do diagrama de tipos do negócio. Desta forma, deve ser oferecida uma regra de agrupamento dos tipos. É isto, basicamente, que vai determinar a identificação dos componentes. Em um trabalho recente [18], podemos encontrar algumas destas regras.

Catalysis

O *Catalysis* não tem uma forma sistemática para a identificação de componentes. Entretanto, é sugerido que se use as ações e as colaborações para identificar as ações (ou métodos) dos componentes. Isso é uma grande diferença com relação ao *UML Components*, já que este apresenta um método sistemático.

UML Components

Este método expõe de forma bastante simples e didática a identificação sistemática de componentes. Analisando as associações do diagrama de tipos, deve ser identificado qual "informação" independe das outras. Isto identificará o tipo-núcleo. Todos os outros tipos deverão estar agrupados junto a algum tipo núcleo. Esta escolha, muita vezes, fica a cargo da interpretação pessoal do analista. Sendo assim, não existe garantia de que dois analistas que utilizam *UML Components* em seu projeto encontrarão os mesmos componentes de negócio do sistema.

UML Components também faz a distinção entre estes componentes de negócio e o que chama de componentes de sistema. Estes nada mais são do que um mapeamento da realização dos casos de uso identificados sobre os componentes de negócio.

7.3.3 Identificação de serviços e restrições de cada componente

Curiosamente, a identificação dos métodos dos componentes não tomou tanta atenção quanto a identificação dos componentes. De certa forma, todas as decisões arquiteturais já foram tomadas na fase anterior. A identificação dos serviços devem apenas seguir as políticas definidas anteriormente.

Os dois métodos pregam que os métodos identificados em cada componente devem ter contratos associados. Este contratos podem ser escritos em linguagem natural, mas preferencialmente em *OCL*.

Catalysis

A identificação de serviços não é sistemática, entretanto, é sugerido que os serviços (ou ações, como chamado no *Catalysis*) sejam identificados a partir das ações e colaborações modeladas. Utilizando-se o método anterior, consegue-se de imediato as restrições desses serviços e do componente em geral, já que todas as ações e as colaborações tem suas restrições modeladas utilizando linguagem textual, num nível mais abstrato, e a linguagem *OCL*, num nível menos abstrato.

UML Components

Cada especificação de componente é composta por um conjunto de interfaces. Na verdade, no *UML Components*, procuramos pelos métodos das interfaces.

A identificação dos serviços ocorre de forma massante. Cada caso de uso é representado por uma interface de sistema. Cada interface de sistema tem a seu dispor algumas interfaces de negócio para realizar seus métodos. Esta descoberta é extremamente cansativa porque, para cada método de uma interface de sistema, é criado um diagrama de colaboração. As restrições dos componentes também são obtidas a partir das restrições dos tipos do componente.

7.3.4 Representação de componentes

As notações dos diagramas do *UML* são utilizadas para representação dos modelos em cada um dos métodos. Mesmo utilizando esta linguagem padrão, tivemos dificuldades para criar os modelos de acordo com a representação proposta. Isto ocorre porque cada método procurou estender a linguagem da sua própria maneira para atender às suas necessidades. As ferramentas *CASE* [1] disponíveis não acomodaram muitas das extensões propostas. Mostramos aqui tanto a diferença de

[1]Para a realização da modelagem dos estudos de caso, utilizamos ferramentas conhecidas como *computer aided software engineering (CASE)*. Inicialmente testamos as seguintes ferramentas: *Dia* [1], *Rational Rose* [4] e *Together Control Center* [6]. A ferramenta utilizada foi a *Together Control Center v6.0*, pois foi a que mais ofereceu liberdade para construção de diagramas.

notação entre os métodos quanto a diferença entre os métodos e as utilizadas nas ferramentas.

Catalysis

O Catalysis utilizou notações próprias que dificilmente podem ser acomodadas em ferramentas *CASE*. Desta forma, tivemos que fazer certas adaptações sobre o modelo pretendido pelo *Catalysis*. Podemos ilustrar este problema por meio de alguns modelos. A modelagem de um componente deveria ser como mostrado na 7.1. Nesta figura, ilustra-se o componente, com suas classes internas, suas ações e a especificação dessas ações.

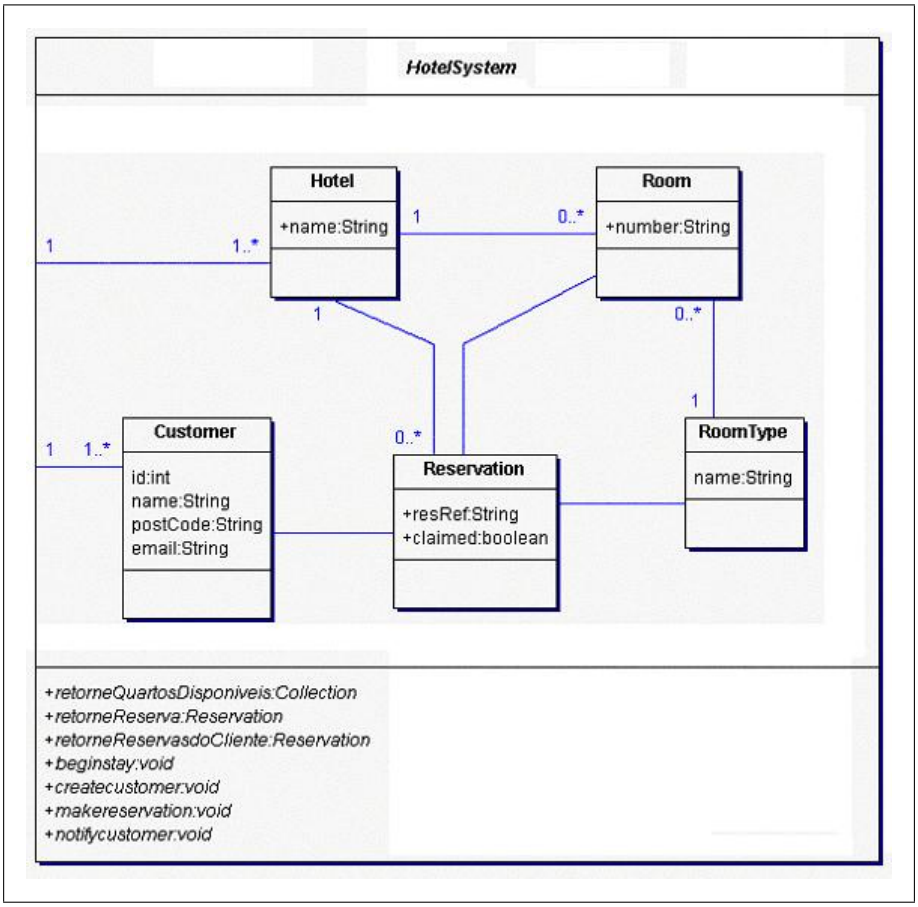


Figura 7.1: Notação ideal para representação de componentes para o *Catalysis*.

Mas, por causa da limitação da ferramenta, usamos a notação da 7.2, que contém a interface do componente em cinza, e suas especificações detalhadas textualmente. Da mesma forma, a modelagem das ações teve que ser adaptada, como observamos

nas figuras 7.3 e 7.4.

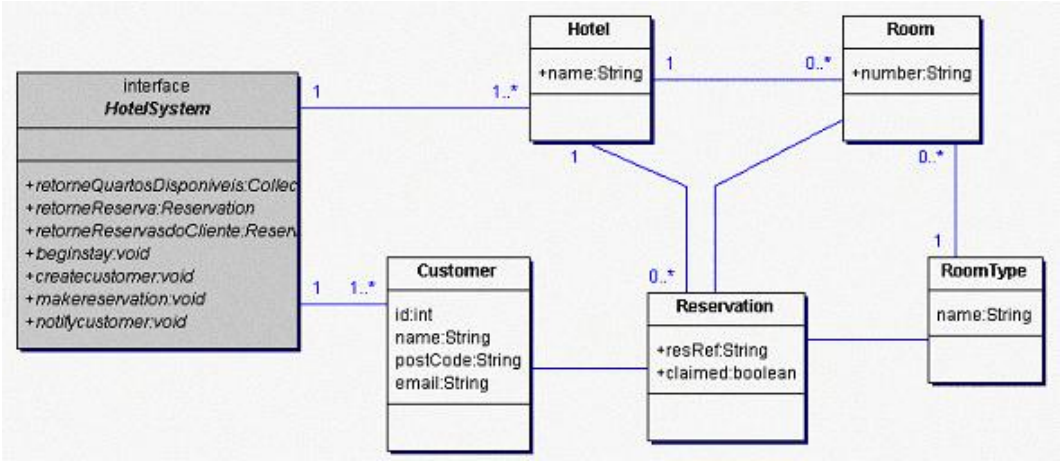


Figura 7.2: Notação possível para representação de componentes para o *Catalysis*.

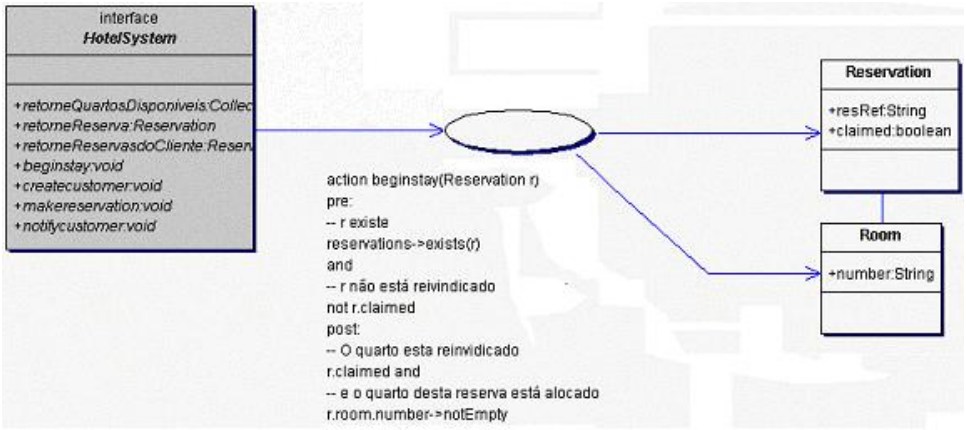


Figura 7.3: Notação para representação de ações no *Catalysis*.

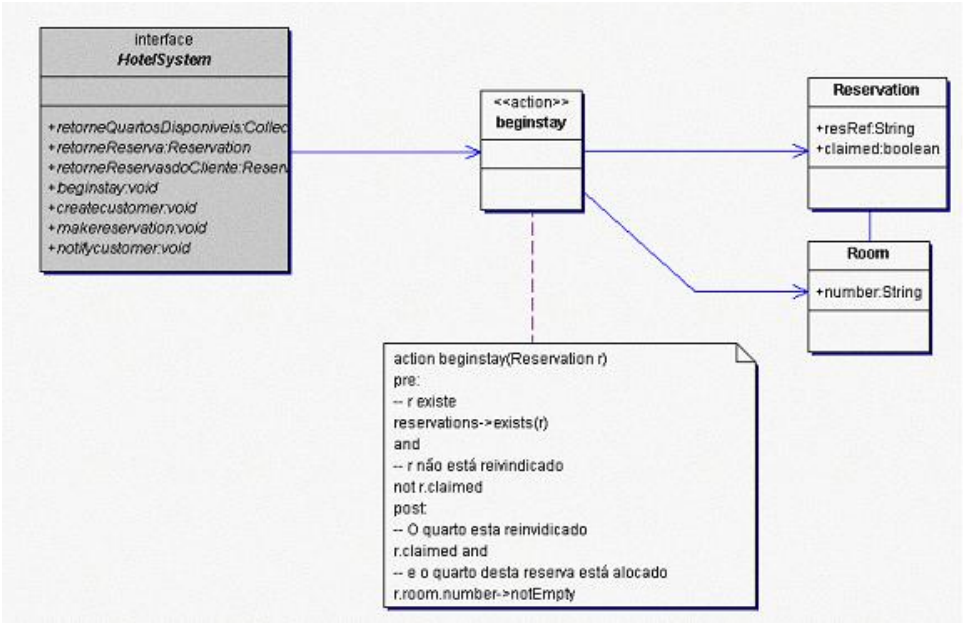


Figura 7.4: Notação utilizada para representação de ações no *Catalysis*.

UML Components

Para os autores do *UML Componentes*, o interesse na utilização da *UML* para modelagem de componentes está em capturar sua característica e comportamento visível exteriormente. Apesar da *UML* ter o propósito inicial de modelar sistemas orientados a objeto, esta linguagem provê alguns mecanismos de extensão, e o mecanismo utilizado foi o de **estereótipos**. Praticamente qualquer elemento da *UML* pode receber um estereótipo. Desta forma, os autores utilizaram intensivamente o diagrama de classes da própria *UML* e utilizaram estereótipos para adicionar a semântica necessária aos modelos. Observe o resumo dos estereótipos utilizados na tabela 7.1.

Conceito	Construtor *UML*	Estereótipo
Especificação de componente	Class	≪comp spec≫
Tipo de interface	Type (Class ≪type≫)	≪interface type≫
Espec. de componente oferece tipo de interface	Dependency	≪offers≫
Conceito de negócio	Class	≪concept≫ (opcional)
Tipo de negócio	Type (Class ≪type≫)	≪types≫
Tipo de dado estruturado	Type (Class ≪type≫)	≪datatype≫
Tipo de informação de interface	Type (Class ≪type≫)	≪info type≫ (freqüentemente omitido)

Tabela 7.1: Resumo do que é utilizado de esteriótipos em *UML Components*

A tabela 7.2 a seguir resume os principais pontos identificados na comparação dos dois métodos estudados neste trabalho.

Itens	*Catalysis*	*UML Components*
Especificação de requisitos	Não trata especificamente da especificação de requisitos. Sugere o uso de casos de uso, rascunhos de interface e glossário.	Não descreve em detalhes como deve ser feito, mas é sugerido o uso do modelo conceitual de negócios e diagramas de casos de uso.
Identificação de componentes	Não existe um método sistemático, entretanto faz algumas sugestões, como a utilização das colaborações.	Contém um método sistemático para a identificação de componentes.
Identificação de serviços	Usa modelos de colaboração e também as descrições em *OCL* das invariantes dos tipos e das ações.	Por meio do exame de cada interface do sistema, usando a arquitetura do componente.
Representação de componentes	Os componentes são representados por sua interface e pelos tipos de suporte dessa interface. É representado também pelo modelo de tipos e pela especificação de cada ação desta interface, seja através de *OCL* ou através de um diagrama de colaboração. Diagramas extras, como o de estados e de atividades podem ser usados como suporte.	Tímido nas sugestões de extensão da *UML*. Usa bastante esteriótipos para dar semântica aos elementos dos seus modelos.

Tabela 7.2: Resumo em tabela da comparação dos métodos

Capítulo 8

Conclusão

O objetivo principal deste trabalho foi a comparação na prática de dois dos mais conhecidos métodos para o desenvolvimento baseado em componentes. Infelizmente, na literatura, há poucas referências para trabalhos semelhantes com o enfoque mais prático e de aplicação.

O que a literatura descreve como métodos funcionais e práticos nem sempre se mostrou verdade [9][20]. Ao mesmo tempo, algumas das técnicas propostas pelos métodos realmente provaram ser eficientes para os exemplos propostos.

Embora a notação do *Catalysis* não ter sido totalmente explorada neste trabalho, mostrou ser muito mais completa do que a do *UML Components*; o que permite uma visão mais abrangente de um sistema baseado em componente. Entretanto, o *UML Components* mostrou-se muito mais didático e simples de ser aplicado, em função da sua maior simplicidade.

Em função desta simplicidade, o *UML Components* não trata muito bem de assuntos que deveriam ter um enfoque maior, como a identificação de tipos e a não orientação com relação a implementação. O enfoque maior do *UML Components* é na diferença entre a especificação de componentes e a interface.

O *Catalysis* enfatiza mais a própria especificação do componente em si. Mesmo assim, o *Catalysis* mostrou-se eficiente na ajuda da construção dessa interface,

técnica esta que poderia inclusive ajudar o desenvolvimento de sistemas sem o enfoque de componentes.

8.1 Contribuições

Este trabalho teve as seguintes contribuições principais:

1. Comparar dois métodos de desenvolvimento baseados em componentes, com o objetivo de identificar pontos fortes e fracos de cada método. Esta comparação possivelmente pode ser útil, se usada como referência para escolha de um método baseado em componentes.

2. Verificar a aplicabilidade de cada método. Os métodos apresentam várias técnicas e um dos objetivos deste trabalho foi justamente utilizar estas técnicas para verificar a sua aplicabilidade.

Podemos ainda citar como contribuição complementar a identificação da inadequação das atuais ferramentas. Foram testadas várias ferramentas com o objetivo de realizar a modelagem, dentre elas o Dia [1], o Rational Rose [4] e o Together Control Center [6] . Entretanto, todas elas não se mostraram eficientes no sentido de facilitar a criação dos diagramas, principalmente do *Catalysis*, que usa diagramas e elementos fora do padrão UML.

Uma contribuição final é a apresentação da modelagem dos estudos de casos que pode servir de referência para outros trabalhos utilizando *Catalysis* e *UML Components*. Existem poucos estudos de casos disponíveis sobre estes métodos e nosso trabalho pode servir como mais uma fonte de referência.

8.2 Limitações

Ao longo do trabalho foi possível identificar várias limitações. Estas limitações podem servir de base para trabalhos futuros. Dentre elas, estão:

Implementação

Uma grande limitação do trabalho foi a falta da efetiva implementação dos estudos de caso propostos. Através da efetiva criação de código executável, provavelmente seria possível determinar outros pontos positivos e negativos dos métodos, além de permitir fazer mais sugestões em relação a outras etapas do processo de desenvolvimento, como a etapa de projeto.

Exemplos

O nosso estudo de caso focou-se em dois exemplos descritos na literatura de referência, mas cada um usando o estudo de caso do outro. Os resultados mostrados aqui levam em consideração somente um estudo de caso para cada método. O ideal seria que fosse possível ter mais estudos de caso e fazer esta comparação a partir desta experiência.

Simplicidade do Estudo de Caso

Os estudos de caso propostos são muito simples, e não representariam um sistema na vida real. O ideal seria utilizar os métodos em sistemas reais e com regras de negócio complexas e da vida real.

Falta de mudança de requisitos

Um problema comum no desenvolvimento de qualquer tipo de software é a mudança de requisitos durante o desenvolvimento do projeto. Neste trabalho, o escopo estava fixo e definido. Seria muito interessante testar a aplicação dos métodos em um ambiente em que há mudanças de requisitos, assim como na vida real.

Técnicas dos Métodos

Em função da simplicidade dos estudos de caso, não foi possível utilizar todas as técnicas disponíveis dos métodos, principalmente do *Catalysis*. Dentre as técnicas que não foram utilizadas estão os *frameworks* e a realização de mais níveis de refinamentos. Nem todos os tipos de colaborações foram utilizados.

Referências Bibliográficas

[1] Dia, http://www.lysator.liu.se/ alla/dia/ (último acesso em 27/08/2003).

[2] Enterprise JavaBeans, http://java.sun.com/products/ejb (último acesso em 27/08/2003).

[3] Microsoft COM+, http://msdn.microsoft.com (último acesso em 27/08/2003).

[4] Rational Rose, http://www.rational.com (último acesso em 27/08/2003).

[5] The Unified Modeling Language v1.3, Object Management Group, http://www.omg.org/uml (último acesso em 27/08/2003).

[6] Together Control Center, http://www.borland.com (último acesso em 27/08/2003).

[7] Aoyama, M. New age of software development: How component-based software engineering changes the way of software development. In *Proceedings of International Workshop on Component-Based Software Engineering, Kyoto, Japão (http://www.sei.cmu.edu/cbs/icse98/papers/p14.html)* (1998).

[8] Buschmann F., Meunier R., Rohnert H., Sommerlad P. e Stal M. *Software Engineering with Reusable Components*. John Wiley & Sons, 1996.

[9] Cheesman, J., e Daniels, J. *UML Components*. The Component Software Series. Addison-Wesley, 2001.

[10] Crnkovic, I. Component-based software engineering - new challenges in software development. In *Software Focus* (2001), vol. 2, pp. 127–133.

[11] Gamma E., Helm R., Johnson R. e Vlissides J. *Design Patterns - Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[12] Lim, W. *Managing Software Reuse.* Prentice Hall, 1998.

[13] Liu, Y., e Cunningham, C. Software Component Specification Using Design by Contract. In *Proceedings of the South East Software Engineering Conference, Tennessee Valley Chapter, National Defense Industry Association, Huntsville, AL* (2002).

[14] Meyer B. *Object-Oriented Software Construction.* Prentice Hall, 1997.

[15] Sametinger, J. *Software Engineering with Reusable Components.* Springer Verlag, 1997.

[16] Szperski, C. *Component Software: Beyond Object-Oriented Programming.* Addison-Wesley, 1998.

[17] Szperski, C. Component and objects together. In *Software Development Magazine* (1999), vol. 7.

[18] Teixeira, H. V. Geração de componentes de negócio a partir de modelos de análise. Tese de Mestrado, COPPE/UFRJ, 2003.

[19] Warmer J. e Kleppe A. *The Object Constraint Language : Precise Modeling with UML.* Addison-Wesley, 1999.

[20] Wills, A. C., e D'Souza, D. *Objects, Components and Frameworks with UML: The Catalysis Approach.* Addison-Wesley, 1999.