FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA – UNIVEM
CURSO DE CIÊNCIA DA COMPUTAÇÃO

CLAUDINEI DE OLIVEIRA

DIRETRIZES PARA DOCUMENTAÇÃO DE COMPONENTES DE SOFTWARE EXISTENTES

MARÍLIA
2006

CLAUDINEI DE OLIVEIRA

DIRETRIZES PARA DOCUMENTAÇÃO DE COMPONENTES DE SOFTWARE EXISTENTES

Monografia apresentada ao Curso de Ciência da Computação da Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador:
Prof. Dr. Valter Vieira de Camargo
Co-Orientadora:
Profª. Drª. Maria Istela Cagnin Machado

MARÍLIA
2006

CLAUDINEI DE OLIVEIRA
RA. N.º 317209

DIRETRIZES PARA DOCUMENTAÇÃO DE COMPONENTES DE SOFTWARE EXISTENTES

BANCA EXAMINADORA DA MONOGRAFIA PARA OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

CONCEITO FINAL: _____(_____)

ORIENTADOR: _____
Prof. Dr. Valter Vieira de Camargo

1º EXAMINADOR: _____
Profª. Drª. Ana Paula Piovesan Melchiori Peruzza

2º EXAMINADOR: _____
Prof. Dr. Edmundo Sergio Spoto

Marília, 04 de Dezembro de 2006.

Aos meus pais, Antônio e Maria de Lourdes.

AGRADECIMENTOS

Começo agradecendo sempre a Deus por a minha vida e por me dar saúde.

Aos meus pais, Antônio Ferraz de Oliveira e Maria de Lourdes de Oliveira por todo o amor, carinho, orgulho e confiança depositada em mim, o que fazia tudo parecer fácil, mesmo quando parecia impossível. Sem eles não seria possível ter sequer iniciado esta jornada.

Agradeço a Co-Orientadora Profª. Drª. Maria Istela Cagnin Machado pelo seu apoio, dedicação e confiança depositada em mim.

Agradeço principalmente ao meu Orientador Prof. Dr. Valter Vieira de Camargo por orientar com muita paciência, dedicação, compreensão e amizade. E por todas as sugestões e questionamento que contribuiu para o andamento dessa monografia.

Aos meus amigos, colegas que direta ou indiretamente contribuíram para a realização desta monografia.

A todos os professores do UNIVEM no qual tive a oportunidade e o prazer de aprender.

Finalmente, um grande e especial agradecimento à minha namorada Patrícia pela compreensão e incentivo durante o período da elaboração desta monografia.

“A melhor parte da vida de uma pessoa está nas sua amizades”
Abraham Lincon

“Ter muitos amigos é não ter nenhum”
Aristóteles

OLIVEIRA, Claudinei de. **Diretrizes para Documentação de Componentes de Software Existentes.** 2006. 70 f. Trabalho de Conclusão de Curso – Centro Universitário Eurípides Soares da Rocha, Marília, 2006.

RESUMO

Neste trabalho são apresentadas diretrizes para documentação de componentes de software existentes. O objetivo é facilitar o entendimento, análise e documentação de componentes, visando também facilitar seu reúso.

Para apoiar as diretrizes para documentação é utilizada a técnica UML *Components*, para tratar a especificação e modelagem das interfaces dos componentes e a GRN para auxiliar na obtenção da documentação, pertencente ao mesmo domínio dos componentes. As diretrizes só servem para sistemas que são baseados em componentes, e não possuam nenhuma documentação a respeito do componente.

Partindo do sistema desenvolvido com componente, é utilizado o código-fonte e o programa em execução, para recuperar os requisitos e a especificação dos componentes, fornecendo a documentação do componente.

Assim, é apresentado um conjunto de diretrizes que represente o domínio e a análise dos componentes existentes, permitindo posteriormente que outros componentes sejam documentados por meio dessas diretrizes.

Para validar as diretrizes apresentadas foi conduzido um estudo de caso de um sistema baseado em componentes, chegando a produzir toda a documentação pertinente ao componente.

Palavras-Chave: Componente de software, UML *Components, Catalysis, Rational Unified Process* (RUP) e Padrões de Análise.

OLIVEIRA, Claudinei de. **Diretrizes para Documentação de Componentes de Software Existentes.** 2006. 70 f. Trabalho de Conclusão de Curso – Centro Universitário Eurípides Soares da Rocha, Marília, 2006.

ABSTRACT

In this work, guidelines for the documentation of existing softwares will be presented the objective is so make the understanding, analyses and documentation of the components easier, and by doing so also making its reuse easier.

To support the guidelines for documentation has bee used the UML *Components* techniques to dead with the specification and modularization of the components' interfaces and the GRN to help in the documentation obtaining, belonging to the same domain of the components. The guidelines only work to systems that are based on components and do not own any documentation about it.

From the developed system with the components, it is used the source-code and the program in execution to recover the commands and the specification of the components supplying the documentation of the components.

Therefore, it's presented a group of guidelines which represents the domain and the analyses of the existing components allowing, afterwards, that ofher components be documented by this guidelines.

To validate the presented guidelines it was conduted a case study of a system based on components even producing a wee of documentation to the system.

Keywords: Software Component, UML *Components*, *Catalysis*, *Rational Unified Process* (RUP), Patterns of Analysis.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AD – Análise de Domínio

DBC – Desenvolvimento Baseado em Componentes

DDCS –Diretrizes para Documentação de Componentes Software

ED – Engenharia de Domínio

RUP – *Rational Unified Process*

UML – *Unified Modeling Language*

SUMÁRIO

INTRODUÇÃO

A Engenharia Reversa é o processo de análise de um sistema para identificar seus componentes e criar representações num nível mais alto de abstração, produzido preferencialmente de forma automática documentos que ajudem a aumentar o conhecimento geral de sistemas de software, facilitando o reúso (WERNER e BRAGA, 2005).

O conceito de reutilização de software não é novo. Desde que começaram a aparecer os primeiros sistemas computacionais havia interesse em reutilizar partes do código em outros sistemas de modo a reduzir a complexidade, os custos e aumentar a qualidade, uma vez que os artefatos reutilizados já foram testados (PRESSMAN, 2002).

De acordo com Sanches (2001), os maiores especialistas na área de Engenharia de Software reconhecem que a falta de um projeto de documentação tem prejudicado a manutenção do software durante toda a história da computação. De um modo geral a utilização do software tem sido degradada por não se ter disponíveis informações sobre uma documentação bem elaborada. Nesse contexto, a utilização de padrões na documentação de processos é fundamental.

Barroca et al., (2005) cita que padrões de software, que fornecem soluções para problemas recorrentes, no contexto de Desenvolvimento Baseado em Componente (DBC) caracterizam-se como um importante meio de documentação, tanto para o desenvolvedor do componente, que pode encontrar nos padrões um meio para desenvolver o componente; quanto para o reutilizador de componentes, que pode encontrar uma forma de enxergar a organização da especificação do componente.

A análise de domínio está relacionada como um processo que identifica, coleta e organiza informações provenientes do desenvolvimento de um sistema, para reutilizar estas informações em novos softwares que serão desenvolvidos posteriormente.

Dentre os métodos que apóiam o DBC têm-se o UML *Components*. Esse método concentra-se em considerar os componentes como uma visão arquitetural da análise realizada sobre um problema. Para isso os componentes adotam os princípios da orientação a objetos, comportamentos como unificação de dados e funções, encapsulamento e identidade para se adequarem ao papel da interface e das especificações, independentes de tecnologia (WERNER e BRAGA, 2005).

Ressalta-se que padrões de software quando pertencem ao nível de análise, podem ser utilizados para apoiar o entendimento e a documentação de sistemas legados[1] (CAGNIN, 2005). A partir disso, este trabalho tem como objetivo utilizar padrões de software de análise existentes para apoiar no entendimento e na documentação de componentes de software. Com isso, será definido uma diretriz de documentação de componentes de software já desenvolvidos com o apoio de padrões de análise existentes. Isso porque é importante que componentes de software desenvolvidos possuam documentação a fim de facilitar o seu reúso.

Em geral, existem diversos componentes disponíveis que não possuem documentação referentes ao projeto, implementação e a análise.

Motivação

Cada vez mais há interesse, por parte dos engenheiros de software, em reutilizar componentes de software para apoiar no desenvolvimento e na manutenção de software. No entanto, para que isso possa ser efetivamente praticado, é necessário que a documentação de tais componentes sejam também disponibilizadas, o que não ocorre normalmente. Assim, é de

[1]É um sistema que ainda está em funcionamento, mas sua qualidade e vida operacional estão constantemente deterioradas devido a diversas atividades de manutenção e atualizações tecnológicas, rodam em hardware e software ultrapassados e possui geralmente uma documentação incompleta ou até inexistente (ZOU E KONTOGIANNIS, 2002 **apud** CAGNIN).

grande importância a preocupação com a elaboração da documentação de componentes de software existentes.

A documentação auxilia no reúso de sistemas já existentes, facilitando seu entendimento.

Objetivos

Como a documentação de componentes de software dão mais enfoque, em geral, no projeto e na implementação e considerando a importância da documentação no nível de análise dos componentes, para facilitar o entendimento do domínio ao qual pertencem bem como facilitar o desenvolvimento com o apoio de componentes, observou-se à importância de desenvolver este trabalho.

Este trabalho tem como objetivo teórico estudar os conceitos relacionados a componentes e documentação de componentes em geral; selecionar componentes existentes de um determinado domínio, estudar padrões de software de análise, existentes na literatura; pertencentes ao mesmo domínio dos componentes selecionados e por fim; esboçar uma diretriz de documentação componentes de software existentes baseado em padrões de análise.

Organização da Monografia

Este trabalho está organizado em três capítulos. Foram apresentados neste capítulo o contexto, a motivação e os objetivos deste trabalho. O restante do documento está organizado de seguinte maneira: No Capítulo 1 é apresentada a revisão bibliográfica deste trabalho, fornecendo os conceitos que proporcionam o embasamento teórico necessário para a compreensão dos principais aspectos relacionados à reutilização de software, importância dos componentes de software, bem como definições, características e benefícios dos componentes, conceitos de documentação de software e de padrões de software.

No Capítulo 2, o processo de documentação é apresentado. Os passos do processo, juntamente com as diretrizes são detalhadas em cada seção do capítulo. Para apresentar o processo proposto, é utilizado com estudo de caso um Sistema de Reserva de Hotel, extraído do livro de Cheesman e daniels (2001).

No último capítulo são apresentadas as conclusões, contribuições deste trabalho e propostas de trabalhos futuros.

Finalmente, no Apêndice A, apresenta o documento de requisitos do sistema de reserva de hotel.

1 REVISÃO BIBLIOGRÁFICA

1.1 Considerações Iniciais

Neste capítulo é apresentada a revisão bibliográfica, abordando conceitos teóricos importantes referentes a componentes de software, necessários para a compreensão do desenvolvimento do trabalho. Na Seção 1.2 é apresentada a importância da reutilização de software. Na Seção 1.3 é apresentada a definição de componente de software, fornecendo embasamento teórico sobre os principais aspectos relacionados à importância dos componentes de software. Na Seção 1.3.1 aborda-se sobre a documentação de componentes de software, destacando os processos *Rational Unified Process* (RUP), *Catalysis* e UML *Components*. Uma visão geral e cada método citado é apresentado nas subseções seguintes, com destaque para o método de Cheesman e Daniel (2001), que será utilizando para especificar componente, na diretriz de documentação de componentes de software existentes que constituiu a base para este trabalho. Na Seção 1.4 define-se padrão de software e linguagem de padrão e apresenta-se a linguagem de padrões (GRN), bem como outras linguagens de padrões de análise existentes. Por último, na Seção 1.5, discutem-se as considerações finais deste capítulo.

1.2 Reutilização de Software

A reutilização de software tem como objetivo a eficiência no processo de desenvolvimento. Os projetistas podem adquirir diversos benefícios por reutilizarem software existente, como: aumento da qualidade e redução do esforço de desenvolvimento (BARROCA *et al*., 2005; SOMMERVILLE, 2004).

Existem várias definições de reutilização de componente. De acordo com Lim (1998 **apud** ROSSI, 2004) e Sommerville (2003) a reutilização tem como objetivo melhorar a

produtividade, aumentar a qualidade, reduzir o tempo de desenvolvimento e proporcionar usabilidade. Neste contexto, Lim (1998 **apud** ROSSI, 2004) complementa que componentes são os produtos ou subprodutos do processo de desenvolvimento de software, incluindo tanto elementos tangíveis (código, projeto, algoritmos, planos de teste e documentação) quanto elementos intangíveis (conhecimento e metodologia).

Segundo Sommerville (2004), "o reúso sistemático requer um processo de projeto que considere como os projetos existentes podem ser reutilizados e que organize explicitamente o projeto em torno de componentes de software disponíveis".

A reutilização de software se baseia na programação modular em que se pode fazer o uso de procedimentos, funções, classes pré-existentes e componentes, afim de outros desenvolvedores reaproveitam em sua aplicação (SOMMERVILLE, 2003).

Para Freeman ((1987); (BASILI e ROMBACH, 1988); (COOPER, 1994) **apud** (BARROCA *et al.*, 2005)), a reutilização é o uso de qualquer informação já disponível que um desenvolvedor pode necessitar no processo de criação de software. Basili e Rombach destacam ainda que essa informação inclui o conhecimento e Cooper cita que a reutilização pode ser com ou sem modificação.

Para possibilitar a reutilização de software existem diversas abordagens, dentre elas componentes e padrões de software, que são de interesse deste trabalho.

De acordo com Sommerville (2003), o desenvolvimento baseado em componente ideal deve ser construído a partir de componentes existente, nos quais já tenham sido reutilizados.

Existe uma inevitável conciliação entre a facilidade de reúso e a de uso de um componente. Tornar o componente reutilizável implica fornecer uma interface mínima e simples, que seja fácil de compreender. Já a facilidade do reúso aumenta a complexidade, desse modo, reduz a facilidade de compreensão do componente (SOMMERVILLE, 2003).

No desenvolvimento de software, uma quantidade considerável de documentos é gerada. Como exemplos desses documentos têm-se: diagramas, documentos de requisitos, código-fonte, manuais, etc. Em empresas de desenvolvimento de software é comum a utilização entre 20 e 30% de todo o esforço do desenvolvimento na documentação (PRESSMAN, 2002). Diante disso, ferramentas de produção e gerência de documentos apóiam praticamente toda a Engenharia de Software.

As unidades de software que são reutilizadas podem ser de tamanhos diferentes (SOMMERVILLE, 2004):

- Reúso de sistemas de aplicações: o software pode ser reutilizado em outra aplicação sem mudanças.
- Reúso de componentes: o componente de uma aplicação com tamanho e função variados pode ser reutilizado.
- Reúso de funções: uma única função pode ser reutilizada.

Para facilitar o uso de componentes de software é importante que eles sejam documentados de maneira apropriada. Em geral, nenhuma documentação é fornecida sobre o componente. Ressalta-se a importância da documentação de componentes de software no nível de análise, a fim de facilitar o entendimento do componente para reusar na construção de novos sistemas. Sob essa perspectiva, este trabalho tem como objetivo geral disponibilizar um conjunto de diretrizes para documentação de componentes software existentes.

1.3 Conceitos de Componentes de Software

Segundo Barroca *et al*. (2005), um componente pode ser definido como uma unidade de software independente, que encapsula, dentro de si, seu projeto e também sua implementação; e oferece serviços por meio de interfaces bem definidas para o meio externo. A motivação para componentes não está unicamente relacionada à reutilização. Atualmente,

as pressões para a liberação de produtos no mercado, fizeram com que o processo de desenvolvimento de curto prazo seja uma tendência que, cada vez mais, está sendo considerado na construção de software.

Existem diversas definições de componentes de software. Segundo Sommerville (2003), os softwares são montados com partes menores, previamente testadas e confiáveis, que agilizam o processo de desenvolvimento facilitando o trabalho das equipes de desenvolvimento e reduzindo os custos, com o objetivo de obter a versão final do software com menor tempo de desenvolvimento. Os softwares grandes e complexos estão sendo construídos por meio de uma série de componentes, podendo ser desenvolvidos em estruturas de tempo mais realistas.

Bass *et al*. (2001) definem componente de software como uma implementação de alguma funcionalidade, que é reutilizada em diferentes aplicações, acessada por meio de interfaces e pode ser vendida como um produto comercial.

Os componentes podem ser entendidos como uma caixa preta provedora de serviços. Quando um serviço precisa de uma funcionalidade, ele usa um componente para fornecer esse serviço, sem se preocupar de onde esse componente está sendo executado ou em qual linguagem de programação está implementado (SOMMERVILLE, 2003).

Na Figura 1.1 é mostrada a representação gráfica das interfaces de um componente. Os componentes são definidos por suas interfaces e nos casos mais genéricos, podem ser imaginados como duas interfaces relacionadas.

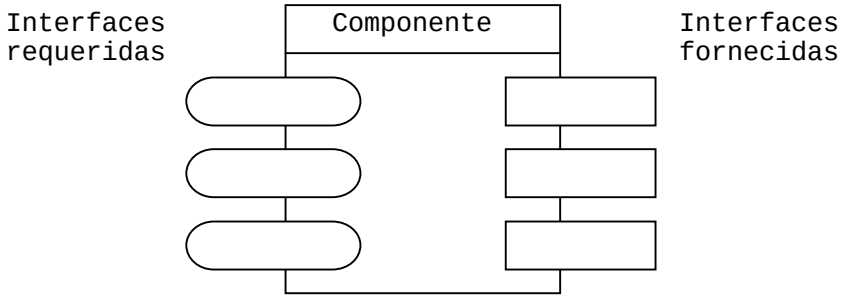


Figura 1.1 – Interfaces de componente (Sommerville, 2003).

As interfaces são pontos de interconexão do componente, podendo ser classificadas em dois tipos: interfaces fornecidas (provided interfaces) definem os serviços oferecidos pelo componente por meio de operações e interfaces requeridas (required interfaces) definem os serviços que o componente necessita de outros componentes (BARROCA et al., 2005).

A seguir são descritas algumas características de componentes, comentadas por Barroca *et at*., (2005):

- Deve fornecer uma especificação clara dos seus serviços e interfaces bem identificadas. Cada interface consiste em serviços especificados, mediante uma ou mais operações, sendo cada uma delas separadamente identificada de acordo com seus parâmetros de entrada e saída e respectivos tipos estabelecidos;
- As interfaces requeridas definem os serviços necessários de outros componentes para complementar seu próprio serviço, e também devem ser definidas explicitamente;
- Um componente deve garantir o encapsulamento e a ocultação de seus dados e processos;
- A especificação do serviço oferecido deve ser completa e não ambígua.
- A comunicação entre componente que utiliza um outro componente deve levar em consideração apenas suas interfaces, não sendo feito suposição alguma sobre a sua implementação;
- Informações sobre propriedades não funcionais como velocidade devem ser fornecidas.

1.3.1 Documentação de Componente de Software

A documentação de um componente é algo concreto que disponibilizará um material que facilitará o seu entendimento, bem como o seu uso e a sua evolução (KRUCHTEN, 2000 **apud** CAGNIN, 2005).

As informações que devem conter no documento de um componente são: a especificação do componente, que devem garantir que as especificações de suas interfaces sejam compatíveis; o relatório de validação, que consiste de uma série de testes que validam o componente nos ambientes para os quais ele foi projetado; as propriedades não-funcionais pertinentes ao componente, como segurança, desempenho e confiabilidade (BARROCA *et al*., 2005). Sendo de interesse para esse trabalho a parte da especificação de componentes.

Para Cheesman e Daniels (2001), dois tipos de contato que um componente pode ter: uso (usage) com o cliente e implementação (realization) como o implementador. O primeiro corresponde à especificação para um usuário de componentes, que consiste em uma lista de operações fornecidas na sua interface. Enquanto o segundo é voltado para o implementador, é visto como a parte da especificação do componente que não precisa ser conhecida pelo usuário. Essa separação é feita para facilitar a manutenção, para que as modificações de implementação não afetam os usuários de um componente. Além das especificações de uso e implementação, é necessária uma especificação dos conectores.

Existem várias maneiras diferentes de conectar componentes, sendo uma delas a clamada explícita de procedimentos, apresentados por D'Souza e Wills (1998, **apud** BARROCA *et al*., 2005).

Numa visão geral, percebe-se que fica indispensável a documentação de componente, seja ela para o reúso ou para o próprio entendimento do componente.

Existem vários processos que apóiam o desenvolvimento baseado em componentes. Dentre eles tem-se: *Rational Unified Process* (RUP) (KROLL e KRUCHTEN, 2000),

Catalysis (D'SOUZA e WILLS, 1998) e UML *Components* (CHEESMAN e DANIELS, 2001).

1.3.2 *Rational Unified Process (RUP)*

O *Rational Unified Process* (RUP) (KRUCHTEN, 2000) embora seja chamado de processo, na verdade trata-se de uma estrutura de processo, fornecendo um vocabulário e uma estrutura para falar sobre processos. Embora seja independente da UML, freqüentemente são mencionados juntos (FOWLER, 2005).

Às vezes, o RUP é mencionado como UP (*Unified Process* – Processo Unificado). Organizações utilizam a terminologia e o estilo global do RUP, sem usar os produtos licenciados da *Rational Software*.

O RUP é um modelo de processo de software, composto por duas dimensões ou estruturas: a dinâmica e a estática, conforme apresentado na Figura 1.2. A estrutura dinâmica representa o tempo de um processo e mostra aspectos do ciclo de vida de um processo (KROLL e KRUCHTEN, 2000). A estrutura estática representa o aspecto estático do processo e descreve como os componentes de processo, atividades, disciplinas, artefatos e papéis estão logicamente integrados nas atividades do processo.

A estrutura dinâmica divide o ciclo de vida do software em quatro fases: concepção, elaboração, construção e transição. Cada fase possui um conjunto bem definido de objetivos que podem ser utilizados como guia para decidir quais atividades executar e quais artefatos produzir.

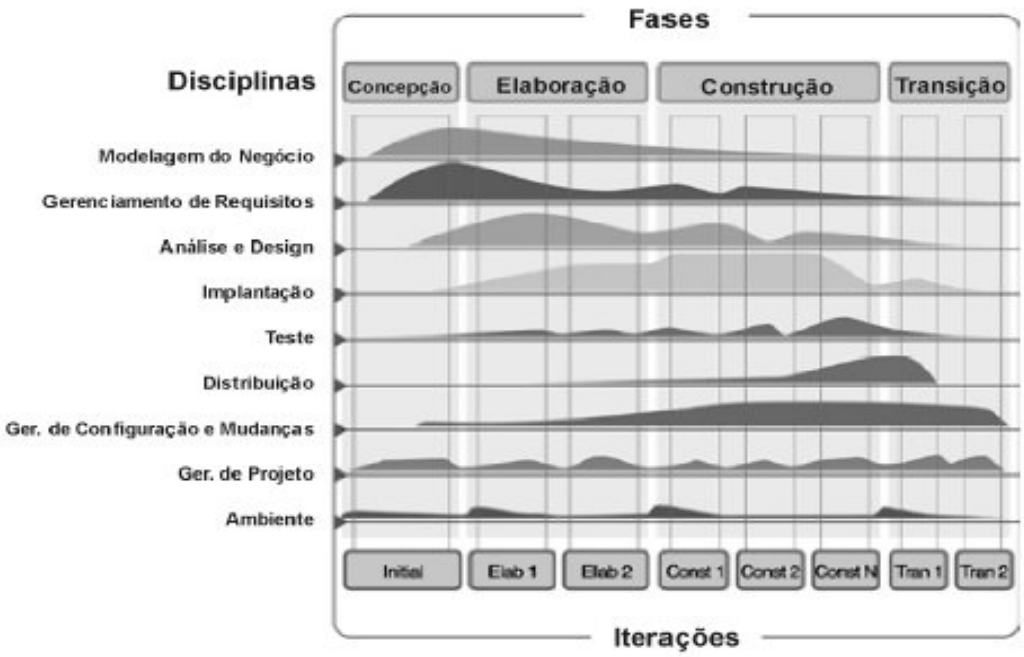


Figura 1.2 – As duas estrutura do RUP (KRUCHTEN, 2000).

A seguir é descrita cada fase do RUP:

- **Concepção:** é uma visão inicial de um projeto, visando estabelecer um bom entendimento da funcionalidade do sistema; descreve todos os requisitos e escopo do sistema, e obtém a permissão dos interessados para a continuidade do projeto.
- **Elaboração:** é responsável em identificar os casos de uso principais do projeto, concentra-se nas atividades mais complexas tecnicamente: projeto, implementação, teste e criação da arquitetura executável. Fornece ao final uma idéia de solucionar os riscos de desempenho e de contenção de recursos, por meio da implementação e da validação do código-fonte.
- **Construção:** dá seqüência ao processo, desenvolvendo a maior parte da implementação para obter a primeira versão executável do sistema. São geradas diversas versões internas para assegurar que o sistema é utilizável e atende as solicitações do usuário. Ao concluir essa fase, o cliente recebe uma

versão executável do sistema, a documentação de apoio e o material de treinamento, embora possa haver ainda ajustes de funcionalidade.

- **Transição**: garante que o software atenda as necessidades do cliente. Alguns testes do produto são feitos em preparação para o seu lançamento. As informações do usuário nesse ponto, geralmente diz respeito a pequenos ajustes no produto, configuração e instalação. Quando um problema crítico é descoberto, deve ser resolvido em fases anteriores.

A estrutura estática é definida em função de quatro elementos primários de modelagem, de acordo com a Figura 1.3:

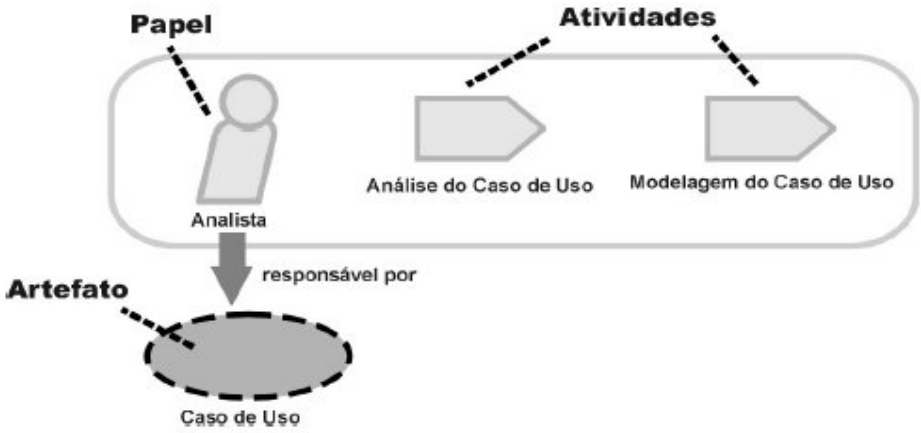


Figura 1.3 – Papéis de executores, atividades e artefatos (KRUCHTEN, 2000).

- **Papel:** define a responsabilidade de um indivíduo ou grupo trabalhando junto em uma equipe e também o comportamento por meio de atividades que o indivíduo executa. Diversas funções ou papéis podem ser desempenhados pelo mesmo individuo. Exemplos de papéis: análista, projetista e programadores.
- **Atividades:** é uma atividade de trabalho que um indivíduo pode executar e produz um resultado positivo no projeto. Pode ser quebrada em passos para facilitar a sua execução.

- **Artefatos:** um artefato é uma informação que é produzida, gerando o resultado do trabalho realizado.
- **Disciplina ou fluxo de trabalho:** define a importância de manter uma seqüência significativa das atividades, produzindo um resultado. Ressalta-se que somente papéis, atividades e artefatos não formam um processo complexo.

Uma disciplina no modelo RUP é uma coleção de atividades produzindo um resultado importante no desenvolvimento de software. As principais disciplinas do RUP são:

- **Modelagem do Negócio:** descreve os processos a fim de compreender melhor o negócio, capaz de criar requisitos adequados para o sistema a ser desenvolvido.
- **Gerenciamento de Requisitos:** Levanta, organiza e documenta requisitos.
- **Análise e Projeto:** Cria a arquitetura e o projeto do sistema.
- **Implementação:** Escreve o código-fonte, efetua testes e gerencia o código-fonte e os executáveis.
- **Teste:** Efetua testes de integração, de sistema e de aceitação.
- **Implantação:** Torna o software disponível aos seus usuários, elabora a documentação final e empacota o software.
- **Gerenciamento de Configuração e de Mudanças:** Efetua atividades ligadas ao gerenciamento de versões, disponibilizarão e mudanças.
- **Ambiente:** Faz uma adequação ao processo às necessidades de uma organização, ou de um projeto.

1.3.3 *Catalysis*

Catalysis é um método de desenvolvimento baseado em componentes, considerado como um dos mais complexos, que aborda conceitos relevantes nesse tipo de

desenvolvimento, cobrindo todas as fases de desenvolvimento de um componente, desde a sua especificação até sua implementação (WERNER e BRAGA, 2005).

Catalysis utiliza a UML com alterações principalmente no que se refere à especificação de componentes.

De acordo com D'Souza e Wills (1998, **apud** ROSSI, 2005), o método *Catalysis* apresenta alguns princípios fundamentais para o processo de desenvolvimento de software, se divide nas seguintes fases:

- Especificação de requisitos ou Modelo de negócio: nessa primeira etapa deve-se entender as necessidades do cliente, capturando as regras de negócio, comportamento e as limitações do problema a ser solucionado, produzindo um modelo de domínio.
- Especificação do sistema: nesta fase deve-se identificar as funcionalidades do sistema e estabelecer como será a interação do sistema com pessoas e sistemas; refinar a especificação de requisitos com o objetivo de construir interações precisas das ações executadas pelo sistema.
- Arquitetura: nesta fase defini-se a arquitetura da aplicação para identificar os componentes necessários à implementação do sistema e também para projetar as interações e dependências entre os componentes.
- Projeto: nesta fase são especificados os componentes e o contexto de utilização, tendo uma importante preocupação com o desacoplamento dos componentes.
- Implementação: concentra-se em definir a implementação interna dos componentes e suas interações de forma que sejam satisfeitos os requisitos de todos os componentes.

Catalysis possui um conjunto de características, que são:

- Processo formal que destaca a precisão;
- Rastreabilidade dos modelos;

- Baseada em UML;
- Baseado em componentes;
- Reutilização de modelos e frameworks.

O Catalysis permite uma adaptação do processo de desenvolvimento de acordo com as características do projeto, definindo melhor a ordem das atividades e os artefatos que devem ser produzidos para cada caso (WERNER e BRAGA, 2005).

1.3.4 UML *Components*

A UML (*Unified Modeling Language*) (BOOCH *et al*., 1998) nasceu de três linguagens gráficas de modelagem orientada a objetos. Desde 1997 é uma linguagem de terceira geração para especificação e modelagem de sistemas. É de uso ilimitado podendo ser utilizada para especificar, visualizar, construir e documentar artefatos de um sistema orientado a objetos. Seu conteúdo é uma união das melhores práticas de Engenharia de Software já utilizadas na modelagem de sistemas complexos, especialmente em suas arquiteturas (BIANCHINI, 2004).

UML *Components* é um método proposto por CHEESMAN e DANIELS (2001) que possui um processo bem definido para o desenvolvimento de software baseado em componentes e utiliza a UML como notação para os elementos do projeto.

De acordo com Werner e Braga (2005), a UML Components concentra-se principalmente em considerar os componentes como uma visão arquitetural da análise realizada sobre um problema. Os componentes devem adotar os princípios da orientação a objetos, ou seja, devem se comportar como unificação de dados e funções, encapsulamento e identidade, para se adequarem ao papel da interface e das especificações, independentes de tecnologia.

O processo UML *Components* é uma adaptação do RUP, sendo que atividades foram acrescentadas ou modificadas, como a especificação, o fornecimento e a montagem, substituindo algumas atividades do processo RUP, como as disciplinas de análise e projeto. Além disso, vários novos diagramas foram incluídos, a fim de suprir as necessidades de modelagem (WERNER e BRAGA, 2005).

De acordo com Werner e Braga (2005) *UML Components* não apóia qualquer prática de um processo de reutilização de componentes. Isso ocorre porque aspectos de ED – engenharia de domínio[2] não são apresentados, ou seja, para a reutilização de componentes, a metodologia deveria se preocupar e considerar de maneira adequada os aspectos de ED. Considerado ED antes e ao longo do processo proposto pela UML *Components* poderia trazer grandes benefícios para a abordagem em si.

UML *Components* também contextualiza os mesmos diagramas tradicionais de UML.

O método propõe a utilização da UML para modelar todas as fases do desenvolvimento de sistemas baseados em componentes, contendo as atividades de definição de requisitos, identificação e descrição das interfaces entre componentes, especificação e modelagem, além de implementação e montagem. Além disso, foram criados novos diagramas para o DBC, visto que havia essa necessidade (WERNER e BRAGA, 2005).

No Quadro 1.1 apresentam-se os diagramas utilizados pela abordagem UML Components.

[2] É a atividade de coletar, organizar e armazenar experiências no desenvolvimento de sistemas ou de partes específicas de sistemas em um domínio particular, permitindo o reúso de recursos para a construção de novos sistemas.

Quadro 1.1 – Diagramas utilizados pela abordagem UML *Components* (Werner e Braga, 2005, p. 94).

	Diagrama	***Descrição***
Análise	Diagrama do Modelo do Conceito do Negócio (*Business Concept Model Diagram*)	Modelo conceitual das informações que existem no domínio.
	Diagrama de Tipo do Negócio (*Business Type Diagram*)	Diagrama com os tipos de negócios (e seus relacionamentos) que precisam ser mantidos pelo software.
	Diagrama de Casos de Uso (*Use Case Diagram*)	Diagrama de casos de uso do software.
Especificação da arquitetura	Diagrama de Especificação de Interface (*Interface Specification Diagram*)	Definição precisa sobre as ações de uma interface, possuindo um modelo de informação, a especificação das operações e as invariantes.
	Diagrama de Especificação do Componente (*Component Specification Diagram*)	Diagrama que descreve as interfaces fornecidas e exigidas de uma especificação de componente.
	Diagrama de Arquitetura do Componente (*Component Architecture Diagram*)	Diagrama da arquitetura de componentes.
	Diagrama de Responsabilidade da Interface (*Interface Responsibility Diagram*)	Diagrama que mostra o conhecimento de uma interface sobre os elementos de negócio.
	Diagrama de Interação do Componente (*Component Interaction Diagram*)	Diagrama de interação entre uma especificação de componente e suas interfaces em tempo de execução.

No Quadro 1.2 apresenta-se um resumo das principais características do método UML *components*.

Quadro 1.2 – Principais características do método UML *Components* (Werner e Braga, 2005, p. 95).

Processo	O processo de DBC é detalhado, sendo uma modificação do RUP para atender aspectos específicos de DBC. Aspectos relacionados à ED não são apresentados.
Produto	Um conjunto de modelos UML modificados para incluir características DBC. É clara a falta de um modelo que guie o desenvolvedor no processo de reutilização de componentes em uma aplicação.
Ferramental automatizado	Utilização de ferramental proposto pela Rational.

Analisado esse métodos algumas características são descritas:

1. **Documentação**: o método UML Components apresenta vasta documentação de fácil acesso;
2. **Fácil integração com UML**: o método apresenta fácil integração com UML por causa do uso de estereótipos;

3. **Processo flexível e simples**: o método somente adiciona o conceito de seis estereótipos e por isso torna-se simples para modelagem de componentes;
4. **Específico para DBC**: ao contrário do RUP, por exemplo, que é um processo genérico para desenvolvimento de software, o UML Components se apresenta voltado exclusivamente para DBC;
5. **Plataforma aberta**: UML Components pode ser usado para especificar componentes em qualquer plataforma de desenvolvimento;
6. **Bem aceito na indústria e academia**: a idéia deste método já vem sendo bem usada tanto pela indústria de software quanto pela academia.

1.3.4.1 Extensão de UML com estereótipos

Para a extensão para UML existe um grande número de mecanismos diferentes, provavelmente a mais utilizada de acordo com Cheesman e Daniels (2001 apud NASCIMENTO, 2005) é utilizar estereótipos. Permite que qualquer elemento tenha um estereótipo anexado, facilitando a extensão da linguagem. Ele é encapsulado pelo símbolo "<< >>" permitindo ao usuário criar novos elementos de modelagem.

Serão apresentados os estereótipos utilizados no método:

1. **<<type>>.** Usado para representar uma classe no modelo de negócios de um componente em nível de especificação. Vale ressaltar que esse conceito não é o mesmo que classe em uma linguagem de programação como Java.
2. **<<datatype>>.** Possui a mesma representação de Type, mas é usado para classes que utilizam persistência.
3. **<<interface type>>.** Representa uma interface em nível de especificação. Por convenção, o nome da classe desse tipo são iniciadas com um "I". A ressalva deste estereótipo é não confundir com o estereótipo: <<interface>>

já presente em UML. Este último serve para modelagem orientada a objetos e isso é o que exatamente se deseja evitar quando for feita a modelagem ou especificação de componentes.

4. **<<comp spec>>.** Usado para indicar uma especificação de um componente. Este estereótipo, normalmente, está associado a um outro estereótipo <<interface type>> através de uma ligação de <<offers>>, que será explicado na próxima seção.
5. **<<offers>>.** Liga uma especificação de um componente à sua interface. Este estereótipo é análogo a <<realize>> de UML. Ou seja, ao invés de um componente implementar uma determinada interface, o conceito muda, de tal forma que, a especificação de um componente oferece uma interface.
6. **<<core>>.** O "core" subentende um "type" sem ocorrência de dependências, sendo assim, o tipo "core" pode ser encontrado a partir de um refinamento do modelo de um componente, observando quais são os tipos que não apresentam dependência com outros componentes.

1.3.4.2 Visão Geral do Processo de UML Components

O método *UML Components* tem objetivo a partir dos requisitos de negócio de um sistema produzir uma arquitetura de componentes e suas especificações no nível de análise.

O processo global tem como entrada os requisitos de negócio de um sistema que é usado pela etapa de requisitos para produzir um modelo conceitual de negócio e modelo de casos de uso. Esses modelos, juntamente como os recursos existentes do sistema, formam as entradas necessárias para que a etapa de especificação produza as especificações dos componentes e a arquitetura de componentes. As saídas (especificações e arquitetura) serão

usadas na etapa de provisionamento para decidir quais componentes serão reutilizados, comprados ou construídos, conforme representadas pela Figura 1.4.

A etapa de montagem recebe como entrada os recursos existentes, os modelos de casos de uso e dos componentes, para poder juntá-los e formar uma nova aplicação ou integrá-los em uma aplicação já existente. Após a etapa de montagem, a aplicação é testada e em seguida disponibilizada para a operação.

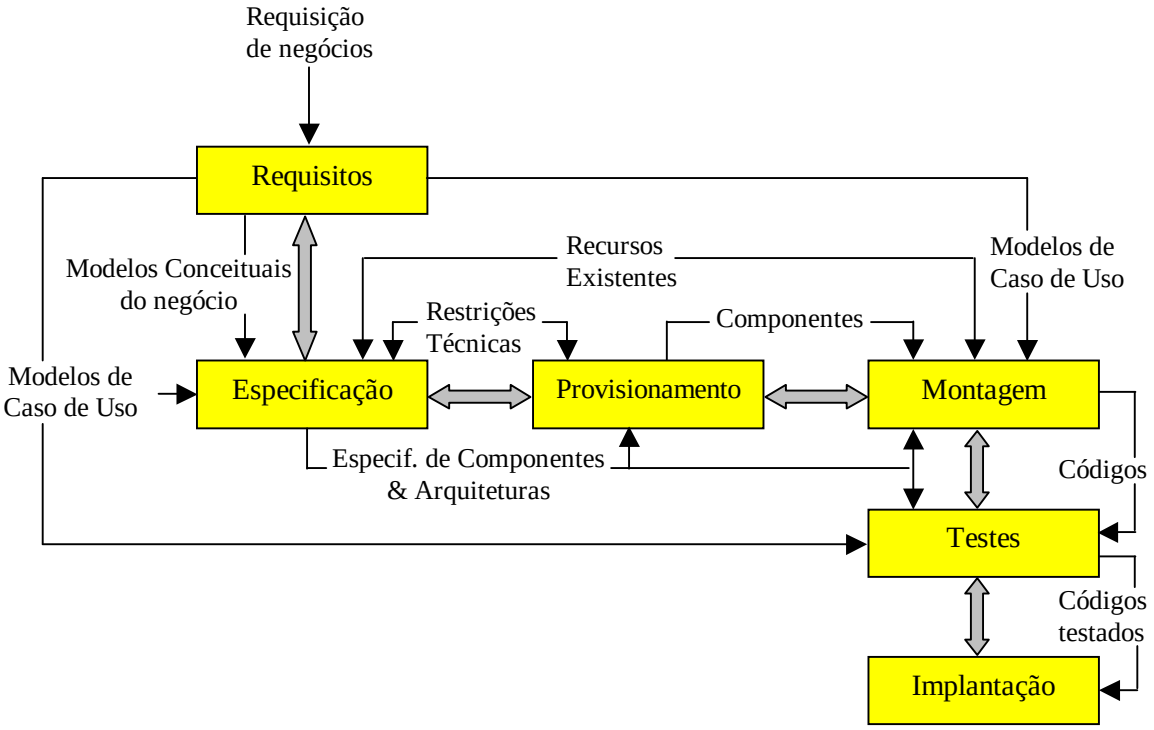


Figura 1.4 – Etapas do método de DBC (Cheesman e Daniels, 2001, p. 27).

De acordo com Nascimento (2005), as atividades Requisitos, Testes e Implantação correspondem diretamente às atividades encontradas no método RUP. Assim será dada somente ênfase nas atividades de Especificação, Provisionamento e Montagem, que substituem diretamente as atividades de Análise, Projeto e Implementação do RUP.

O foco principal deste processo trata-se da especificação dos componentes, dividida em três sub-etapas: identificação, interação e especificação dos componentes, como é mostrado na Figura 1.5. Ela recebe como entrada da atividade de requisitos o modelo de casos de uso e o modelo conceitual de negócio. A atividades de especificação gera como resultado final um conjunto de arquiteturas e especificações de componentes.

Estas saídas serão usadas pela atividade de provisionamento, garantindo que o componente necessário estará disponível, seja ele adquirido de qual for à forma. Exemplo, comprado, construído e reutilizado.

A montagem integra todos os componentes, utilizando uma interface com o usuário adequada, formando uma aplicação que atenda ás necessidades do negócio, ou seja, aos requisitos levantados.

Nas próximas seções que se seguem serão detalhados as três fases da atividade de especificação.

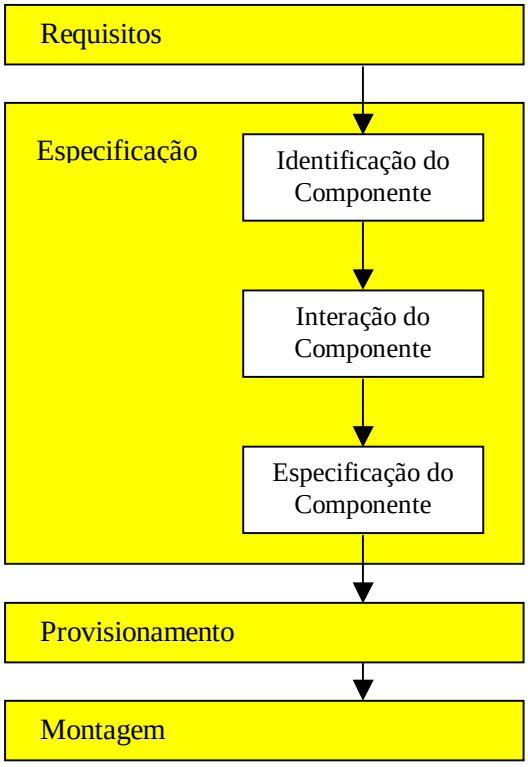


Figura 1.5 – Detalhamento das atividades do processo de DBC utilizando UML Components (Cheesman e Daniels, 2001, p. 27).

1.3.4.3 Identificação dos Componentes

Segundo Cheesman e Daniels (2001) ênfase desta etapa é a descoberta de quais informações precisam ser gerenciadas pelos componentes, quais interfaces são necessárias para gerir essas informações, quais componentes são necessários para prover as funcionalidades. O foco principal é identificar um conjunto inicial de interfaces de negócio

que possuem operações que fazem a gestão do sistema para os componentes de negócio e um conjunto inicial de interfaces e operações do sistema que implementam as regras de negócio para os componentes do sistema.

Outro artefato produzido nesta etapa é modelo de tipos de negócios, que representa a visão do sistema sobre o modelo conceitual de negócio utilizado numa etapa posterior.

As interfaces e operações do sistema serão descobertas por meio de casos de uso, que serão utilizados na próxima etapa. Para cada passo de um caso que uso precisa considerar se há ou não responsabilidade do sistema que precisam ser modeladas para representá-las como uma ou mais operações de uma interface apropriada do sistema.

Após a identificação das interfaces de negócio e de sistema, um conjunto inicial de especificações de componentes é criado, tendo-se uma visão de como eles são integrados para formar a arquitetura inicial dos componentes.

A etapa de identificação dos componentes fornece um conjunto inicial de interfaces e componentes a serem utilizados na aplicação. As Atividades desta etapa são mostradas na Figura 1.6.

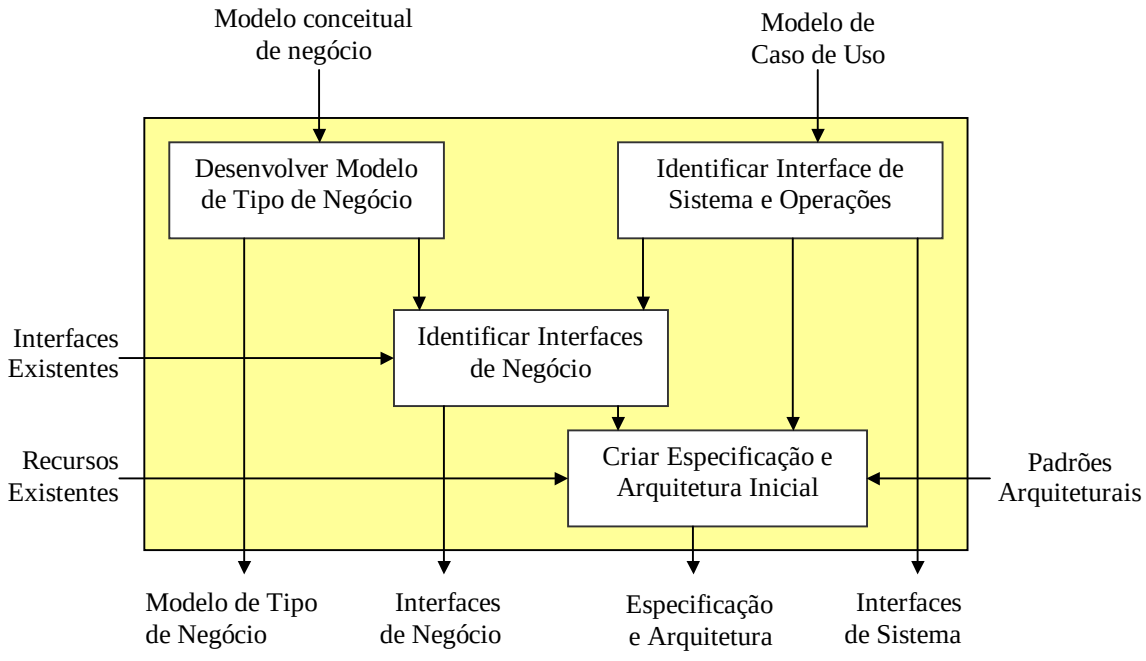


Figura 1.6 – Etapa de Identificação dos Componentes (Cheesman e Daniels, 2001).

1.3.4.4 Interação dos Componentes

Na etapa de interação dos componentes, decide-se como os componentes vão interagir para implementar os requisitos do sistema. Cada operação do sistema é modelado como o uso dos diagramas de interação como o da UML, para descobrir operações nas interfaces de negócio (Cheesman e Daniels, 2001).

Após a descoberta das operações é necessário refinar a responsabilidade das interfaces e quebrar as dependências entre os componentes. Para representar a especificação das interfaces é utilizado o estereótipo de classe <<interface type>> (Cheesman e Daniels, 2001).

Segundo Nascimento (2005) para atingir o objetivo desta fase, alguns passos devem ser seguidos, podendo ser vistas na Figura 1.7 na ordem que foram descritos:

- **Desenvolver modelos de interação para cada operação do sistema**: todos os casos de uso são analisados e são verificados quais passos precisam de uma operação correspondente no sistema. Essas operações descobertas podem ser auxiliadas pela UML, utilizando diagramas de colaboração.
- **Descobrir operações das interfaces do sistema e suas assinaturas**: a partir de diagramas de colaboração, é possível identificar as operações necessárias, e com isso, especificar as assinaturas de cada operação, mas não necessariamente todas, podendo algumas passar por outros refinamentos.
- **Refinar responsabilidades**: uma vez encontradas as operações das interfaces e suas assinaturas, as responsabilidades dos tipos de negócio podem, não obrigatoriamente, serem refinadas.
- **Definir restrições necessárias**: esse é um passo importante para definir as regras de negócio do sistema. São ressaltadas as restrições que levam a definição de pré e pós-condições das operações do sistema na próxima fase de especificação do componente.

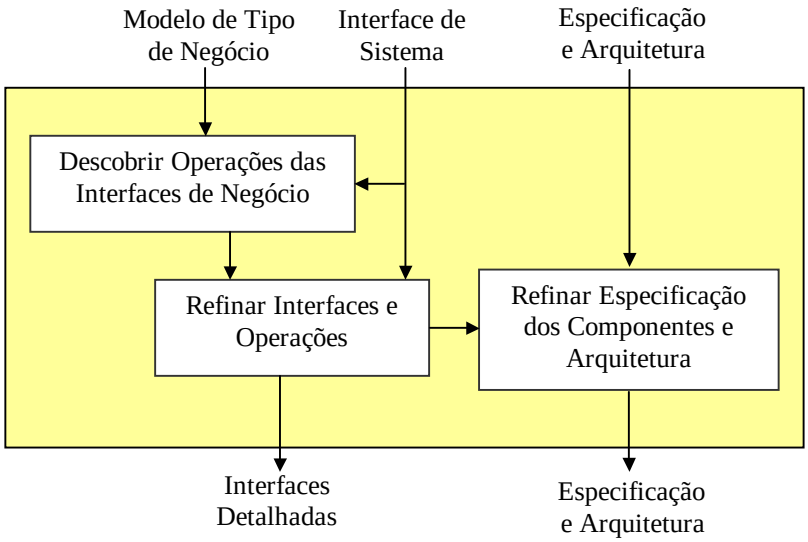


Figura 1.7 – Etapa de Interação de Componentes (Cheesman e Daniels, 2001).

1.3.4.5 Especificação dos Componentes

O objetivo geral desta etapa é especificar os contratos de uso que é definido pela especificação de interface e o contrato de realização dos componentes e interfaces que é de especificação do componente (Cheesman e Daniels, 2001).

Segundo Nascimento (2005) o contrato de uso pode ser entendido como um contrato entre a interface de um componente e os outros objetos que a utilizam, representado pelo estereótipo <<interface type>>. Um contrato de realização entendido como um contrato entre a interface do componente e sua realização ou implementação, representado pelo estereótipo <<comp spec>>.

Deve-se fazer a especificação detalhada das interfaces de negócio e de sistema, por meio das interfaces pode-se saber como gerenciar as dependências entre componentes. O primeiro passo e descrever o estado dos objetos componentes e assim especificar pré e pós-condições para cada operação. Cada interface deve possuir um modelo de informação que especifica as operações das interfaces, sendo derivado do modelo de tipos de negócios e mostra os possíveis estados de um objeto componente e descreve as mudanças de estado dos

objetos componentes causados pelas operações. O estereótipo de classe <<data type>> é utilizado para representar tipos de dados estruturados como classes da UML.

A especificação dos componentes é necessária estabelecer a quais interfaces sua realização apóia. Um diagrama de especificação de componentes mostra a dependência de interfaces de objetos componentes e as interfaces oferecidas.

Por fim, uma análise das restrições entre interfaces deve ser feita, tratando-se dos relacionamentos entre os modelos de informação de interface, de como as interfaces oferecidas se relacionam entre si e com as interfaces usadas.

As atividades desta etapa podem ser vista na Figura 1.8.

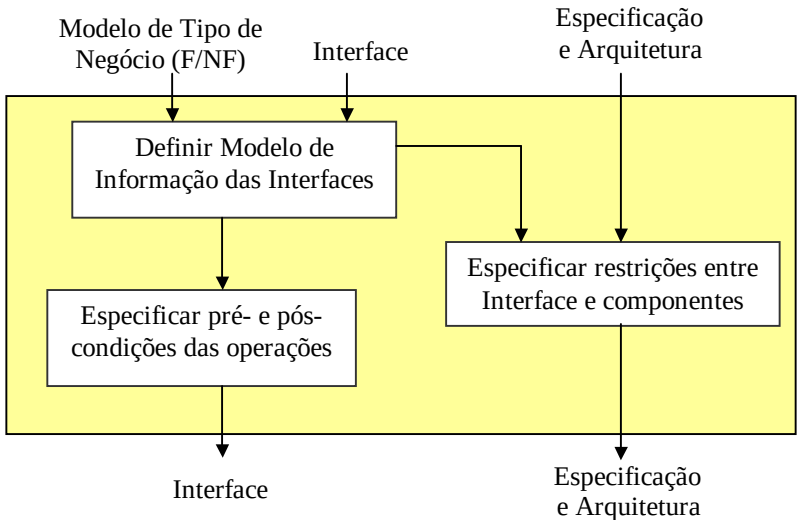


Figura 1.8 – Etapa de Especificação de Componentes (Cheesman e Daniels, 2001).

1.4 Padrões e Linguagens de Padrões

No início da década 90, desenvolvedores de software perceberam a existência de diversas soluções para problemas parecidos e fomentaram a iniciativa de organizá-las em padrões de software e divulgá-las para outras pessoas com a finalidade de poupar tempo e esforço no desenvolvimento de software.

A idéia de padrões de software foi inspirada na área de arquitetura, mais especificamente por Alexander *et al.*, (1977, **apud** CAGNIN), para documentar conhecimento

relacionado à construção de casas e prédios, que diz: “Cada padrão descreve um problema que ocorre diversas vezes em nosso ambiente e então descreve o núcleo da solução para esse problema, de forma que você possa utilizar essa solução milhares de vezes sem usá-la do mesmo modo duas vezes”.

De maneira análoga aos padrões de Alexander *et al*., (1977, **apud** BRAGA), padrões de software descrevem soluções para problemas que ocorrem com freqüência no desenvolvimento de software, com o objetivo de captar a experiência adquirida pelos desenvolvedores durante anos de prática profissional.

Vários autores iniciaram a criação de um catálogo de padrões de projeto, que foram publicados no livro de Gamma *et* al., (1995), que segundo ele com o emprego de padrões espera-se diminuir os esforços e o tempo de implementação, produzir aplicações flexíveis, reusáveis e mais confiáveis, já que as soluções foram testadas em outras aplicações.

Um padrão apresenta a essência de uma solução para um problema recorrente, que significa que cada padrão identifica um problema e apresenta uma solução bem sucedida para ele, sendo descrito para problemas que já ocorreram diversas vezes (FOWLER, 1997 **apud** FILHO e PEREIRA, 2002) e (BARROCA *et al*., 2005). Segundo Fowler (1997 **apud** FILHO e PEREIRA, 2002), a contribuição realmente importante de um padrão não está na solução fornecida, mas sim no raciocínio que está por trás desta solução.

De acordo com Pressman (1995), os padrões referem-se à comunicação de problemas e soluções. Em outras palavras, os padrões permitem documentar um problema conhecido recorrente e sua solução em um contexto específico e comunicar esse conhecimento para outras pessoas.

> “Cada padrão é uma regra de três partes, que expressa uma relação entre um certo contexto, um problema e uma solução” (PRESSMAN, 2002).

Um padrão é mais bem aplicado quando ele estiver inserido no contexto ao qual foi designado, ou seja, se existe um problema deve-se procurar entre os padrões já conhecidos qual a melhor solução (PRESSMAN, 2002).

Um padrão de análise descreve um conjunto de classes, possivelmente pertencentes a diferentes hierarquias de classes, podendo ser visto como uma forma de descrever subesquemas de projeto mais complexos, que ocorrem durante o desenvolvimento de software (FOWLER, 1997 **apud** FILHO e PEREIRA, 2002).

A importância do padrão de análise melhora o tempo de desenvolvimento de novos softwares, aumenta a qualidade e a produtividade no desenvolvimento de software, e principalmente, facilita o entendimento e a compreensão no nível de análise (FILHO e PEREIRA, 2002). Isso não é diferente no contexto de desenvolvimento baseado em componentes.

Uma linguagem de padrões é uma coleção estruturada de padrões que se apóiam uns nos outros para transformar requisitos e restrições numa arquitetura (COPLIEN, 1998).

Como já mencionado, no contexto de DBC, os padrões de software caracterizam como um importante meio de documentação para o desenvolvedor do componente, que pode encontrar nos padrões um meio para descrever os seus componentes, desde o nível arquitetural até o nível de análise (BARROCA *et* al., 2005). Neste trabalho, isso será explorado para documentar componentes existentes.

1.4.1 A Linguagem de Padrão GRN

Embora tenha surgido nos últimos anos uma variedade de linguagens de padrões, são poucas destinadas ao contexto de negócios. Braga *et* al. (1999), propuseram uma linguagem de padrões para Gestão de Negócio.

A linguagem para Gestão de Recursos de Negócio (GRN) é composta por quinze padrões de análise, sendo alguns deles aplicações ou extensões de padrões existentes na literatura. Entretanto, a linguagem de padrões GRN fornece uma abstração superior em relação de padrões existentes na literatura.

A linguagem de padrões auxilia aos engenheiros de software menos experientes, oferecendo informação suficiente para tratar o desenvolvimento de novos sistemas no domínio de Gestão de Recursos de Negocio, juntamente com soluções alternativas, quando necessário.

GRN possui um domínio especifico e bem definido, nas quais seja necessário registrar transações de aluguel, comercialização ou manutenção de recursos de negócio. Exemplificando, o aluguel de recursos enfoca principalmente a utilização temporária de um bem ou serviço, por exemplo, como uma fita de vídeo, a locação de um carro. A comercialização de recursos enfoca na transferência de propriedade de um bem, como por exemplo, uma venda ou leilão de produto. A manutenção de recursos enfoca o reparo ou conservação de um determinado bem, utilizando mão-de-obra e peças para execução, por exemplo, de uma oficina.

Na Figura 1.9, extraída de Braga *et al.*, (1999), exemplifica a GRN, mostrando os relacionamentos e a dependência entre os padrões existentes, bem como a ordem na qual eles são geralmente aplicados.

Essa linguagem possui três padrões principais: LOCAR O RECURSO (4), COMERCIALIZAR O RECURSO (6) E MANTER O RECURSO (9). O uso desses padrões não é mutuamente exclusivo, uma vez que existem sistemas nos quais eles podem ser usados em conjuntos, por exemplo, em uma oficina mecânica de veículos que também compra, vende e conserta seus próprios carros.

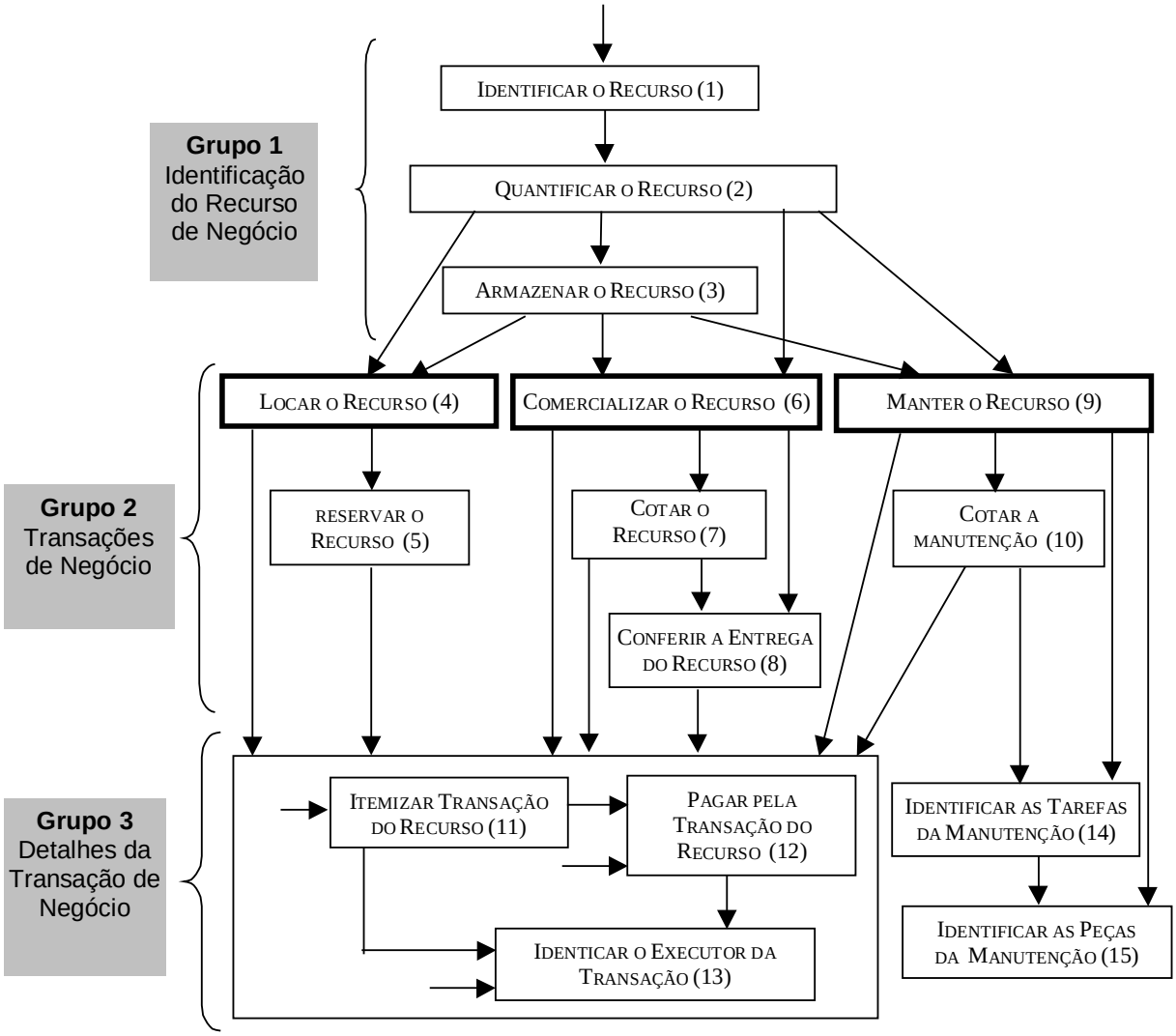


Figura 1.9 – Relacionamento da linguagem de padrões GRN (Braga et al., 1999).

Conforme mostrado na Figura 2.9, os padrões estão agrupados de acordo com seu propósito, formando três grupos. No primeiro grupo (Identificação de Recurso de Negócio) possui três padrões: (1) (2) (3), que dizem respeito à identificação e possível qualificação, quantificação e armazenagem dos recursos gerenciados pelo negocio. No segundo grupo (Transações de Negócio) possui sete padrões: (4) (5) (6) (7) (8) (9) (10) que estão relacionados à manipulação dos recursos de negocio pela aplicação. No terceiro grupo (Detalhes da Transação de Negócio) contém cinco padrões: (11) (12) (13) (14) (15) que tratam os detalhes das transações efetuadas com o recurso. Eles são aplicados a quaisquer das

transações do grupo dois. Os dois últimos padrões (14) e (15) são aplicáveis às transações contidas nos padrões (9) e (10).

A estrutura utilizada para expressar os padrões é representada em UML (Larman, 1998 **apud** Aragón (2004)). Para cada padrão é incluído um exemplo de sua instanciação, na qual novos atributos podem ser adicionados às classes, de acordo com a aplicação específica, tornando a GRN flexível.

1.4.2 Outras Linguagens de Padrões de Análise

A Linguagem de Padrões LV (Linguagem de Padrões para Leilões Virtuais) proposta por Ré (2002), aplica-se ao desenvolvimento de sistemas para gestão de vendas por intermédio de leilões virtuais. Ela é constituída de 10 padrões e podem ser divididos em duas categorias principais. A primeira categoria – Padrões Requeridos – representa requisitos essenciais para que um recurso possa ser leiloado. A segunda categoria – Padrões Opcionais – possui padrões desejáveis mas não são obrigatoriamente necessários.

Pode ser considerada como uma extensão da GRN, pois trata de uma transação (leilão) não coberta pela GRN.

Outra linguagem de padrões foi proposta por Pazin (2004), denominada SiGCli (Sistemas de Gerenciamento de Clínicas). Essa linguagem de padrões foi elaborada para apoiar a análise de Sistemas de Gestão de Clínicas de Reabilitação.

A SiGCli trata de registrar e controlar as vendas de produtos e serviços da clínica para seus pacientes. Nesse contexto, em uma clínica que comercializam produtos e serviços, os serviços são atividades realizadas durante os tratamentos dos pacientes e os produtos são quaisquer bens que sofrem uma troca de propriedade, ou seja, pertenciam à clínica e passam a pertencer aos pacientes que os adquiriram.

Os padrões da GRN não cobrem totalmente algumas das principais funcionalidades do domínio das clínicas de reabilitação, como por exemplo, o acompanhamento detalhado do tratamento dos pacientes. Apesar dessa linguagem de padrões ter nome diferente, ela difere na essência da GRN. Assim, a SiGCli pode ser considerada como uma extensão da GRN.

1.5 Considerações Finais

Este capítulo apresentou os conceitos necessários para o entendimento deste trabalho. Foi exposta a relevância sobre o reúso de software, componentes de software foram definidos e foi apresentada a ausência da documentação desses no nível de análise. Com isso observou-se a motivação deste trabalho.

Foram estudados também os padrões e linguagens padrões de análise, mais especificamente, a GRN; já que são também importantes para apoiar a documentação de componentes. Além disso, foram estudados também processos de desenvolvimento baseados em componentes, como o *Rational Unified Process*, Catalysis e UML *Components*, a fim de observar como a documentação de componentes é tratada por eles.

2 DIRETRIZES PARA DOCUMENTAÇÃO DE COMPONENTES DE SOFTWARE EXISTENTES (DDCS)

2.1 Considerações Iniciais

Neste capítulo são apresentadas as diretrizes para documentação de componentes de software existentes. As diretrizes utilizam o método UML *Components* para que os componentes sejam identificados, especificados e documentados, a partir dos requisitos dos componentes, utilizando a modelagem UML como notação. Além do método UML *Components* (CHEESMAN e DANIELS, 2001), é utilizada uma linguagem de padrões de análise, a GRN (BRAGA *et al*., 1999), que trata Gestão de Recursos de Negócio, para apoiar a recuperar os requisitos dos componentes como o auxílio de padrões de análise.

Para ilustrar a apresentação das diretrizes para documentação é apresentado um estudo de caso de um sistema de reservas de hotel baseado em componente. Este sistema já está desenvolvido e foi utilizado apenas como um estudo de caso.

O Capítulo está organizado da seguinte forma: Na Seção 2.2 apresenta-se a visão geral do processo proposto detalhando as atividades das diretrizes. Por fim, a consideração final do capítulo é apresentada na Seção 2.3.

2.2 Visão Geral do Processo

Na Figura 2.1 são ilustrados os três passos que compõem as diretrizes para documentação de componentes de software. O primeiro passo tem por finalidade conseguir identificar os componentes existentes no sistema, fazendo sua representação como pacote da UML.

O segundo passo recebe como entrada o componente modelado como pacotes da UML para poder recuperar requisitos de análise de cada componente, tendo o apoio a linguagem de

padrões de análise (GRN). O resultado final desse passo é obter o documento de requisitos, modelo de projeto e de análise e o modelo de caso de uso. Nesse passo também podem ser identificados novos componentes de software que não foram identificados anteriormente, voltando para o passo 1.

O terceiro passo pode ser considerado o foco principal desse trabalho por tratar as especificações dos componentes, com o auxílio da técnica UML *Components*. Um ponto fundamental quando se fala de componentes, refere-se as interfaces providas e requeridas que serão modelados nesse passo. O resultado final desse passo é obter cada componente de software modelado, gerando sua especificação.

No final dessas atividades do DDCS, espera-se obter a documentação de análise completa dos componentes de software, para facilitar o entendimento e análise do componente, a fim de reutilizar esses componentes em outros sistemas.

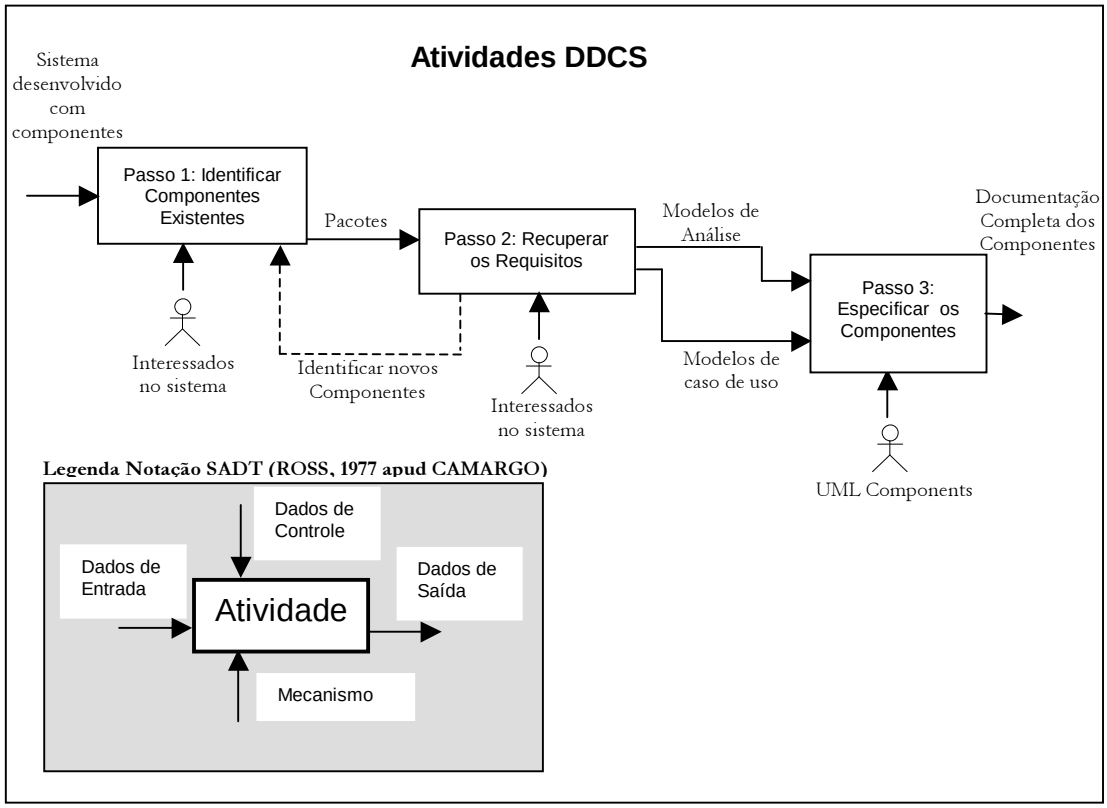


Figura 2.1 – Processo formal do DDCS.

Cada passo do DDCS proposto é apresentada em detalhes nas seções a seguir.

2.2.1 Passo 1: Identificar Componentes Existentes

Esse passo tem como objetivo identificar todos os componentes de software existentes. Para iniciar esse passo é necessário ter o sistema baseado em componentes e possuir o código fonte e o sistema em operação. Esse sistema deve lidar com o domínio de Gestão de Recurso de Negócio em função do segundo passo ter o apoio de padrões de análise para esse domínio.

No DDCS, para identificar os componentes de software do sistema é necessário identificar todas as classes do sistema. As classes são identificadas inspecionando o código fonte e executando suas funcionalidades. Como podem existir classes cujo único objetivo é auxiliar em alguma funcionalidade extra, essas podem ser desconsideradas.

Depois de identificadas todas as classes pertinentes ao sistema, o conjunto de classes relacionadas que executa uma determinada função do sistema são identificados como componentes de software existentes no sistema.

Como já comentado anteriormente, um sistema de reserva de uma rede de hotéis foi utilizado como estudo de caso. O sistema já estava modularizado com componentes e funcionando adequadamente.

Com a realização deste passo, foram identificados cinco componentes, os quais foram nomeados de `GesHotel`, `GesCliente`, `GesEmpresa`, `GesFaturamento` e `GesREeserva`. Os nomes foram dados conforme a funcionalidade das classes. Por exemplo, a identificação do componente `GesHotel` foi feita mediante a identificação das classes: `hotel`, `acomodação`, `tipoAcomodação`, `reservas`, `estadas` e uma interface `IGesHotel`. O componente `GesReserva` é formado pela classe reserva com as interfaces `IReserva`, `IAlterarReserva`, `IFazerReserva`, `INaoComparecer`, `IOcuparReserva`. A existência da classe `cliente` permitiu a identificação do componente `GesCliente`. O componente `GesFaturamento` é formado pelas classes `gesFaturamento`, `lançamento`, `pagamento`,`conta` e a interface `IFaturamento`.

O último componente identificado foi o `GesEmpresa` formado pelas classes `cliente`, `empresa`, `gesEmpresa` e a interface `IGesEmpresa`.

No Quadro 2.1 são ilustrados trechos de código existentes do estudo de caso do sistema de reserva, inspecionando a classe `Hotel` foram identificado o relacionamento com os tipos `Acomodações` `tiposAcomodação`, `reservas` e `estada`.

Quadro 2.1 – Classe `Hotel`.

```
Package GesHotel;

      class Hotel
      {

            private int hotelID;
            private String nomeHotel;
            private Vector acomodacoes;
            private Vector tiposAcomodacao;
            private Vector reservas = new Vector();
            private Vector estadas = new Vector();

            public Hotel()
            {
                  hotelID=0;
                  nomeHotel="";

                  acomodacoes = new Vector();
                  tiposAcomodacao = new Vector();
                  reservas = new Vector();
            }
      . . .
```

Cada tipo identificado anteriormente representa uma classe na qual possuem atríbutos especificos. No Quadro 2.2 é apresentado a classe `Acomodação`.

Quadro 2.2 – Classe `Acomodação`.

```
package GesHotel;

      class Acomodacao
      {
            private int hotelID;
            private int numero;
            private String tipo;
            private boolean status;
      . . .
```

Depois que os componentes foram identificados, eles foram modelados como pacotes UML, como mostrada na Figura 2.2.

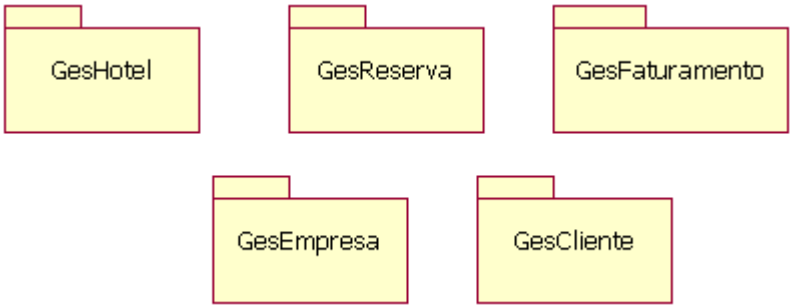


Figura 2.2 –Componentes identificados no estudo de caso.

Depois de identificado os componentes, o próximo passo é recuperar os requisitos do componente de software.

2.2.2 Passo 2: Recuperar os Requisitos

Esse passo tem como objetivo recuperar os requisitos do sistema, elaborar o documento de requisitos, elaborar o modelo de projeto e de análise e o modelo de casos de uso.

Este passo possui as seguintes atividades: 1) Recuperar o documento de requisitos do sistema; 2) Elaborar modelo de projeto; 3) Elaborar modelo de análise; 4) Identificar os casos de uso funcionais; 5) Elaborar modelo de casos de uso funcionais; 6) Descrever casos de uso funcionais.

Na Figura 2.3 são mostradas as atividades deste passo.

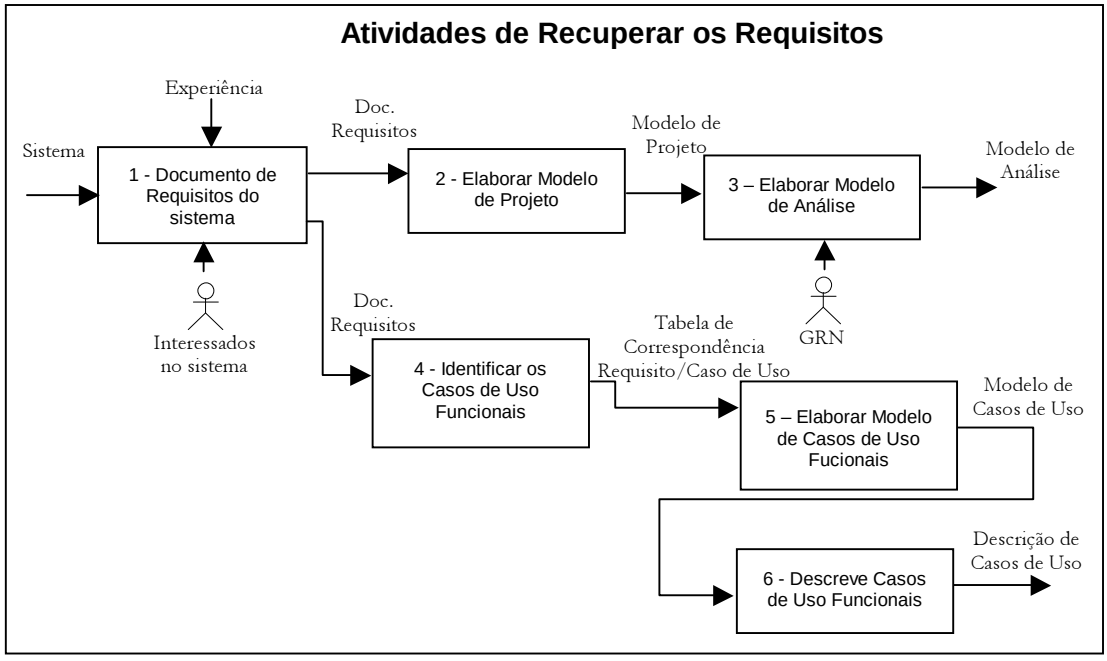


Figura 2.3 – Atividades de recuperar os requisitos.

Na primeira atividade deve ser recuperado o documento de requisito do sistema. Para recuperar o documento de requisito deve-se analisar e identificar os interesses dos componentes de software, executando o sistema e se possível elaborar um questionário que deve ser respondido pelo desenvolvedor do sistema. Por exemplo um sistema que gerencia reservas de hotéis possui funcionaldade para efetuar reserva, cadastrar clientes e etc. O documento de requisitos obtido a partir do estudo de caso é descrito no Apêndice A deste trabalho.

Na segunda atividade deve-se elaborar o modelo de projeto a partir do documento de requisitos. Essa atividade tem como objetivo entender o funcionamento do sistema e fazer sua representação do projeto arquitetural de baixo nível por meio do código fonte para modelo de projeto, conforme a Figura 2.4.

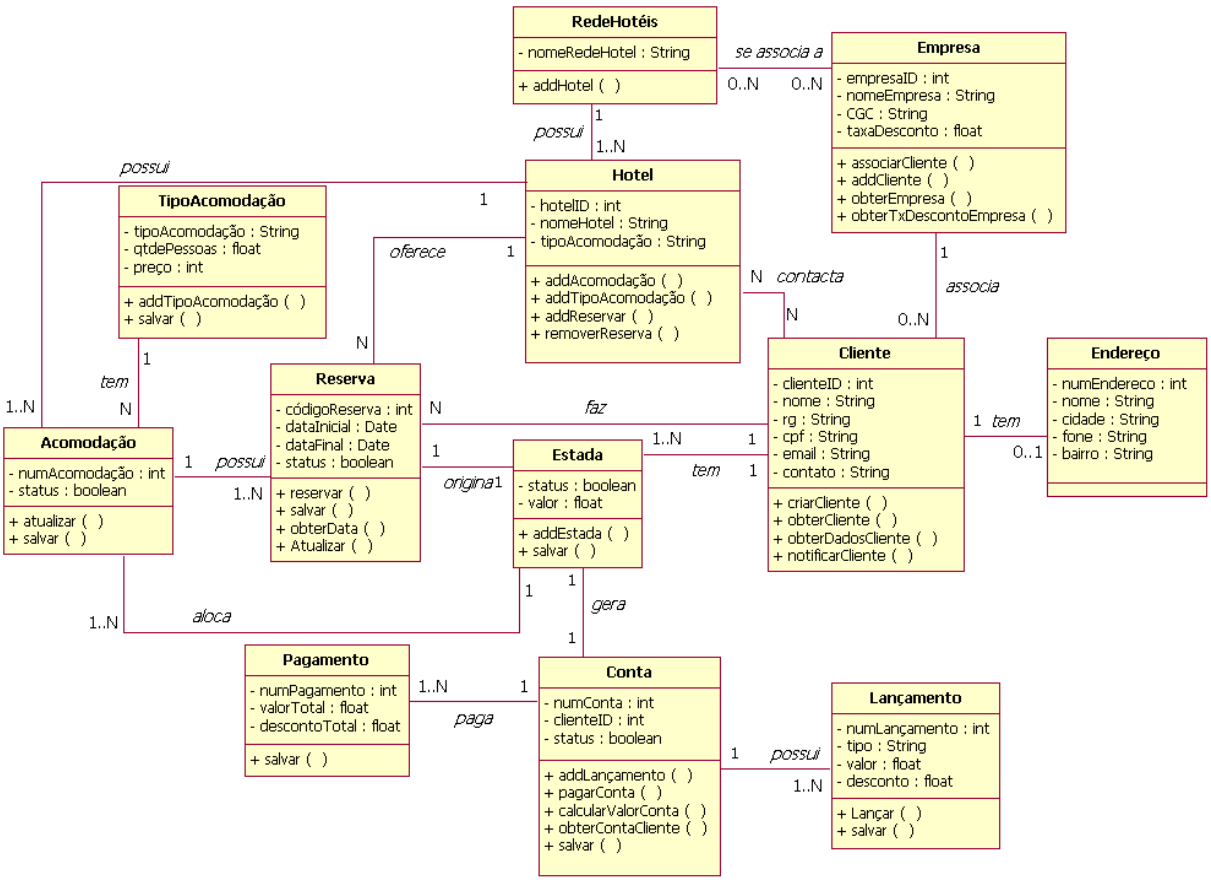


Figura 2.4 – Modelo de projeto do sistema de reserva de hotel.

Na terceira atividade é elaborado o modelo de análise com o auxilio dos padrões de análise da GRN, fazendo o refinamento sobre o modelo de projeto. O modelo elaborado deve manter os mesmos relacionamentos identificados no modelo de projeto, fazendo o refinamento das classes, identificando apenas os atributos. O modelo pertencente ao nível de análise, deve o auxilio dos padrões da GRN, por tratar do mesmo domínio de negócio do componente escolhido para gerar toda a documentação.

Partindo do modelo de projeto foram identificados as classes que compõem os padrões da GRN. O Padrão Identificar o Recurso representam os recursos de negócio envolvido nas transações processadas pelo sistema, sendo assim a classe identificada para cobrir esse padrão foram as classes Reserva e Acomodação.

O Padrão Quantificar o Recurso, são identificado nos sistemas, aquele que são necessários ter controle sobre instâncias específicas do recurso. Para sistema de hotel, a classe Acomodação possui um relacionamento com a classe TipodeAcomodação, satisfazendo o padrão Quantificar o Recurso.

O Padrão Locar o Recurso lida com aluguel de recursos, para o sistema de hotel a classe Estada lida com as diárias efetuadas nas reservas, juntamente com as classes Hotel e Reserva, satisfazendo o padrão Locar o Recurso.

O Padrão Reservar o Recurso foram identificado de imediato no modelo de projeto pelas classes Reservar, Hotel e a Acomodação, por tratar os recursos que permitem que os mesmos sejam reservados. O último padrão que ajudou a identificar a classe no modelo de análise, refere-se ao Padrão Pagar pela Transação do Recurso, identificados pelas classes Estada, Conta, Pagamento e Lançamento.

Dentre os quinze padrões da GRN foram utilizados apenas cinco para o estudo de caso: Identificar o Recurso (1), Quantificar o Recurso (2), Locar o Recurso (3), Reservar o Recurso (4) e o Pagar pela Transação do Recurso (5).

Na Figura 2.5 é mostrado o modelo de análise do sistema de reserva de hotel.

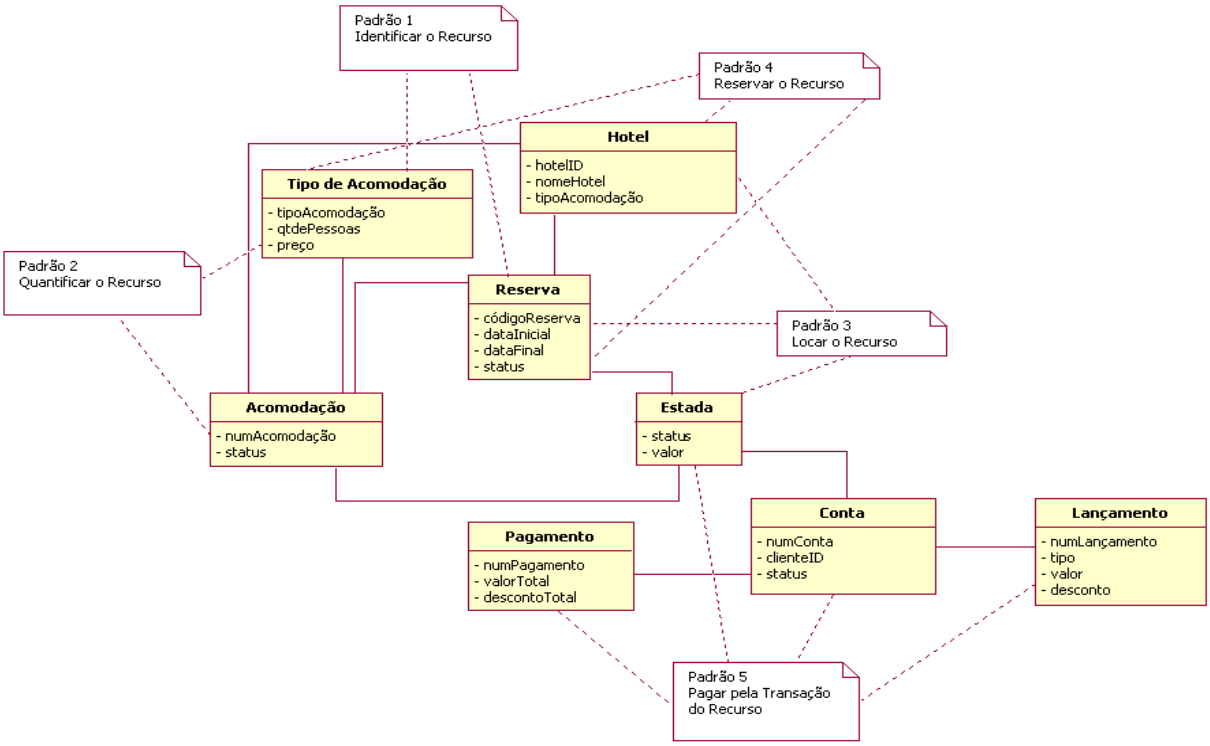


Figura 2.5 – Modelo de análise.

Na quarta atividade deve-se representar os requisitos funcionais do sistema em um diagrama de casos de uso. Para executar essa tarefa e necessário ter o documento de requisitos. Na Figura 2.6 relaciona o documento de requisitos com os casos de uso para verificar se todos os requisitos foram coberto com os casos de uso identificados. O requisito coberto encontra-se sua descrição no documento de requisito que segue no anexo A.

Requisito Coberto	Caso de Uso
Requisito 1	Incluir/Alterar/Remover Clientes
Requisito 2	Incluir/Alterar/ Remover Empresas
Requisito 3	Incluir/Alterar/ Remover Hotel
Requisito 4	Incluir/Alterar/ Remover Acomodação
Requisito 5	Incluir/Alterar/ Remover Tipo de Acomodação
Requisito 6	Fazer Reserva
Requisito 7	Ocupar Reserva
Requisito 8	Gerar Fatura
Requisito 9	Cancelar/Alterar Reserva
Requisito 10	Listar Clientes
Requisito 11	Listar Reservas
Requisito 12	Listar Faturamento
Requisito 13	Consultar se uma acomodação está disponível no momento
Requisito 14	Gerar nota de saída para cliente

Figura 2.6 – Casos de Uso identificados no documento de requisitos.

Na quinta atividade deve-se construir o diagrama de casos de uso funcional do sistema. A construção do diagrama fornece a visão externa do sistema e suas interações com o mundo externo, representando o comportamento geral do sistema.

Na Figura 2.7 são mostrado alguns casos de uso do sistema de reserva de hotel. O caso de uso cadastrar cliente o operador de reserva envia os dados do cliente para o sistema. O caso de uso efetuar uma reserva, o cliente solicita ao operador reservar a acomodação de um hotel. O caso de uso ocupar reserva tem o objetivo de registrar o início da reserva efetuada. O caso de uso alterar reserva tem o objetivo de alterar detalhes de uma reserva. O caso de uso verificar disponibilidade têm o objetivo de verificar se acomodação está disponível em um período. O caso de uso gerar dados para faturamento têm o objetivo de efetuar a fatura para o cliente.

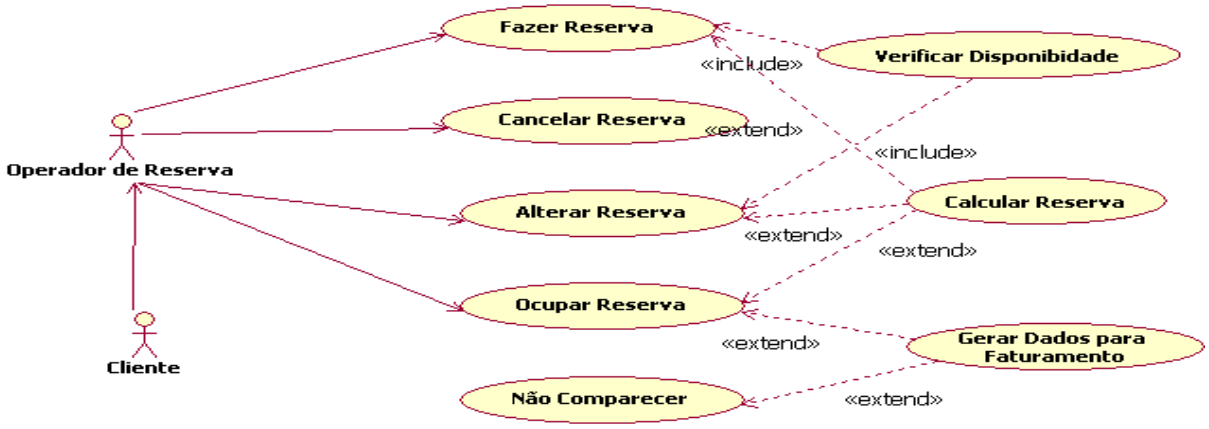


Figura 2.7 – Modelo de casos de uso funcional.

Na sexta atividade descreve-se detalhes de cada caso de uso funcional, usando o modelo composto pelos seguintes campos: nome do caso de uso, atores que integram o caso de uso, objetivo do caso de uso, cenário de sucesso principal e vários fluxos alternativos.

É apresentado na Figura 2.8 a descrição do caso de uso Fazer Reserva.

Caso de Uso: Fazer Reseva **Ator:** Operador de Reserva **Objetivo:** Reservar tipos de acomodação de um hotel
Cenário de Sucesso Principal: 1. O Operador de Reservas solicita fazer uma reserva. 2. O Operador de Reservas escolhe, em ordem, o hotel, o tipo de acomodação, datas de início e fim da reserva. 3. O sistema Verificar disponibilidade de Acomodação. 4. O Operador de Reservas informa o nome do cliente e o Sistema realiza a busca dos dados do cliente. 5. O Sistema registra a reserva e aloca um código para a reserva efetuada. 6. O Sistema mostra o código da reserva para o Operador da Reserva. Fluxo Alternativo: 3. Acomodação não disponível a. O Sistema oferece alternativas de hotéis. 4. O Cliente não está cadastrado no sistema. a. O Operador de Reservas fornece ao sistema o nome e dados para contato do cliente.

Figura 2.8 – Descrição do caso de uso fazer reserva.

Depois de recuperado os requisitos o próximo passo são abordados uma descrição sobre Interfaces de Sistema, Interfaces de Negócio, baseadas no estudo de caso realizado neste trabalho e a Especificação da Arquitetura do Componente.

2.2.3 Passo 3: Especificar os Componentes

Esse passo tem como objetivo especificar arquitetura dos componentes identificados no sistema, principalmente identificar as interfaces de negócio e sistema dos componentes.

Para apoiar esse passo são utilizados técnicas do método UML *Components* de Cheesman e Daniels (2001). Conforme já foi explicado em detalhes na Seção 1.3.4 respectivamente.

Este passo possui as seguintes atividades: 1) Identificar Interfaces dos Componentes; 2) Identificar Interações dos Componentes; 3) Especificar Interfaces dos Componentes.

Na Figura 2.9 são mostradas as atividades deste passo.

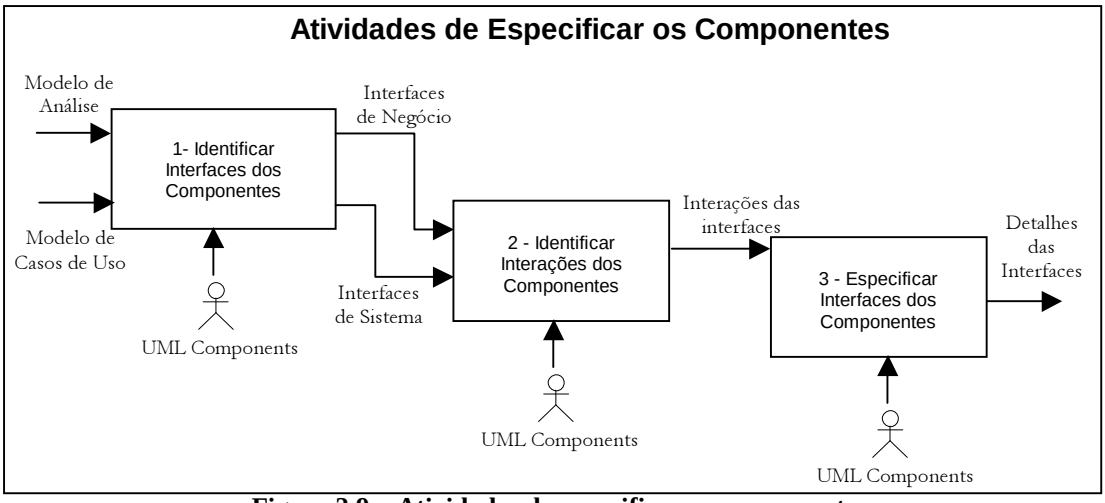


Figura 2.9 – Atividades de especificar os componentes.

Na primeira atividade tem o objetivo de identificar as interfaces de sistema e de negócio. Para identificar as interfaces de sistema é utilizado o modelo de casos de uso, cada caso de uso identifica-se uma interface de sistema. Para identificar as interfaces de negócio é construído o modelo de tipo de negócio que define um identificador de negócio para cada componente. Para construir o modelo tipo de negócio são identificados os tipos básicos do sistema. Conforme mostrado na Figura 2.10.

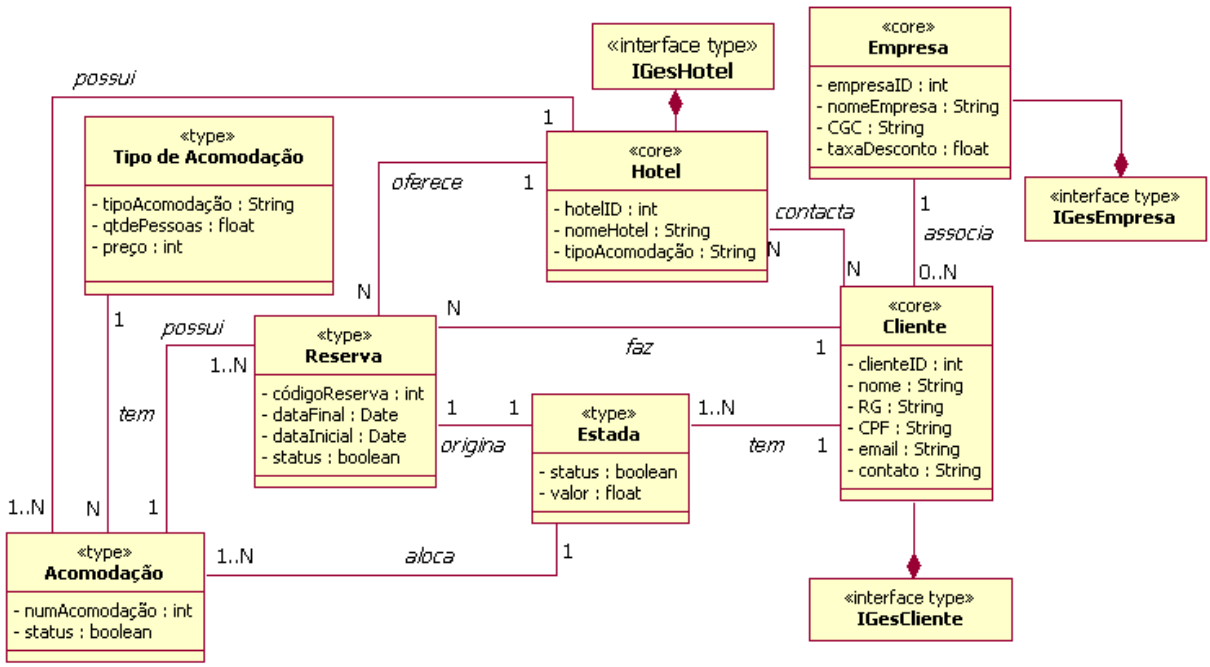


Figura 2.10 – Modelo de tipo de negócio.

Com a arquitetura dos componentes desenvolvida neste trabalho, foram identificado as interfaces de sistema e de negócio. Para os componentes `GesHotel`, `GesCliente`, `GesEmpresa` e `GesFaturamento` foram identificado as interfaces de negócio para cada componente. Enquanto para o componente `GesReserva` foram identificado as interfaces do sistema associada a um único componente, que se comunica com todas as interfaces dos outros componentes, conforme a Figura 2.11.

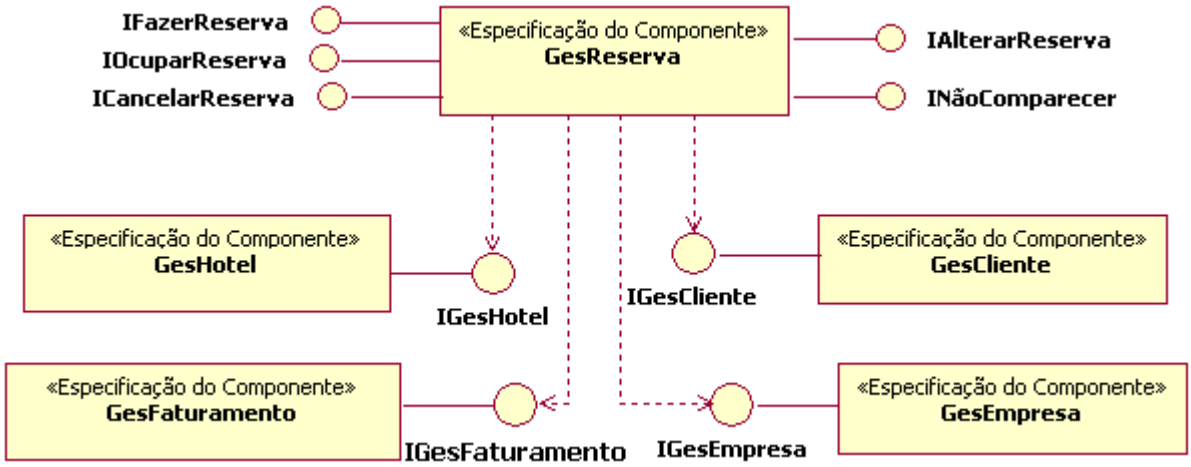


Figura 2.11 – Especificação da arquitetura dos componentes.

No Quadro 2.3 são mostrados as interfaces de sistema do componente GesReserva identificado no código-fonte.

Quadro 2.3 – Interface `GesHotel`.

```
package GesReserva;

      public class GesReserva implements IFazerReserva, IOcuparReserva,
      ICancelarReserva, IAlterarReserva, INaoComparecer
      {
      . . .
```

No segundo atividade tem o objetivo de mostrar como os componentes devem interagir entre si, para isso é realizado a interação de cada operação identificada na interface do sistema. Com isso é possível identificar as operações de negócio necessárias para que a operação de sistema possa ser completada.

O processo de interação entre componente do sistema e de negócio, é feito modelando as operações das interfaces de sistema com o diagrama de colaboração da UML. Na Figura 2.12 é apresentada a interação entre as interfaces para a realização da operação fazerReserva. No Item um é chamado a operação fazerReserva da interface `IFazerReserva` do componente `GesReserva`, no Item 1.1 é invocado a operação SelecionaHotel para a interface `IGesHotel`. No Item 1.2 é invocado a operação obterCliente para a interface `IGesCliente`. No item 1.3 é invocado a operação verificarDisponibilidade para mesma interface. O último item é invocado a operação fazerReserva para interface `IGesHotel`.

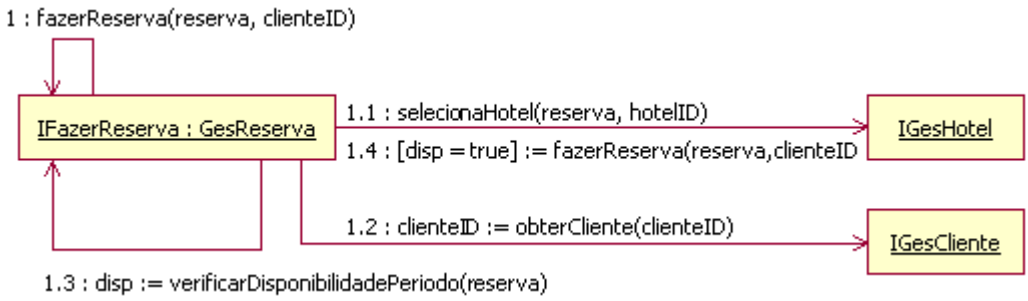


Figura 2.12 – Operação fazerReserva da interface IFazerReserva.

Na terceira atividade deve-se detalhar as operações das interfaces de negócio e de sistema. As operações da cada interface foram descobertas com as interações entre as interfaces. Na Figura 2.13 são mostrados detalhes da interface de negócio `IGesHotel`.

<< interface >> **IGesHotel**
obterHoteis(): String[ ] obterNomeHotel(in cliID: int): String selecionaHotel(in nomeHotel:String) selecionaHotel(in hotelID: int) obterDetalheHotel(in nomeHotel: String): DetalhesHotel obterInfoTipoAcomodacao(in tipoAcomodacao: String): InfoTipoAcomodacao obterQtdeAcomodacoesTipo(in tipoAcomodacao: String): int obterQtdeAcomodacoes(): int obterQtdeReservasPeriodo(in dataIni:Date, in dataFim: Date): int obterQtdeReservasTipo(in dia:Date): int fazerReserva(in res:Detalheseserva, in cliID: int): String {codReserva} obterReserva(in codReserva: String): DetalhesReserva criarEstada (in codReserva:String): int cancelarReserva(in codReserva: String): boolean alterarReserva(in codReserva: String, in novaRes: DetalhesReserva): boolean obterReservasVencidas(in hotelID: int): String[ ]

Figura 2.13 – Interface IGesHotel detalhada.

Na Figura 2.14 é mostrada as operações da interface IFazerReserva do componente GesReserva, descrevendo todos os seus métodos definidos para a realização de suas funções. Esse mesmo processo deve ser seguido para especificar as demais interfaces do sistema.

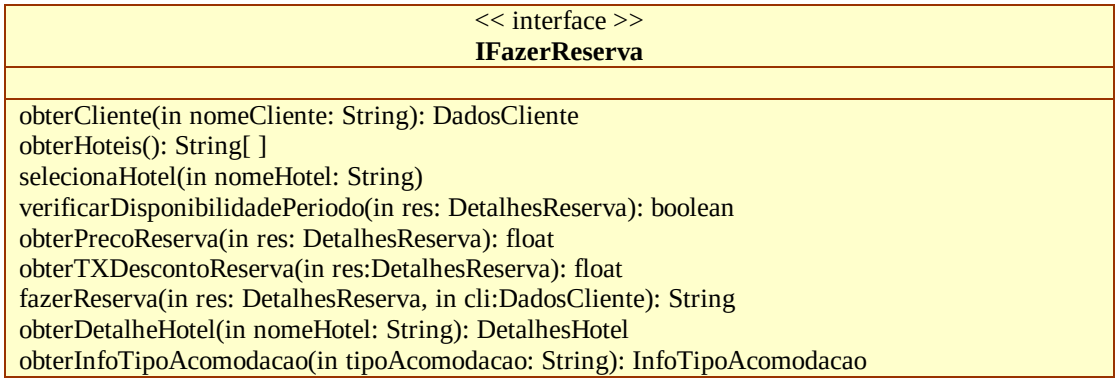


Figura 2.14 – Interface IFazerReseva detalhada.

Com os detalhes das interfaces de sistema e de negócio os componentes de software foi finalizado as diretrizes de documentação de componentes de software.

A documentação completa dos componentes obteve como resultado a modelagem dos componentes de software em pacotes UML visto no primeiro passo, já no segundo passo foram recuperado os requisitos do sistema, como por exemplo, documento de requisitos, modelo de projeto e de análise com o apoio dos padrões da GRN e o modelo de casos do uso. No terceiro passo como o auxilio do método UML *Components* foram tratado a especificação dos componentes partindo do modelo de casos de uso e do modelo de análise e também foi elaborado o modelo de tipo de negócio para identificar as interfaces de sistema e de negócio. Identificados as interfaces de cada componente, foram modelados as interações dos componentes afim de descobrir as operações das interfaces. Passa concluir a documentação dos componentes detalhes das interfaces são apresentados.

2.3 Considerações Finais

Neste capítulo, a diretrizes de documentação de componentes de software foi apresentado.

Para o reúso do componente, deve haver a documentação, pois abrange todas as características necessárias para o desenvolvedor.

O processo DDCS foi aplicado no estudo de caso do sistema de reserva de hotéis.

CONCLUSÕES

Neste Capítulo apresenta-se o resultado do trabalho realizado, destacando-se as contribuições obtidas para a área de desenvolvimento baseado em componentes. Apontando sugestões de trabalhos futuros, decorrente da pesquisa realizada nesta monografia, que podem dar continualidade a este trabalho, que poderão gerar outras contribuições.

O objetivo do processo é documentar componentes de software existentes, tendo como resultado o componentes de software documentado.

A principal contribuição deste trabalho para a área de desenvolvimento de software baseado em componente é as diretrizes DDCS proposto. As diretrizes possuem passos desde identificar, até especificação dos componentes. A principal característica do processo é utilizar o método UML *Components* (Cheesman e Daniels, 2001), para guiar todo o processo de documentação de componente.

A linguagem de padrões para gestão de recursos de negócio foi utilizado como apoio na elaboração do modelo de análise.

A documentação dos componentes desenvolvidos com objetivo de serem reutilizados em futuras aplicações baseados em componentes. Assim, o método UML *Components* utiliza a Linguagem UML para modelar todas as fases de desenvolvimento de sistemas baseados em componentes.

A limitação a ser considerada na diretriz DDCS exige que o sistema documentado seja baseado em componente.

Umas das dificuldades nesse processo foi encontrar componentes para ser documentados.

O trabalho realizado nesta monografia pode ser aperfeiçoado por meio de trabalhos futuros:

Primeiramente, seria de grande importância o desenvolvimento de uma ferramenta para gerar a documentação do componente de software.

Em segundo lugar, pode ser feita investigação de outras abordagens de reúso existente, como por exemplo, framewoks.

Em terceiro lugar, outros sistemas que foram baseados em componentes, só que não foram aplicados a GRN, verificar se vão ter padrões que auxilia na documentação.

Em quarto lugar, o trabalho poderia ser aplicado em outros domínios, realizando outro estudo de caso.

REFERÊNCIAS

ARAGÓN, C.R. **Processo de Desenvolvimento de uma Linha de Produtos para Sistemas de Gestão de Bibliotecas.** Dissertação (Mestrado em Ciência da Computação) – Departamento de Computação e Estatística. Fundação Universidade Federal de Mato Grosso do Sul, Campo Grande, 2004.

BARROCA, L.; GIMENES, I. M. S.; HUZITA, E. H. M. Desenvolvimento Baseado em Componentes. cap 1. **Conceitos Básicos.** In: (GIMENES e HUZITA, 2005), p. 1-26, 2005.

BARROCA, L.; GIMENES, I. M. S.; HUZITA, E. H. M. Desenvolvimento Baseado em Componentes. cap 2. **Desenvolvimento Baseado em Componentes.** In: (GIMENES e HUZITA, 2005), p. 27-56, 2005a.

BASILI, V. R.; ROMBACH, H. D**. Towards a comprehensive framework for reuse**: **A reuse enabling software evolution environment**. Maryland, University of Maryland, 1988.

BASS, L.; BUHMAN, C.; COMELLA-DORDA, S.; LONG, F.; SEACORD, R.; WALLNAU, K. **Volume I: Market Assessment of Component-Based Software Engineering**. Pittsburgh: Software Engineering Institute, May 2001 (Technical Note CMU/SEI-2001-TN-007).

BRAGA, R.T.V. **Um Processo para Construção e Instanciação de Frameworks baseados em uma Linguagem de Padrões para um Domínio Específico.** Tese de Doutorado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2003.

BRAGA, R. T. V.; GERMANO, F. S. R.; MASIERO, P. C. **A Pattern Language for Business Resource Management.** In: PLOP'1999, 6th Conference on Pattern Languages of Programs, p. 1-33, 1999.

BIANCHINI, S. L. **Aspecto de Automatização da Documentação de Processo**. Dissertação (Graduação em Ciência da Computação) – Departamento de Computação. Instituto de Ciências Matemáticas e de Computação. Universidade de São Paulo, São Carlos, 2004.

BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. **The Unified Modeling Language User Guide.** Addison-Wesley, 1998.

CAGNIN, M. I. **PARFAIT: uma contribuição para a reengenharia de software baseada em linguagens de padrões e frameworks.** Tese (Doutorado em Ciência da Computação) – Instituto de Ciências Matemáticas e de Computação. Universidade de São Paulo, São Carlos, 2005.

CAMARGO, V.V. **Frameworks Transversais: definição, classificações, arquitetura e utilização em um processo de desenvolvimento de software.** Tese (Doutorado em Ciência da Computação) - Instituto de Ciências Matemáticas e de Computação. Universidade de São Paulo, São Carlos, 2006.

CHEESMAN, J.; DANIELS, J. **UML Components, a simple process for specifying component-based software.** 1.ed. Reading: Addison-Wesley, 2001.

COOPER, J. **Reuse business implications.** In: Encyclopedia of software engineering with syntropy. 1.ed. Nova York: Prentice Hall, 1994.

COPLIEN, J. O. **The patterns handbook: Techniques, strategies, and applications, cap. Software Design Patterns:** Common Questions and Answers Cambridge University Press, p. 311-320, 1998.

D'SOUZA, D.; WILLS, A. **Objects, components, and frameworks: The catalysis approach.** 1.ed. Reading: Addison-Wesley, 1998.

FILHO, J. L.; PEREIRA. M. **Desenvolvimento de uma Ferramenta CASE para o Modelo UML-GeoFrame com suporte para Padrões de Análise.** In: IV Simpósio Brasileiro de Geoinformática, 2002, Belo Horizonte, 2002. p. 147-154.

FOWLER, M. **Analysis Patterns: Reusable Object Models**. Addison Wesley Longman, 1997.

FOWLER M; **UML Essencial.** 3ª ed. Porto Alegre: Bookman, 2005.

FREEMAN, P. **"Reusable software engineering concepts and research directions"**. In: Software reusabily (tutorial). Washington, D.C.; IEEE Computer Society Press, 1987.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design patterns elements of reusable of object-oriented software.** 2.ed. Addison-Wesley, 1995.

GIMENES, I. M. S.; HUZITA E. H. M. **Desenvolvimento Baseado em Componentes.** Rio de Janeiro: Ciência Moderna, 2005.

KRUCHTEN, P. **The Rational Unifiel Process: An Introduction**. Second ed. Addison-Wesley, 298 p., 2000.

LARMAN, C. **Applying UML and Patterns.** Prentice Hall, 1998.

NASCIMENTO, L. B. **Componentização de Software em JAVA™ 2 Micro Edition - Um *framework* para Desenvolvimento de Interface Gráfica para Dispositivos Móveis**. Dissertação (Graduação em Ciência da Computação) – Centro de Informática. Universidade Federal de Pernambuco, Recife, 2005.

PAZIN, A. **GAwCRe: Um Gerador de Aplicações baseadas na Web para o Domínio de Clínicas de Reabilitação.** Dissertação de mestrado, Departamento de Computação, Universidade Federal de São Carlos, São Carlos, 2004.

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

RÉ, R. **Um Processo para Construção de Frameworks a partir da Engenharia Reversa de Sistemas de Informação baseados na Web: Aplicação do Domínio dos Leilões Virtuais.** Dissertação de mestrado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2002.

ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, K. C. **Qualidade de Software: Teoria e prática.** New Jersey: Prentice Hall, 659 p, 2001.

ROSSI, A. C. **Representação de Componentes de Software na FARCSoft: Ferramenta de Apoio à Reutilização de Componentes de Software**. Dissertação (Mestrado em Engenharia) - Departamento de Engenharia de Computação e Sistemas Digitais. Escola Politécnica da Universidade de São Paulo, São Paulo, 2004.

ROSS, D. Structured Analysis: **A Language for Communicating Ideas.** IEEE Transaction on Software Engineering. In: (CAMARGO *et al*., 2006),v.3, n.1, 1977.

SANCHES, R. Qualidade de software: Teoria e prática. cap. **Processos do Ciclo de Vida: Processos de Apoio.** In: (ROCHA *et al*., 2001),p. 54–57, 659 p, 2001.

SOMMERVILLE, I. **Engenharia de Software**. 6ª ed. São Paulo: Addison Wesley, 2003.

WERNER, C. M. L.; BRAGA, R. M. M. Desenvolvimento Baseado em Componentes. cap 3. **A Engenharia de domínio e o Desenvolvimento Baseado em Componente.** In: (GIMENES e HUZITA, 2005), p. 57-104, 2005.

ZOU, Y.; KONTONGIANNIS, K. **Migration to object oriented platforms: A state transformation approach.** In: ICSM'2002, 18th International Conference on Software Maintenance, Montreal, Quebec, Canada, p. 530-539, 2002.

APÊNDICE A – Documento de Requisitos de Sistema de Reservas de uma Rede de Hotéis

A.1 Visão Geral do Sistema de Hotel

O Sistema de Hotel consiste basicamente do gerenciamento das reservas das acomodações de uma rede de hotel. O sistema a ser desenvolvido oferecerá a seus clientes, facilidades de reservas on-line ou por telefone a uma central de reserva. Um diferencial desse tipo de sistema é que por gerenciar uma rede de hotel ele possibilitará quando o hotel já estiver cheio, proporcionará alternativas de acomodações em hotéis conveniados com o sistema.
O hotel disponibilizará acomodações desde quartos, apartamentos e suítes do hotel para acomodar seus clientes durante sua estada.
O sistema de hotel tem por objetivo ser ágil para fazer a reserva, levando em média quatro minutos. Os dados dos clientes já cadastrados serão atualizados ao efetuar uma nova reserva.

A.2 Requisitos Funcionais

A.2.1 Lançamentos Diversos

1. O Sistema deve permitir a inclusão, alteração e remoção de clientes do hotel, contendo os seguintes atributos: nome, endereço, cidade, bairro, CEP, telefone, RG, CPF, e-mail e contato.

2. O Sistema deve permitir a inclusão, alteração e remoção de empresas que reservam com freqüência uma acomodação, contendo os seguintes atributos: nome, endereço, cidade, bairro, CEP, telefone, contato, e-mail, CGC e taxa de desconto.

3. O Sistema deve permitir a inclusão, alteração e remoção de hotéis, contendo os seguintes atributos: nome, endereço, telefone, acomodações, tipos de acomodações, reservas e estadas.

4. O Sistema deve permitir a inclusão, alteração e remoção acomodações do hotel, contendo os seguintes atributos: número da acomodação, descrição, tipo de acomodação e status da situação atual.

5. O Sistema deve permitir a inclusão, alteração e remoção de tipos de acomodações oferecidas pelo hotel, contendo os seguintes atributos: código do tipo de acomodação, descrição do tipo de acomodação, quantidade de pessoas e preço da diária.

6. O Sistema deve permitir a reserva de acomodações do hotel. Cada reserva deve conter os seguintes atributos: identificação do hotel, identificação do cliente principal (previamente cadastrado), tipo de acomodação desejada, data e hora da entrada do cliente, data e hora de saída do cliente. A reserva somente deve ser concretizada se houver vagas suficientes para atendê-la, caso contrário deverá ser mostrada uma mensagem alertando que não há disponibilidade de acomodações para o período indicado.

7. O Sistema deve permitir o processamento da entrada do cliente no hotel, contendo os seguintes atributos: data e hora da entrada do cliente, data e hora prevista para saída do cliente, identificação do cliente principal (previamente cadastrado), número da acomodação utilizada, valor da diária e identificação do funcionário responsável pelo recebimento do cliente. Se tiver sido feita a reserva prévia da acomodação, o sistema recupera automaticamente os dados da reserva durante a entrada do cliente.

8. O Sistema deve permitir o processamento da quitação de fatura do sistema de pagamento, contendo os seguintes atributos: número da conta, número do pagamento, valor total, total de descontos e opção de pagamento: à vista (em dinheiro, cheque ou cartão de crédito), ou cheque para 30 dias.

9. O Sistema deve permitir alteração ou cancelamento de reservas.

A.2.2 Impressão de Diversos Tipos de Relatórios e Consultas

10. O Sistema deve permitir a impressão de uma listagem dos clientes que estão no hotel no momento, contendo o nome do cliente principal, data de entrada, data prevista para saída, número da acomodação e o tipo da acomodação.

11. O Sistema deve permitir a impressão de uma listagem das reservas efetuadas em um determinado período, contendo o nome do cliente, data prevista para a entrada e saída, tipo de acomodação e telefone para contato.

12. O Sistema deve permitir a impressão de um relatório, contendo o faturamento do hotel no período: data inicial e data final, e um resumo das estadas pagas, com colunas dos dias, nome do cliente e valor total.

13. O Sistema deve permitir que consultas sejam realizadas na hora da reserva, para saber se acomodação está ocupada ou não.

14. O Sistema deve permitir a impressão de um comprovante de pagamento para o cliente, contendo os dados da reserva.

A.3 Requisitos Não Funcionais

A.3.1 Segurança do Sistema

17. O Sistema deve permitir a inclusão, alteração e remoção de funcionário do hotel, contendo os seguintes atributos: nome, endereço, cidade, estado, telefone, data de nascimento e cargo do funcionário.

18. O Sistema deve possuir meios de autentificação para cada funcionário registrado no sistema. Ex: gerente e funcionários do hotel.

19. O Sistema deve gerar logs das operações realizadas no sistema. Os logs deve conter data/hora, nome do funcionário e a operação executada.

20. O Sistema deve oferecer recursos para efetuar cópias dos arquivos do sistema.

A.3.2 Eficiência do Sistema

22. O Sistema deve responder as consultas on-line em segundos e toda reserva no sistema deve gastar na média 4 minutos.

23. O Sistema deve iniciar a impressão de relatórios solicitados pelo funcionário do hotel, no máximo 20 segundos após sua requisição.

A.3.3 Portabilidade do Sistema

24. O Sistema requer computadores com processamento no mínimo de 500 MHz, com 64 MB, espaço livre em disco de 100 MB, Windows 98 ou superior e uma máquina virtual instalada.

25. O Sistema deve ser capaz de armazenar os dados necessários para o gerenciamento do hotel em um banco de dados livre HSQLDB.

A.4 Glossário

A.1 Glossário

Glossário	**Termo Descrição**
Rede de Hotéis	Empreendimento que possui vários hotéis.
Hotel	Estabelecimento comercial que possui acomodações que podem ser alugadas por um período de tempo.
Acomodação	Quarto, apartamento e suíte do hotel para acomodar os clientes durante sua estada no hotel.
Diária	Valor a ser pago pelo cliente referente a estada no hotel.
Tipo de Acomodação	Padrão de conforto do hotel. Exemplo: padrão, luxo e suíte.
Usuário do Sistema	Funcionário do hotel que possui acesso às operações do sistema de reservas do hotel.
Papel	Função que um usuário desempenha no sistema. Exemplo: cliente, operador de reservas e gerente do hotel.
Cliente	Pessoa que faz reserva do apartamento.
Gerente de Hotel	Pessoa responsável pelo gerenciamento do hotel.
Reserva de Acomodação	Procedimento no qual um tipo de acomodação fica reservado para um cliente, garantindo há disponibilidade de acomodação, para quando o cliente fizer sua entrada no hotel.
Estada do Hotel	Procedimento no qual o cliente ocupa a reserva, recebendo um quarto para acomodar-se e iniciar sua estada.
Empresa	Empreendimento conveniada com a rede de hotel, obtendo descontos no valor das diárias.

APÊNDICE B - Documentação dos Componentes do Sistema de Hotel

B.2 Descrever os Casos de Uso Funcionais.

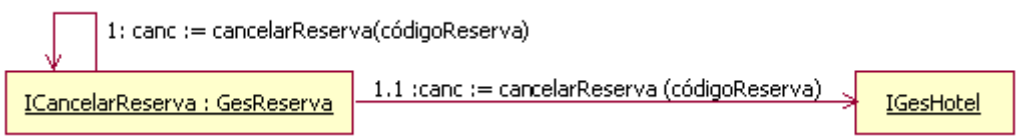
Caso de Uso: Ocupar Reseva. **Ator:** Operador de Reserva. **Objetivo:** Registrar-se em um hotel a partir de uma reserva efetuada.
Cenário de Sucesso Principal: 1. O Cliente chega ao hotel, informa que fez uma reserva e informa ao Operador de Reservas o código de sua reserva. 2. Inclui Identificar Reserva. 3. O Sistema verifica se o Cliente já está cadastrado no Sistema. 4. O Operador de Reservas informa ao Sistema os dados de cadastro completo do Cliente. 5. O Cliente confirma os detalhes da duração de estada e tipo de acomodação e o Operador de Reservas confirma a operação. 6. O Sistema aloca uma acomodação. 7. Inclui Calcular Desconto. 8. Inclui Gerar Fatura. Fluxo Alternativo: 3. Reserva não identificada. a. Fim do caso de uso. 4. O Cliente já está cadastrado no Sistema. a. Vai para 5. 5. O Cliente deseja alterar os detalhes de sua estada. a. Inclui Alterar Reserva.

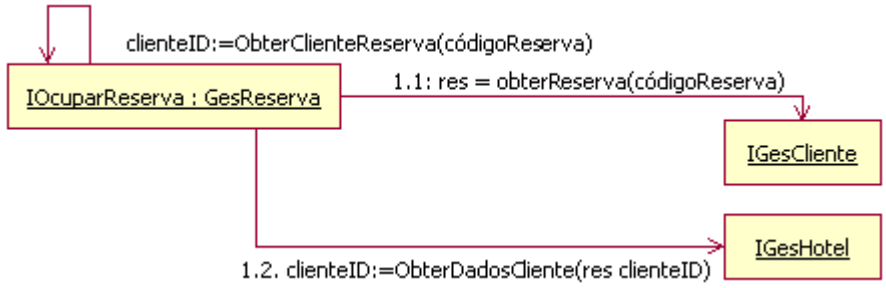
Caso de Uso: Cancelar Reseva. **Ator:** Operador de Reserva. **Objetivo:** Cancelar um reserva.
Cenário de Sucesso Principal: 1. O Cliente deseja cancelar sua reserva e informa ao Operador de Reservas o código de sua reserva. 2. Inclui Identificar Reserva. 3. O Cliente confirma o cancelamento da reserva e o Operador de Reservas confirma a operação. 4. O Sistema registra o cancelamento da reserva. Fluxo Alternativo: 3. Reserva não identificada. a. Fim do caso de uso.

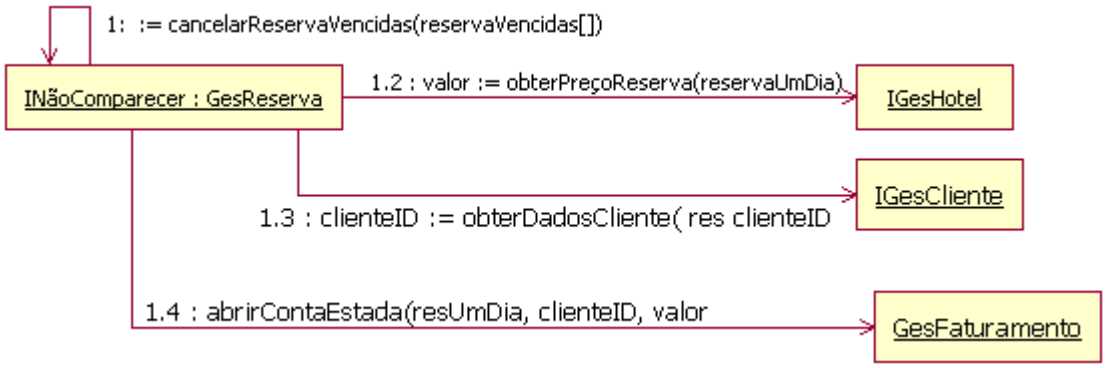
Caso de Uso: Não Comparecer. **Ator:** Operador de Reserva. **Objetivo:** Identificar Clientes que fizeram reserva e não compareceram para ocupá-la.
Cenário de Sucesso Principal: 1. O Auxiliar de Reservas deseja verificar as reservas que já venceram nos dias anteriores não foram ocupadas pelos Clientes. 2. O Sistema mostra as reservas vencidas nos dias anteriores e que não foram canceladas e os Clientes não compareceram para ocupá-las. 3. O Auxiliar de Reservas seleciona as reservas não ocupadas que devem ser faturadas. 4. Inclui Calcular Desconto. 5. Para cada reserva selecionada inclui Gerar Faturas. 6. As reservas não selecionadas são canceladas.

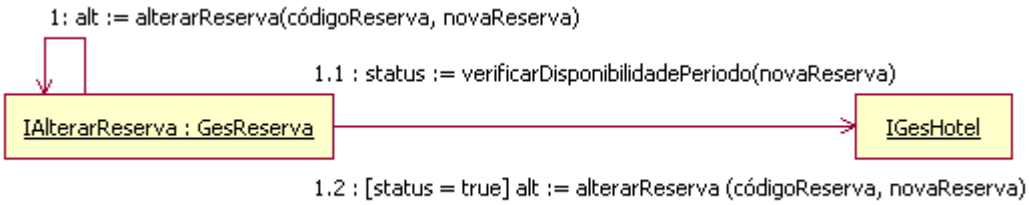
Caso de Uso: Alterar Reserva. **Ator:** Operador de Reserva. **Objetivo:** Alterar detalhes de uma reserva.
Cenário de Sucesso Principal: 1. O Cliente deseja alterar os detalhes de sua reserva e informa ao Operador de Reservas o código da reserva. 2. Inclui Identificar Reserva. 3. O Cliente informa ao Operador de Reservas os novos dados da reserva (hotel, datas do período e tipo de acomodação). 4. Incluir Verificar Disponibilidade de Acomodação. 5. O Sistema fornece o novo preço. 6. O Cliente confirma a alteração da reserva e o Operador de Reservas confirma a operação. 7. O Sistema atualiza a reserva do cliente. Fluxo Alternativo: 3. Reserva não identificada. a. Fim do caso de uso. 5. Acomodação não disponível a. O Sistema oferece alternativas de hotéis. b. O Cliente escolhe um hotel alternativo. 6. O Cliente recusa a oferta de preço. a. Fim do caso de uso.

B.2 Interação entre as Interfaces.









B.3 Operações das Interfaces de Sistema.

<< interface >> **IFazerReserva**
obterCliente(in nomeCliente: String): DadosCliente obterHoteis(): String[ ] selecionaHotel(in nomeHotel: String) verificarDisponibilidadePeriodo(in res: DetalhesReserva): boolean obterPrecoReserva(in res: DetalhesReserva): float obterTXDescontoReserva(in res:DetalhesReserva): float fazerReserva(in res: DetalhesReserva, in cli:DadosCliente): String obterDetalheHotel(in nomeHotel: String): DetalhesHotel obterInfoTipoAcomodacao(in tipoAcomodacao: String): InfoTipoAcomodacao

<< interface >> **IGesFaturamento**
abrirContaEstada(in res: DetalhesReserva, in cli: DadosCliente, in valor:float)

<< interface >> **IOcuparReserva**
obterReserva(in codReserva: String): DetalhesReserva obterClienteReserva(in codReserva: String): DadosClienteCompleto completarDadosCliente(in cliCompleto: DadosClienteCompleto) obterPrecoReserva(in res: DetalhesReserva): float obterTXDescontoReserva(in res:DetalhesReserva): float iniciarEstada(in codReserva: String): int

<< interface >> **ICancelarReserva**
obterReserva(in codReserva: String): DetalhesReserva obterNomeHotel(in hotelID: int):String obterNomeCliente(in cliID: int):String cancelarReserva(in codReserva: String): boolean

<< interface >> **IAlterarReserva**
obterReserva(in codReserva: String): DetalhesReserva obterDetalheHotel(in nomeHotel: String): DetalhesHotel obterInfoTipoAcomodacao(in tipoAcomodacao: String): InfoTipoAcomodacao obterPrecoReserva(in res: DetalhesReserva): float obterTXDescontoReserva(in res:DetalhesReserva): float verificarDisponibilidadePeriodo(in res: DetalhesReserva): boolean alterarReserva(in codReserva: String, in novaRes: DetalhesReserva): boolean

<< interface >> **INaoComparecer**
obterReservasVencidas(in hotelID: int): String[ ] obterTXDescontoReserva(in res:DetalhesReserva): float cancelarReservasVencidas(in reservasVencidas: String[ ]) gerarFaturasReservas(in reservas: String[ ])

B.4 Operações das Interfaces de Negócio.

<< interface >> **IGesCliente**
obterNomeCliente(in cliID: int): String obterCliente(in nomeCliente:String): DadosCliente obterDadosCliente(in cliID: int): DadosCliente obterCliente(in cli:DadosCliente): int criarCliente(in cli:DadosCliente) notificarCliente(in cliID:int, in codReserva:String, in res:DetalhesReserva) completarDadosCliente(in cliCompleto:DAdosCompletoCliente)

<< interface >> **IGesEmpresa**

associaClienteEmpresa(DadosCliente cli, String nomeEmpresa);
obterEmpresas();
obterClientesEmpresa(String nomeEmpresa);
obterTXDescontoEmpresa(String nomeEmpresa);
obterEmpresaCliente(int cliID);
addEmpresa(DadosEmpresa emp);