

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO DO RN – FARN**

ANÁLISE E PROJETO DE UM SIMULADOR PARA EXAMES DE CERTIFICAÇÃO

ITALO SOARES DE OLIVEIRA LIMA

**Natal
2007**

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO DO RN – FARN**

ANÁLISE E PROJETO DE UM SIMULADOR PARA EXAMES DE CERTIFICAÇÃO

ITALO SOARES DE OLIVEIRA LIMA

**Natal
2007**

ITALO SOARES DE OLIVEIRA LIMA

ANÁLISE E PROJETO DE UM SIMULADOR PARA EXAMES DE CERTIFICAÇÃO

Monografia apresentada ao Curso de Pós Graduação em Desenvolvimento de Sistemas Corporativos da Faculdade Natalense para o Desenvolvimento do RN como requisito obrigatório para a obtenção do título de Especialista em Informática.

Orientador: Ulisses Telemaco Neto.

Natal
2007

REC 5
FARN
BIBLIOT
10/05/07
10/05/07

Catálogo na Publicação – Biblioteca da FARN
Setor de Processos Técnicos

Lima, Italo Soares de Oliveira

Análise e projeto de um simulador para exames de certificação / Italo
Soares de Oliveira Lima. – Natal: FARN, 2007.
49f.

Orientador: Ulisses Telêmaco Neto

Monografia (Especialização em Desenvolvimento de Sistemas
Corporativos) – Faculdade Natalense para o Desenvolvimento do Rio
Grande do Norte.

1. Desenvolvimento de Softwares – Monografia. 2. Sistema Simulador
de Certificação – Monografia. 3. Análise e Projetos Orientados a Objetos –
Monografia. I. Título.

RN/FARN/BC

CDU 004.4

B.B. 30.110

FARN

BIBLIOTECA

Nº Reg

10000901

16.04.08

004.4

L6980v

2007

ex 01

ITALO SOARES DE OLIVEIRA LIMA

ANÁLISE E PROJETO DE UM SIMULADOR PARA EXAMES DE CERTIFICAÇÃO

Monografia apresentada ao Curso de especialização em Desenvolvimento de Sistemas Corporativos da Faculdade Natalense para o Desenvolvimento do RN e aprovado em ____ de dezembro de 2007, pela banca examinadora composta pelos seguintes membros:



Ulisses Telemaco Neto
Orientador



Gleydson de A. F. Lima
Examinador

*Aos meus pais Ivan e Eva, ao meu irmão
Ivan Júnior e ao meu sobrinho Ícaro.*

AGRADECIMENTOS

A minha namorada Tásia pelo apoio e companheirismo em todos os momentos. Suas observações inteligentes e pertinentes em muitas vezes me ajudaram no meio do caminho.

A minha mãe Eva pelo pensamento positivo e incentivo. Seu otimismo serviu de inspiração nessa jornada.

RESUMO

A grande quantidade de currículos e a forte concorrência na seleção de candidatos para as empresas de TI atualmente, tem levado os empregadores a usar alguns filtros na hora de selecionar recursos. Um diferencial para os profissionais de TI são as certificações emitidas por entidades competentes na aprovação do candidato em exames que testam sua proficiência em tecnologias específicas. A preparação para esses exames exige disciplina e dedicação por parte do candidato. Uma ferramenta que auxilia nos estudos são softwares simuladores das provas que imitam o exame real. Nesse trabalho, será modelado um Sistema Simulador para Certificação de Programador Java, usando a Análise e Projeto orientado a objetos, com a UML para documentação de diagramas e princípios GRASP e padrões de projeto GoF para elaborar o projeto OO.

Palavras-chave: Análise e projeto OO. UML. Padrões de Projeto. Simulador. Certificação.

ABSTRACT

Nowadays the big quantity of applications and the high competition in candidate's selection for IT companies have induced the employers to apply some filters in the resources' selection. A differential for the IT professionals are the certifications issued for entities responsible for the candidate's approval in exams that test his proficiency in specific technologies. The preparation for these exams needs discipline and dedication. A tool that helps in the studies is simulators softwares of the tests that imitate the real exam. In this work will model a Simulator System for Certification of Java Programmer, using the Analysis and Project guided to objects, with the UML for diagrams documentation and GRASP principles and design patterns GoF to elaborate the project.

Key words: OO analysis and project. UML. Design Patterns. Simulator. Certification.

LISTA DE FIGURAS

Figura 1: Exemplo de Diagrama de Caso de Uso	18
Figura 2: Exemplo de Classe e UML	20
Figura 3: Representação Visual de uma Generalização em UML	21
Figura 4: Representação Visual de uma Associação em UML	21
Figura 5: Representação Visual de uma Agregação em UML.....	22
Figura 6: Representação Visual de uma Composição em UML	22
Figura 7: Exemplo de Diagrama de Seqüência	23
Figura 8: Exemplo de Modelo de Domínio	24
Figura 9: Diagrama de Casos de Uso do Simulador	32
Figura 10: Modelo de Domínio do Sistema	37
Figura 11: Diagrama de Seqüência Realizar Login	38
Figura 12: Diagrama de Seqüência Iniciar Prova	39
Figura 13: Diagrama de Seqüência Fazer Prova	39
Figura 14: Diagrama de Seqüência Consultar Resultados	40
Figura 15: Diagrama de Classes da Aplicação	41
Figura 16: Diagrama de Classes mostrando a persistência da Aplicação.....	42
Figura 17: Diagrama de Seqüência para cenário Realizar Login	43
Figura 18: Diagrama de Seqüência para cenário Iniciar Prova	44
Figura 19: Diagrama de Seqüência para cenário Fazer Prova	45
Figura 20: Diagrama de Seqüência para cenário Consultar Resultados	46

SUMÁRIO

1 INTRODUÇÃO.....	11
1.1 APRESENTAÇÃO.....	11
1.2 CONTEXTUALIZAÇÃO DO PROBLEMA.....	11
1.2.1 Principais simuladores existentes.....	12
1.2.1.1 <i>Master Exam</i>	13
1.2.1.2 <i>Whizlabs Preparation Kit V6</i>	13
1.3 OBJETIVOS.....	14
1.3.1 Geral.....	14
1.3.2 Específicos.....	14
1.4 JUSTIFICATIVA.....	14
2 REFERENCIAL TEÓRICO.....	16
2.1 ANÁLISE E PROJETO ORIENTADO A OBJETOS.....	16
2.2 UML	16
2.2.1 Diagrama de Caso de Uso	17
2.2.2 Casos de Uso	18
2.2.3 Diagramas de Classes	19
2.2.3.1 <i>Associações de Classe</i>	20
2.2.3.1.1 Generalização	20
2.2.3.1.2 Associação	21
2.2.3.1.3 Agregação	21
2.2.3.1.4 Composição	22
2.2.4 Diagramas de Seqüência	22
2.3 MODELO DE DOMÍNIO.....	23
2.4 RESPONSABILIDADES.....	24
2.5 PADRÕES GRASP	25
2.5.1 Criador	25
2.5.2 Coesão Alta	25
2.5.3 Acoplamento Baixo	25
2.5.4 Especialista	26

2.5.5 Controlador	26
2.6 PADRÕES DE PROJETO.....	26
2.6.1 Abstract Factory	27
2.6.2 Adapter	28
2.6.3 Bridge	28
2.6.4 Builder	28
2.6.5 Chain of Responsibility	28
2.6.6 Command	28
2.6.7 Composite	28
2.6.8 Data Access Object (DAO)	28
2.6.9 Façade	28
2.6.10 Factory Method	29
2.6.11 Mediator	29
2.6.12 Observer	29
2.6.13 Singleton	29
2.6.14 Template Method	29
2.7 FRAMEWORKS.....	29
3 ANÁLISE E PROJETO.....	31
3.1 ANÁLISE DE REQUISITOS.....	31
3.1.1 Diagrama de Caso de Uso	31
3.1.2 Documentação dos Casos de Uso	32
3.1.2.1 Caso de Uso Realizar Login	33
3.1.2.2 Caso de Uso Iniciar Prova	33
3.1.2.3 Caso de Uso Fazer Prova	34
3.1.2.4 Caso de Uso Consultar Resultados	36
3.1.3 Modelo de Domínio	37
3.1.4 Diagramas de Seqüência	37
3.1.4.1 Realizar Login	38
3.1.4.2 Iniciar Prova	38
3.1.4.3 Fazer Prova	39
3.1.4.4 Consultar Resultados	40
3.2 PROJETO DE OBJETOS.....	40
3.2.1 Diagrama de Classes	40

3.2.2 Diagramas de Seqüência	43
3.2.2.1 <i>Realizar Login</i>	43
3.2.2.2 <i>Iniciar Prova</i>	44
3.2.2.3 <i>Fazer Prova</i>	44
3.2.2.4 <i>Consultar Resultados</i>	45
4 CONCLUSÃO.....	47
REFERÊNCIAS.....	48

1 INTRODUÇÃO

1.1 APRESENTAÇÃO

Este trabalho tem como objetivo elaborar a análise e projeto de um software simulador de exames de certificação que possa auxiliar o aluno no estudo e na preparação para a prova real, guardando dados estatísticos de desempenho para consultas futuras.

Através do projeto proposto, será possível implementar a solução em qualquer tecnologia.

Existem espalhados na internet muitos simuladores de exames de certificação, cada um com suas características, muitas vezes repetidas, alguns com mais recursos, outros mais limitados.

A idéia é especificar um simulador com as principais características utilizadas nas implementações já existentes com um diferencial que é a geração de histórico sobre o resultado obtido pelo usuário cada vez que uma prova é respondida.

1.2 CONTEXTUALIZAÇÃO E DEFINIÇÃO DO PROBLEMA

Nos dias atuais, o mercado exige cada vez mais o aperfeiçoamento dos profissionais da área de informática e a rapidez com que as informações mudam os obriga a sempre estarem atualizados com novas tecnologias.

Com tanta gente tentando entrar no mercado de trabalho, os empregadores necessitam de uma forma de saber se os pretendentes às vagas possuem conhecimento em determinadas tecnologias ou pelo menos selecionar de alguma maneira padronizada, fazendo um primeiro filtro entre os candidatos.

Uma das formas do candidato provar seu conhecimento numa determinada área é possuir certificações de órgãos e/ou empresas que aplicam provas, de modo a medir o conhecimento do candidato naquele assunto.

Ter certificações que interessam ao mercado é uma das maneiras de mostrar, no mínimo disciplina e determinação de ter estudado e passado numa prova de conhecimento.

O simulador proposto nesse trabalho se baseia no exame de Certificação de Programador Java (SCJP). A *Sun Microsystems*, criadora do exame, é uma das maiores empresas de tecnologia em conectividade do mundo. O site oficial é <http://www.sun.com>.

A prova do exame é composta de questões de múltipla escolha, classificadas em vários assuntos, que são: declaração, controle de acesso, orientação a objetos, atribuições, operadores, controle de fluxo, exceções, assertivas, manipulação de strings, E/S, formatação, *parsing*, *generics*, *collections*, classes internas, *threads* e ferramentas do JDK (*Java Development Kit*).

O candidato deve adquirir um *voucher* que dá o direito de agendar um dia para realização prova em uma das unidades da *Prometric* credenciadas. Isso pode ser feito em um prazo máximo de um ano após a compra do *voucher*. A *Prometric* é uma empresa parceira da *Sun* que aplica suas provas de certificação no mundo todo.

Tais exames de certificação têm custo elevado, o que obriga uma boa preparação do candidato para não ser reprovado.

Na fase de preparação, costuma-se usar programas "*mocks*" que simulam o exame original de forma que o candidato treine e se habitue ao que encontrará na prova real.

A proposta dessa monografia é, usando UML, modelar um simulador de exames de certificação para auxiliar candidatos na fase de preparação para a prova.

O diferencial desse sistema para os já existentes é a proposta de um módulo de dados históricos, onde será gerada informação sobre o andamento do candidato nas simulações e sua progressão com várias tentativas, identificando percentualmente as melhoras ou pioras em cada tipo de assunto abordado nas provas.

1.2.1 Principais simuladores existentes

Dois dos principais simuladores usados como base para levantamento de requisitos do Sistema Simulador são descritos de maneira resumida abaixo.

1.2.1.1 Master Exam

Software de preparação para exames desenvolvido pela empresa *LearnKey* que tem como objetivo a prática de testes para medir e melhorar o nível de experiência do aluno e prepará-los para os atuais exames de certificação.

As principais características do *Master Exam* são:

- Descobrir fraquezas e forças na compreensão dos objetivos do teste;
- Medir e acompanhar a preparação do candidato para fazer a prova de certificação;
- Praticar usando o mesmo tipo de questões e cenários encontrados no exame;
- Focar os estudos em tópicos críticos;
- Testar o conhecimento em todos os objetivos.

1.2.1.2 Whizlabs Preparation Kit V6

Esse é um popular software da *Whizlabs* que, segundo a empresa, garante o sucesso no exame de certificação. Preparado por profissionais altamente experientes, simula com grande fidelidade o ambiente real da prova.

Suas principais características são:

- Grande variedade de simulados e questões. São seis simulados completos com mais de 400 questões diferentes para cada simulado;
- Padrões das questões são baseados no do exame de certificação real;
- Cerca de 150 *Quiz* para revisar e reforçar o entendimento dos conceitos;
- Dicas de revisão rápida para ajudar a revisar informações importantes da prova;
- Explicação detalhada de cada questão com uma explanação exhaustiva das respostas;
- Relatórios de Compreensão do andamento do candidato em cada objetivo do exame.

1.3 OBJETIVOS

1.3.1 Geral

O objetivo geral do trabalho é elaborar a análise e projeto de um modelo orientado a objetos de um software simulador de exames de certificação para treinamento.

1.3.2 Específico

- Estudar modelagem orientada a objetos.
- Verificar a aplicação prática das técnicas de análise e projeto orientados a objetos com auxílio da UML para melhor visualização da modelagem.
- Levantar os pontos fortes de simuladores já existentes e uni-los no mesmo projeto.
- Preparar e documentar um modelo que sirva de base para implementação do software de simulação de exames.
- Estudar e aplicar padrões de projeto.

1.4 JUSTIFICATIVA

Uma preocupação de quem está se preparando para prestar exame de certificação é medir seu conhecimento e saber se ele está preparado para enfrentar a prova e obter sucesso.

O papel de um simulador é imitar ao máximo o comportamento e dificuldade das questões no exame real, dando *feedback* para o indivíduo saber se está preparado ou se seu estudo está surtindo o efeito desejado. Comparar as tentativas e indicar melhoras ou não ao longo dos estudos pode ser uma interessante funcionalidade de um simulador.

Este trabalho busca projetar e documentar um simulador de exames de certificação que possa ajudar o aluno mostrando através de relatórios estatísticos o seu nível de conhecimento. Tal funcionalidade permite que o candidato identifique em quais áreas precisa melhorar até o ponto que se sinta confortável para prestar o exame real.

2 REFERENCIAL TEÓRICO

2.1 ANÁLISE E PROJETO ORIENTADOS A OBJETOS

Segundo Larman (2007, p.34),

a análise enfatiza uma investigação dos objetos do domínio do problema e dos requisitos, em vez de uma solução. Já o projeto enfatiza uma solução conceitual, focando a definição dos objetos de software e como eles colaboram para a satisfação dos requisitos e não sua implementação.

De uma maneira resumida a ênfase durante a Análise OO está em identificar e descrever conceitos no domínio do problema no mundo real. Já na fase de Projeto OO, a ênfase está em identificar e modelar objetos lógicos de software que possam ser compreendidos e implementados usando uma linguagem de programação OO.

2.2 UML

A *Unified Modeling Language* (UML) é uma linguagem para especificação, documentação, visualização e desenvolvimento de sistemas orientados a objetos.

É considerada uma das linguagens de modelagem mais expressivas para sistemas orientados a objetos.

Através de seus diagramas é possível representar visualmente sistemas de *softwares* sob diversas perspectivas, facilitando a comunicação das pessoas envolvidas no processo de desenvolvimento de um sistema.

Segundo o *Object Management Group* (OMG) apud Larman (2007, p.39) a UML é uma linguagem visual para especificar, construir e documentar os artefatos dos sistemas.

A UML pode ser usada para visualizar, especificar, construir e documentar os artefatos de um sistema de uma forma completa. (BOOCH, 1999, p.20, tradução nossa).

De acordo com Guedes (2006), a UML é uma linguagem visual utilizada para modelar sistemas computacionais por meio do paradigma de Orientação a Objetos.

2.2.1 Diagrama de Casos de Uso

Para Guedes (2006) o Diagrama de Casos de Uso procura, por meio de uma linguagem simples, possibilitar a compreensão do comportamento externo do Sistema por qualquer pessoa, tentando apresentar o Sistema através de uma perspectiva do Usuário.

Geralmente esse artefato é usado no levantamento e análise dos requisitos, podendo servir para a modelagem de outros diagramas.

Por apresentar uma visão externa geral das funções e serviços oferecidos aos usuários, pode ser apresentado em reuniões iniciais como forma de entendimento do comportamento do sistema.

“É bastante útil e recomendável que um Diagrama de Casos de Uso seja apresentado aos clientes juntamente com um protótipo, o que permitirá que um complemente o outro”. (GUEDES, 2006, p. 45).

A importância dada aos Diagramas de Caso de Uso deve ser vista com cuidado, para Larman (2007, p.89) “enfocar os Diagramas de Caso de Uso UML, de valor secundário, em vez do importante texto do Caso de Uso é um erro comum”.

Larman (2007, p.116-7) sobre os Diagramas de Caso de Uso afirma que “um Diagrama de Caso de Uso é uma excelente imagem do contexto do sistema; ele é um bom diagrama de contexto, ou seja, mostra a fronteira de um sistema, o que está fora dele e como o sistema é usado”.

Na Figura 1 é mostrado um exemplo do Diagrama de Casos de Uso.



Figura 1: Exemplo de Diagrama de Caso de Uso
Fonte: Fundamentos do UML (2007)

2.2.2 Casos de Uso

Casos de Uso são narrativas em texto de algum ator usando um sistema, utilizadas principalmente para descobrir e registrar requisitos, influenciando em muitos aspectos de um projeto. Servem de entrada e para vários artefatos e influenciam outros artefatos de análise, projeto, implementação, gestão de projeto e teste. (LARMAN, 2007).

Para Larman (2007, p.89), ator é algo com comportamento, tal como uma pessoa (identificada por seu papel), um sistema de computador ou uma organização. Cenário é uma sequência específica de ações e interações entre os Atores e o Sistema

Um Caso de Uso contém um contrato entre os *stakeholders* de um Sistema sobre seu comportamento. O Caso de Uso descreve o comportamento do Sistema sob várias condições, como ele responde à requisição de um dos *stakeholders*, chamado de ator principal. O ator principal inicia uma interação com o Sistema para cumprir algum objetivo. O Sistema responde, protegendo os interesses de todos os *stakeholders*.

(COCKBURN, 2000, p.15, tradução nossa).

De forma resumida “Casos de Uso são narrativas em texto de algum ator usando um Sistema para atingir objetivos”. (LARMAN, 2007, p.89):

De acordo com Larman (2007, p.89) “pode-se então dizer que um Caso de Uso é uma coleção de cenários relacionados de sucesso e fracasso, que descrevem um ator usando um Sistema como meio para atingir um objetivo”.

Guedes (2006, p.47), sobre documentação de Casos de Uso, diz:

não existe um formato específico de documentação para Casos de Uso definido pela UML, o que está de acordo com a característica do próprio diagrama, ou seja, o formato de documentação de um Caso de Uso é bastante flexível, permitindo que se documente o Caso do Uso da forma que se considerar melhor.

Segundo Larman (2007), podem-se classificar os Casos de Uso em três diferentes formatos e níveis de formalidade:

- **Resumido** – resumo sucinto de um parágrafo para um cenário, geralmente de sucesso.
- **Informal** – múltiplos parágrafos informais que cobrem vários cenários.
- **Completo** – Todos os passos e variantes são escritos detalhadamente, existindo seções de suporte.

Larman (2007, p.109) descreve algumas ações para encontrar casos de uso:

- Escolher a fronteira do Sistema;
- Identificar os atores principais;
- Identificar os objetivos para cada ator principal;
- Definir casos de uso que satisfaçam os objetivos dos usuários.

2.2.3 Diagramas de Classes

O Diagrama de Classes é considerado o mais importante e mais usado diagrama da UML. Apresenta uma visão estática de como as classes estão organizadas, focado em como definir a estrutura lógica das mesmas, servindo dessa forma como base para a construção da maioria dos outros diagramas da UML (GUEDES, 2006).

Já Booch et al. (1998, p.88, Tradução Nossa) diz que um Diagrama de Classes mostra um conjunto de classes, interfaces, e colaborações e seus relacionamentos, sendo o diagrama

mais comumente encontrado na modelagem de sistemas orientados a objetos, ilustrando uma visão estática do projeto de um sistema.

De acordo com Guedes (2006, p.69) “O Diagrama de Classes foi intencionalmente projetado para ser uma evolução do modelo entidade-relacionamento utilizado para definir as estruturas de tabelas em bancos de dados relacionais”.

Enfocando a parte de projeto, na UML as classes são representadas por retângulos, com o nome da classe, podendo mostrar também os atributos e métodos da classe.

Abaixo na Figura 2 a representação visual de uma classe em UML.

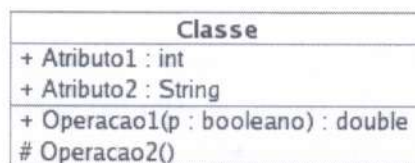


Figura 2: Exemplo de Classe e UML
Fonte: Fundamentos do UML (2007)

2.2.3.1 Associações de Classe

As classes dentro do diagrama podem relacionar-se entre si de diferentes maneiras, como: generalização, associação, agregação, composição.

2.2.3.1.1 Generalização

Em UML, uma associação de generalização entre duas classes coloca-as numa hierarquia representando o conceito de herança de uma classe derivada de uma classe base. É representada por uma linha conectando duas classes, com uma seta no lado da classe base como na Figura 3.

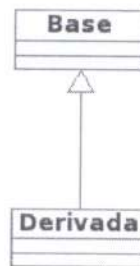


Figura 3: Representação Visual de uma Generalização em UML
Fonte: Fundamentos do UML (2007)

2.2.3.1.2 Associação

Uma associação representa um relacionamento entre classes e fornece a semântica comum e estrutura para muitos tipos de ligações entre objetos. Permite objetos comunicar-se entre si.

Associações podem ter uma regra que especifica o propósito da associação e pode ser unidirecional ou bidirecional. Cada ponta da associação também possui um valor de multiplicidade, que dita como muitos objetos neste lado da associação podem relacionar-se com o outro lado.

São representadas como linhas conectando as classes participantes do relacionamento, opcionalmente com regras e multiplicidade como na Figura 4.



Figura 4: Representação visual de uma associação em UML
Fonte: Fundamentos do UML (2007)

2.2.3.1.3 Agregação

Agregação é um tipo especial de associação onde as duas classes participantes não possuem em nível igual, mas fazem um relacionamento "todo-parte". Uma agregação descreve como a classe que possui a regra do todo, é composta de outras classes, que possuem a regra das partes.

Agregações são representadas por uma associação que mostra um romboide no lado do todo conforme ilustrado da Figura 5.



Figura 5: Representação Visual de uma Agregação em UML
Fonte: Fundamentos do UML (2007)

2.2.3.1.4 Composição

Composições são associações que representam agregações muito fortes. Isto significa que Composições formam relacionamentos todo-parte também, mas o relacionamento é tão forte que as partes não podem existir independentes. Elas existem somente dentro do todo, e se esse é destruído as partes morrem também.

Em UML, composições são representadas por um rombóide sólido no lado do todo. Abaixo na Figura 6 um exemplo de composição.



Figura 6: Representação Visual de uma Composição em UML
Fonte: Fundamentos do UML (2007)

2.2.4 Diagramas de Seqüência

Diagrama de Seqüência é um recurso visual para mostrar um cenário específico de um Caso de Uso, descrevendo parcialmente um comportamento do sistema com os eventos gerados pelos atores externos e os eventos entre sistemas em ordem temporal.

Para Guedes (2006, p.104) o objetivo principal do Diagrama de Seqüência é: "Determinar a ordem em que os eventos ocorrem, as mensagens que são enviadas, os métodos que são chamados e como os objetos interagem entre si dentro de um determinado processo".

Um Diagrama de Seqüência muitas vezes se identifica com um Caso de Uso específico, porque um Caso de Uso, em geral, refere-se a um processo disparado por um usuário. Assim um Diagrama de Seqüência também permite documentar um Caso de Uso.

Guedes (2006, p.104) diz que os Diagramas de Seqüência dependem também do Diagrama de Classes, uma vez que as classes dos objetos declarados no diagrama estão descritas nele, bem como os métodos disparados entre os objetos.

Um Diagrama de Seqüência costuma identificar o evento gerador do processo modelado, bem como, o ator responsável por este evento e determina como o processo deve se desenrolar e ser concluído por meio da chamada de métodos disparados por mensagens enviadas entre objetos.

A função dos Diagramas de Seqüência é fazer uma análise comportamental do Sistema, através dos seus eventos visualizados em ordem temporal.

Na Figura 7 abaixo, um exemplo de Diagrama de Seqüência.

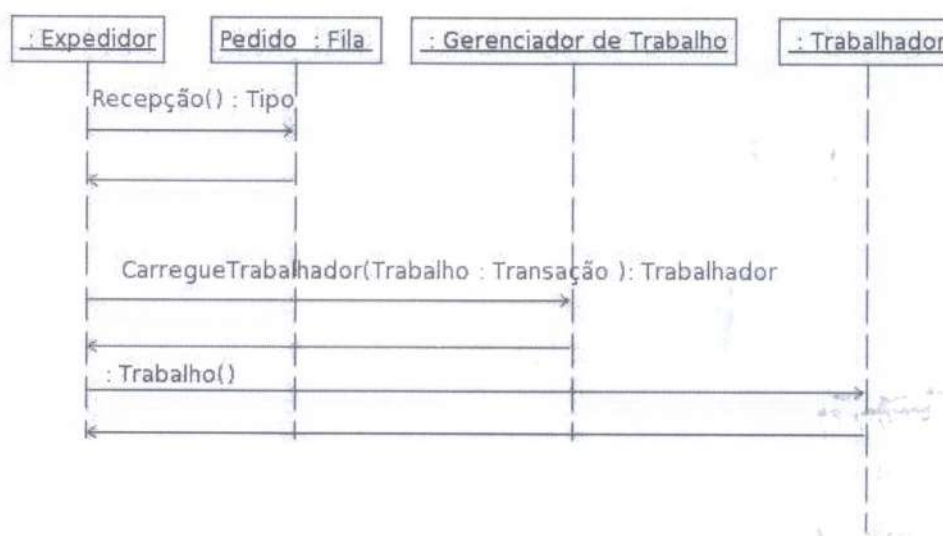


Figura 7: Exemplo de Diagrama de Seqüência
Fonte: Fundamentos do UML (2007)

2.3 MODELO DE DOMÍNIO

Para Larman (2007, p.159) “o passo mais essencialmente orientado a objetos na análise é a decomposição de um domínio em conceitos ou objetos importantes”.

Um Modelo de Domínio é uma representação visual, ou objetos do mundo real, em um domínio (MARTIN, FOWLER apud LARMAN, 2007, p.160).

Visualmente, aplicando notação UML, um Modelo de Domínio é um Diagrama de Classes sem operações definidas, fornecendo uma perspectiva conceitual do problema. Geralmente são mostradas classes conceituais, seus atributos e associações entre elas.

Modelo de Domínio é um dicionário visual das abstrações relevantes, do vocabulário do domínio e do conteúdo de informação do domínio (LARMAN, 2007, p.161).

Não se deve confundir o Modelo de Domínio com a arquitetura do *software*, nem com o modelo de dados.

Abaixo na Figura 8, um exemplo de um Modelo de Domínio.

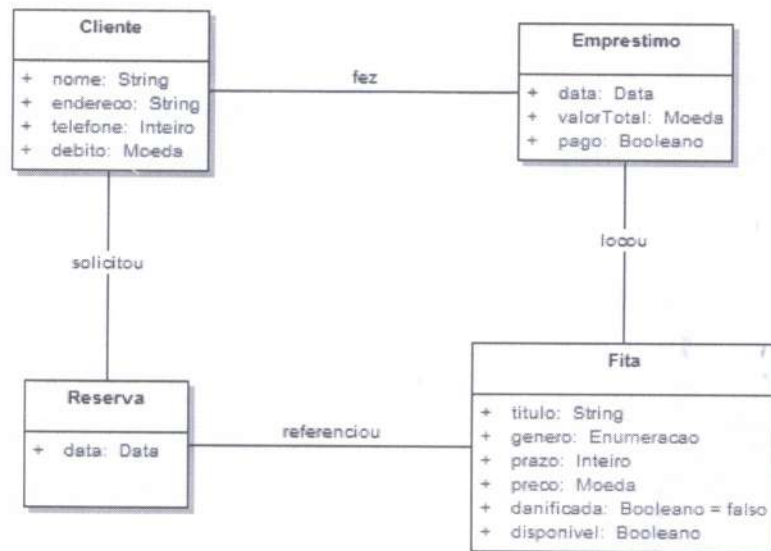


Figura 8: Exemplo de Modelo de Domínio
Fonte: Fundamentos do UML (2007)

2.4 RESPONSABILIDADES

A UML define responsabilidade como “um contrato ou obrigação de um classificador” (*OBJECT MANAGEMENT GROUP* apud LARMAN, 2007, p.292).

Responsabilidades estão relacionadas com as obrigações de um objeto em termos do seu papel. Durante o projeto orientado a objetos, são atribuídas a classes de objetos. Sua classificação, basicamente, é de dois tipos: fazer e conhecer (LARMAN, 2007).

2.5 PADRÕES GRASP

GRASP, sigla para *General Responsibility Assignment Software Patterns* (Padrões de Software para Atribuição de Responsabilidade Geral) são a base para se projetar sistemas OO (LARMAN, 2007).

Os princípios ou padrões GRASP são um apoio ao aprendizado para ajudar a entender o projeto de objetos essencial e aplicar o raciocínio de projeto de um modo metódico, racional, explicável. Essa abordagem para entender e usar princípios de projeto é baseada em padrões para atribuir responsabilidades (LARMAN, 2007, p.293).

São nove os padrões GRASP:

- Criador (*Creator*);
- Especialista na Informação (*Information Expert*);
- Acoplamento Baixo (*Low Coupling*);
- Controlador (*Controller*);
- Coesão Alta (*High Cohesion*);
- Polimorfismo (*Polimorphism*);
- Indireção (*Indirecton*);
- Invenção Pura (*Pure Fabrication*);
- Variações Protegidas (*Protected Variations*).

Abaixo cinco dos mais usados princípios são detalhados de maneira breve:

2.5.1 Criador

Problema: quem é o responsável por criar instâncias de uma classe?

Solução: um objeto deve ser criado por outro que esteja fortemente associado a ele o possua como parte (geralmente por agregação).

2.5.2 Coesão Alta

A coesão é uma medida do quão fortemente as responsabilidades de uma classe estão relacionadas e focalizadas.

É fortemente sugerido que as responsabilidades atribuídas a cada classe sejam altamente relacionadas.

2.5.3 Acoplamento Baixo

O acoplamento é a medida de quão fortemente uma classe está conectada a outras classes, tem conhecimento das mesmas ou depende delas.

Uma classe com acoplamento forte é mais difícil de compreender isoladamente, mais difícil de reutilizar (seu uso depende da reutilização das outras classes da qual ela depende) e mais sensível a mudanças nas classes associadas.

2.5.4 Especialista

É o padrão mais usado para atribuir responsabilidades.

Problema: dado um comportamento (responsabilidade), definir a qual classe essa responsabilidade deve ser alocada.

Solução: atribuir essa responsabilidade ao especialista da informação – a classe que tem a informação necessária para satisfazer a responsabilidade.

2.5.5 Controlador

Problema: quem deveria ser responsável por tratar um evento do sistema?

Solução: atribuir a responsabilidade do tratamento de um evento do sistema a uma classe que representa uma das seguintes escolhas:

- Representa o “sistema” todo (controlador fachada)
- Representa um tratador oficial de todos os eventos de sistema de um caso de uso (controlador de caso de uso).

2.6 PADRÕES DE PROJETO

A idéia original dos Padrões de Projeto foi descrita por Gamma et al. (2000, p.20) da seguinte forma:

Um padrão de projeto nomeia, abstrai e identifica os aspectos-chave de uma estrutura de projeto comum para torná-la útil para a criação de um projeto orientado a objetos reutilizável. O padrão de projeto identifica as classes e instâncias participantes, seus papéis, colaborações e a distribuição de responsabilidades. Cada padrão de projeto focaliza um problema ou tópico particular de projeto orientado a objetos. Ele descreve quando pode ser aplicado, se pode ser aplicado em função de outras restrições de projeto e as consequências, custos e benefícios de sua utilização.

Dessa maneira, torna-se mais fácil reutilizar projetos e arquiteturas de sucesso. Documentar técnicas testadas e aprovadas as deixam mais acessíveis para a utilização das mesmas idéias em futuros sistemas. Padrões de projeto ajudam à construção de sistemas

reutilizáveis, facilitando a escolha de alternativas de projeto, evitando soluções que comprometam a reutilização (GAMMA et al., 2000, p.18).

De acordo com Larman (2007, p.295) “um bom padrão é um par problema/solução denominado e bem conhecido que pode ser aplicado em novos contextos, com conselhos sobre como aplicá-lo em situações novas e discussão sobre seus compromissos, implementações variações, etc”.

Um padrão de projeto orientado a objetos pode utilizar uma linguagem como UML para apresentar Diagramas de Classes e objetos que compõem o padrão.

Um padrão de projeto provê um esquema para um refinamento de subsistemas ou componentes de um sistema de software. Estes tipos de padrões definem uma micro arquitetura, sendo que sua influência restringe-se à região do sistema onde são utilizados.

Os padrões de projeto são classificados da seguinte forma (GAMMA et al., 2000):

▪ Padrões de criação (*Creational Patterns*)

Especificam o processo de criação de objetos. Podem abstrair os detalhes de criação para que a implementação seja independente dos tipos de objetos envolvidos. Desta forma, ao se adicionar um novo tipo de objeto àqueles que podem ser criados, não será necessário promover grandes mudanças no código.

▪ Padrões estruturais (*Structural Patterns*)

Lidam com as conexões ou relacionamentos entre objetos, ou seja, especificam composições de objetos ou classes.

▪ Padrões comportamentais (*Behavioral Patterns*)

Caracterizam interações de objetos entre si, ou seja, controlam determinados tipos de ações no sistema e distribuem responsabilidades.

Para o propósito desse trabalho, alguns dos principais padrões de projeto são descritos com um resumo dos problemas a serem resolvidos por eles (Booch, 2000).

2.6.1 Abstract Factory

Prover uma interface para se criar famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.

2.6.2 Adapter

Converter a interface de uma classe em outra interface esperada pelos clientes. Adapter permite a comunicação entre classes que não poderiam trabalhar juntas devido à incompatibilidade de suas interfaces.

2.6.3 Bridge

Desacoplar uma abstração de sua implementação para permitir que ambos variem independentemente.

2.6.4 Builder

Separar a construção de um objeto complexo de sua representação para que o mesmo processo de construção possa criar diferentes representações.

2.6.5 Chain of Responsibility

Evitar o acoplamento do remetente de uma requisição com o seu destinatário ao permitir que mais de um objeto trate a requisição. Compõe os objetos em cascata e passa a requisição adiante até que um objeto a trate.

2.6.6 Command

Encapsular uma requisição como um objeto, permitindo que clientes parametrizem diferentes tipos de requisições.

2.6.7 Composite

Compor objetos em estruturas de árvore para representar hierarquias parcialmente completas. *Composite* permite que clientes tratem objetos individuais e composições de objetos de maneira uniforme.

2.6.8 Data Access Object (DAO)

Padrão para persistência de dados que permite separar regras de negócio das regras de acesso a banco de dados.

2.6.9 Façade

Oferecer uma interface única para um conjunto de interfaces de um subsistema. *Façade* define uma interface de nível mais elevado que reduz a complexidade do subsistema.

2.6.10 Factory Method

Definir uma interface para criar um objeto, mas permitir que as subclasses decidam o tipo da classe a ser instanciada. *Factory Method* permite que uma classe delegue a responsabilidade de instanciamento às subclasses.

2.6.11 Mediator

Definir um objeto que encapsula a forma como um conjunto de objetos interage entre si. *Mediator* promove acoplamento fraco ao evitar que os objetos se refiram uns aos outros explicitamente, permitindo que a interação desses objetos de seja alterada de maneira independente.

2.6.12 Observer

Definir uma dependência um-para-muitos entre objetos de modo que quando um objeto mudar de estado, todos os seus dependentes sejam notificados e atualizados automaticamente.

2.6.13 Singleton

Garantir que uma classe só tenha uma única instância, e prover um ponto de acesso global a ela.

2.6.14 Template Method

Definir o esqueleto de um algoritmo, para que as subclasses possam prover especialidade específica posteriormente. *Template Method* permite que suas subclasses redefinam certos passos de um algoritmo sem mudar sua estrutura.

2.7 FRAMEWORKS

“Um *framework* é um conjunto de classes cooperantes que constroem um projeto reutilizável para uma classe específica de software” (DEUTSCH, JOHNSON apud GAMMA et al., 2000, p.41).

Para Gamma et al. (2000), um *framework* de *software* orientado a objetos, é definido como um conjunto de classes que cooperam entre si para formar um projeto específico para

um tipo de sistema, fornecendo um esqueleto do sistema ao dividir o projeto em classes abstratas, definindo suas responsabilidades e colaborações.

Nesse contexto o desenvolvedor concretiza uma determinada aplicação por herança e composição de instâncias de classes do *framework*.

Entre *frameworks* e padrões, podem-se citar algumas diferenças:

- Os *frameworks* determinam a arquitetura de um sistema, enquanto os padrões definem micro arquiteturas (ou seja, influenciam apenas uma determinada parte da aplicação). Desta forma, um *framework* pode utilizar vários padrões para especificar a sua arquitetura.
- Alterações em um *framework* implicam em efeitos sobre a arquitetura do sistema, enquanto que alterações introduzidas por padrões possuem efeitos locais.
- *Frameworks* podem ser reutilizados imediatamente (já estão escritos em uma determinada linguagem de programação), enquanto os padrões precisam ser implementados.
- *Frameworks* são escritos para um determinado tipo de aplicação (domínio) e, por essa razão, são mais especializados do que os padrões.

3 ANÁLISE E PROJETO

Nessa seção será abordado o estudo de caso do presente trabalho: o Sistema Simulador de Provas de Certificação. No domínio do problema proposto, serão tratados e resolvidos problemas pertinentes de requisitos e projeto.

O Sistema Simulador é usado para treinar e medir o conhecimento num determinado exame de certificação dentre os previamente cadastrados na aplicação. Tipicamente utilizado por estudantes que estão se preparando para realizar a prova real de certificação. O Sistema proposto possui um módulo de exames, onde o usuário escolhe o tipo do exame, inicia uma nova prova, fazendo com que o sistema aleatoriamente escolha as questões e gere a prova com questões de múltipla escolha.

Todas as respostas são armazenadas sempre que a prova for finalizada, gerando, dessa forma, informação histórica de cada tentativa do usuário.

Pela concepção inicial, o Sistema possui também um módulo de cadastro (inclusão, alteração e exclusão) de usuários, provas e questões. No entanto, esse módulo não fará parte do escopo do atual trabalho. Essa decisão foi tomada para dar um foco maior nas regras de negócio do Simulador do ponto de visão do funcionamento para quem vai fazer a prova.

A partir de agora será apresentado a documentação UML desde os casos de uso e modelagem de domínio até o projeto de objetos com padrões, fazendo um estudo mais detalhado das decisões tomadas durante essa atividade.

O levantamento de requisitos foi baseado em simulados já existentes, cada um com certas características relevantes que uma vez estudadas e compreendidas, foram selecionadas, resultando na elaboração dos casos de uso principais do sistema.

3.1 ANÁLISE DE REQUISITOS

3.1.1 Diagrama de Casos de Uso

De forma a ilustrar os nomes dos Casos de Uso e dos Atores, bem como os relacionamentos entre eles, o Diagrama de Casos de Uso da aplicação é ilustrado na Figura 9 e dá uma visão geral do Sistema.

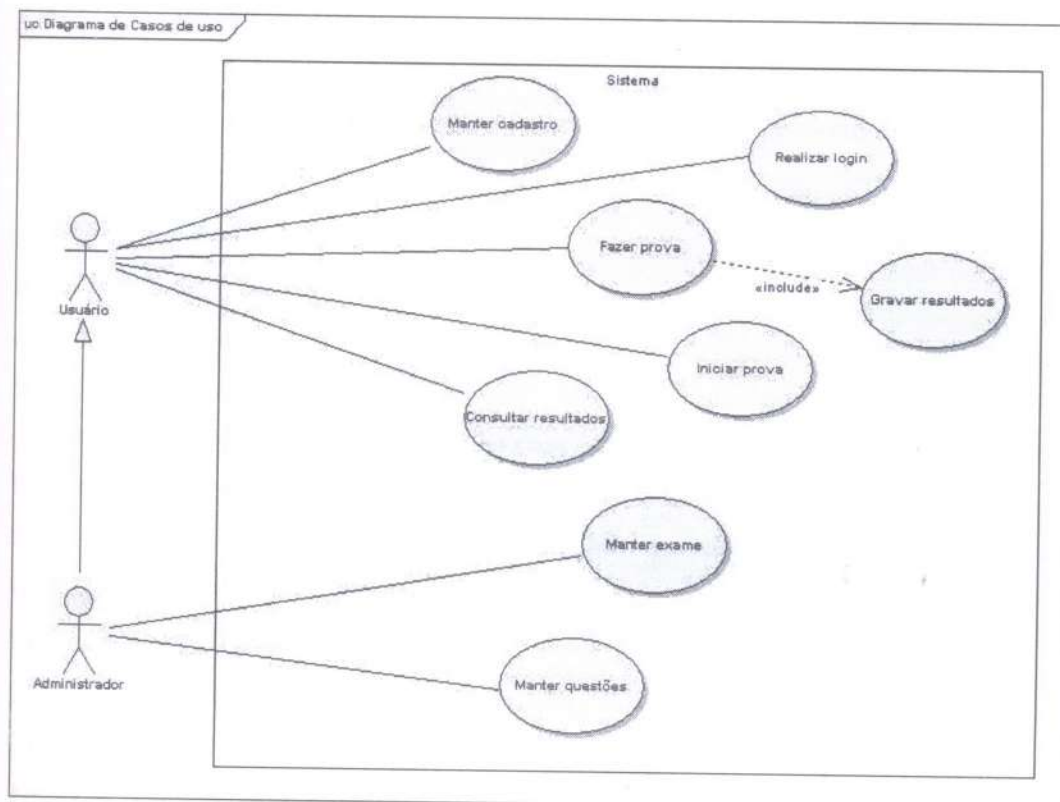


Figura 9: Diagrama de Casos de Uso do Simulador
Fonte: Dados da Pesquisa (2007)

O Diagrama de Casos de Uso acima passa uma imagem do contexto do Sistema e seus principais Atores, onde o foco é sobre as tarefas realizadas pelo ator *Usuário*, mostrando a fronteira existente e deixando claro o que está fora da aplicação.

3.1.2 Documentação dos Casos de Uso

Os Casos de Uso a seguir descrevem todos os passos e variantes dos serviços oferecidos pelo sistema com base nos principais requisitos funcionais, classificados por cenários relacionados entre si. Todos os Casos de Uso usam o ator *Usuário* com ênfase em atingir seu objetivo.

3.1.2.1 Caso de Uso Realizar Login

Descrição

Permite que o Usuário tenha acesso ao Sistema, informando seu *login* e senha.

Escopo: Aplicação Simuladora

Nível: objetivo do Usuário

Ator Principal: Usuário

Interessados e interesses:

-Usuário: Deseja se autenticar no Sistema com seu *login*/senha.

Pré-Condições: Sistema está com a opção de *login* do usuário disponível.

Pós-Condições: Usuário foi autenticado. Opções de iniciar prova e consultar resultados visíveis.

Cenário de Sucesso Principal:

- 1.Sistema disponibiliza tela inicial com a opção de *login* do Usuário;
- 2.Usuário escolhe a opção de entrar no Sistema;
- 3.Sistema apresenta tela de autenticação solicitando *login* e senha;
- 4.Usuário informa seu *login* e senha;
- 5.Sistema verifica dados informados e realiza o *login* do Usuário.

Fluxos Alternativos:

- 1.Sistema autentica negativamente o *login*/senha informados;
- 2.Sistema apresenta novamente a tela inicial com a opção de *login*, informando erro na autenticação.

3.1.2.2 Caso de Uso Iniciar Prova

Descrição

Permite que o Usuário solicite ao Sistema a geração da prova de acordo com o tipo escolhido, com a opção de parametrizar o número de questões da prova.

Escopo: Aplicação Simuladora

Nível: objetivo do Usuário

Ator Principal: Usuário

Interessados e interesses:

-Usuário: deseja escolher numa relação um dos exames disponíveis e opcionalmente informar o número de questões que pretende responder.

Pré-Condições: Usuário está identificado e autenticado.

Pós-Condições: Prova foi iniciada. Primeira questão e tempo correndo visível ao usuário.

Cenário de Sucesso Principal:

- 1.Sistema disponibiliza os tipos de exames existentes e opção de escolha da quantidade de questões;
- 2.Usuário escolhe o tipo de exame e opcionalmente o número de questões da simulação e dispara o início da prova. Caso não informe a quantidade, o valor padrão será 72 questões;
- 3.Sistema escolhe aleatoriamente questões, monta e apresenta a prova para o usuário e inicia a contagem do tempo.

Fluxos Alternativos:

- 1.A qualquer momento usuário clica no botão Sair;
- 2.Sistema apresenta tela inicial com a opção de *login*.

3.1.2.3 Caso de Uso Fazer Prova

Descrição

Permite que o Usuário percorra as questões da prova gerada e marque as respostas, com a opção de selecionar questões para alteração posterior.

Escopo: Aplicação Simuladora

Nível: objetivo do Usuário

Ator Principal: Usuário

Interessados e interesses:

-Usuário: deseja responder a prova. Treinar e medir seu conhecimento, respondendo questões semelhantes às do exame real com a opção de marcar questões para revisar ou responder no final.

Pré-Condições: Executar Caso de Uso Iniciar Prova.

Pós-Condições: Sistema mostra pontuação do Usuário e informa se foi aprovado ou não.

Cenário de Sucesso Principal:

1. Sistema exibe as questões (uma por vez) e o tempo de prova;

- 2.Usuário escolhe as respostas (opcionalmente marca a questão para uma verificação posterior) e avança ou retrocede nas questões livremente;
- 3.Sistema computa a resposta selecionada;
Usuário e Sistema repetem os passos 2 e 3 até o tempo esgotar ou usuário avance após última questão.
- 4.Usuário responde última questão e clica em avançar;
- 5.Sistema exibe quadro de revisão listando as questões com as respostas do Usuário e classificadas como: respondidas, não respondidas ou marcadas para revisão, disponibilizando opções do Usuário alterar questões e finalizar prova.
- 6.Usuário Finaliza a prova;
7. Sistema armazena as informações obtidas na prova, gera os resultados e apresenta-os ao usuário.

Fluxos Alternativos:

- a.A qualquer momento Usuário clica no botão Sair:
 - 1.Sistema apresenta tela inicial com a opção de Login.
- b.A qualquer momento Usuário clica no botão de visualizar progresso:
 - 1.Sistema exibe quadro com resumo de todas as questões classificadas em: respondidas, não respondidas e marcadas;
 - 2.Usuário vê as informações, fecha o quadro e segue fazendo a prova.
- c.Tempo da prova se esgota:
 - 1.Sistema exibe mensagem informando que o tempo esgotou;
 - 2.Usuário confirma a mensagem;
 3. Sistema armazena as informações obtidas na prova, gera os resultados e apresenta-os ao Usuário.
- d.Usuário finaliza a prova com questões não respondidas ou marcadas para revisão:
 - 1.Sistema mostra quadro sinalizando de forma diferente questões não respondidas e marcadas,
 - 2.Usuário escolhe questões para revisar, responder ou alterar resposta e clica em revisar;
 - 3.Sistema apresenta questões escolhidas novamente;
Usuário segue a partir do passo 2 do fluxo principal.

3.1.2.4 Caso de Uso Consultar Resultados

Descrição

Permite que o Usuário consulte os resultados de provas anteriores feitas no Sistema, detalhando informações de porcentagem de acertos e erros de cada prova, classificados por seção (assunto) com a explicação de cada resposta.

Gera um relatório geral de acompanhamento histórico que informa os resultados ao longo do tempo, indicando possíveis melhoras ou pioras do Usuário.

Escopo: Aplicação Simuladora

Nível: objetivo do Usuário

Ator Principal: Usuário

Interessados e interesses:

-Usuário: deseja consultar informações sobre suas provas já realizadas, com opção de classificar dados por assunto ou gerar relatório geral de resultados históricos.

Pré-Condições: Usuário está identificado e autenticado.

Pós-Condições: Relatório contém todos os dados históricos do usuário.

Cenário de Sucesso Principal:

- 1.Usuário seleciona filtro para restringir consulta por período de data ou tipo de exame;
- 2.Sistema lista uma relação de provas realizadas com opção de detalhar cada prova ou gerar relatório com a evolução do usuário por assunto;
- 3.Usuário escolhe qual relatório deseja ver,
- 4.Sistema gera relatório das provas.

Fluxos Alternativos:

a.A qualquer momento Usuário clica no botão voltar tela de relação:

- 1.Sistema exibe tela de filtro da consulta.

b.A qualquer momento Usuário clica no botão Sair:

- 1.Sistema apresenta tela inicial após o *login*.

c.Usuário escolhe uma prova específica da relação:

- 1.Sistema apresenta quadro detalhando a prova, com a pontuação e a explicação de cada resposta.

3.1.3 Modelo de Domínio

O Modelo de Domínio abaixo mostra os principais conceitos do domínio da aplicação. Identificando classes conceituais relacionadas com os requisitos, tem-se uma útil representação visual de objetos do sistema simulador no mundo real.

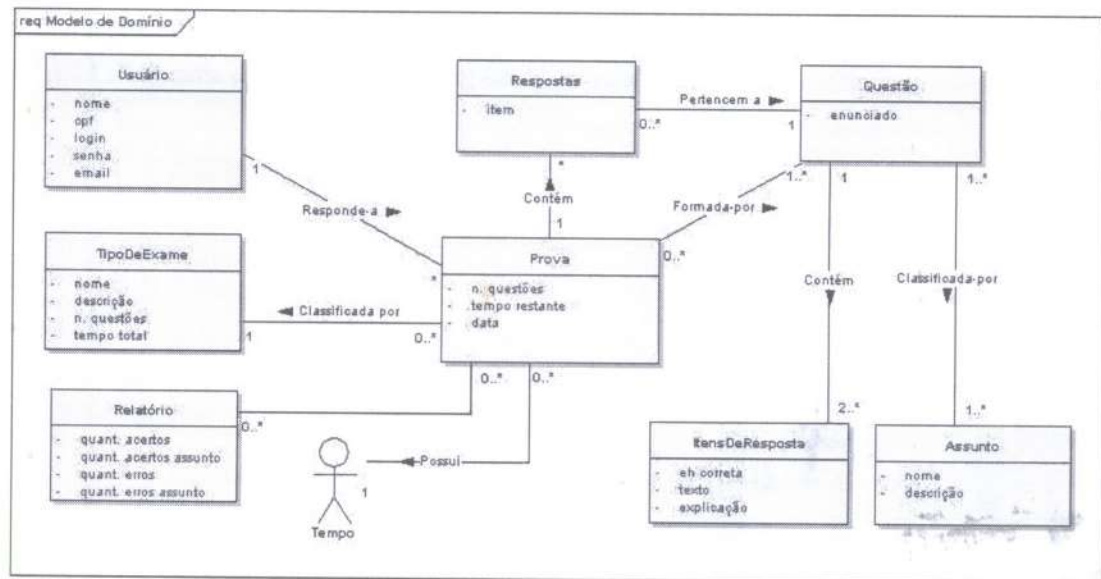


Figura 10: Modelo de Domínio do Sistema Simulador
Fonte: Dados da Pesquisa (2007)

A utilização do Modelo Conceitual acima é justificada pela diminuição do hiato representacional, que é a tentativa do analista de deixar seu trabalho o mais parecido possível com o modelo mental pensado, tentando diminuir a distância entre o mundo real e sua representação no Sistema.

As estratégias usadas para construir tal modelo foram basicamente duas: identificar substantivos no texto dos casos de uso e reusar idéias de modelos já publicados.

3.1.4 Diagramas de Sequência

Nos Diagramas de Sequência do Sistema Simulador os principais cenários identificados na análise são descritos de forma dinâmica. As mensagens passadas do Usuário para o Sistema identificam a ocorrência de eventos que mais adiante servirão como artefatos

de entrada na fase de projeto. Nas seções abaixo, os Diagramas de Seqüência representam a temporalidade dos passos descritos nos cenários de Casos de Uso da aplicação.

3.1.4.1 Realizar Login

Cenário onde o Usuário se autentica para entrar no sistema informando seu login e senha. O Diagrama de Seqüência da Figura 11 ilustra essa ação.

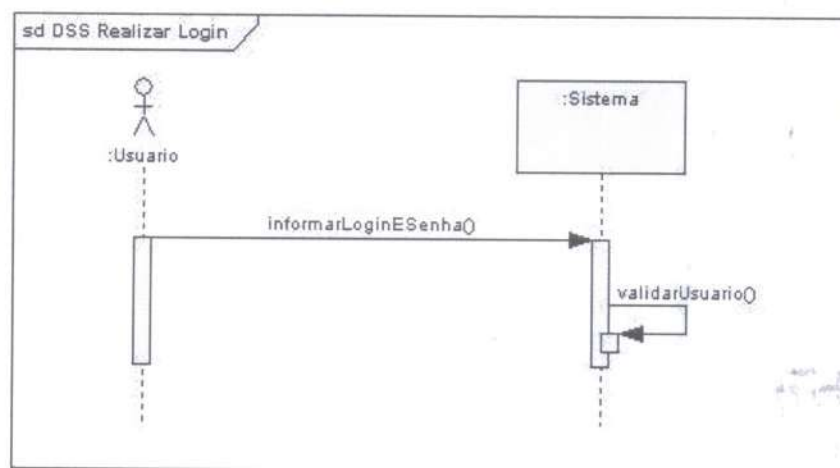


Figura 11: Diagrama de Seqüência Realizar Login
Fonte: Dados da Pesquisa (2007)

3.1.4.2 Iniciar Prova

Cenário onde o Usuário escolhe um tipo de exame e inicia uma prova do exame selecionado. O diagrama seguinte mostra visualmente os eventos.

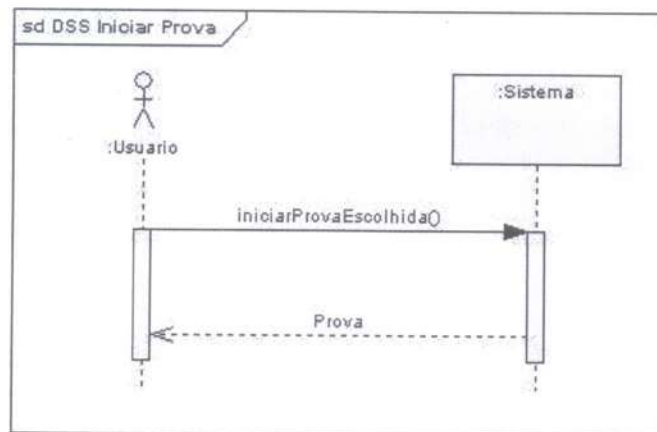


Figura 12: Diagrama de Seqüência Iniciar Prova
Fonte: Dados da Pesquisa (2007)

3.1.4.3 Fazer Prova

Cenário onde o Usuário responde questões de uma prova e no final um resultado é apresentado indicando se o Usuário teve sucesso ou não no teste. Abaixo, o diagrama da situação descrita.

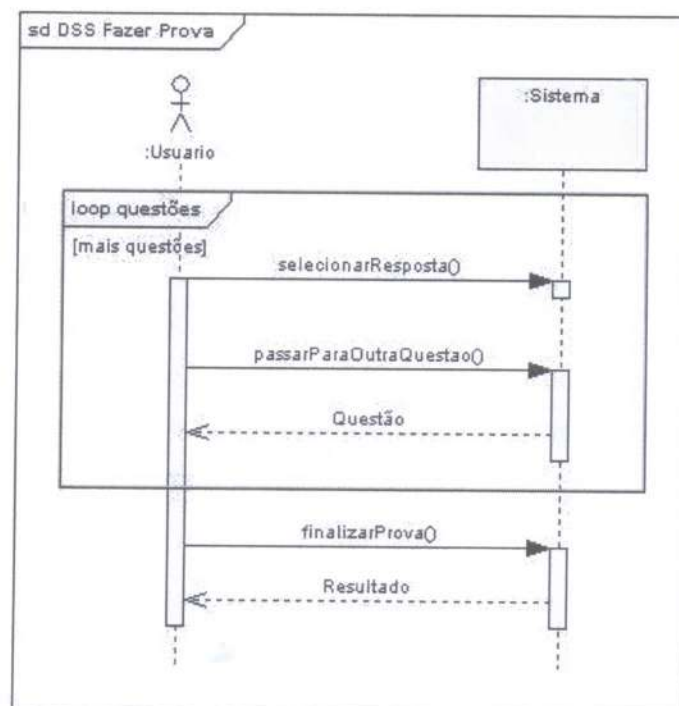


Figura 13: Diagrama de seqüência Fazer Prova
Fonte: Dados da Pesquisa (2007)

3.1.4.4 Consultar Resultados

Cenário onde o Usuário realiza uma consulta, aplicando um filtro para listar uma relação de provas já feitas, com a opção de ver um relatório detalhado do desempenho do Usuário ao longo das tentativas. Abaixo na Figura 14 o diagrama que reflete o descrito acima.

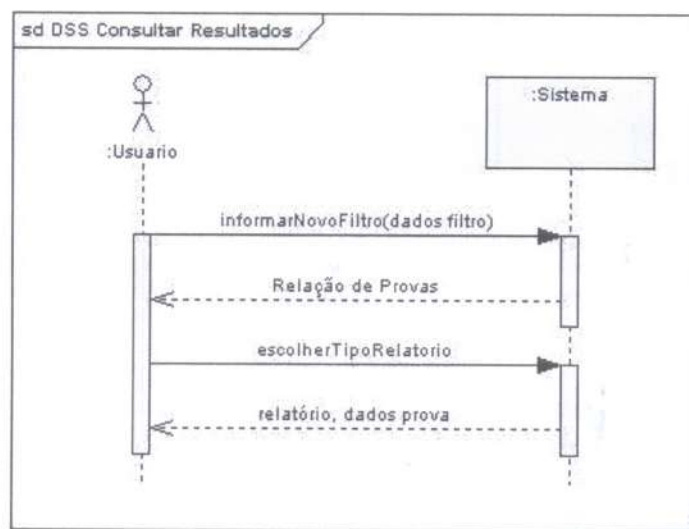


Figura 14: Diagrama de Seqüência Consultar Resultados
Fonte: Dados da Pesquisa (2007)

3.2 PROJETO DE OBJETOS

O projeto de objetos do Sistema escrito foi montado baseando-se nos principais princípios GRASP e padrões GoF.

Para implementar a atribuição de responsabilidades e o projeto de colaborações foram usados vários princípios GRASP como: especialista na informação, controlador, criador, acoplamento baixo e invenção pura e alguns padrões GoF como: *singleton*, *abstract factory*, *façade*, *DO (Data Object)* e *DAO (Data Access Object)*.

3.2.1 Diagrama de Classes

Na Figura 15 pode-se ver o Diagrama de Classes do Sistema. Estão representados todos os elementos de domínio.

As classes de Business foram usadas para separar os dados e lógica de negócio. Essas classes fazem o encapsulamento das regras de negócio. O gerenciamento da camada de persistência é feito pelas classes DAO.

As justificativas para o uso de vários cada padrão serão apresentadas em seguida.

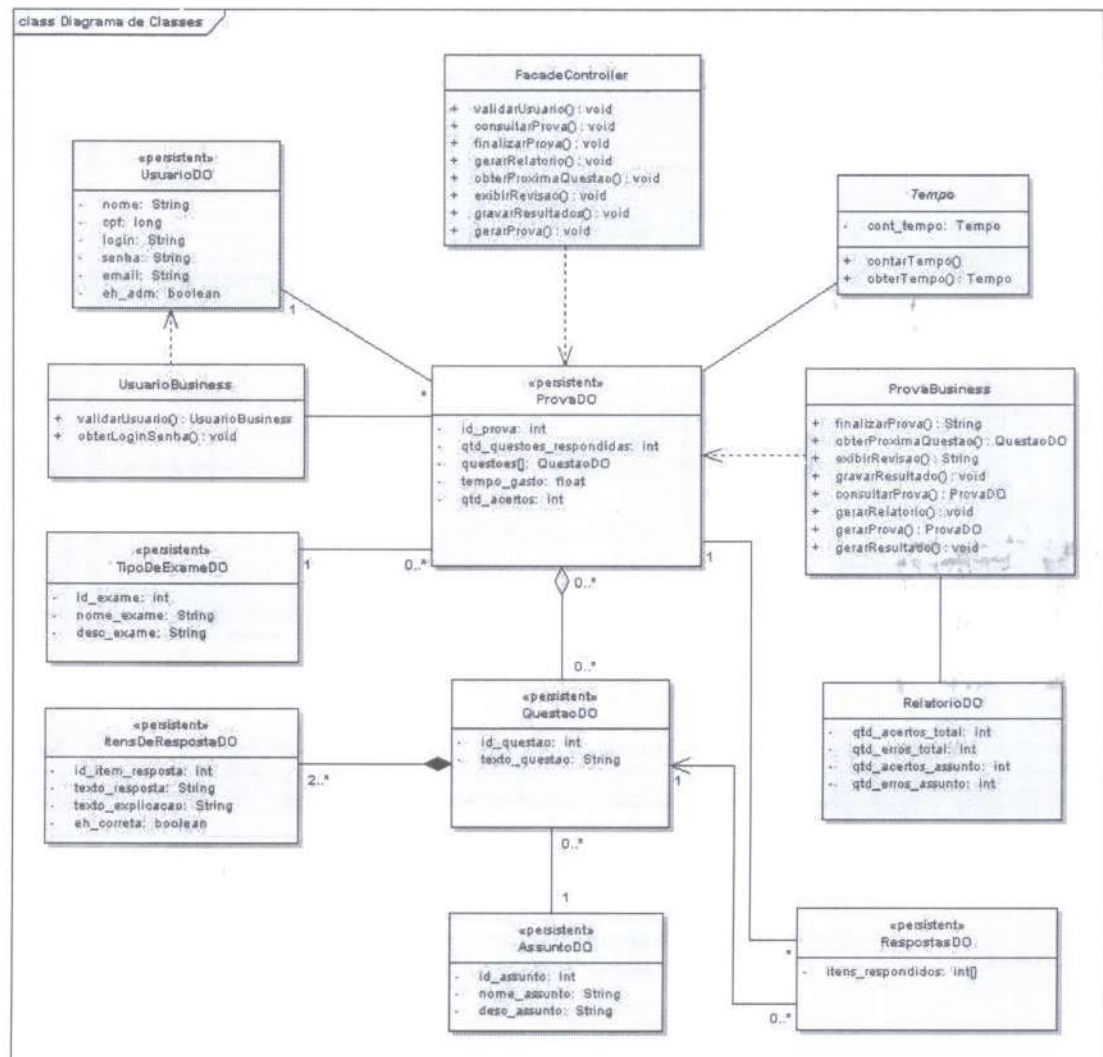


Figura 15: Diagrama de Classes da Aplicação
Fonte: Dados da Pesquisa (2007)

O primeiro princípio usado no projeto foi o *controller* que identifica o primeiro objeto depois da interface que gerencia uma operação do Sistema e delega a outros objetos o serviço que precisa ser feito.

Na aplicação foi usado um controlador fachada, que é uma espécie de fachada, chamado de *facade controller*.

Outro princípio utilizado no projeto com frequência é o Especialista, que sugere atribuir responsabilidade a classe que tem a informação necessária para satisfazer tal responsabilidade. Podem-se citar como exemplo as operações da classe Prova que, segundo esse padrão, contém as informações sobre as questões apresentadas ao Usuário.

Para evitar a poluição visual do Diagrama de Classes do Sistema, a parte que gerencia a persistência foi desenhada em outro Diagrama de Classes, que pode ser visto na Figura 16.

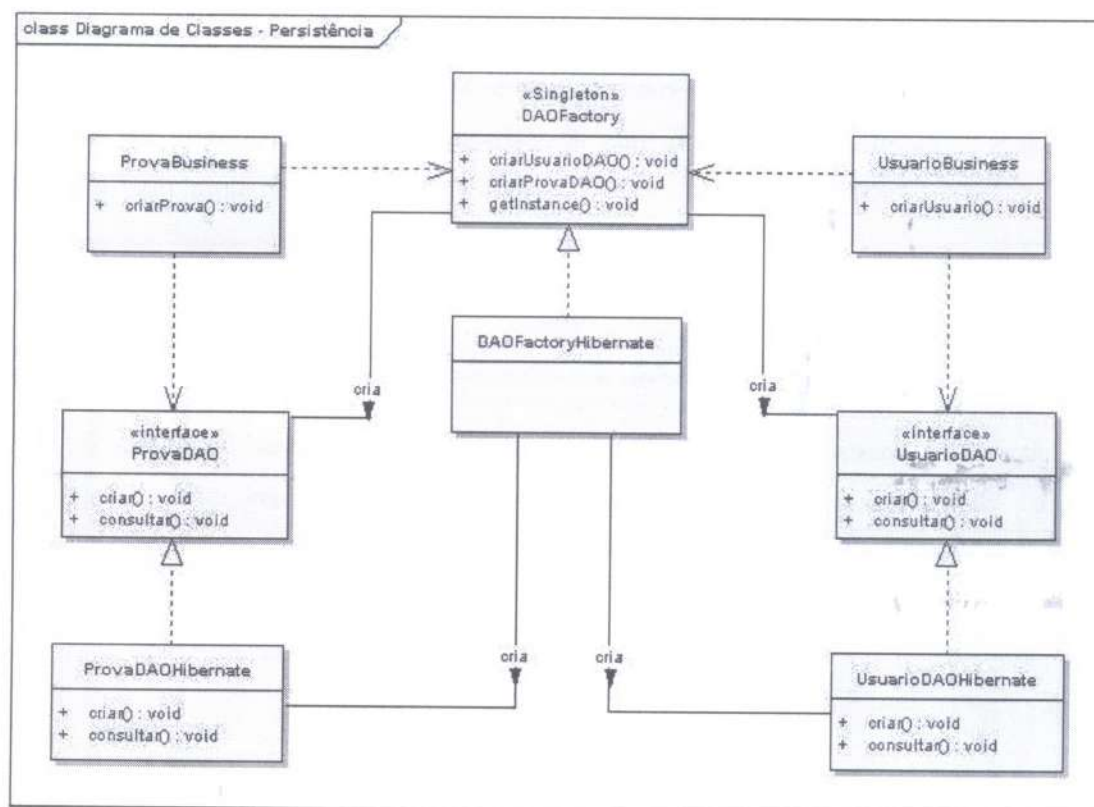


Figura 16: Diagrama de Classes mostrando a camada de persistência
Fonte: Dados da Pesquisa (2007)

Através da utilização do padrão *Data Access Object* (DAO) foi possível extrair e encapsular todos os acessos à origem de dados em uma única camada.

As classes DAO gerenciam a conexão com a fonte de dados para obter e armazenar os dados do Sistema.

3.2.2 Diagramas de Seqüência

Nessa seção são apresentados os Diagramas de Seqüência utilizados no Projeto de Objetos do Sistema Simulador.

Nota-se o uso de alguns princípios GRASP, como a escolha do objeto *Façade Controller* como controlador, por ser o primeiro objeto além da camada de interface que recebe e controla as operações do Sistema.

Outros princípios interessantes que valem a pena citar são: *acoplamento baixo e especialista* que podem ser observados no uso de classes que isolam o acesso ao repositório de dados, deixando o trabalho de persistência para classes que conhecem a informação necessária para implementar a responsabilidade.

3.2.2.1 Realizar Login

Diagrama de Seqüência que ilustra o cenário de entrada no Sistema, mostrando como os objetos interagem entre si para validação do *login/senha* do usuário.

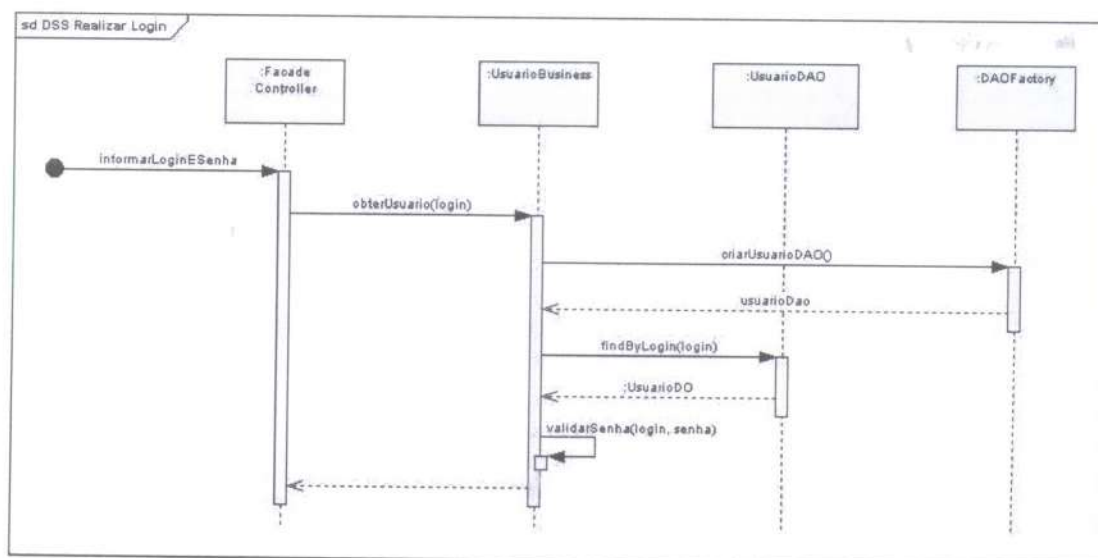


Figura 17: Diagrama de Seqüência para cenário Realizar Login
Fonte: Dados da Pesquisa (2007)

3.2.2.2 Iniciar Prova

Cenário onde o usuário escolhe um novo exame e, em seguida, inicia uma prova, disparando a contagem do tempo.

Foi usada uma fachada (controlador) para gerenciar a solicitação da prova. O responsável pela geração é o *ProvaBusiness*, objeto de negócio fortemente ligado ao objeto *ProvaDO*.

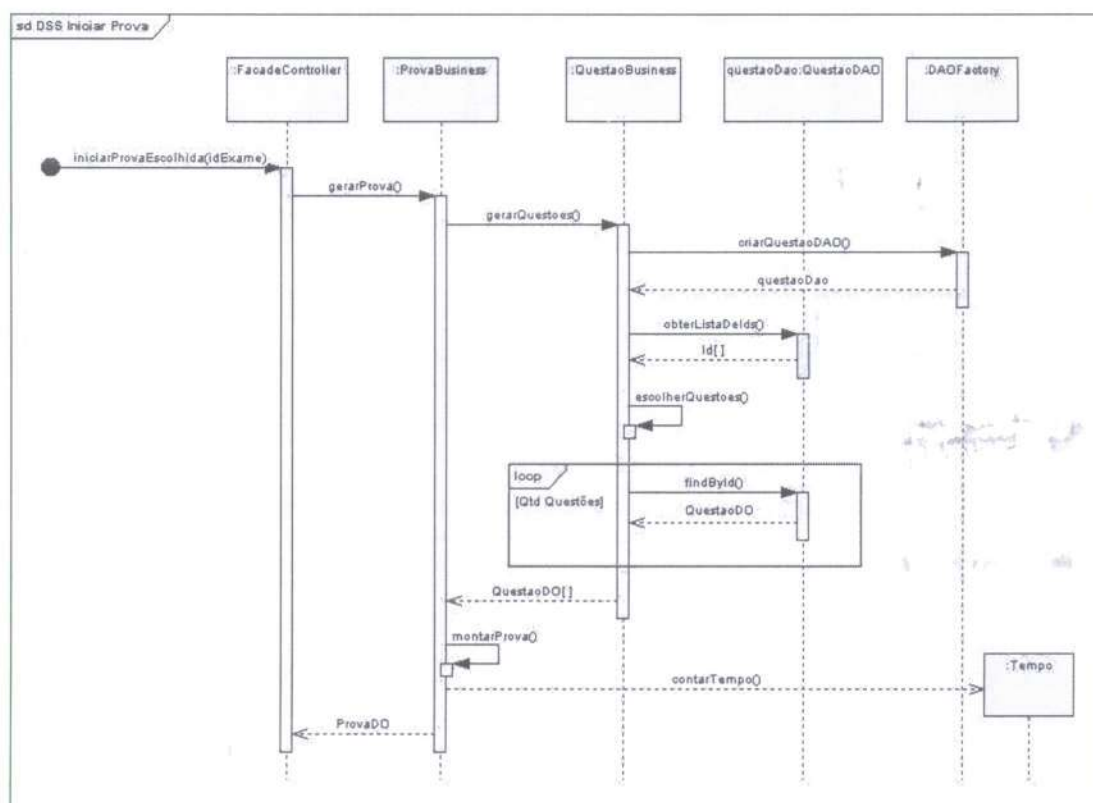


Figura 18: Diagrama de Seqüência para cenário Iniciar Prova
Fonte: Dados da Pesquisa (2007)

3.2.2.3 Fazer Prova

O cenário da Figura 18 abaixo mostra a comunicação entre os objetos que participam da realização da prova pelo Usuário.

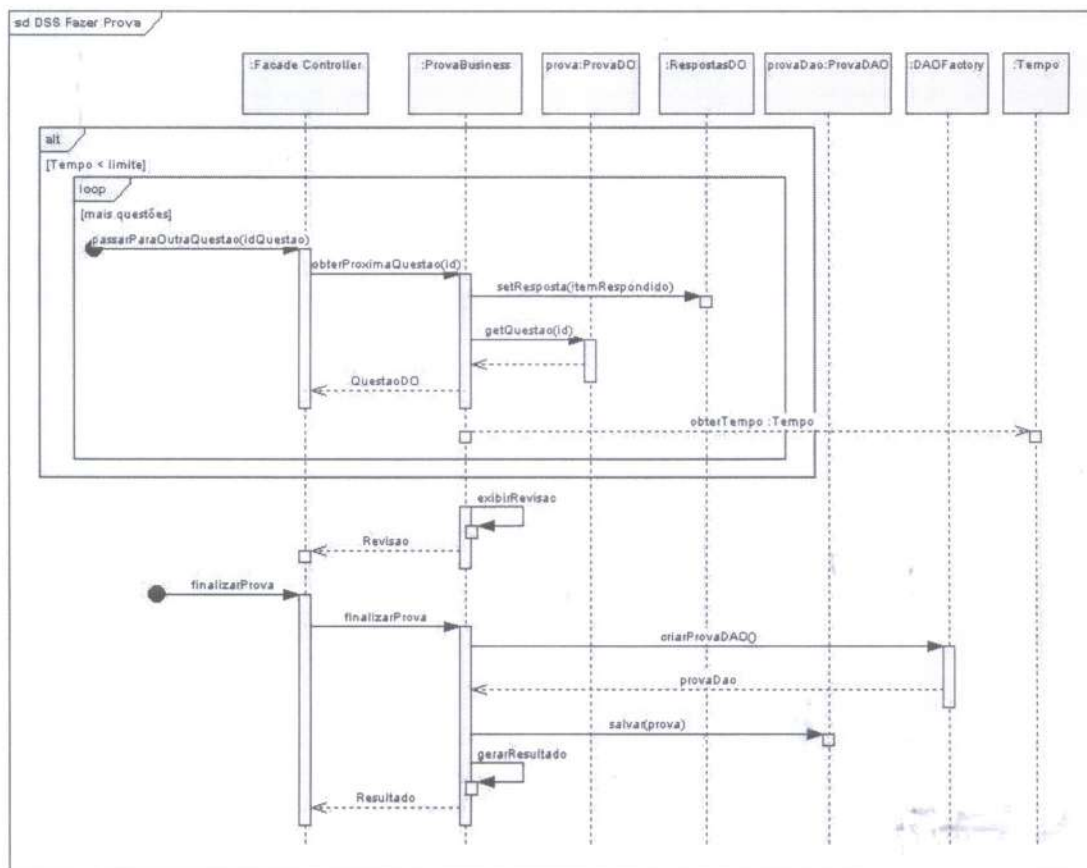


Figura 19: Diagrama de Seqüência para cenário Fazer Prova
Fonte: Dados da Pesquisa (2007)

3.2.2.4 Consultar Resultados

O cenário no Sistema para recuperação de provas já realizadas e a geração de relatórios para exibição de informações da progressão nas provas é observado na Figura 19 abaixo.

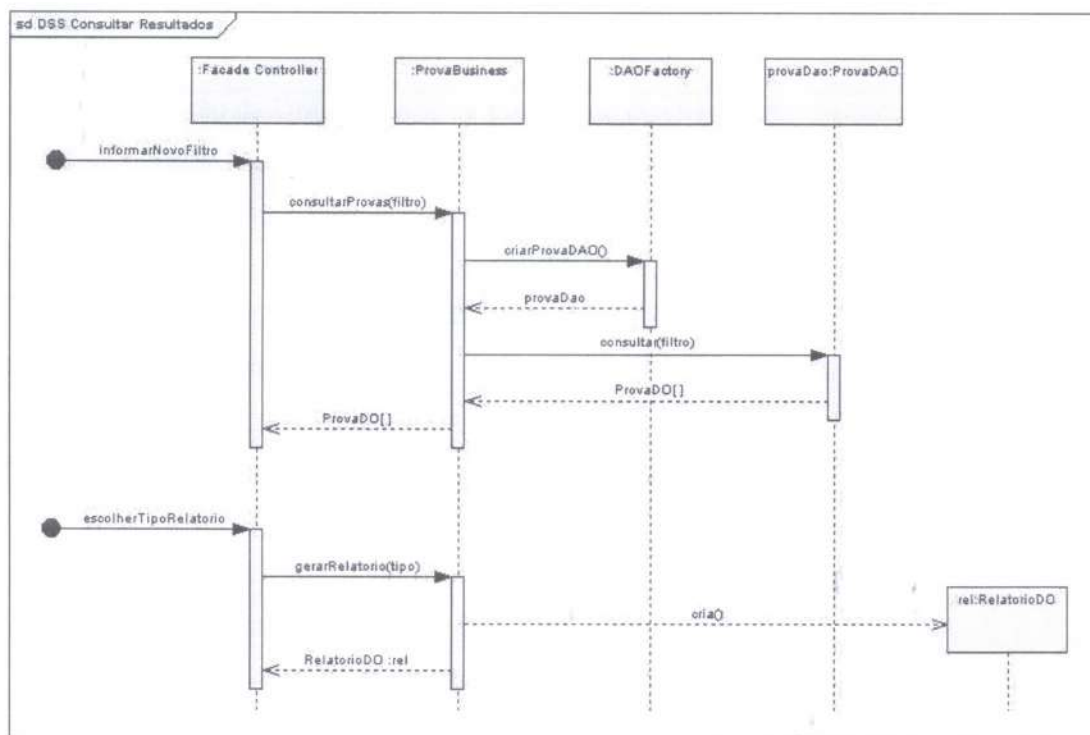


Figura 20: Diagrama de Seqüência para cenário Consultar Resultados
 Fonte: Dados da Pesquisa (2007)

4 CONCLUSÃO

A utilização da UML, através da geração de diagramas que documentam e transmitem uma idéia bem próxima do funcionamento do Sistema, tornou mais clara a visualização e compreensão da comunicação entre os objetos da Aplicação e de como esses devem ser implementados.

A atribuição de responsabilidades no Projeto e o uso de padrões GRASP e GoF garantiram maior modularização e facilidade de reutilização de código, com a diminuição do acoplamento e uma maior coesão entre os objetos.

Foi possível, usando técnicas simples de levantamento de requisitos, identificar as principais características dos simuladores existentes e reuni-las na Aplicação proposta nesse trabalho, construindo um Projeto Orientado a Objetos que pode ser usado em uma futura implementação do Sistema Simulador de Provas.

Uma das dificuldades encontradas foi a transição da análise de requisitos OO para o Projeto de Objetos do Simulador, tendo em vista que escolher e adicionar princípios e padrões para montar uma arquitetura com características que melhorem o reuso e facilitem a manutenção do Sistema é uma tarefa trabalhosa e os objetos da fase de análise são escolhidos de forma a representar entidades do mundo real, não existindo uma ligação de um-para-um com as classes usadas no Projeto do Sistema.

Portanto, embasado nos objetivos propostos e nos resultados apresentados, conclui-se que o uso de técnicas de análise e projeto orientadas a objetos simplificou o entendimento do problema e apontou para uma solução de fácil manutenção e diminuição dos impactos na adição de novos módulos ao Sistema Simulador.

REFERÊNCIAS

Análise e Projeto de Sistemas Orientados a Objeto. Disponível em <<http://www.dsc.ufcg.edu.br/~jacques/cursos/apoo/html/apoo2.htm>>. Acessado em 15 dez. 2007.

BOOCH, Grady et al. **The unified modeling language user guide**. Massachusetts Addison-Wesley, 1998.

COCKBURN, A. **Writing Effective Use Cases**. Addison-Wesley, 2000.

Diagrama de Caso de Uso. Disponível em <http://www.unimep.br/~aeasilva/topico3_AOO.pdf>. Acessado em 04 dez. 2007.

Diretrizes: Caso de Uso. Disponível em <http://www.wthreex.com/rup/process/modguide/md_uc.htm>. Acessado em 04 nov. 2007.

Fundamentos do UML. Disponível em <http://docs.kde.org/stable/pt_BR/kdesdk/umbrello/uml-basics.html>. Acessado em 12 nov. 2007.

GAMMA, Erich et al. **Padrões de projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2000.

GUEDES, Gilleanes. **UML: uma abordagem prática**. São Paulo: Novatec, 2006.

LARMAN, Craig. **Utilizando UML e padrões: introdução à análise e ao projeto orientado a objetos e ao desenvolvimento iterativo**. Porto Alegre: Bookman, 2007.

SIERRA, Kathy. **Certificação Sun para Programador Java**. Rio de Janeiro: Alta Books, 2006.

Sun Microsystems Home. Disponível em <<http://www.sun.com/training/certification/faq/#java>>. Acessado em 04 nov. 2007.

UML – Wikipédia. **UML**. Disponível em <http://pt.wikipedia.org/wiki/UML#%20Objetivos_da_UML>. Acessado em 05 dez. 2007.

Unified Model Language. Disponível em <<http://www.uml.org>>. Acessado em 15 dez. 2007.