

**Universidade Federal do Rio Grande do Sul
Instituto de Informática
Bacharelado em Ciência da Computação**

**Estudo de uma Metodologia
de Modelagem Orientada a Objetos
Aplicada à Implementação de um
Protótipo de Sistema de Apoio a
Avaliação de Conteúdos**

Ademar Latorre Júnior

**Prof. Roberto Cabral de Mello Borges
Orientador**



Porto Alegre, Julho 1997.

**UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA**

AGRADECIMENTOS

Antes de mais nada agradeço à todos aqueles que acreditaram em mim e me apoiaram na minha decisão de adquirir na Universidade não só o conhecimento, mas uma vivência bastante diversificada entre pessoas convictas de suas potencialidades e com pensamentos divergentes em relação à vida.

Agradeço aos meus amigos André A. Parmegiani, Cássio Luís Bastos, Carlos Alex Dávila de Ávila, Walter Jeck Júnior, por serem quem são e terem me apoiado neste final de curso e me darem a chance de retribuí-los com esse trabalho.

Agradeço muito ao meu orientador, Prof. Roberto Cabral de Mello Borges pelo interesse em meu trabalho desde o início e sempre se por inteiramente à disposição para eventuais dúvidas e problemas.

À Eliene, funcionária do Instituto de Informática, que mais do que ninguém me ajudou na preparação dos documentos e na orientação dos alunos formandos.

Agradeço a todos os colegas que, mesmo distantes, estejam torcendo por mim.

Agradeço ao meu atual chefe e amigo Alexandre de Moura Gomes por ter tornado possível a matrícula nas últimas cadeiras que me restavam cursar e por me incentivar na conclusão do Curso.

Finalmente agradeço à todos aqueles que passaram por mim como amigos ou colegas, interferindo de uma forma positiva na minha formação.

SUMÁRIO

Agradecimentos.....	2
Abstract	5
Resumo.....	6
1. Introdução	7
2. Paradigma de Orientação a Objetos.	8
2.1. Conceitos Básicos.....	8
2.1.1. Objeto.....	8
2.1.2. Mensagem	8
2.1.3. Método	9
2.1.4. Classe	9
2.1.5. Instância	10
2.1.6. Herança	10
2.1.7. Agregação	11
2.1.8. Generalização e Especialização	11
2.2. Metodologias Orientadas a Objetos.....	12
2.3. As Partes da Linguagem Java.....	13
2.3.1. Tokens.....	13
2.3.2. Tipos.....	13
2.3.3. Expressões.....	13
2.3.4. Instruções	14
2.3.5. Classes.....	15
2.3.6. Interfaces	15
2.3.7. Pacotes	15
3. O que é o método Fusion?	15
3.1. O Processo do Método Fusion.....	16
3.1.1. Fase de Análise	16
3.1.1.1. Desenvolvimento do Modelo de Objetos	17
3.1.1.2. Desenvolvimento Interface do Sistema	18
3.1.1.3. Desenvolvimento do Modelo de Interfaces	20
3.1.1.4. Verificar os Modelos de Análise	23
3.1.2. Fase de Projeto.....	25
3.1.3. Fase de Implementação.....	25
4. Apresentação do problema para aplicação	25
5. Solução do problema utilizando o método Fusion.	26
5.1. Modelos de Objetos do Sistema	27
5.2. Cenários do Sistema	31
5.3. Modelos de Ciclo-de-vida	33
5.4. Esquemas do Sistema	35
5.4.1. Esquema para o evento Aplicar_exercício.....	35

5.4.2. Esquema para o evento Consultar exercício	35
5.4.3. Esquema para o evento Aplicar_prova	36
5.4.4. Esquema para o evento Cadastrar_professor	36
5.4.5. Esquema para o evento Cadastrar_aluno	37
5.4.6. Esquema para o evento Corrigir_prova.....	37
5.4.7. Esquema para o evento Alterar_prova	38
5.4.8. Esquema para o evento Alterar_exercício.....	38
5.5. Diagrama de Estados do Sistema	39
6. Dicionário de Dados.....	44
6.1. Dicionário de Dados: Sistema	44
6.2. Dicionário de Dados: Interface do Sistema	46
6.3. Dicionário de Dados: Agentes.....	48
7. Tabelas do Sistema	49
8. Telas do sistema	53
8.1. Tela principal do sistema.....	53
8.2. Tela do módulo administrador	53
8.3. Tela de consulta de aluno	54
8.4. Tela de consulta de professor	54
8.5. Tela do módulo aluno.....	55
8.6. Tela de aplicação de exercício.....	55
8.7. Tela de aplicação de prova	56
8.8. Tela do módulo professor.....	56
8.9. Tela para alteração de exercício	57
8.10. Tela para alteração de prova.....	57
8.11. Tela para correção de prova.....	58
9. Softwares utilizados para elaboração do protótipo	59
10. Dificuldades Encontradas na construção do protótipo	60
11. Conclusões	61
Bibliografia.....	62

ABSTRACT

The systems development is a very fascinating activity, but it still being made in a confusing way or without a methodology that comprise since the requirement document creation until his implementation. The thuth is that the existing funcional metodologies become impractical when applied in complex systems on what refers to his inspection and maintenance.

To make the process more simple was created object oriented systems development, becoming this process easier and more intuitive. The main goal of this paper is to present one of these metodologies and apply that in a problem described later.

RESUMO

O desenvolvimento de sistemas é uma atividade sedutora, mas até hoje é realizada de maneira desorganizada ou sem uma metodologia que abrangesse o problema desde a criação do documento de requisitos do sistema até a sua implementação. A verdade é que as metodologias funcionais existentes se tornam inviáveis quando aplicadas a sistemas muito complexos no que tange a sua inspeção e manutenção.

A fim de tornar esse processo mais simples foram criadas metodologias para desenvolvimento de sistemas orientadas a objetos, tornando esse processo mais fácil e intuitivo. A finalidade principal deste trabalho é apresentar uma destas metodologias e aplicá-la a um problema que será descrito mais adiante.

1. INTRODUÇÃO

A motivação maior desse trabalho surgiu na vontade de expandir meus conhecimentos de análise em relação a metodologias de desenvolvimento orientada a objetos.

A finalidade desse trabalho é a apresentação do método Fusion para o desenvolvimento de sistemas utilizando o paradigma de orientação a objetos. A fim de aplicarmos essa metodologia no desenvolvimento de sistemas, durante a sua explanação faremos a análise de um sistema cujo protótipo será apresentado no final dessa exposição.

Esse trabalho foi dividido em três partes. Primeiramente, apresentaremos conceitos básicos de orientação a objetos (OO), metodologias OO e linguagens de programação que utilizam esse paradigma. Em uma segunda etapa, será definido o método FUSION e finalmente, será proposto o problema para aplicação seguido da solução para esse problema com o método FUSION e sua posterior implementação.

2. PARADIGMA DE ORIENTAÇÃO A OBJETOS.

2.1. CONCEITOS BÁSICOS

A seguir é apresentado uma definição de alguns termos básicos aplicados em orientação a objetos. O paradigma de programação orientada a objetos permite, devido a sua constante evolução, que abordemos esses conceitos de uma forma adaptada ao método utilizado no processo de desenvolvimento do sistema. O método utilizado nesse trabalho para a modelagem orientada a objetos é o método *FUSION*.

2.1.1. OBJETO

Um objeto representa um elemento que pode ser identificado de maneira única e exclusiva. Não temos condições, em uma etapa inicial do processo de análise, definirmos se tal elemento será um objeto, classe, relacionamento ou um simples atributo (definições que trataremos mais adiante). Um objeto, por sua vez, pode possuir um ou mais valores por ele associados. Em outras palavras, um objeto pode possuir diversos campos que armazenam valores, denominados atributos, cujos valores poderão ser alterados, sem que a quantidade e os seus nomes não sejam modificados.

Exemplo de uma classe Professor com os atributos nome e área, contendo os seguintes objetos:

Professor	
Nome	Código
Paulo	1123
Carlos	3452
Denise	2435

2.1.2. MENSAGEM

É o mecanismo pelo qual um objeto pode comunicar-se com outros e através do qual é desencadeada a execução dos métodos.

Uma mensagem é composta pela indicação de receptor, ou seja, o objeto destinatário da mensagem, pelo seletor do método a ser executado e, opcionalmente, por uma lista de parâmetros requeridos pelo método selecionado.

O receptor responderá à mensagem através da seleção e execução de um método componente de seu comportamento. O objeto emissor da mensagem interrompe sua execução e aguarda o retorno do controle a partir do receptor.

Enquanto não recebe nenhuma mensagem, um objeto pode comportar-se de duas maneiras distintas: pode permanecer inativo, aguardando a chegada de uma mensagem, sendo denominado objeto passivo; ou pode permanecer executando algo até ser interrompido pela recepção de uma mensagem, sendo denominado objeto ativo.

2.1.3. MÉTODO

Consiste na descrição de uma ação, componente do comportamento associado a um ou mais objetos, que executam um conjunto de operações sobre seu estado interno.

Métodos são ativados através da recepção de mensagens, enviadas explicitamente por outro objeto. Um método é formado por uma interface, ou seja, a descrição do tipo de ação executada pelo método, e por sua implementação, onde descreve-se como o faz.

Exemplo de método cuja descrição seria: Um professor (na verdade um objeto da classe Professor) corrige (altera o estado interno) a prova (dos objetos da classe Prova). Corrige, portanto, seria a designação de um método.



Código em Java que declara o método isEmpty:

```

public synchronized boolean isEmpty(int x, int y) {
    if (board[x][y] != EMPTY)
        return(true);
    return(false);
}
  
```

2.1.4. CLASSE

Os objetos são agrupados em conjuntos denominados classes. Uma classe é uma abstração que representa a idéia ou noção geral de um conjunto de objetos similares. Cada classe possui um predicado que irá definir o critério para a inserção de um novo objeto na mesma. Nesse trabalho iremos adotar a convenção de sempre ser utilizada uma letra maiúscula para denotarmos os nomes das classes. Enquanto a

referência a uma classe com letras minúsculas denotará apenas um membro da classe. Graficamente, uma classe é representada por um retângulo com seu nome colocado na parte superior, separado por uma linha como representado na figura.

Professor
Nome
Código
Senha

Código em Java que define uma classe:

```
public class Teste{
    public static void main (String Args[] ) {
        System.out.println ("Teste");
    }
}
```

2.1.5. INSTÂNCIA

É a materialização de um dos objetos descritos por uma classe. Toda instância deve estar associada a uma classe.

Exemplo:

Professor	
Nome	Código
Paulo	1123
Carlos	3452
Denise	2435

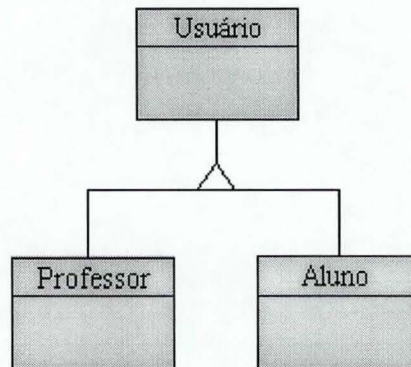
Onde as tuplas (Paulo, 1123; Carlos,3452 e Denise, 2435) são instâncias da classe Professor.

2.1.6. HERANÇA

É o mecanismo que permite definir uma hierarquia de especializações (*Generalizações e Especializações*), de forma que uma classe mais especializada herde as propriedades (*variáveis, interface e métodos*) da classe mais geral a que ela

subordina-se na hierarquia. A classe mais geral é denominada *super-classe* e a mais especializada, *subclasse*.

Exemplo de Especialização onde as subclasses (Professor e Aluno) poderiam herdar da super-classe Usuário os atributos nome, código ou até mesmo métodos aplicados a esta mesma super-classe, tais como é_cadastrado.



2.1.7. AGREGAÇÃO

A agregação é um mecanismo para estruturar o modelo de objetos. Permite a construção de uma classe agregada a partir de outras classes componentes. As agregações são similares aos relacionamentos, pois ambos são formados ao se considerar tuplas de instâncias de classes. Uma agregação pode ser usada para “encapsular um relacionamento. Nesse caso, as tuplas da classe agregada deve respeitar o relacionamento nela contido.

2.1.8. GENERALIZAÇÃO E ESPECIALIZAÇÃO

A generalização permite que uma classe, denominada classe supertipo, seja formada pela fatoração das propriedades comuns de várias classes, denominadas subtipos. Especialização representa o caso inverso, onde um novo subtipo será definido na forma de uma versão mais especializada de um supertipo. Em uma generalização, os atributos e os relacionamentos pertencentes ao supertipo serão “herdados” por todos os subtipos. Cada subtipo pode possuir atributos adicionais e participar de outros relacionamentos. A generalização possui a propriedade importante de que todos os objetos de um subtipo também pertencem ao supertipo. A notação para a generalização de uma classe utiliza um triângulo preenchido, o qual conecta esta classe supertipo a seus subtipos. Os subtipos particionam o supertipo, ou seja, os subtipos são disjuntos entre si, e sua união é o supertipo.

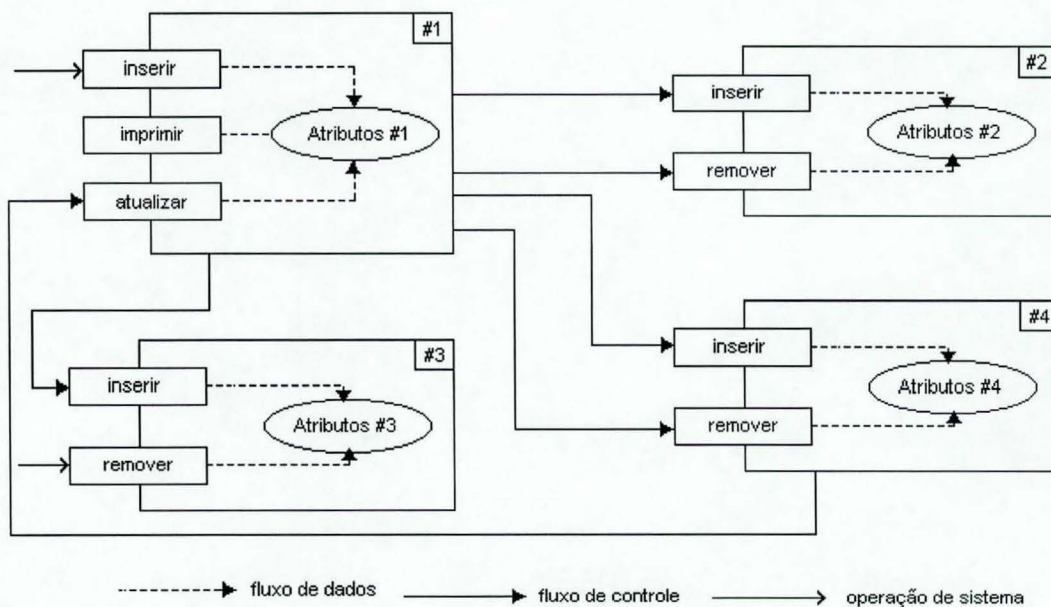
2.2. METODOLOGIAS ORIENTADAS A OBJETOS

Orientação a objetos é uma abordagem de programação que procura explorar o nosso lado intuitivo. Os “átomos” da computação orientada, os objetos, são análogos aos objetos existentes no mundo físico. Isso propõe um modelo de programação que difere profundamente da tradicional visão “funcional”. Se por um lado a orientação a objetos se mostra produtiva, tanto na teoria quanto na prática, por outro lado, ela não combina com métodos mais tradicionais de desenvolvimento de sistema.

Atualmente existem vários métodos diferentes específicos para o desenvolvimento de sistemas orientados a objetos. Esses métodos, poderiam ser denominados, métodos de primeira geração por terem surgido pela aplicação de noções de orientação a objetos nos métodos não orientados a objetos existentes.

No enfoque orientado a objetos, os átomos do processo de computação são os objetos, que trocam mensagens entre si. Estas mensagens resultam na ativação de métodos, os quais realizam as ações necessárias.

MODELO COMPUTACIONAL ORIENTADO A OBJETOS



2.3. AS PARTES DA LINGUAGEM JAVA

O principal motivo que me levou a escolher o Java para implementar esse sistema é que ele utiliza o paradigma de orientação a objetos. Isto porque eu estava interessado em conhecer mais sobre modelagens orientadas a objetos e poderia ser um problema se eu utilizasse uma linguagem procedural para esse fim, visto que o modelo computacional para esses dois paradigmas são completamente diferentes. Por outro lado, eu não iria perder a chance de conhecer a linguagem mais divulgada na atualidade.

Outro motivo, e já prevendo o futuro, de ter escolhido o Java, é por esta linguagem ser multiencadeamento, podendo disparar vários processos em paralelo, e distribuída, o que nos permite prever antes mesmo do início da análise uma maior interação a nível de rede.

A utilização de uma ferramenta de programação, por outro lado, me ajudou no sentido de ter mais tempo para estudar a funcionalidade do sistema, não me preocupando muito em como implementar, o que me parece uma tendência irreversível.

A seguir será apresentada uma visão geral das partes que compõem a linguagem Java: Token, Tipos, Expressões, Instruções, Classes, Interfaces e Pacotes.

2.3.1. TOKENS

Para que o computador entenda o código desenvolvido pelo programador é necessário determinar quais símbolos ou tokens estão expressos no código. Assim como na maioria das linguagens de programação, no Java existem cinco tipos de Tokens: identificadores, palavras-chave, literais, operandos e separadores. O código poderá, também, conter espaços em branco e comentários para facilitar a sua compreensão.

2.3.2. TIPOS

Na linguagem de computador, o tipo refere-se à maneira como um objeto primitivo é representado e armazenado na máquina. Em Java são eles: byte, short, int, long, float, double, char e boolean.

2.3.3. EXPRESSÕES

Elas podem calcular um valor ou servir de meio para calcular esse valor. Elas podem, ainda, ser expressões de fluxo de controle, que expressem a seqüência de execução do programa. Essas expressões podem incluir constantes, variáveis, palavras

chave, operadores e outros. O Java não suporta a criação de instruções compostas que utilizem o operador de vírgula, a não ser que seja dentro das frases de inicialização e continuidade das instruções de loop.

2.3.4. INSTRUÇÕES

As instruções em Java são similares às instruções em C ou C++. As instruções são executadas em seqüência, fugindo a essa regra apenas as instruções de fluxo de controle ou de exceção. O Java possui vários tipos de instruções:

Instrução Vazia: Não realizam nada, podendo ser úteis durante o desenvolvimento do programa como espaço reservado.

Instrução Rotulada: Não devem ser palavra-chave, variáveis locais já declaradas, nem rótulos já utilizados nesse módulo. Podem ser usados como argumentos de instruções Jump.

Instrução de expressão: São sete as relacionadas em Java: Atribuição, Pré-incremento, pós-incremento, pré-decremento, pós-decremento, chamada de método e expressão de alocação.

Instrução de seleção: Escolhe um dentre os diversos controles de fluxo. Existem três tipos de instruções de seleção no Java: **if**, **if-else** e **switch-case-default**.

Instruções de Iteração: Especifica como e quando ocorrerá o looping. Existem três tipos de instruções de iteração que exceto pelos jumps e rótulos, são idênticos a seus correspondentes em C e C++: **while**, **do**, **for**.

Instruções Jump: As instruções Jump passam o controle para o início ou final do bloco atual ou para uma instrução rotulada. Os quatro tipos de instruções Jump são: **break**, **continue**, **return** e **throw**.

Instrução de Sincronização: São usadas para tratar assuntos com multiencadeamento (vários processos sendo ativados ao mesmo tempo). As palavras chave **synchronized** e **threadsafe** são usadas para marcar variáveis que podem ou não exigir bloqueios que impeçam usos simultâneos.

Instrução de Restrição: São usadas para tratar de forma segura os códigos que possam causar exceções (como dividir por zero). Essas instruções utilizam as palavras-chave **try**, **catch** e **finally**.

Instrução inatingível: Geram um erro no momento da compilação.

2.3.5. CLASSES

No contexto das linguagens de programação, as classes representam uma parte do paradigma de objeto. As classes podem ser compostas por uma combinação variada de tipos e outras classes, e podem herdar propriedades de classes principais. Elas podem apresentar operações (chamadas Métodos no paradigma de objeto) como parte de sua definição.

2.3.6. INTERFACES

Uma interface é a reunião de declaração de métodos e constantes que uma ou mais Classes de Objetos irão utilizar. As interfaces não podem conter variáveis ou códigos para implementar especificamente nenhum método que seja.

Os exemplos mais comuns de interface tendem a ser listas de métodos para imprimir ou obter dados, especificar a aparência de formulários na tela, e definir como as classes podem lidar com algum recurso controlado como as imagens ou corpos de texto.

2.3.7. PACOTES

No Java, os Pacotes são grupos de Classes. Eles são como biblioteca em várias linguagens de computador. Um pacote de classes Java, geralmente, conterà classes relacionadas. Por exemplo: Java.applet contém classes necessárias para criar applets Java que irão rodar no netscape2.0 (ou posterior), HotJava ou outros navegadores compatíveis com o Java.

3. O QUE É O MÉTODO FUSION?

O método Fusion é um método de desenvolvimento de sistemas voltado para a produção de sistemas orientados a objetos cujo lançamento ocorreu em 1992. Seus autores são Derek Coleman, bacharel em física e mestrado em Ciência da Computação pela Universidade de Londres, Patick Arnold, bacharel em Ciência da Computação, entre outros que sintetizaram essa metodologia no livro intitulado O Método Fusion - Desenvolvimento Orientado a Objetos . Trata-se de um método de completa abrangência, fornecendo todos os recursos para análise, projeto e implementação. As notações do Fusion permitem, de forma sistemática, descobrir e preservar as estruturas dos objetos do sistema. Integrando e estendendo abordagens existentes, o Fusion fornece uma rota direta entre a definição dos requisitos e a implementação com o uso de uma linguagem de programação. Este método está baseado em um conjunto conciso, mas completo, de notações bem definidas para a captura das decisões de análise e projeto.

Fusion suporta, por outro lado, os aspectos técnicos e gerenciais relativos ao desenvolvimento de software.

- Fornece um processo para o desenvolvimento de software. Ele divide o processo em fases e indica o que deve ser feito em cada uma dessas fases. Fornece uma orientação sobre a ordem na qual as ações devem ser realizadas dentro de cada fase, de modo que o responsável pelo desenvolvimento saiba como progredir. Além disso, estabelece critérios para informar à equipe de desenvolvimento quando será adequada a passagem para a fase seguinte.
- Fornece notações completas, simples e bem definidas para todos os seus modelos. Como estas notações são baseadas na prática existente, serão de fácil aprendizagem.
- Fornece ferramentas de gerenciamento para o desenvolvimento de software. Os resultados das diferentes fases são claramente identificados. Há uma série de verificações que asseguram a consistência entre as fases e também no âmbito de cada uma delas. As fases possuem técnicas próprias e esse destinam a diferentes aspectos da tradução que precisa ser feita para que o documento de requisitos se transforme em um código executável.

3.1. O PROCESSO DO MÉTODO FUSION

O método do processo Fusion adota a divisão do processo de desenvolvimento em análise, projeto e implementação caracterizadas anteriormente.

3.1.1. FASE DE ANÁLISE

Nesta fase tomamos como ponto de partida os requisitos que o usuário nos passa, transformando-o em um conjunto de modelos precisos e sem ambigüidades. Como consequência, as decisões estarão sujeitas a reavaliações e revisões contínuas a medida que os requisitos se tornarem mais claras. Dessa forma, a fase de análise criará uma especificação completa e consistente que captura os requisitos do usuário. Para isso o método divide a fase de análise em quatro etapas:

1. Desenvolver um modelo de objetos para o domínio do problema.
2. Determinar a interface do sistema.
3. Desenvolver um modelo de interfaces.
4. Verificar os modelos de análise.

3.1.1.1. DESENVOLVIMENTO DO MODELO DE OBJETOS

Nesta etapa iremos o método inspira-se no documento de requisitos do sistema e nos propõe listar os possíveis candidatos a classes e relacionamentos. Para orientar como encontrar classes e relacionamentos o método orienta:

Classes: Todo o substantivo ou frase substantiva pode dar origem a uma classe. Entretanto, para incluir o substantivo, este deve estar relacionado a um conceito importante ao entendimento do domínio. Objetos físicos, pessoas, organizações e abstrações são possíveis fontes de candidatos a classe.

Exemplos:

- Objetos físicos: sala de aula, prédio.
- Pessoas e organizações: professor, aluno, administrador
- Abstrações: lista de exercícios, listagem de alunos, listagem de professores.

Relacionamentos: Os relacionamentos indicam relações entre objetos. Associações físicas, inclusões e ações são fortes candidatas a relacionamento.

Exemplos:

- Associações: Por exemplo, questões pertencentes a uma prova.
- Inclusões: Por exemplo, alunos são cadastrados em uma cadeira.
- Ações: Por exemplo, professores corrigem provas.

A divisão da lista por assuntos será muito importante para dividirmos o problema em partes independentes. Cada item de cada lista deve ser essencial a explanação do problema. Logo devemos manter os candidatos mais gerais em cada lista.. Lembre-se que o modelo de objetos utiliza classes enquanto que o documento de requisitos é expresso em termos de objetos específicos. O processo será repetido até que as listas se estabilizem. Quando isso acontecer, as listas estarão completas, isto é, nenhum detalhe foi esquecido em relação ao documento de requisitos. A partir desse momento as classes e relacionamentos deverão ser incluídos no dicionário de dados.

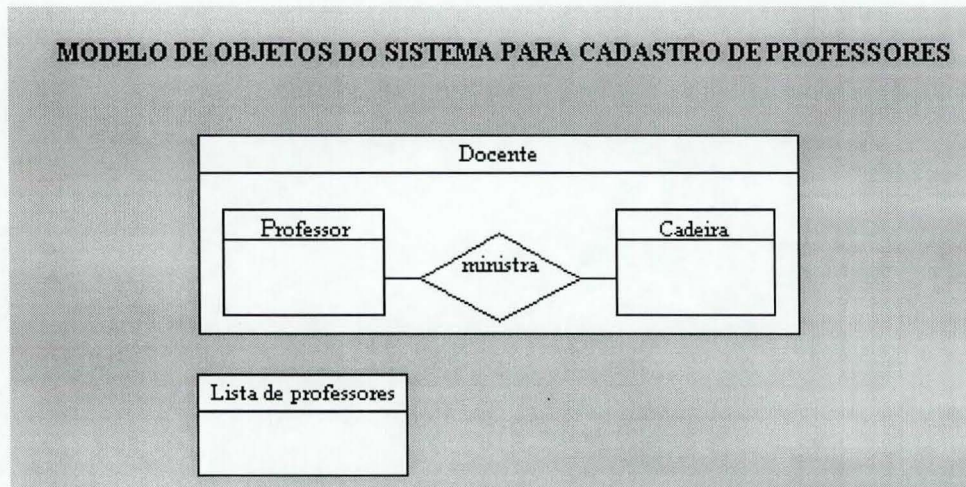
Para exemplificarmos a criação do modelo de objetos, trataremos de um requisito proposto a seguir que, na verdade, já será suficiente para implementarmos um módulo do sistema.

Requisito proposto: Um professor deve ser cadastrado no sistema para ministrar uma cadeira em uma determinada turma.. O sistema pode, também, emitir uma listagem de todos os professores cadastrados neste sistema.

As classes consideradas para esse modelo são: Professor, Cadeira, Lista de Professores, sem uma das quais o modelo não estaria completo.

Os relacionamentos considerados são: ministra.

Poderíamos considerar estes relacionamentos e classes considerados como sendo parte de uma lista denominada **Cadastro de Professor**. Esta lista, por sua vez está representada no seguinte modelo de objetos.



Nesse momento já devemos incluir estas classes e relacionamentos no dicionário de dados do sistema.

Para cada lista, devemos criar um modelo semelhante a este com suas classes e relacionamentos baseados nas listas propostas.

Obs.: Não devemos nos preocupar com a perfeição ou complexidade do modelo, pois este poderá ser alterado durante toda a fase de análise.

3.1.1.2. DESENVOLVIMENTO INTERFACE DO SISTEMA

É interessante lembrarmos que o modelo de objetos que acabamos de criar é uma entidade ativa que irá cooperar com outras entidades ativas, denominadas *agentes*. Como já foi comentado anteriormente, o sistema e os agentes se comunicam através do envio e recebimento de eventos. Quando os elementos são recebidos pelo sistema, podem causar uma mudança de estado e a geração de outros eventos. Um evento de entrada é uma operação de sistema.

A interface do sistema é composto pelo conjunto de operações que ele pode responder e pelos eventos que ele pode gerar. Uma operação do sistema é sempre ativada por um agente, e não por um objeto. Para representarmos estes eventos e operações utilizaremos os chamados cenários.

Um cenário nada mais é do que a representação em forma de diagrama de tempo, mostrando a ordenação temporal exibida pelas operações do sistema e também pelos eventos que fluem para os agentes. Esses diagramas não conseguem mostrar caminhos alternativos para a comunicação. Portanto, múltiplos diagramas podem ser necessários para um único cenário.

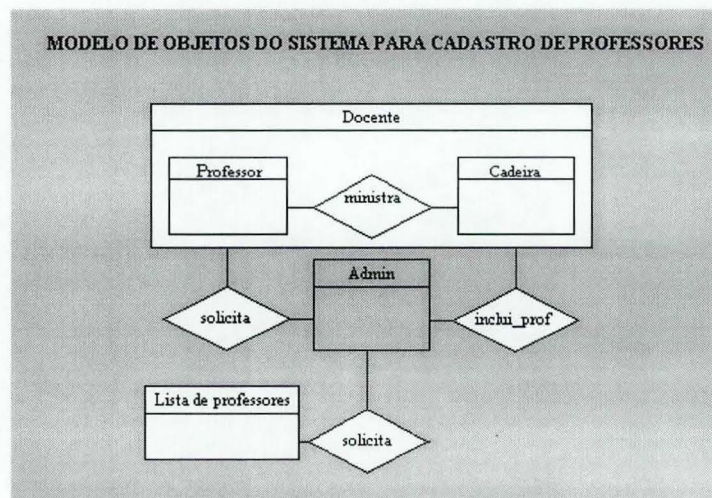
Para continuarmos com o exemplo, vamos criar uma interface para o modelo de objetos criado anteriormente.

Primeiramente teremos que definir os agentes desse modelo, as operações que o mesmo irá poder realizar e os eventos que o sistema poderá enviar a esse agente.

Agentes considerados: Administrador.

Operações: Cadastrar professores, solicitar listagem de professores

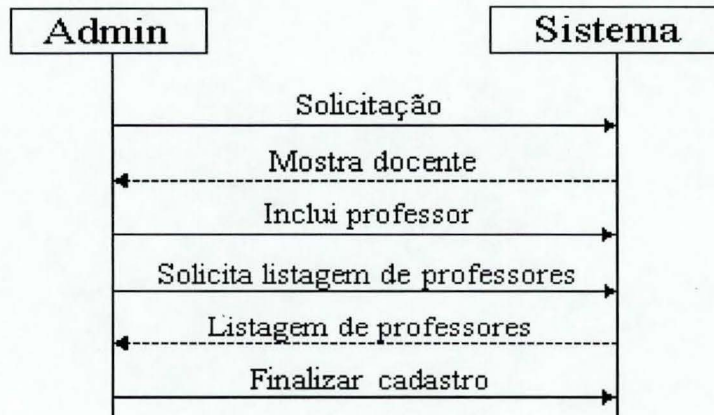
Considerando estes agentes e operações, nosso modelo ficaria dessa forma:



Assim como na fase da criação do modelo inicial, devemos incluir esses elementos novos do modelo no dicionário de dados específico para a interface do sistema. A utilização do dicionário de dados se tornará mais importante na medida que formos criando e modificando os modelos de objetos do sistema, pois o modelo começa a se tornar complexo, confundindo-nos quanto às classes e operações existentes no mesmo. Para que isso não ocorra devemos inserir os agentes no dicionário de dados para agentes do sistema e as operações de sistema e eventos serão inseridos do dicionário de dados para a interface do sistema, sendo que, neste último, devemos preencher da forma mais clara possível a descrição de cada operação de sistema e eventos a fim de facilitar futuras consultas.

O cenário respectivo para esse modelo é representado dessa forma:

Cadastrar Professor



As linhas pontilhadas no sentido do Admin (agente) são os eventos respondendo às operações de sistema ativadas pelo agente Admin. Como já foi dito, esse diagrama é limitado se quisermos alternar operações. Por exemplo, no diagrama anterior, seguindo a ordenação temporal, o Admin não poderia mais cadastrar um outro professor após ter solicitado uma listagem de professores. A única solução é construir outro diagrama para esse mesmo cenário.

Classes e objetos que existem no sistema.

- ◆ Relacionamentos entre essas classes.
- ◆ Operações que podem ser executadas no sistema.
- ◆ Seqüências permissíveis para essas operações.

Em contraste com alguns outros métodos, o estágio de análise no FUSION não associa métodos a classes particulares: isso será feito posteriormente.

3.1.1.3. DESENVOLVIMENTO DO MODELO DE INTERFACES

O modelo de interfaces é composto pelo modelo ciclo-de-vida e um modelo de operações, não importando qual será primeiramente desenvolvido. É interessante, por outro lado, iniciarmos pelo modelo ciclo-de-vida, pois ele nos auxiliará no desenvolvimento dos esquemas para o modelo de operações.

As expressões ciclo-de-vida são mais descritivas que os diagramas de tempo, porque podem expressar repetição, alternância e opcionalmente concatenação de eventos. Uma expressão ciclo-de-vida pode definir um conjunto de cenários, enquanto que um diagrama de tempo pode mostrar apenas um único cenário.

O processo para a formação do modelo ciclo-de-vida é:

- Generalizar os cenários, de modo a formar expressões ciclo-de-vida;
- Combinar as expressões ciclo-de-vida para formar o modelo ciclo-de-vida.

Vamos considerar o cenário de cadastro de professor, mostrado anteriormente. Esse cenário pode ser traduzido diretamente para a expressão ciclo-de-vida identificada por **Cadastro_Professor** a seguir:

Cadastro_Professor=

```
solicitação. # mostra_docente
[inclui_professor]
([solicita_listagem_professores . #
listagem_professores_cadastrados])*
finalizar_cadastro.
```

O operador de repetição ‘*’ indica que cada solicitação de listagem resulta na emissão da listagem de professores cadastrados. O operador ‘#’ significa resposta do sistema (evento). Observe que os conchetes “[]” significam que as operação e eventos internos a ele são opcionais.

Estando prontas todas as expressões ciclo-de-vida, precisaremos considerar como elas se relacionam, a fim de definirmos o comportamento de um modelo de sistema mais abrangente. O relacionamento entre estas expressões se fará com a utilização de dois símbolos adicionais: “ | “ e “|| “, o primeiro indicando que as expressões são exclusivas e a segunda que são intercalas. Se um dos requisitos do sistema fosse que enquanto o administrador estivesse cadastrando um professor, ele não pudesse cadastrar um aluno, teríamos a seguinte combinação:

```
(Cadastro_professor | Cadastro_aluno)*
```

O modelo de operação define a semântica de cada operação do sistema que faz parte da interface do sistema.. Cada esquema contém uma especificação informal das pré-condições e das pró-condições, ou seja, cláusulas **Assumes** e **Result**. Chama-se de contrato entre a operação e seus usuários a esse tipo de especificação. O contrato garante que a operação atinja um estado final, onde a cláusula **Result** é verdadeira, contanto que a operação tenha sido ativada com a cláusula **Assumes** satisfeita. Assim. É tarefa do analista o delineamento dos contratos das operações do sistema, de modo que satisfaçam aos requisitos dos usuários.

A próxima figura apresenta o esquema para a operação Cadastrar professor.

Operação: Cadastrar_professor

Descrição: O Admin solicita o cadastro de um professor para ministrar as aulas de determinada turma de uma cadeira.

Reads: cadeira: código_da_disciplina, Professor: nome, cod_prof: código do professor, senha: senha_inicial do professor, turma: letra referente à turma da cadeira a qual o professor irá ministrar.

Changes: Docente
new professor

Sends: admin: {listagem_professores}

Assumes:

Result: professor ministra cadeira.
O atributo docente.cód_cadeira foi iniciado com o código da cadeira emitida pelo Admin.
O atributo docente.turma foi iniciado com o valor de turma emitida pelo Admin.
O atributo docente.nome foi iniciado com o conteúdo de Nome do Professor.
O atributo docente.cod_prof foi iniciado com o código do Professor.
O atributo docente.senha foi iniciada com a expressão Senha Inicial.

Nesse ponto, novamente, é interessante que se atualize o dicionário de dados com as alterações que porventura tenham ocorrido em relação às operações do sistema.

Após termos traduzido todas as operações de sistema para estes esquemas, partiremos para a última fase.

3.1.1.4. VERIFICAR OS MODELOS DE ANÁLISE

Neste momento, apresentamos algumas diretrizes para a verificação dos modelos de análise. Essas verificações não são exaustivas, mas, de qualquer forma, são úteis.

I. *Exatidão contra requisitos*. Devemos reler os requisitos do sistema e tirar quaisquer dúvidas que ainda existam com o cliente e, em seguida, verificar se:

- ⇒ Todos os cenários possíveis forma cobertos pelo ciclo de vida.
- ⇒ Todas as operações de sistema foram definidas com o uso de um esquema
- ⇒ Todas as informações estáticas foram capturadas pelo modelo de objetos do sistema.
- ⇒ Todas as outras informações (por exemplo, definições técnicas e invariantes) se encontram no dicionário de dados.

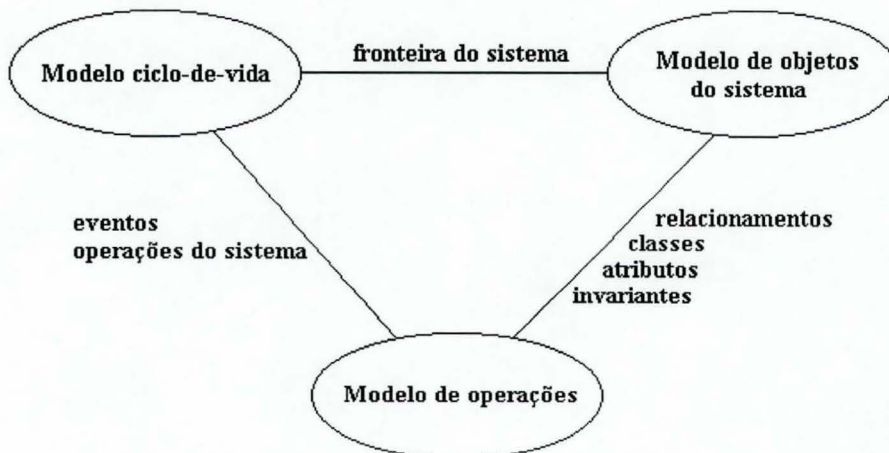
II. *Consistência simples*. Essas verificações tratam das áreas de sobreposição entre os modelos de análise. Após essas verificações devemos observar se:

- ⇒ Todas as classes, relacionamentos e atributos mencionados no modelo de operações aparecem no modelo de objetos do sistema. Todos os outros conceitos (por exemplo, os predicados) devem estar definidos no dicionário de dados ou em alguma outra fonte de referência.
- ⇒ A fronteira do modelo de objetos do sistema é consistente com a interface do sistema fornecida pelo modelo ciclo-de-vida.
- ⇒ Todas as operações do sistema, presentes no modelo ciclo-de-vida, possuem um esquema.
- ⇒ Todos os identificadores utilizados em todos modelos se encontram no dicionário de dados.

III. *Consistência Semântica*. Aqui tenta-se garantir que as implicações dos modelos sejam consistentes.

- ⇒ Os eventos gerados pelo modelo ciclo-de-vida e pelo modelo de operações devem ser consistentes. O esquema para uma operação do sistema deve gerar os eventos de saída que aparecem descritos, logo após a operação, nos cenários do modelo ciclo-de-vida.
- ⇒ O modelo de operações deve preservar os invariantes do modelo de objetos do sistema. Se houver um invariante associado a um relacionamento ou classe, então qualquer operação que possa alterar esses dois elementos deve respeitar o invariante representado no respectivo esquema.
- ⇒ Confira os cenários manualmente através dos esquemas. Selecione exemplos de cenário e defina a mudança de estado que cada um deveria causar. Em seguida “execute” os cenários, usando os esquemas para definir o comportamento de cada operação do sistema. Verifique se os resultados obtidos são aqueles esperados.

A seguinte figura ilustra as interseções entre os modelos.



Estas verificações que foram apresentadas podem ser aplicadas de várias formas. Idealmente, deveriam ser lembradas durante o desenvolvimento. Entretanto, serão provavelmente de uso mais prático como critérios de avaliação para as inspeções.

3.1.2. FASE DE PROJETO

O projetista decide como as operações do sistema serão implementadas pelo comportamento, em tempo de execução, dos objetos em interação. Diversas formas para subdividir uma interações de objetos podem ser testadas. Durante esse processo, as operações serão associadas a classes. O projetista também escolhe como os objetos farão referências mútuas e quais relacionamentos de herança serão apropriados entre as classes.

A fase de projeto gera modelos que mostram o seguinte:

- Como as operações do sistema serão implementadas pelas interações entre os objetos.
- Como as classes farão referências mútuas e como estarão relacionadas através da herança.
- Os atributos das classes bem como suas operações.

Os responsáveis pelo projeto podem ter de investigar com maiores detalhes as subestruturas de algumas classes e suas operações. Isso será feito com a *decomposição hierárquica* - aplicando as técnicas de análise e projeto a essas classes, considerando-as como um subsistema.

3.1.3. FASE DE IMPLEMENTAÇÃO.

O responsável pela implementação deve transformar o projeto em código, usando uma linguagem de programação específica. O Fusion orienta como isso deve ser feito.

- Herança, referências e atributos das classes serão implementados nas classes da linguagem de programação.
- As interações entre os objetos são codificadas na forma de métodos pertencentes a uma classe selecionada.
- As seqüências permitidas para as operações são reconhecidas por máquina de estado.

4. APRESENTAÇÃO DO PROBLEMA PARA APLICAÇÃO

Tomando como base minha experiência como aluno, levanto três questões que, a meu ver, foram bastante importantes no que diz respeito a minha aprendizagem. A maior dificuldade que senti durante esses anos todos como aluno, foi a de não haver possibilidade de testar o meu conhecimento em relação ao conteúdo apresentado nas

prática, isso seria inviável pela escassez de tempo e desagradável se cobrado fora do horário de aula.

Então qual seria a solução? Uma delas seria o professor fazer uma lista de exercícios para cada aula. Por outro lado, quem não iria gostar disso é o próprio professor, por diversos motivos: porque é demorado o processo de impressão e xerox, além do que esses exercícios acabam sempre perdidos. Segundo pelo fato do aluno ter que pagar pelo xerox, além do professor não ser ressarcido do que gastou. A solução seria tornar esse processo mais fácil e sem qualquer custo para o aluno, mesmo que ele não esteja presente nas aulas. O único recurso que nos permite ter acesso vinte e quatro horas por dia de uma informação e em qualquer lugar que estivermos é a Internet.

A idéia de disponibilizar exercícios para alunos disponibilizando-o em um servidor WWW foi apenas a idéia inicial. A necessidade de se criar uma aplicação para isso surgiu da idéia de construir um sistema que fizesse com que o aluno interagisse mais com o professor sem que isso custasse tempo para qualquer um dos dois. A aplicação, por sua vez, permitiria que o aluno fizesse os exercícios que achasse conveniente e que o aluno solicitasse ao professor a correção de alguns exercícios referentes a uma aula qualquer, corrigindo e esclarecendo, se fosse o caso, qualquer erro na resolução desses exercícios. A atualização destes exercícios ficaria a critério do professor. É claro que esse recurso não é ilimitado. Se, por acaso, cem alunos decidirem acessar esse servidor, por uma limitação de acessos, esse site não ficará mais acessível aos demais, tornando necessário a requisição de senhas para acessar o sistema. Finalmente por que não criar um sistema completo? A necessidade da senha torna impreterível a implementação de módulos de cadastro de alunos e professores no sistema. E para aproveitar a carona, por que não possibilitar a aplicação de provas e a correção das mesmas via este sistema?

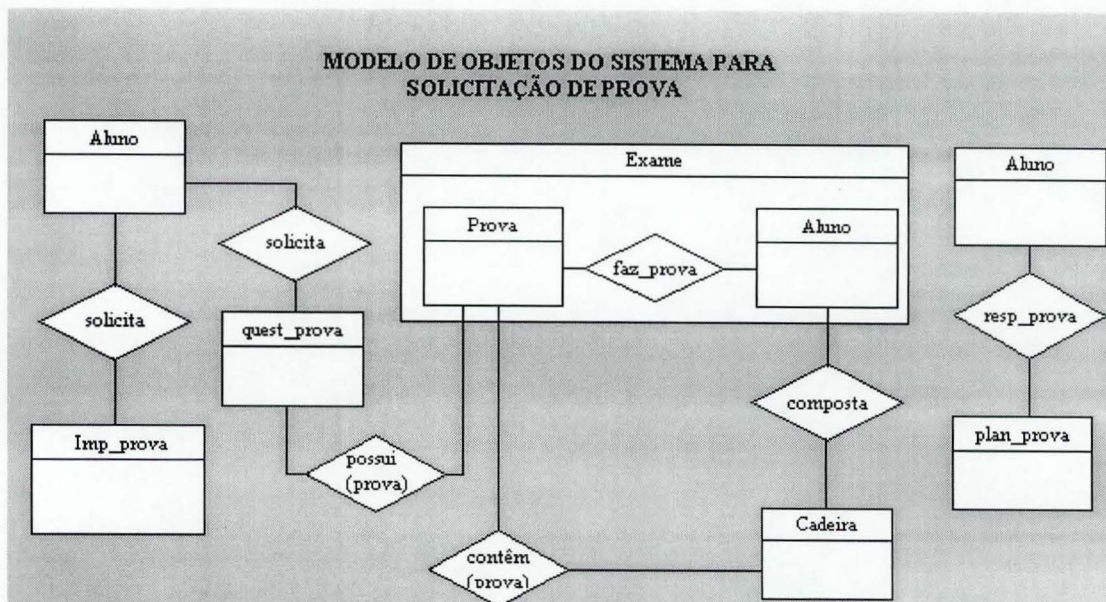
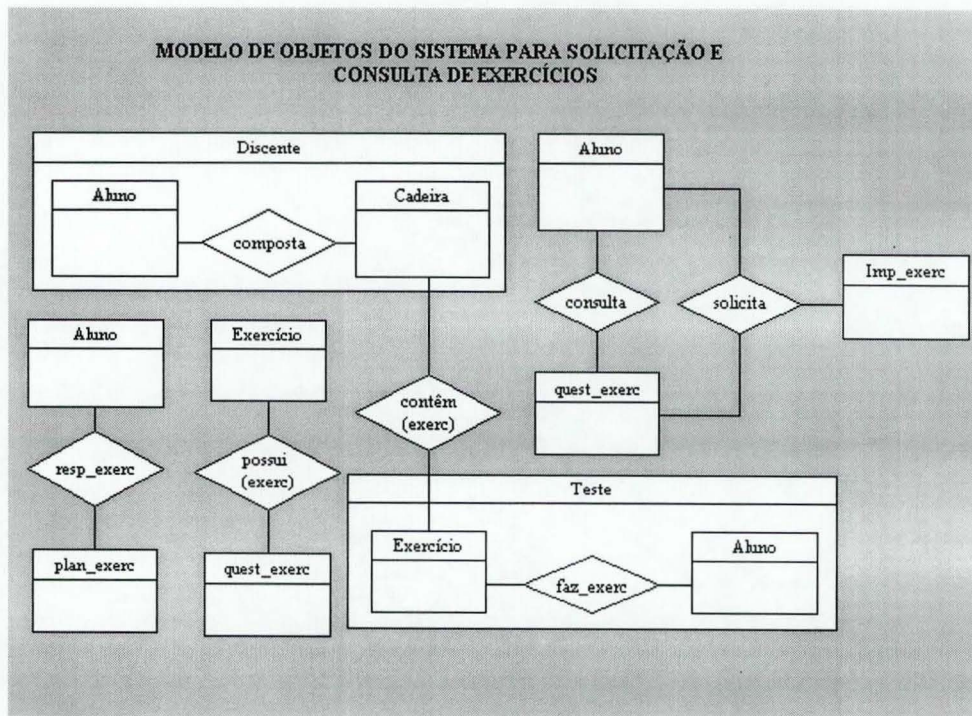
Daí a idéia de desenvolver um protótipo utilizando o processo FUSION e a linguagem de programação JAVA junto ao ambiente de programação Java Workshop.

5. SOLUÇÃO DO PROBLEMA UTILIZANDO O MÉTODO FUSION.

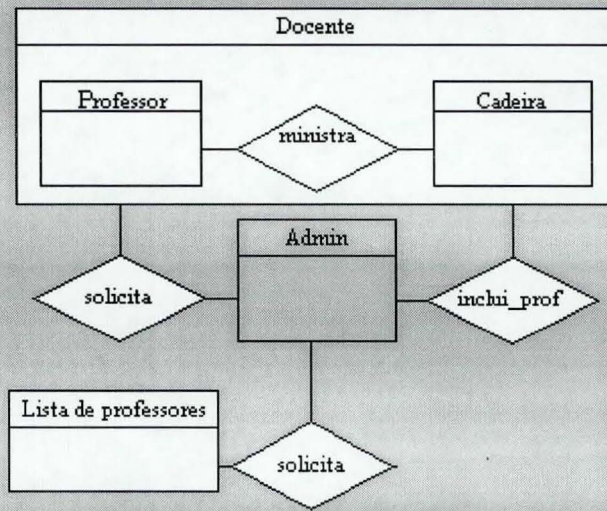
Nesse capítulo, veremos o resultado de cada etapa do método FUSION para a análise inicial do problema proposto, sem que se façam observações sobre cada decisão, visto que as etapas e critérios apontados por este método já foram devidamente apresentados no capítulo 3.1.

Apenas recordando, devemos criar as listas de requisitos e separá-las por assuntos, criando um modelo de objetos para cada uma destas listas.

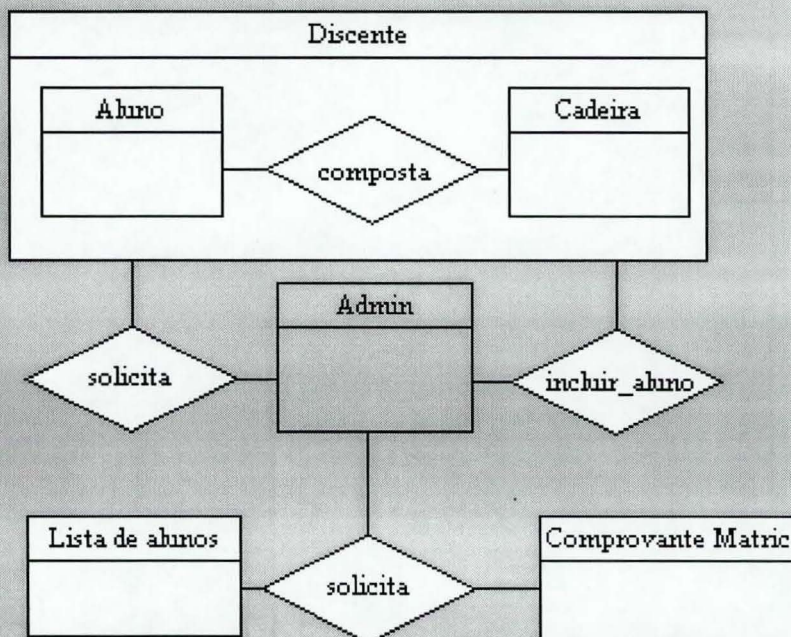
5.1. MODELOS DE OBJETOS DO SISTEMA



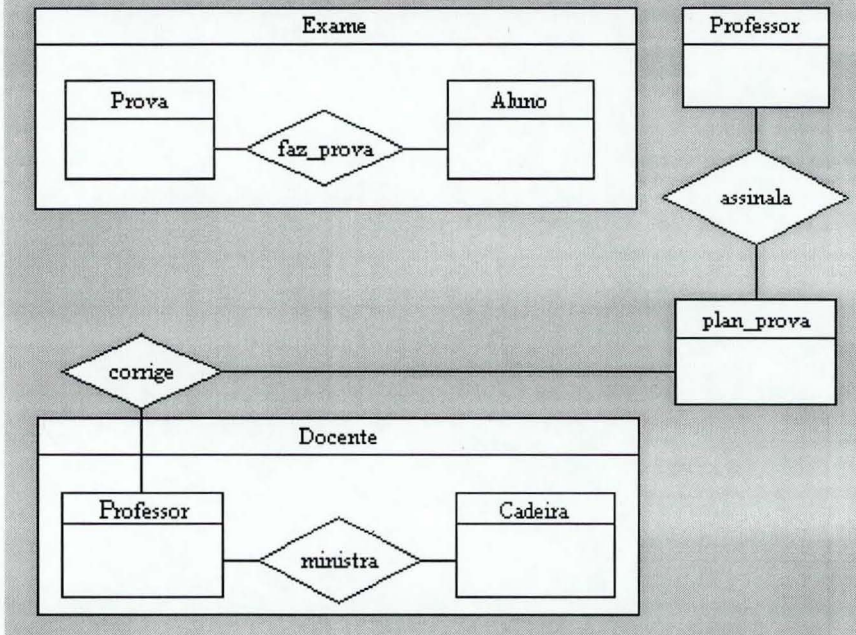
MODELO DE OBJETOS DO SISTEMA PARA CADASTRO DE PROFESSORES



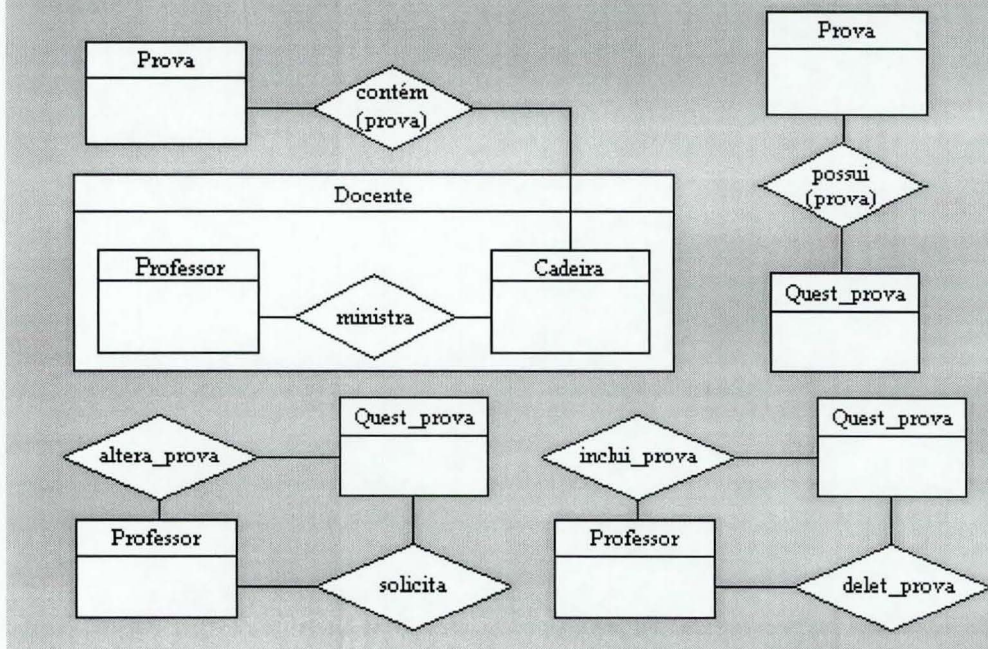
MODELO DE OBJETOS DO SISTEMA PARA CADASTRO DE ALUNOS



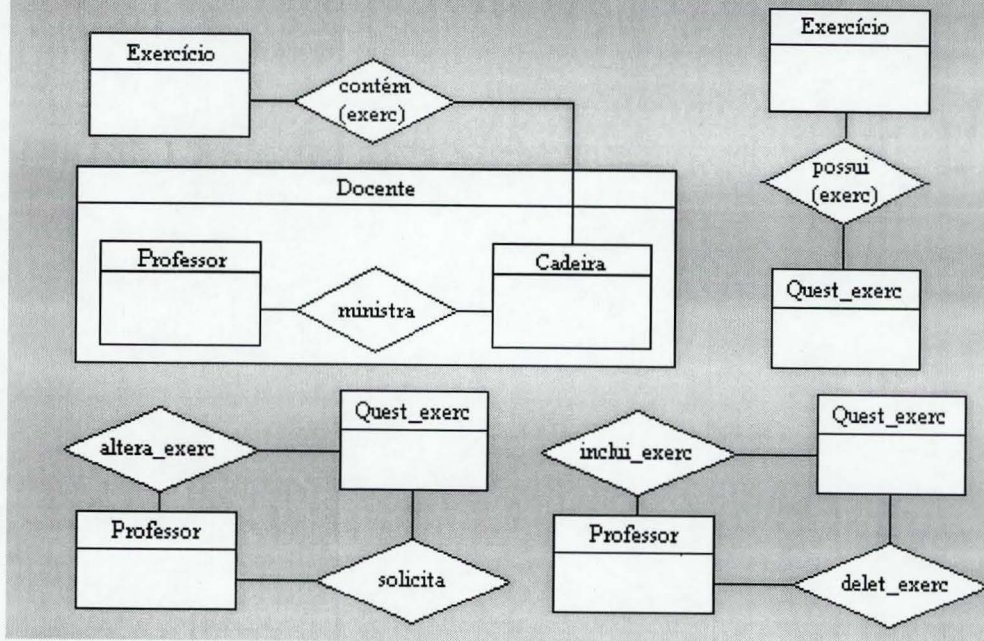
MODELO DE OBJETOS DO SISTEMA PARA CORRIGIR PROVA



MODELO DE OBJETOS DO SISTEMA PARA ALTERAR PROVA

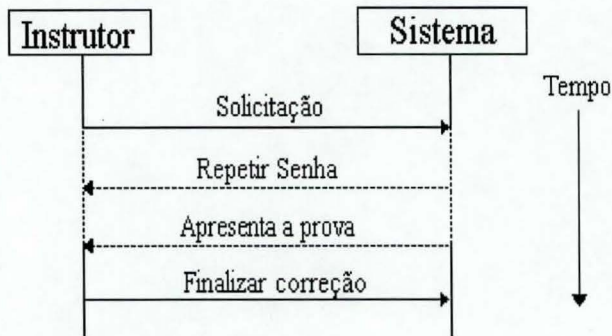


MODELO DE OBJETOS DO SISTEMA PARA
ALTERAR EXERCÍCIO

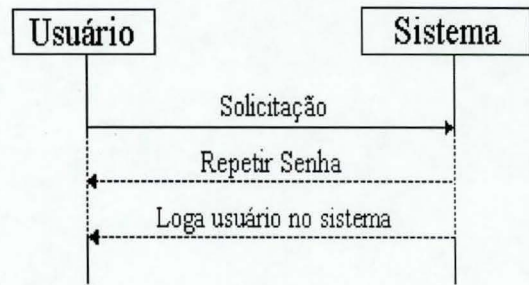


5.2. CENÁRIOS DO SISTEMA

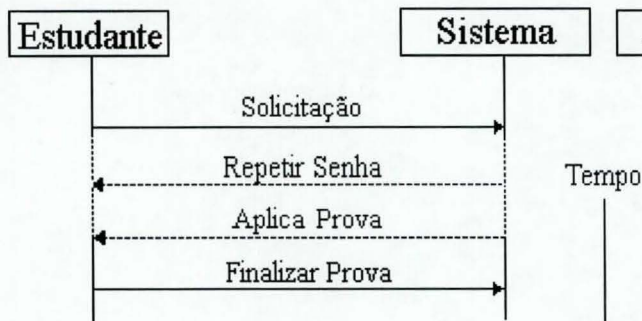
Corrigir Provas



Logar no sistema



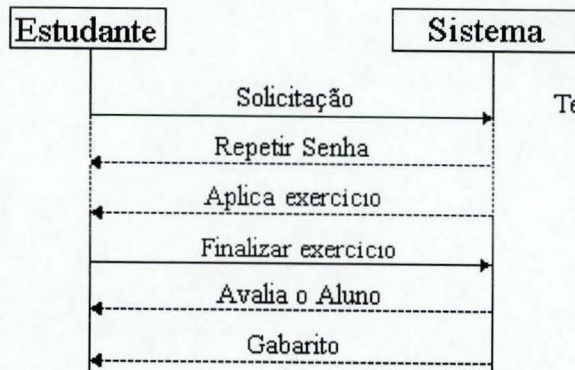
Aplicar Prova



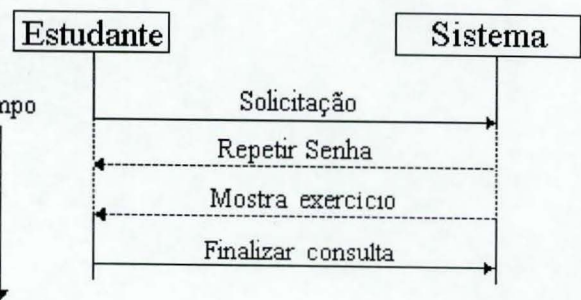
Cadastrar Aluno



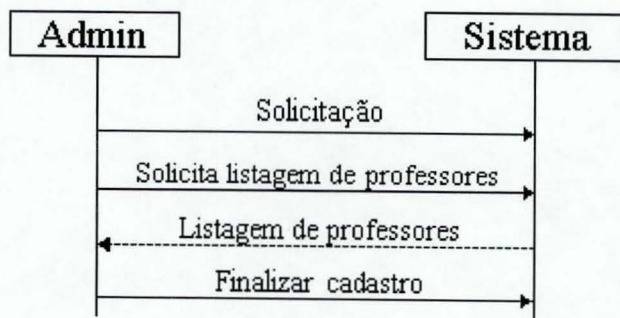
Aplicar Exercício



Consultar Exercício



Cadastrar Professor



5.3. MODELOS DE CICLO-DE-VIDA

Logar_no_sistema =

```
solicitação.
senha . [# repetir_senha]
([senha . # repetir_senha])*
[senha].# loga_usuario_sistema
```

Aplicar_exercício =

```
solicitação.
senha . [# repetir_senha]
([senha . # repetir_senha])*
# plan_exerc
([voltar_questao | adiantar_questao | resp_exerc] |
[solicita_impressao.# imprime_exercicio])*
(finalizar_exercicio. # avalia_aluno) | sair
```

Aplicar_prova =

```
solicitação.
senha . [# repetir_senha]
([senha . # repetir_senha])*
# plan_prova.
([voltar_questao | adiantar_questao] | resp_prova) |
[solicita_impressao.# imprime_prova])*
sair
```

Consulta_aluno =

```
solicitação. # discente
([solicita_comprovante.# comprovante_de_matricula] |
[solicita_listagem_alunos. # listagem_alunos] |
Inc_aluno | del_aluno | voltar_aluno | adiantar_aluno)*
sair.
```

Alterar_prova =

```
solicitação.
senha . [# repetir_senha]
([senha . # repetir_senha])*
# quest_prova.
([voltar_questao | adiantar_questao] |
ins_prova | del_prova )*
sair.
```

Alterar_exercício =

```
solicitação.  
senha . [# repetir_senha]  
([senha . # repetir_senha])*  
# quest_exerc.  
([voltar_questao | adiantar_questao] |  
ins_exerc | del_exerc )*  
sair.
```

Consulta_Professor=

```
solicitação. # docente  
[inc_professor]  
([solicita_listagem_professores . # listagem_professores] |  
inc_prof | del_prof | voltar_prof | adiantar_prof )*  
sair.
```

Corrigir_prova =

```
solicitação.  
senha . [# repetir_senha]  
([senha . # repetir_senha])*  
# plan_prova  
([voltar_questao | adiantar_questao] |  
sair.
```

5.4. ESQUEMAS DO SISTEMA

5.4.1. ESQUEMA PARA O EVENTO APLICAR_EXERCÍCIO

Operação: Aplicar_exercício

Descrição: O aluno solicita a aplicação de um teste referente a aula correspondente .

Reads: aula: aula_correspondente, senha: senha_aluno, cadeira: código_da_disciplina, turma: letra correspondente à turma da cadeira que o aluno está matriculado.

Changes: faz_exerc: conceito

Sends: aluno: {aplica_exercício}{repetir_senha}{imprime_exercício}
{avalia_aluno}{imprime_gabarito}

Assumes: que a senha do aluno é uma senha válida

Result: Se (senha foi validada) então faz_exerc.conceito recebeu como valor o conceito do aluno no teste.
Caso contrário repetir_senha foi enviado ao aluno.

5.4.2. ESQUEMA PARA O EVENTO CONSULTAR EXERCÍCIO

Operação: Consultar_exercício

Descrição: O aluno solicita a consulta de um exercício que fez em uma aula de uma disciplina.

Reads: aula: aula_correspondente, senha: senha_aluno, cadeira: código_da_disciplina, turma: letra correspondente à turma da cadeira que o aluno está matriculado.

Changes:

Sends: aluno: {mostra_exercício}{imprime_exercício}{redigitar_senha}

Assumes: que a senha do aluno é uma senha válida

Result: Se (senha foi validada) então mostra_exercício foi enviado ao aluno.
Caso contrário repetir_senha foi enviado ao aluno.

5.4.3. ESQUEMA PARA O EVENTO APLICAR_PROVA

Operação: Aplicar_prova

Descrição: O usuário (aluno) solicita a aplicação de uma prova em uma turma de determinada disciplina.

Reads: senha: senha_aluno, cadeira: código_da_disciplina, turma: letra correspondente à turma da cadeira que o aluno está matriculado.

Changes:

Sends: aluno: {aplica_prova}{imprime_prova}{repetir_senha}

Assumes: que a senha do aluno é uma senha válida

Result: Se (senha foi validada) então aplica_prova foi enviado ao aluno.
Caso contrário repetir_senha foi enviado ao aluno.

5.4.4. ESQUEMA PARA O EVENTO CADASTRAR_PROFESSOR

Operação: Cadastrar_professor

Descrição: O Admin solicita o cadastro de um professor para ministrar as aulas de determinada turma de uma cadeira.

Reads: senha: senha_admin, cadeira: código_da_disciplina, Professor: nome, senha: senha_inicial do professor, turma: letra referente à turma da cadeira a qual o professor irá ministrar.

Changes: Docente
new professor

Sends: admin: {listagem_professores}

Assumes:

Result: professor ministra cadeira.
O atributo docente.código foi iniciado com o código da cadeira emitida pelo Admin.
O atributo docente.turma foi iniciado com o valor de turma emitida pelo Admin.
O atributo docente.nome foi iniciado com o conteúdo de Nome do Professor.
O atributo docente.senha foi iniciada com a expressão Senha Inicial.

5.4.5. ESQUEMA PARA O EVENTO CADASTRAR_ALUNO

Operação: Cadastrar_aluno

Descrição: O Admin solicita o cadastro de um aluno em uma determinada turma de uma cadeira.

Reads: nome: nome do aluno, cadeira: código_da_disciplina, Turma: Turma_da_disciplina, senha: senha_inicial do aluno

Changes: Discente
new aluno

Sends: admin: {comprovante_matrícula}{listagem_alunos}

Assumes:

Result: cadeira composta por aluno.
O atributo discente.código foi iniciado com o código da cadeira emitida pelo Admin.
O atributo discente.turma foi iniciado com o valor de turma emitida pelo Admin.
O atributo discente.nome foi iniciado com o conteúdo de Nome do Aluno.
O atributo discente.senha foi iniciada com a expressão Senha Inicial.

5.4.6. ESQUEMA PARA O EVENTO CORRIGIR_PROVA

Operação: Corrigir_prova

Descrição: O professor solicita a correção das provas de uma turma de determinada cadeira.

Reads: senha: senha do professor, nome: nome do professor, cadeira: código_da_disciplina, Turma: Turma_da_disciplina

Changes: faz_prova: conceito

Sends: professor: {repetir_senha}{apresentar_prova}

Assumes: que a senha do professor é uma senha válida

Result: Se (senha foi validada) então:
O atributo conceito foi iniciado com o conceito final do professor.
Caso contrário repetir_senha foi enviado ao professor.

5.4.7. ESQUEMA PARA O EVENTO ALTERAR_PROVA

Operação: Alterar_prova

Descrição: O professor solicita a alteração das questões de uma prova de uma determinada cadeira.

Reads: senha: senha do professor, nome: nome do professor, cadeira: código da disciplina, Turma: letra que define a turma da cadeira na qual o aluno está matriculado.

Changes: A(s) questão(es) da prova da cadeira e turma selecionadas.

Sends: professor: {repetir_senha}{mostra_prova}

Assumes: que a senha do professor é uma senha válida

Result: Se (senha foi validada) então:
As questões da prova foram alteradas pelo professor.
Caso contrário repetir_senha foi enviado ao professor.

5.4.8. ESQUEMA PARA O EVENTO ALTERAR_EXERCÍCIO

Operação: Alterar_exercício

Descrição: O professor solicita a alteração das questões de um exercício de uma aula de determinadas turma e cadeira.

Reads: senha: senha do professor, nome: nome do professor, cadeira: código_da_disciplina, Turma: Turma_da_disciplina, aula: aula referente aos exercícios.

Changes: A(s) questão(es) do exercício da cadeira, turma e aula selecionadas.

Sends: professor: {repetir_senha}{mostra_exercício}

Assumes: que a senha do professor é uma senha válida

Result: Se (senha foi validada) então:
As questões do exercício foram alteradas pelo professor.
Caso contrário repetir_senha foi enviado ao professor.

5.5. DIAGRAMA DE ESTADOS DO SISTEMA



Diagrama de estado para solicitação de consulta de exercício

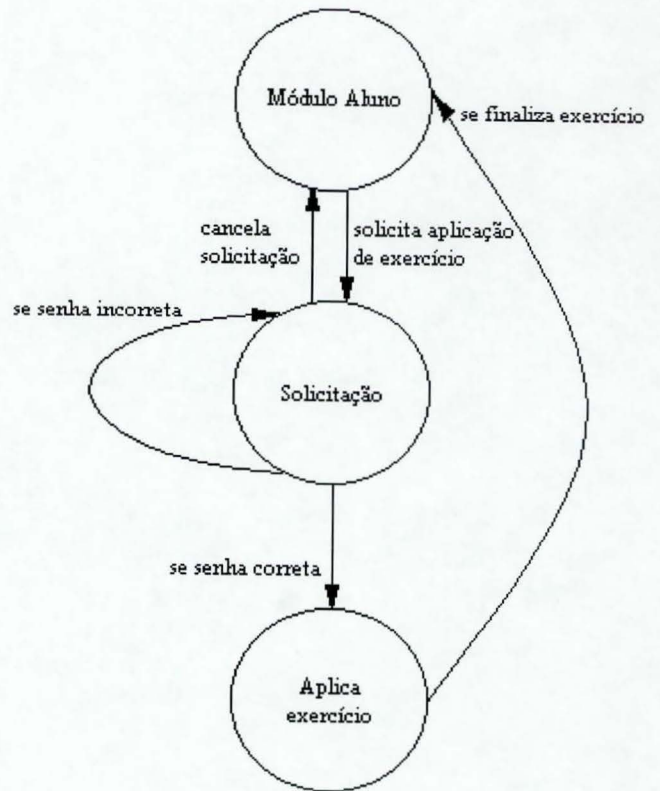


Diagrama de estado para solicitação de aplicação de exercício



Diagrama de estado para solicitação de aplicação de prova

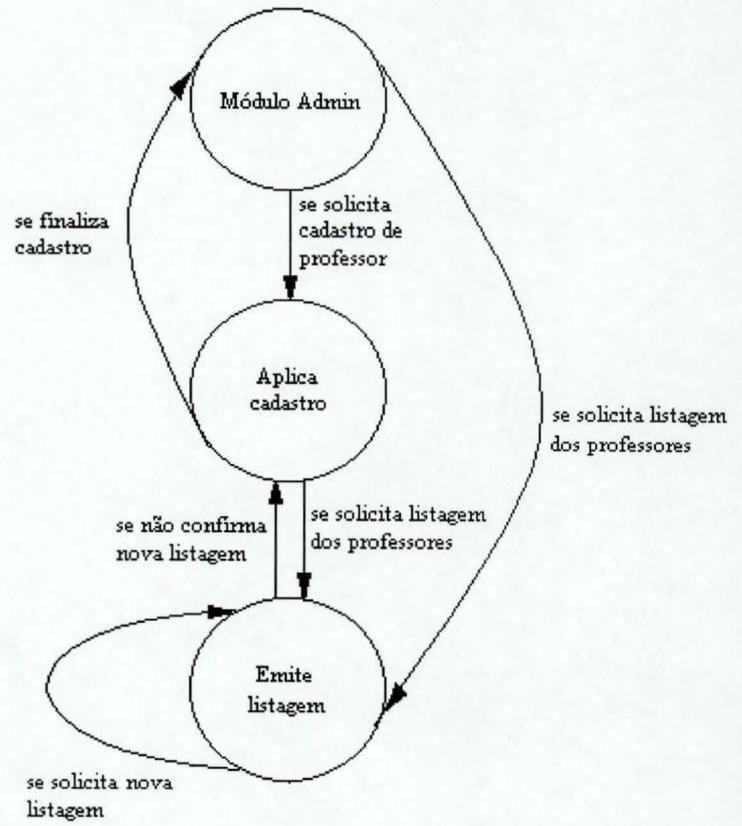


Diagrama de estado para solicitação de cadastro de professores

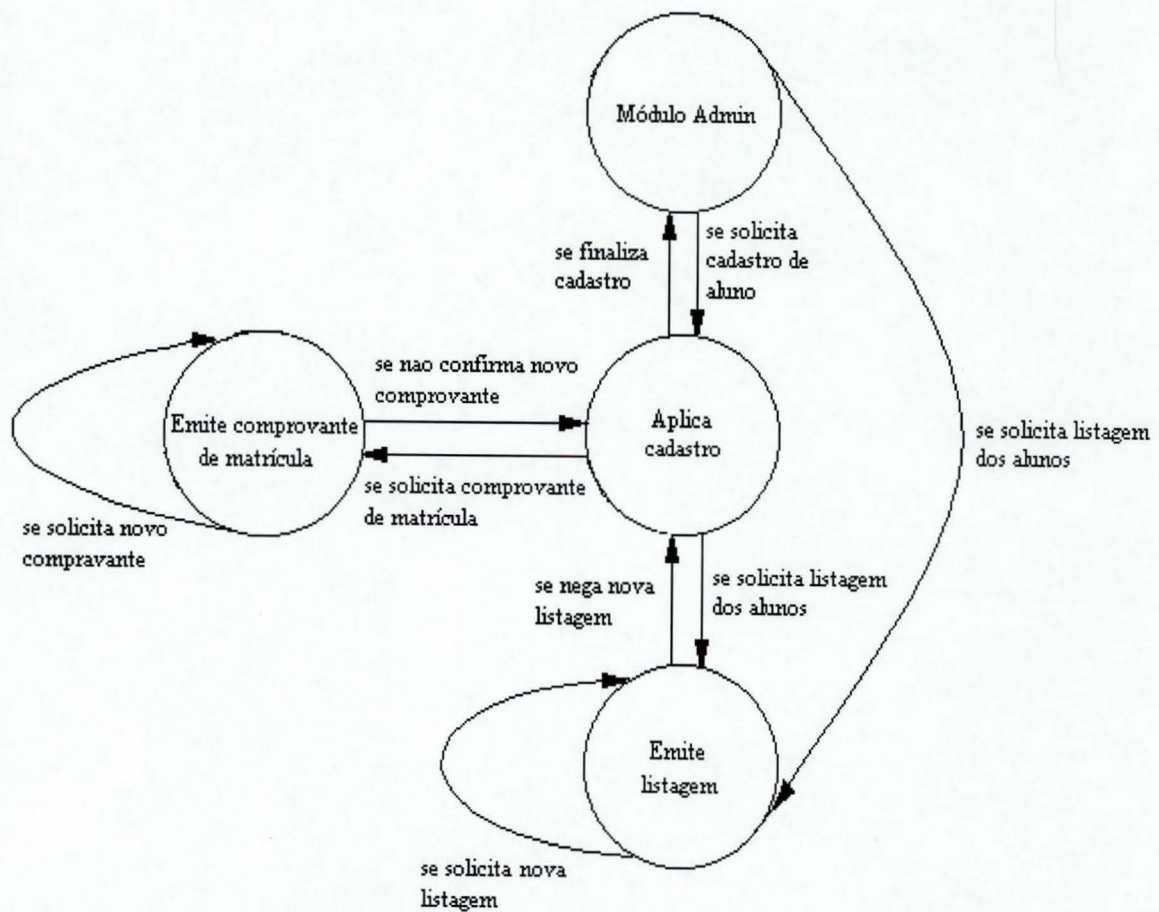


Diagrama de estado para solicitação de cadastro de aluno

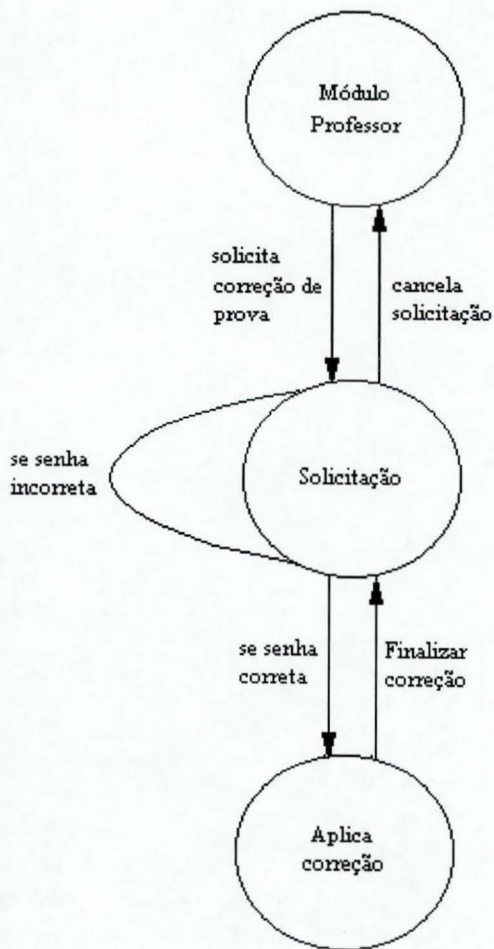


Diagrama de estado para solicitação de correção de prova

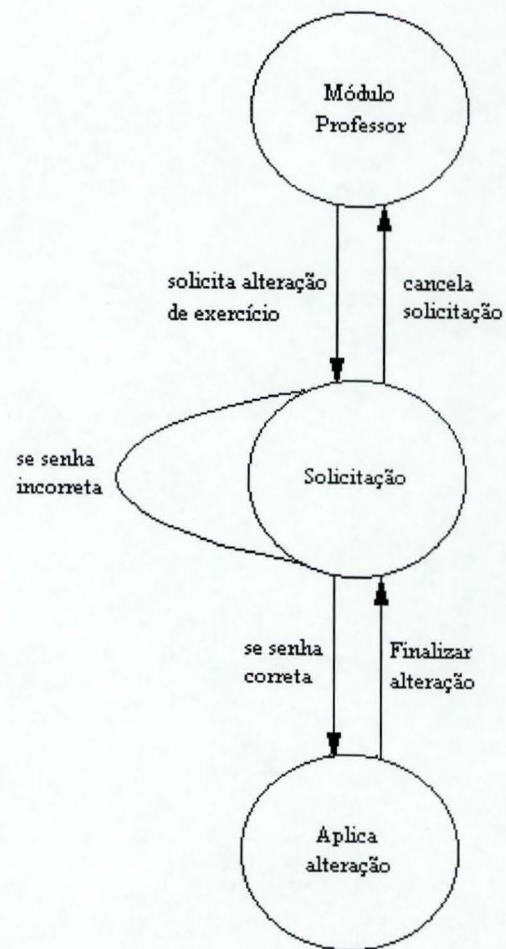


Diagrama de estados para solicitação de alterar exercício



Diagrama de estados para solicitação de alteração de prova

6. DICIONÁRIO DE DADOS

6.1. DICIONÁRIO DE DADOS: SISTEMA

Nome	Categoria	Descrição	Atributos
Admin	classe	Administrador ou super usuário do Sistema	nome, senha, username
Aluno	classe	Aluno que está cursando uma cadeira ou trancou sua matrícula	nome, username, cod_aluno
Cadeira	classe	Uma disciplina oferecida para o curso.	Nome_cadeira, cod_cadeira
Exercício	classe	Exercício referente a uma aula de uma turma de uma certa cadeira	cod_exerc, turma, aula, cod_cadeira
Plan_exerc	classe	Planilha com questões referentes a um exercício de uma aula de uma certa turma de uma disciplina específica	Cod_exercicio, nr, cod_aluno, alternativa
Plan_prova	classe	Planilha com questões e respostas referentes à prova de uma disciplina específica	Cod_prova, nr (da questão), cod_aluno, alternativa, resp_desc
Professor	classe	Uma professor que assessora nas disciplinas em turmas específicas.	Nome, cod_professor, username
Prova	classe	Prova referente a uma certa cadeira e turma	cod_prova, cod_cadeira, turma
Quest_exerc	classe	Questões referentes a um exercício específico	Cod_exercicio, nr, conteudo
Quest_prova	classe	Questões referentes à uma prova específica	Cod_prova, nr, conteudo
Discente	classe (agregação)	Lista de todos os alunos cadastrados em quaisquer cadeiras	
Docente	classe (agregação)	Tuplas contendo Professor-ministra-Cadeira onde ministra tem como atributo turma.	

Nome	Categoria	Descrição	Atributos
Exame	classe (agregação)	Tupla contendo as classes Prova e Aluno e o relacionamento faz_prova que tem como atributo cadeira e nota.	
Teste	classe (agregação)	Tupla que contém as classes Exercício e Aluno e o relacionamento faz_exerc que acrescenta o atributo nota.	
Possui (exerc)	relacionamento	Um exercício possui uma ou mais questões.	Cod_cadeira, cod_exerc, turma, nr
Possui (prova)	relacionamento	Uma prova possui uma ou mais questões.	Cod_cadeira, cod_prova, turma, nr
Composta	relacionamento	Cadeira(s) é (são) composta(s) por aluno(s).	cod_aluno, cod_cadeira, turma
Contém (exerc)	relacionamento	Cada aula de uma turma de uma determinada cadeira contém um exercício	cod_cadeira, turma, aula
Contém (prova)	relacionamento	Cada cadeira contém uma prova	cod_cadeira, turma
Faz_exerc	relacionamento	Cada aluno faz um ou nenhum exercício de determinada aula de uma turma de uma disciplina	cod_aluno, cod_exerc, nota, turma, cod_cadeira, senha, aula
Faz_prova	relacionamento	Cada aluno pode fazer ou não uma ou mais provas em uma disciplina.	Cod_aluno, cod_cadeira, turma, nota
Ministra	relacionamento	Professor(es) ministra(m) cadeira(s) em determinada turma	cod_professor, senha, cod_cadeira, turma

6.2. DICIONÁRIO DE DADOS: INTERFACE DO SISTEMA

Nome	Categoria	Agente	Descrição	Parâmetros
Avalia_aluno	evento	aluno	Avalia o aluno de acordo com as alternativas escolhidas no exercício	
Repetir senha	evento	aluno, professor, admin	Solicita repetir a senha para o nível de operação requerido.	
Altera_prova	operação de sistema	professor	Altera a prova de uma determinada turma de uma disciplina.	Nome (professor), senha, cod_cadeira, turma
Cad_aluno	operação de sistema	admin (SU)	Faz o cadastro de um aluno em uma determinada turma de uma cadeira	Cod_cadeira, turma, nome (aluno), cod_aluno, senha inicial
Cad_prof	operação de sistema	admin (SU)	Faz o cadastro de um professor para lecionar uma certa disciplina.	Cod_cadeira, turma, cod_prof, nome (professor), senha inicial
Consulta (Quest_exerc)	operação de sistema	aluno	Solicita consultar um teste de uma determinada aula e disciplina.	Cod_cadeira, turma, aula, senha
Corrige_prova	operação de sistema	professor	Corrige as provas de uma certa turma de uma disciplina.	Nome (professor), senha, cod_cadeira, turma
Del_aluno	operação de sistema	admin	remove um aluno de uma determinada turma de uma cadeira	Cod_cadeira, turma, cod_aluno
Del_exerc	operação de sistema	professor	deleta uma questão de um exercício	
Del_prof	operação de sistema	admin	remove um professor de uma determinada turma de uma cadeira	Cod_cadeira, turma, cod_professor
Del_prova	operação de sistema	professor	deleta uma questão de uma prova	

Nome	Categoria	Agente	Descrição	Parâmetros
Finalizar prova	operação de sistema	aluno	Termina a aplicação da prova.	
Incluir_aluno	operação de sistema	admin (SU)	cadastra um novo aluno em uma turma de uma determinada cadeira	Nome (aluno) , cod_aluno, cod_cadeira, turma, senha
Incluir_prof	operação de sistema	admin (SU)	cadastra um novo professor em uma turma de uma determinada cadeira	Nome (professor) , cod_prof, cod_cadeira, turma, senha
Ins_exerc	operação de sistema	professor	insere uma nova questão em um exercício	
Ins_prova	operação de sistema	professor	insere uma nova questão em uma prova	
Solicita (Comprovante matric)	operação de sistema	admin (SU)	Imprime um comprovante para o aluno cadastrado no sistema	
Solicita (Imp_exerc)	operação de sistema	aluno	Solicita a impressão das questões relativas ao exercício de uma aula de uma turma de uma determinada cadeira	
Solicita (Imp_prova)	operação de sistema	aluno	Imprime a prova para o aluno conferir.	
Solicita (Lista de alunos)	operação de sistema	admin (SU)	Imprime a lista dos alunos cadastrados no sistema	
Solicita (Lista de professores)	operação de sistema	admin (SU)	Imprime a lista dos professores cadastrados no sistema	

Nome	Categoria	Agente	Descrição	Parâmetros
Solicita (Quest_exerc)	operação de sistema	aluno	Auto-aplicar um exercício para sua avaliação em uma aula de uma turma específica de uma disciplina.	Cod_cadeira, turma, aula, senha
Solicita (Quest_prova)	operação de sistema	aluno	Auto-aplicar uma prova para sua avaliação em uma disciplina.	Cod_cadeira, turma, senha

6.3. DICIONÁRIO DE DADOS: AGENTES

Nome	Categoria	Descrição
admin	agente	Consulta Alunos; Consulta Professores
aluno	agente	Consulta Teste; Aplica prova
professor	agente	Consulta prova; Altera exercício; Corrige as provas

7. TABELAS DO SISTEMA

Tabela: T_admin

<u>Columns</u>		
Name	Type	Size
nome	Texto	50
senha	Texto	9
username	Texto	10

Tabela: T_aluno

<u>Columns</u>		
Name	Type	Size
nome	Texto	50
cod_aluno	Texto	9
username	Texto	10

Tabela: T_cadeira

<u>Columns</u>		
Name	Type	Size
nome_cadeira	Texto	30
cod_cadeira	Texto	8

Tabela: T_composta

<u>Columns</u>		
Name	Type	Size
Cod_aluno	Texto	9
cod_cadeira	Texto	8
turma	Texto	1

Tabela: T_contem_exerc

<u>Columns</u>		
Name	Type	Size
turma	Texto	1
aula	Número (Inteiro)	2
cod_cadeira	Texto	8

Tabela: T_contem_prova

<u>Columns</u>		
Name	Type	Size
cod_cadeira	Texto	8
turma	Texto	1

Tabela: T_exercicio

<u>Columns</u>		
Name	Type	Size
cod_exerc	Número (Inteiro)	2
cod_cadeira	Texto	8
turma	Texto	1
aula	Número (Inteiro)	2

Tabela: T_faz_exerc

<u>Columns</u>		
Name	Type	Size
cod_cadeira	Texto	8
turma	Texto	1
aula	Número (Inteiro)	2
cod_aluno	Texto	9
nota	Número (Inteiro)	2

Tabela: T_faz_prova

<u>Columns</u>		
Name	Type	Size
cod_aluno	Texto	9
cod_cadeira	Texto	8
turma	Texto	1
nota	Número (Inteiro)	2

Tabela: T_ministra

<u>Columns</u>		
Name	Type	Size
cod_professor	Texto	9
cod_cadeira	Texto	8
turma	Texto	1
senha	Texto	6

Tabela: T_plan_exerc

<u>Columns</u>		
Name	Type	Size
cod_exerc	Número (Inteiro)	2
nr	Número (Longo)	4
cod_aluno	Texto	9
alternativa	Texto	1

Tabela: T_plan_prova

<u>Columns</u>		
Name	Type	Size
cod_prova	Número (Inteiro)	2
nr	Número (Inteiro)	2
cod_aluno	Texto	9
alternativa	Texto	1
resp_desc	Memorando	-

Tabela: T_possui_exerc

<u>Columns</u>		
Name	Type	Size
cod_prova	Número (Inteiro)	2
nr	Número (Inteiro)	2

Tabela: T_possui_prova

<u>Columns</u>		
Name	Type	Size
cod_prova	Número (Longo)	4
nr	Número (Longo)	4

Tabela: T_professor

<u>Columns</u>		
Name	Type	Size
nome	Texto	50
codigo	Texto	9
username	Texto	10

Tabela: T_prova

<u>Columns</u>		
Name	Type	Size
cod_prova	Número (Longo)	4
cod_cadeira	Texto	8
turma	Texto	1

Tabela: T_q_exercicio

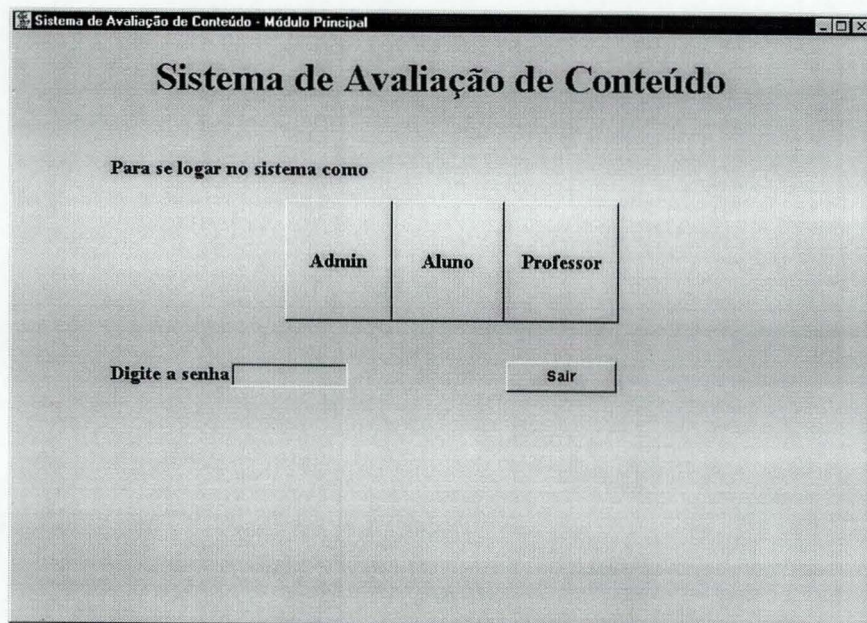
<u>Columns</u>		
Name	Type	Size
cod_exerc	Número (Longo)	4
nr	Número (Longo)	4
conteudo	Memorando	-
alternativa	Texto	1

Tabela: T_q_prova

<u>Columns</u>		
Name	Type	Size
cod_prova	Número (Longo)	4
nr	Número (Longo)	4
conteudo	Memorando	-

8. TELAS DO SISTEMA

8.1. TELA PRINCIPAL DO SISTEMA



Sistema de Avaliação de Conteúdo - Módulo Principal

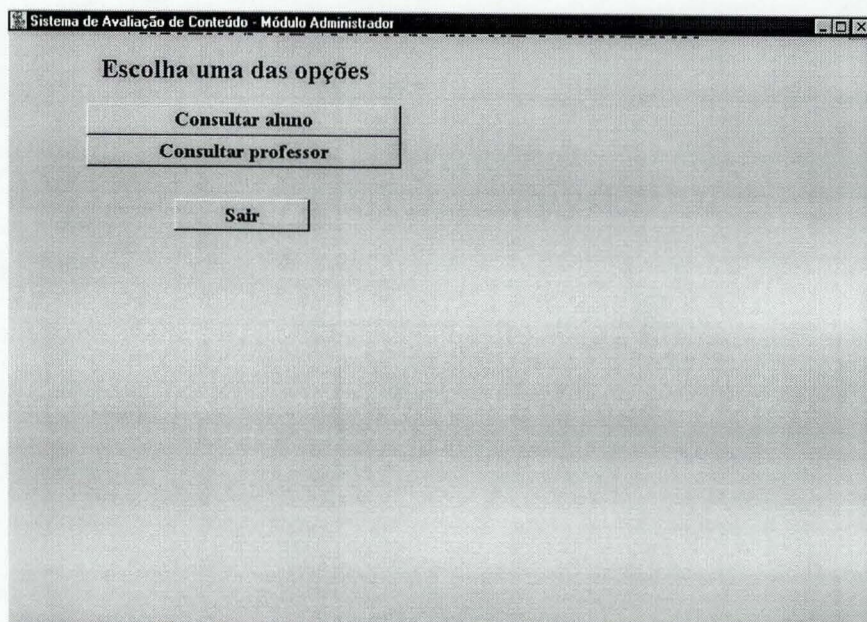
Sistema de Avaliação de Conteúdo

Para se logar no sistema como

Admin	Aluno	Professor
-------	-------	-----------

Digite a senha

8.2. TELA DO MÓDULO ADMINISTRADOR



Sistema de Avaliação de Conteúdo - Módulo Administrador

Escolha uma das opções

Consultar aluno
Consultar professor

8.3. TELA DE CONSULTA DE ALUNO

Consulta de Aluno

Nome

Número de matrícula

Código da cadeira

Turma

Senha inicial

Emite comprovante

Listagem de alunos

Inclui Registro

Apaga Registro

Volta

Avança

Sair

8.4. TELA DE CONSULTA DE PROFESSOR

Consulta de Professor

Nome:

Código do professor:

Código da cadeira:

Turma:

Senha inicial:

Listagem de professores

Inclui registro

Apaga registro

Volta

Avança

Sair

8.5. TELA DO MÓDULO ALUNO

Sistema de Avaliação de Conteúdo - Módulo Aluno

Escolha uma das opções

Aplicar exercício
Aplicar prova
Alterar senha

Digite a nova senha
[senha]

Preencha todos os campos ativados após a escolha da opção

Código do aluno [] Senha []
Código da cadeira [] Turma []
Aula []

Executar opção Sair

8.6. TELA DE APLICAÇÃO DE EXERCÍCIO

Exercício

Disciplina []
Turma [] Aula []
Questão []

[]

Volta Avança Alternativa A B C D E

Imprime

Corrige Percentagem de acerto [] Gabarito Sair

8.7. TELA DE APLICAÇÃO DE PROVA

Prova

Disciplina

Turma Questão

Resposta descritiva

Resposta descritiva

Volta Avança Alternativa A B C D E

Imprime Sair

8.8. TELA DO MÓDULO PROFESSOR

Sistema de Avaliação de Conteúdo- Módulo Professor

Escolha uma das opções

Alterar lista de exercícios

Alterar questões da prova

Alterar senha

Corrigir as provas

Digite a nova senha

Preencha todos os campos ativados após a escolha da opção

Código do professor Turma

Código da cadeira Aula

Senha

Executar opção Sair

8.9. TELA PARA ALTERAÇÃO DE EXERCÍCIO

Alteração de exercícios

Disciplina

Turma Aula

Questão

Volta Avança Insere Apaga

Imprime Sair

8.10. TELA PARA ALTERAÇÃO DE PROVA

Alteração de Prova

Disciplina

Turma

Questão

Volta Avança Insere Apaga

Imprime Sair

8.11. TELA PARA CORREÇÃO DE PROVA

Correção de Prova

Aluno Turma

Disciplina Questão

Resposta Descritiva Alternativa

Volta Avança Pontuação (1-10)

Imprime Sair

9. SOFTWARES UTILIZADOS PARA ELABORAÇÃO DO PROTÓTIPO

Browsers:

- Netscape 3.0
- Netscape 4.0
- Internet Explorer 3.0

Ferramentas de programação:

- Java WorkShop
- JDK 1.0
- JDK 1.0.1

Banco de Dados:

- Microsoft Access 7.0
- ODBC 32 bits
- JDBC
- Bridge ODBC/JDBC

10. DIFICULDADES ENCONTRADAS NA CONSTRUÇÃO DO PROTÓTIPO

A primeira dificuldade que surgiu foi a de entender as ferramentas de programação utilizadas na elaboração do protótipo. Na verdade, a mais utilizada foi o Java Workshop para a construção das API. A partir daí é que começaram a surgir dificuldades para manter a compatibilidade dessa ferramenta, que não implementa banco de dados, com o JDK1.0.1 e JDK1.0 que possuem classes específicas para manipulação de banco de dados. Além disso, foi necessário instalar o ODBC 32 bits no ambiente de desenvolvimento junto com o JDBC e o bridge ODBC/JDBC para haver uma conectividade completa com o banco de dados do Access 7.0. O protótipo, por sua vez, ficou apenas com as interfaces dos três módulos previstos prontas, mesmo porque o sistema é basicamente um sistema de informação, não tendo muito o que implementar. Sem quase processamento algum do banco de dados além de algumas consultas e alterações do mesmo, esse ambiente de programação me pareceu bastante simples e completo. O banco de dados foi implementado no Access 7.0.

11. CONCLUSÕES

A seguir são apresentadas as principais conclusões extraídas das atividades realizadas durante a execução do presente trabalho.

- Este sistema, ao longo de seu desenvolvimento se tornou bastante genérico, visando não só a sua utilização no meio acadêmico como em cursos de quaisquer gênero.
- O método Fusion, embora recente, demonstrou ser um método bastante objetivo com etapas bem definidas, facilitando assim, estabelecer facilmente as metas que eu deveria alcançar com o protótipo desenvolvido.
- A funcionalidade do sistema ficou bastante clara com as estruturas montadas durante o processo de análise, minimizando, desta forma, correções ao longo do desenvolvimento do sistema.
- A aplicação da modelagem orientada a objetos, inicialmente, pareceu-me bastante complicada e redundante, mas depois foi sendo bem assimilada, tornando-se mais objetiva e simples do que a modelagem funcional.
- A principal vantagem do método Fusion é a simplicidade com que ele desmembra o sistema já na fase inicial, tornando a análise bastante simplificada já no início. A riqueza de detalhes, por outro lado, fica a critério do analista já na fase de análise, visto que o desmembramento do sistema não tem limites. A atomicidade dos eventos nos leva, muitas vezes, a desconsiderarmos detalhes que só seria observado na interface com o usuário.
- Finalmente, acho que adquiri bastante experiência já com a implementação desse primeiro protótipo, o que me levou a discutir algumas soluções a nível de Intranet que esta interface poderia oferecer na minha atual empresa.

BIBLIOGRAFIA

- [COL 90] COLEMAN, Derek. Desenvolvimento Orientado a Objetos - O método Fusion. Editora Campus 1996.
- [YOU 90] YOURDON, Edward. Análise Estruturada Moderna. Editora Campus. 1990.
- [FLA 96] FLANAGAN, David. A Desktop quick reference for Java Programmers. 1996.
- [NEW 97] NEWMAN, Alexander. Usando Java – O Guia de Referência Mais Completo. Editora Campos. 1997.
- [DAM 96] DAMASCENO Júnior, Américo. Aprendendo Java: programação na Internet. Editora Érica Ltda. 1996.