

LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

IMPLEMENTAÇÃO DE UM SISTEMA
ACADÊMICO NA WEB USANDO JSP

Tobias Ferreira da Rocha Neto

NATAL
2003

TOBIAS FERREIRA DA ROCHA NETO

**IMPLEMENTAÇÃO DE UM SISTEMA
ACADÊMICO NA WEB USANDO JSP**

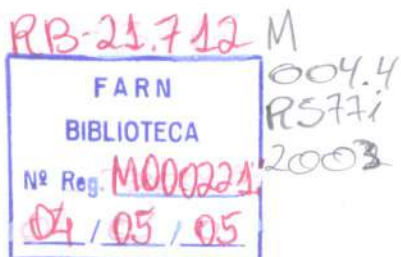
Projeto de Monografia apresentado ao curso de Bacharelado em Sistemas de Informação da Faculdade Natalense para o Desenvolvimento do RN. Como requisito parcial para a obtenção do título em Bacharel em Sistemas de Informação.

Orientador:

Fellipe Araújo Aleixo, MsC.

NATAL

2003



Rocha Neto, Tobias Ferreira da

Implementação de um Sistema Acadêmico na web usando JSP / Tobias Ferreira da Rocha
Neto. Natal: FARN, 2003.

88p.

Monografia (Bacharelado em Sistemas de Informação) - Faculdade Natalense para o
Desenvolvimento do Rio Grande do Norte.

Orientador: Fellipe Araújo Aleixo, MsC.

1. Programação Orientada a Objetos - monografia 2. Tecnologia JAVA para Internet -
monografia - I. Título

RN/FARN/BC

CDU 004.436.4

LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

IMPLEMENTAÇÃO DE UM SISTEMA
ACADÊMICO NA WEB USANDO JSP

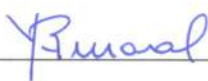
Tobias Ferreira da Rocha Neto

Monografia apresentada e aprovada em 14 de DEZEMBRO de 2002 pela
banca examinadora composta pelos seguintes membros:



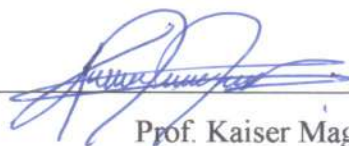
Prof. Fellipe Araújo Aleixo, MsC.

Orientador



Prof. Ytalo Rosendo do Amaral, MsC.

Examinador



Prof. Kaiser Magalde, MsC.

Examinador

Natal, 14 de DEZEMBRO de 2002

"Sonho que se sonha só é só um sonho que se sonha só, mas sonho que se sonha junto é realidade".

Raul Seixas

Dedico este trabalho aos meus pais Valtércio e Ailma e avó Maria que contribuíram com o incentivo necessário para que eu alcançasse meus objetivos, jamais sem eles poderia ter êxito. Ao restante da minha família que de forma indireta, também ajudaram.

Aos meus amigos e namorada Renally que com solidariedade, carinho e alegria, me ajudaram nas horas difíceis com uma palavra amiga e sincera.

AGRADECIMENTOS

Este trabalho foi desenvolvido com o apoio de várias pessoas, venho aqui expressar meus agradecimentos pela presteza e paciência em todas as contribuições.

Agradeço a todos os professores e coordenadora que durante toda a trajetória do curso, contribuíram nos ensinamentos e com certeza, nesse trabalho encontra-se parcelas dos mesmos, incondicionalmente que seja a disciplina ministrada.

Ao meu orientador Fellipe Araújo Aleixo, que acreditou no projeto e em quem estava envolvido, soube conduzir as orientações de forma justa e realmente cumpriu com seu papel, orientado, riscando e ensinando como se poderia fazer esse trabalho da melhor forma.

A Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte, Instituição que escolhi acreditando na proposta de ensino e satisfatoriamente alcançou as minhas expectativas, lembrando que, sendo uma Instituição de ensino nova, poderá melhorar ainda mais nos aspectos de infra-estrutura e ensino.

Ao meu colega e parceiro de projeto Marcelo Figueiredo pelo apoio e incentivo em todas as atividades da análise.

Aos funcionários da FARN, que abdicaram de seu tempo de trabalho para atender-nos para entrevistas, questionamentos e requisições que tiveram também um papel importante no desenvolvimento desse trabalho.

E a todos que, de uma outra forma, contribuíram.

RESUMO

A utilização da programação orientada a objetos para a concepção de sistemas para a *web* é um novo paradigma adotado por desenvolvedores como solução computacional para empresas e instituições.

Este trabalho visa apresentar uma implementação de um protótipo de sistema acadêmico tendo como base uma análise orientada a objetos realizada pelo discente Marcelo Luiz de Figueiredo, integrante do projeto, utilizando as tecnologias para *Internet* como *JAVA*, *Servlets*, *JSP* e *Java Beans*, que será uma solução para melhorar o controle acadêmico e oferecer mais serviços à comunidade acadêmica da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte.

Tal sistema facilitará as atividades realizadas pelos funcionários da secretaria e CPD, descentralizando tarefas que poderão ser distribuídas entre mais pessoas. Professores e alunos estarão também beneficiados no sentido da rapidez da aquisição de informações acadêmicas.

ABSTRACT

The use of the programming guided to objects for the conception of systems for the web is a new adopted paradigm for developers as computacional solution for companies and institutions.

This work seeks to present one prototype of academic system tends as base an analysis guided to objects accomplished by the student Marcelo Luiz de Figueiredo, integral of the project, using the technologies for Internet as JAVA, Servlets, JSP and Java Beans, that it will be one solution to improve the academic control and to offer more services to Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte.

This system will facilitate the activities accomplished by the employees of the secretariat and CPD, decentralizing tasks that they can be distributed among more people. Teachers and students they will be also benefitted in the sense of the speed of the acquisition of academic information.

SUMÁRIO

1. INTRODUÇÃO	12
1.1 APRESENTAÇÃO	12
1.2 PROBLEMA DE PESQUISA	13
1.3 OBJETIVOS GERAIS E ESPECÍFICOS	13
1.4 METODOLOGIA	13
1.5 ESTRUTURA DO TRABALHO	14
2. REFERENCIAL TEÓRICO	15
2.1 PROGRAMAÇÃO ORIENTADA A OBJETOS	15
2.2 JAVA	16
2.2.1 Atributos	17
2.2.2 Métodos	17
2.3 JDBC	18
2.4 JAVA BEAN	18
2.5 SERVLETS	21
2.6 JSP (JAVA SERVER PAGES)	22
2.7 HTML (HYPERTEXT MARKUP LANGUAGE)	23
2.8 SQL (STRUCTURED QUERY LANGUAGE)	27
2.8.1 Instrução SELECT	28
2.8.2 Instrução INSERT	30
2.8.3 Instrução UPDATE	32
2.8.4 Instrução DELETE	32
2.9 FERRAMENTAS UTILIZADAS PARA A IMPLEMENTAÇÃO	34
2.9.1 Macromedia Dreamweaver e Fireworks MX	34
2.9.2 JDK (Java Developer Kit)	34
2.9.3 Servidor Resin	34
2.9.4 Microsoft Access 2000	35
3. IMPLEMENTAÇÃO DO SISTEMA ACADÊMICO	36
3.1 RESUMO DA ANÁLISE DO SISTEMA ACADÊMICO	36
3.2 BANCO DE DADOS	36
3.3 DIAGRAMA DE CLASSES	40
3.4 CARACTERÍSTICAS DA IMPLEMENTAÇÃO	42
4. CONCLUSÃO	78
REFERÊNCIAS BIBLIOGRÁFICAS	79
ANEXO A - EXEMPLO DE CÓDIGO HTML (index.html)	80
ANEXO B - EXEMPLO DE CÓDIGO JSP (servletConsultarAluno.jsp)	81
ANEXO C - EXEMPLO DE CÓDIGO JAVA (SistemaAcademico.java)	83

LISTA DE FIGURAS

1. FIGURA 1 - Forma de declaração de atributos em JAVA.....	17
2. FIGURA 2 - Métodos em JAVA.....	17
3. FIGURA 3 - Exemplo de um construtor JAVA <i>Bean</i>	19
4. FIGURA 4 - Processo numa requisição de um <i>servlet</i>	21
5. FIGURA 5 - Exemplo de código HTML usando tabelas.	25
6. FIGURA 6 - Visualização do código interpretado pelo navegador contendo uma tabela.....	25
7. FIGURA 7 - Exemplo de código contendo um formulário.....	27
8. FIGURA 8 - Visualização do código interpretado pelo navegador contendo um formulário... 27	
9. FIGURA 9 - Diagrama entidade-relacionamento da base de dados.....	36
10. FIGURA 10 - Diagrama de classes do sistema.....	40
11. FIGURA 11 - Seqüência de operações para o cadastramento de alunos.....	42
12. FIGURA 12 - Diagrama de colaboração cadastramento de aluno.....	43
13. FIGURA 13 - Diagrama de atividades do <i>servletCadAluno</i>	43
14. FIGURA 14 - Seqüência de operações para a alteração do cadastro de alunos.....	44
15. FIGURA 15 - Diagrama de colaboração alterar aluno.....	45
16. FIGURA 16 - Diagrama de atividades do <i>servletAltAluno</i>	45
17. FIGURA 17 - Seqüência de operações para a consulta do cadastro de alunos.....	46
18. FIGURA 18 - Diagrama de colaboração consultar aluno.....	47
19. FIGURA 19 - Diagrama de atividades do <i>servletConsAluno</i>	47
20. FIGURA 20 - Seqüência de operações para o cadastramento de professor.....	48
21. FIGURA 21 - Diagrama de colaboração cadastramento de professor.....	49
22. FIGURA 22 - Diagrama de atividades do <i>servletCadProf</i>	49
23. FIGURA 23 - Seqüência de operações para a alteração do cadastro de professor.....	50
24. FIGURA 24 - Diagrama de colaboração alterar professor.. ..	51
25. FIGURA 25 - Diagrama de atividades do <i>servletAltProf</i>	51
26. FIGURA 26 - Seqüência de operações para a consulta do cadastro de professores.....	52
27. FIGURA 27 - Diagrama de colaboração consultar professor.....	53
28. FIGURA 28 - Diagrama de atividades do <i>servletConsProf</i>	53
29. FIGURA 29 - Seqüência de operações para o cadastramento de notas.....	54
30. FIGURA 30 - Diagrama de colaboração cadastramento de notas.....	55
31. FIGURA 31 - Diagrama de atividades do <i>servletCadNotas</i>	55
32. FIGURA 32 - Seqüência de operações para a alteração de notas.....	56
33. FIGURA 33 - Diagrama de colaboração alteração de notas.....	57

34. FIGURA 34 - Diagrama de atividades do servletAltNotas.	57
35. FIGURA 35- Seqüência de operações para a consulta de notas.	58
36. FIGURA 36 - Diagrama de colaboração consulta de notas.	59
37. FIGURA 37 - Diagrama de atividades do servletConsNotas.	59
38. FIGURA 38 - Seqüência de operações para o cadastramento de funcionário.	60
39. FIGURA 39 - Diagrama de colaboração cadastramento de funcionário.	61
40. FIGURA 40 - Diagrama de atividades do servletCadFunc.	61
41. FIGURA 41 - Seqüência de operações para a alteração do cadastro de funcionário.	62
42. FIGURA 42 - Diagrama de colaboração alterar funcionário.	63
43. FIGURA 43 - Diagrama de atividades do servletAltFunc.	63
44. FIGURA 44 - Seqüência de operações para a exclusão de funcionário.	64
45. FIGURA 45 - Diagrama de colaboração excluir funcionário.	65
46. FIGURA 46 - Diagrama de atividades do servletExcFunc.	65
47. FIGURA 47 - Seqüência de operações para a consulta do cadastro de funcionários.	66
48. FIGURA 48 - Diagrama de colaboração consultar funcionário.	67
49. FIGURA 49 - Diagrama de atividades do servletConsFunc.	67
50. FIGURA 50 - Operações para a alteração do cadastro de professor em disciplina.	68
51. FIGURA 51 - Diagrama de colaboração alterar professor em disciplina.	69
52. FIGURA 52 - Diagrama de atividades do servletAltFunc.	69
53. FIGURA 53 - Seqüência de operações para o cadastramento de disciplina.	70
54. FIGURA 54 - Diagrama de colaboração cadastramento de disciplina.	71
55. FIGURA 55 - Diagrama de atividades do servletCadDisc.	71
56. FIGURA 56 - Seqüência de operações para a alteração do cadastro de disciplinas.	72
57. FIGURA 57 - Diagrama de colaboração alterar disciplina.	73
58. FIGURA 58 - Diagrama de atividades do servletAltDisc.	73
59. FIGURA 59 - Seqüência de operações para a exclusão de disciplina.	74
60. FIGURA 60 - Diagrama de colaboração excluir disciplina.	75
61. FIGURA 61 - Diagrama de atividades do servletExcDisc.	75
62. FIGURA 62 - Seqüência de operações para a consulta do cadastro de disciplina.	76
63. FIGURA 63 - Diagrama de colaboração consultar disciplinas.	77
64. FIGURA 64 - Diagrama de atividades do servletConsDisc.	77

1. INTRODUÇÃO

1.1 APRESENTAÇÃO

A rápida disseminação dos computadores pessoais e estações de trabalho, o aumento nas suas capacidades de processamento, e o surgimento de redes locais de alta velocidade (LAN), têm levado a desenvolvimento cada vez maior de aplicações distribuídas voltadas para internet.

Com a possibilidade de computadores autônomos se comunicarem, a informação e recursos passaram a transitar mais rapidamente e ser compartilhada com diversos usuários. Pode-se possibilitar que esses computadores passem a executar serviços, apresentando uma certa transparência de existência de várias máquinas, denominando-se sistemas distribuídos.

Faremos uso do conceito de sistemas distribuídos, haja vista a necessidade das instituições de ensino. Essas necessitam que seus sistemas interoperem, independentemente e que sejam heterogêneos e possam utilizar artifícios que possibilitem sistemas construídos em diferentes linguagens de programação, sistemas operacionais e plataformas de hardware. Um sistema que possibilite comunicação entre as suas partes, compartilhando informação na *Intranet* e *Internet*, para toda comunidade acadêmica. Outro ponto a ser levado em consideração é o balanceamento de carga, que possibilite distribuir processamentos em vários computadores da rede, e somando todos esses, pode-se obter um poder de processamento comparado a de um supercomputador.

Através de uma análise feita na FARN, Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte, este projeto de monografia tem como objetivo apresentar uma implementação de um protótipo de sistema acadêmico tendo como base uma análise orientada a objetos realizada pelo discente Marcelo Luiz de Figueiredo, integrante do projeto, utilizando as tecnologias para *Internet* como JAVA, *Servlets*, JSP e Java *Beans*. Tem como ênfase, o controle de informações do corpo discente, tais como pesquisa ao cadastro, matrícula e diário de classe. Controle de informações do corpo docente como, disciplinas, carga horária entre outras atividades.

1.2 PROBLEMA DE PESQUISA

O sistema acadêmico atual está sendo operacionalizado a partir de somente um microcomputador e o mesmo não compartilha informações com os demais na *Intranet* ou *Internet*, respeitando restrições.

Foi observado também que a FARN disponibiliza a emissão de histórico escolar e boletim via *web*, mas não há integração deste sistema com o sistema acadêmico. Procede-se a exportação de dados no sistema acadêmico e, em seguida, a importação num SGBD integrado com o servidor *web*.

Os dados acadêmicos são geridos por um sistema acadêmico que está sendo operado atualmente na FARN. Isso faz com que haja uma centralização de tarefas realizadas no mesmo a um número pequeno de pessoas. Informações acadêmicas como um simples histórico escolar, solicitado na secretaria da Instituição, é emitido em 72 horas devido ao número de tarefas para uma só pessoa.

1.3 OBJETIVOS GERAIS E ESPECÍFICOS

As informações contidas nos diagramas de caso de uso, colaboração e cenários das atividades, produzidos na etapa da análise, subsidiaram para o início da implementação do protótipo. Caracteriza-se um objetivo, converter em linhas de código todo esse material e também diagramar as telas utilizando um software para edição de gráficos para a *web*.

O objetivo específico deste trabalho é implementar um protótipo de ambiente de controle acadêmico que tenha uma interface amigável e seja operacionalizado a partir de qualquer computador com acesso à *Internet*.

1.4 METODOLOGIA

Para a implementação, as etapas citadas acima são essenciais, pois somente com essas informações, pode-se desenvolver o sistema.

Foram adquiridos, instalados e testados, softwares para a implementação: ambiente de desenvolvimento visual para *web*, *Macromedia Dreamweaver MX* e *Macromedia Fireworks MX*; um SGBD, Sistema Gerenciador de Banco de Dados, *Mysql*. Para o desenvolvimento do protótipo se fez uso de um banco de dados modelado no *Microsoft Access* devido à necessidade de rapidez dos resultados. Para termos os scripts funcionando, foi adquirido um

servidor, o *Resin* versão 2.1.2, da Empresa Caucho, sendo este bastante fácil de instalar e trabalhar.

O próximo passo foi instalar esses softwares em equipamentos cedidos pela própria Instituição FARN para que o sistema seja desenvolvido.

Então, deu-se início a implementação de acordo com os objetivos propostos e paralelamente na análise orientada a objetos. Também será feita a documentação do sistema para que, se houver uma atualização futura, a mesma se dará de forma mais fácil, observando os modelos.

Enfim, serão realizados teste de validação e segurança para que o sistema entre em funcionamento de forma ótima.

1.5 ESTRUTURA DO TRABALHO

Os capítulos subseqüentes apresentarão, por ordem seqüencial. No Capítulo 2, encontra-se o Referencial Teórico, contendo todo material pesquisado em livros e artigos sobre tecnologias e técnicas para a implementação; No capítulo 3, a Implementação do Sistema Acadêmico será apresentada contendo um breve resumo da análise, diagrama de classes e características da implementação, isto é, terá uma especificação de toso os cenários do sistema observando as funcionalidades de cada.

Por fim, teremos a conclusão do trabalho e as referências bibliográficas.

2. REFERENCIAL TEÓRICO

2.1 PROGRAMAÇÃO ORIENTADA A OBJETOS

A Programação Orientada a Objeto é uma técnica de programação criada para permitir o desenvolvimento de programas de grande complexidade.

Atualmente quase todas as linguagens de programação permitem ao programador escolher entre a técnica tradicional de programação e a Programação Orientada a Objeto.

Orientação a objeto utiliza a perspectiva de objetos (coisas, conceitos ou entidades).

A programação orientada a objetos tem como principal objetivo atingir um desenvolvimento interativo e incremental, criação rápida de protótipos, códigos reutilizáveis e extensíveis através do encapsulamento dos dados de uma modelagem do problema a ser solucionado.

Para COLEMAN, ARNOLD, BODOFF, DOLLIN, GILCHRIST, HAYES e JEREMAES (1996), no enfoque 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. O emissor da mensagem não precisa saber como o objeto responde a certas mensagens de maneira bem definida.

A programação orientada a objetos está apoiada em determinados conceitos para o bom desenvolvimento de projetos. Os principais são:

Objetos

Na programação orientada a objeto, um objeto é qualquer coisa, real ou abstrata, sobre a qual armazenamos dados e operações que manipulam os dados.

Classes

Classe é a implementação de um tipo de objeto o qual contém uma estrutura de dados e métodos que especificam operações que podem ser feitas com aquela estrutura de dados.

Métodos e Sobrecarga

São as funções que manipulam sobre o objeto desejado, ou seja, especifica a maneira como as operações são codificadas no software. Sobrecarga é a capacidade de uma operação possuir mais do que um método para sua execução.

Herança

A herança é a principal característica da programação orientada a objetos, pois é através desse mecanismo que a classe mais especializada herda todos os métodos e atributos da classe mais geral.

Encapsulamento

Todas as classes possuem duas interfaces, a Interface Pública e a Interface Privada. A interface pública compreende todas as operações e propriedades que podem ser acessadas diretamente pelas demais classes e a interface privada compreende as operações e propriedades que podem ser acessadas apenas pela própria classe, sendo que a interface privada é protegida pelo encapsulamento.

O empacotamento de dados e operações juntos, e o ocultamento destas informações chamam-se de encapsulamento, com os seguintes objetivos:

- Proteger os dados de um objeto contra corrupção.
- Protege os dados contra utilização arbitrária e que fujam aos objetivos.

O encapsulamento é importante, porque ele separa o comportamento do objeto, da maneira como o objeto é implementado.

Independência

Cada instância criada é totalmente independente da outra, mesmo que venham da mesma classe.

As vantagens de se utilizar a Orientação a Objeto são várias:

- Os mesmos objetos existem em todas as fases e uma notação única (objetos) facilita, portanto a integração entre fases de desenvolvimento (passar de uma fase para outra).
- A Encapsulação facilita as mudanças (através do isolamento).
- A hierarquia (grafo de herança) permite uma melhor reutilização.
- Hierarquia promove a flexibilidade de fazer mudanças de forma localizada.

2.2 JAVA

Java é uma linguagem totalmente orientada a objetos. Tudo, em Java, são objetos, com poucas exceções.

Uma das vantagens de Java é uma linguagem portátil, isto é, o programador é capaz de usar o mesmo código nos sistemas Windows 9x/NT, Solaris, Unix, Macintosh etc.

De acordo com BONFIM JUNIOR (2002):

Surgida em 1995, a linguagem de programação Java talvez seja aquela que mais desperte interesse, e por vários motivos. Por exemplo, sua portabilidade, o que quer dizer sua capacidade de ser deslocada entre sistemas diferentes. Um programa elaborado em Java em uma plataforma X (um sistema operacional qualquer) roda com a mesma Máquina Virtual Java (*Java Virtual Machine* – JVM).

No processo de compilação realizado pelo *javac* ocorre a transformação do código-fonte-texto puro em *bytecodes*. A arquitetura de funcionamento da JVM pode ser descrita como sendo uma plataforma menor sobre uma maior, o sistema operacional do usuário: a aplicação funciona sobre a JVM que, por sua vez, opera sobre o sistema operacional. Já uma aplicação em código nativo opera diretamente sobre o sistema operacional.

O fato de a JVM ter sempre que realizar a interpretação das instruções (*bytecodes*) em código nativo para que um programa Java seja executado implica em uma menor velocidade de execução das aplicações Java, o que não diminui de modo algum a linguagem.

No processo de conversão de código-fonte em *bytecodes* é criado um segundo arquivo de mesmo nome, mas de extensão diferente, (.class). Nele, reside as instruções para a JVM.

2.2.1 Atributos

Os atributos são propriedades de uma classe, que tem a função de armazenar valores empregados no processamento, veja abaixo (fig. 1):

```
private String cor = preto;
private int volume = 10;
private String forma = retangular;
private String material = aço;
```

FIGURA 1 - Forma de declaração de atributos em JAVA.

2.2.2 Métodos

Os métodos em Java são equivalentes às funções usadas nas outras linguagens, entretanto, ao contrário das funções que podem ser utilizadas na construção de classes ou métodos nessas outras linguagens. Os métodos em Java não podem criar uma classe. São declarados de modo semelhante à declaração das variáveis, informando o tipo seguido do nome. Veja no exemplo (fig. 2).

```
tipo_do_método nome_do_método (valores_de_passagem) {
    diretivas;
}
```

FIGURA 2 - Métodos em JAVA.

2.3 JDBC

Java possibilita que sejam feitas operações com banco de dados através da API JDBC(*Java Data Base Connectivity*). Com JDBC, é possível manipular resultados e consultas SQL, independente do banco de dados ou *driver* utilizado.

Para FIELDS (2000), diferentemente das outras linguagens de criação de scripts da *web*, como *ColdFusion*, *SSJ* e *PHP*, a *JSP* não define seu próprio conjunto de *tags* para acesso a banco de dados. Ao invés de desenvolver um outro mecanismo para acesso a banco de dados os *designers* de *JSP* escolheram alavancar a poderosa e popular API de banco de dados de Java - JDBC.

Quando uma aplicação *JSP* precisa se comunicar com um banco de dados ela o faz através de uma classe de *driver* fornecida e gravada na API JDBC. Acessar um banco de dados em *JSP* então não é uma novidade; simplesmente se atenha à API JDBC da *Sun*.

2.4 JAVA BEAN

Para LIMA, *JavaBeans* são componentes de software que são projetados para serem unidades reutilizáveis, que uma vez criados, podem ser reusados sem modificação de código, e em qualquer propósito de aplicação, seja um *applet*, um *servlet* ou qualquer outra.

Um modelo de componente é definido como um conjunto de classes e interfaces na forma de pacotes Java que deve ser usado em uma forma particular para isolar e encapsular um conjunto de funcionalidades.

Existem regras para se escrever *Beans* de forma que esses componentes possam ser reconhecidos na forma geral, sua implementação deve seguir um conjunto de regras que serão usadas pelas ferramentas para inspeção da classe, ou seja, o processo de descobrir quais as funcionalidades do *Bean* e disponibilizá-la para o usuário.

Cabe ao usuário fazer a interface entre o componente e o restante da aplicação, criando assim um novo tipo de programador, o programador de componentes.

Esta classe é um exemplo do que devemos seguir para tornar uma classe um *bean*, ou seja, o conjunto de regras que devemos obedecer para tornar o componente reconhecível por ferramentas e usável.

Um *bean*, como modelo de componente, usa a idéia de encapsulamento. Portanto, as suas variáveis devem ser acessadas somente através de métodos. As variáveis de um *Bean* são suas propriedades.

A primeira regra que devemos respeitar é o construtor. O construtor (fig. 3) de um bean deve ser sem parâmetros.

```
import java.io.Serializable;
public class MyBean implements Serializable {
    private int property1;
    private boolean property2;
    public MyBean() {
    }
    public void setProperty1(int property1) {
        this.property1 = property1;
    }
    public void setProperty2(boolean property2) {
        this.property2 = property2;
    }
    public int getProperty1() {
        return property1;
    }
    public boolean isProperty2() {
        return property2;
    }
}
```

FIGURA 3 - Exemplo de um construtor JAVA *Bean*.

Outra regra que devemos seguir diz respeito ao nome dos métodos que darão acesso as propriedades do *bean*. Os métodos podem ser divididos em duas categorias:

Métodos Acessadores:

Os métodos de leitura de propriedades devem seguir a seguinte convenção:

```
public TipoDaPropriedade getNomeDaPropriedade()
```

Como mostrado no código acima, a propriedade chamada *property1* tem o método de acesso *getProperty1*. Note que a primeira letra da propriedade deve ser minúscula enquanto a primeira letra depois do *get* deve ser em maiúscula. A palavra *TipoDaPropriedade* deve ser substituída pelo tipo da propriedade. Para variáveis "booleanas" vale o mesmo conceito, mas, ao invés do *get* usamos a palavra *is*. No exemplo, temos o acesso à propriedade *property2* que é booleana como:

```
public boolean isProperty2()
```

Métodos Modificadores:

Os métodos de escrita permitem ao usuário do componente modificar o conteúdo da propriedade.

Ele segue a seguinte terminologia:

```
public void setNomeDaPropriedade(TipoDaPropriedade varName)
```

Nos métodos de escrita não há caso especial para variáveis booleanas que também podem ser modificadas através de métodos set.

Pode-se não definir métodos de escrita para uma determinada variável, fazendo com que o usuário não seja capaz de modificar o conteúdo da propriedade.

Outra regra que um *Bean* deve seguir é a implementação da interface *Serializable*. A implementação desta interface fornece ao *bean* a propriedade chamada de persistência.

O conceito de persistência é permitir que o usuário faça uso do componente em um determinado momento e possa salvar o seu estado para o uso posterior partindo do mesmo ponto. A tecnologia que possibilita essa propriedade é a "Serialização de Objetos". Tal tecnologia permite que se salve o objeto em um fluxo para posterior recuperação. Quando houver a recuperação, o objeto deve se comportar como se fosse exatamente o mesmo de quando foi salvo. Se olharmos o objeto como uma máquina de estado, então podemos dizer que o objeto é recuperado no mesmo estado de quando foi salvo.

A tecnologia de serialização de objetos é muito importante em Java, pois permite à linguagem atividades complexas como computação distribuída, além de muitas outras funcionalidades interessantes.

Para serializar objetos basta que a classe implemente a interface *Serializable*.

Propriedades de um Bean:

Os *beans* possuem diversos tipos de propriedades, que são:

Simples: Propriedades simples alteram a aparência ou comportamento do *bean* e são acessadas através dos métodos de escrita e leitura mostrados.

Ligadas: As propriedades deste tipo quando alteradas provocam uma notificação para algum evento.

Reprimidas: As propriedades podem ter sua mudança vetada pelo próprio *bean* ou por objetos externos.

Indexadas: Propriedades indexadas representam coleções de valores acessados por índices, como *arrays*. A seguinte regra deve ser seguida por propriedades indexadas:

Para as propriedades indexadas temos:

Métodos de leitura do array inteiro:

```
public TipoDaPropriedade[ ] getNomeDaPropriedade( );  
public void setNomeDaPropriedade(TipoDaPropriedade[ ] value);
```

Métodos de leitura de elementos individuais:

```
public TipoDaPropriedade getNomeDaPropriedade(int index);  
public void setNomeDaPropriedade(int index, TipoDaPropriedade value);
```

2.5 SERVLETS

Segundo BATISTA, *Servlets* são a alternativa Java para *CGI Scripts*. Um Servidor de HTTP com suporte a *servlets*, ao invés de executar um programa para cada requisição CGI, repassa-o para uma Máquina *Servlet* (fig. 4), que só ativa o programa se ele não foi ativado num passado recente, significando, na maioria dos casos, um aumento substancial de desempenho e redução dos requerimentos de memória.

Por serem escritos em Java, *servlets* têm à disposição todas as APIs e softwares disponíveis na plataforma, o que lhes conferem alta flexibilidade, representada por uma linguagem orientada a objetos, multiplataforma, *multithreading* e com forma de acesso a Bases de Dados padronizada.

A API de *Servlets* oferece classes para gerenciamento de sessões, *parsing* de parâmetros e várias outras atividades, incluindo *pooling* de conexões a SGBDs.

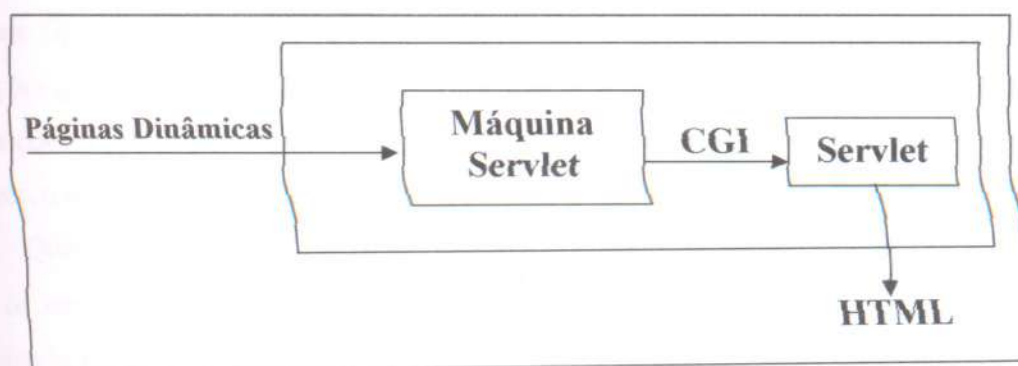


FIGURA 4 - Processo numa requisição de um *servlet*.

Uma desvantagem compartilhada por ambos, *CGI Scripts* e *Servlets*, é a falta de separação entre o trabalho de design e a geração do conteúdo dinâmico (tipicamente uma

atividade de programação). Em ambos os casos é o programador quem escreve todo o HTML para o cliente, mesmo a parte teoricamente fixa de uma página, dificultando sobremaneira a manutenibilidade das páginas.

Desde o princípio surgiram algumas alternativas para tratar da separação entre conteúdo fixo e mutável dentro de uma mesma página. Uma solução razoável é armazenar as partes fixas em arquivos associadas a classes que fazem acesso a bases de dados para obter o conteúdo dinâmico. Um *servlet* pode, no momento de escrever a página (como um todo), usar as classes e combinar os resultados obtidos nas bases de dados com o conteúdo dos arquivos, lidos e repassados sem quaisquer mudanças. Isto aumenta a manutenibilidade, pois mudanças simples de design não são feitas em programas, mas em arquivos HTML.

Uma idéia ainda melhor é criar um arquivo em forma de *template* da página final, formado pelo conteúdo fixo e entremeado de TAGS indicando a parte dinâmica. Esta técnica conhecida de forma geral como *server-side include*, tem muita variante no mercado. Esta solução acomoda todo o conteúdo estático em um único arquivo, sendo, portanto fácil de ser administrado e desenvolvido por designers. A desvantagem principal é a interpretação da página a cada requisição.

2.6 JSP (JAVA SERVER PAGES)

TELEMACO NETO descreve que JSP (Java Server Pages) é uma tecnologia para desenvolvimento de aplicações WEB semelhante ao *Microsoft Active Server Pages* (ASP), porém tem a vantagem da portabilidade de plataforma, podendo ser executado em outros Sistemas Operacionais além dos da Microsoft. Ela permite que o desenvolvedor produza aplicações que permitam o acesso à banco de dados, o acesso a arquivos-texto, a captação de informações a partir de formulários, a captação de informações sobre o visitante e sobre o servidor, o uso de variáveis e *loops*, entre outras coisas.

Quem conhece *servlets* verá que o JSP não oferece nada que você não possa conseguir com os *servlets* puros. O JSP, entretanto, oferece a vantagem de ser facilmente codificado, facilitando assim a elaboração e manutenção de uma aplicação. Além disso, essa tecnologia permite separar a programação lógica (parte dinâmica) da programação visual (parte estática), facilitando o desenvolvimento de aplicações mais robustas, onde programador e designer podem trabalhar no mesmo projeto, mas de forma independente. Outra característica do JSP é produzir conteúdos dinâmicos que possam ser reutilizados.

Quando uma página JSP é requisitada pelo cliente através de um *Browser* esta página é executada pelo servidor e, a partir daí, será gerada uma página HTML que será enviada de volta ao browser do cliente.

Quando o cliente faz a solicitação de um arquivo JSP, é enviado um *object request* para a *JSP engine*. A *JSP engine* envia a solicitação de qualquer componente (podendo ser um *JavaBeans component*, *servlet* ou *enterprise Bean*) especificado no arquivo.

O componente controla a requisição possibilitando a recuperação de arquivos em banco de dados ou outro dado armazenado, em seguida, passa o objeto *response* de volta para a *JSP engine*. A *JSP engine* e o *WEB server* enviam a página JSP revisada de volta para o cliente, onde o usuário pode visualizar os resultados através do *WEB browser*. O protocolo de comunicação usado entre o cliente e o servidor pode ser HTTP ou outro protocolo.

Por definição, JSP usa Java como sua linguagem de scripts. Por esse motivo, O JSP apresenta-se mais flexível e mais robusto do que outras plataformas baseadas simplesmente em *JavaScripts* e *VBScripts*.

2.7 HTML (HYPERTEXT MARKUP LANGUAGE)

HyperText Markup Language ou Linguagem de Marcação HiperTexto é uma linguagem simples composta de marcações de formatação e diagramação de hipertexto/hipermídia (informações em texto, imagens, sons e ações ligadas umas às outras de uma forma complexa e não-sequencial através de chaves relacionadas). É a linguagem da WWW (*World Wide Web*), justamente por essa capacidade de formatação e diagramação de hipertexto/hipermídia. A HTML é interpretada por todos os *browsers* disponíveis (Netscape, Internet Explorer, Mosaic, etc...).

O documento em HTML é um arquivo texto comum (ASCII) que pode ser escrito através de qualquer editor de textos comum e tem a terminação .htm ou .html.

Dentro de um documento HTML temos elementos de marcação denominados *TAGS*. Uma TAG é definida através de seu nome cercado pelos sinais de “menor” (<) e “maior” (>) e normalmente tem uma TAG correspondente para finalização, com o mesmo nome e precedido por uma barra (/).

Ex.: <BODY> corpo do documento ... </BODY> (define o início e o fim do corpo do documento HTML)

Algumas TAGS admitem atributos (parâmetros) que alteram a maneira como o navegador deve interpretá-la e normalmente são colocados na TAG inicializadora.

Ex.: `<BODY BACKGROUND="fundo.gif">` corpo do documento `</BODY>` (define que o documento HTML terá como fundo a figura *fundo.gif*)

Todo documento HTML fica contido entre as tags `<HTML>` e `</HTML>`. Dentro dessas tags temos duas seções indicando a formatação de texto, tabelas, formulários e outros elementos

Segue abaixo o conjunto de marcações essenciais para desenhar tabelas em HTML. Alguns atributos podem ser definidos para cada uma dessas marcações.

- **`<TABLE></TABLE>`**

Toda tabela deve ser iniciada com a marcação `<table>` e encerrada com `</table>`. Para que a tabela apareça com bordas simples, defina dessa forma `<table border></table>`. Antes e depois de uma tabela, acontece sempre uma quebra de linha.

- **`<TR></TR>`**

Cada linha de uma tabela deve sempre aparecer entre as marcações `<tr>` e `</tr>`. (TR = Table Row)

- **`<TD></TD>`**

Esta é a marcação que define cada célula de uma tabela. As células de uma tabela devem sempre aparecer entre as marcações de linhas (`<tr>` e `</tr>`). Como padrão, o texto nas células é alinhado a esquerda.

- **`<TH></TH>`**

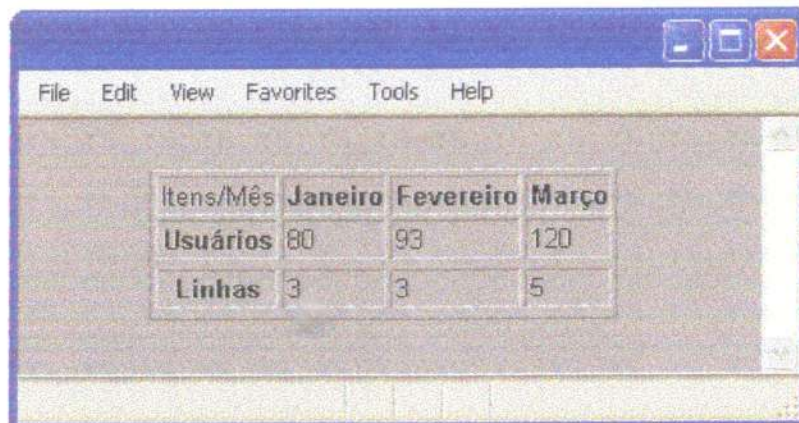
Desta forma são definidos os títulos de uma tabela. Estes podem ser posicionados em qualquer célula. A diferença entre a marcação de célula e título de célula é que o título aparece em negrito.

Exemplo de uma tabela utilizando essas marcações básicas.

```
<html>
  <head>
    <title>Título</title>
  </head>
  <body>
    <table border>
      <tr>
        <td>Itens/Mês</td>
        <th>Janeiro</th><th>Fevereiro</th><th>Março</th>
      </tr>
      <tr>
        <th>Usuários</th><td>80</td><td>93</td><td>120</td>
      </tr>
      <tr>
        <th>Linhas</th><td>3</td><td>3</td><td>5</td>
      </tr>
    </table>
  </body>
</html>
```

FIGURA 5 - Exemplo de código HTML usando tabelas.

Veja como as tags acima serão a exibidas no navegador:



Itens/Mês	Janeiro	Fevereiro	Março
Usuários	80	93	120
Linhas	3	3	5

FIGURA 6 - Visualização do código interpretado pelo navegador contendo uma tabela.

Os "Formulários Eletrônicos" permitem um usuário da Web submeter informações para o provedor do serviço.

Existem vários tipos de campos de entrada, como:

- campos de entrada de texto
- menus de múltipla escolha ou escolha única
- botões sim-ou-não
- botões para submissão ou limpeza de formulário

Um formulário começa com a marcação **<FORM>** e termina com **</FORM>**. Outros itens devem ser especificados:

- **Primeiro**, o formulário precisa saber como enviar a informação para o servidor. Existem dois métodos, GET e POST.

METHOD="GET"

A maioria dos documentos HTML são recuperados a partir da requisição de uma única URL ao servidor. Assim, um formulário que utilize este método, envia toda sua informação ao final da URL ativada.

METHOD="POST"

Este método transmite toda a informação fornecida via formulário, imediatamente após a URL ativada. Ou seja, quando o servidor recebe uma ativação de um formulário utilizando POST ele sabe que precisa continuar "ouvindo" para obter a informação. Este é o melhor método.

Segundo, o formulário precisa saber para onde enviar a informação. Esta é a URL sendo ativada a partir do formulário, e ela é referenciada através da marcação **ACTION**. Esta URL em geral aponta para um script CGI que irá receber e decodificar os resultados. Lembre-se que se você está referenciando um script que reside no mesmo servidor do formulário, você não precisa incluir a URL completa.

Após você construir estas marcações, seu formulário geralmente terá a seguinte estrutura:

```
<html>
  <head>
    <title>Titulo</title>
  </head>
  <body>
    <form method="post" action="/cgi-bin/post-query">
      Qual o seu primeiro nome?
      <input type="text" name="primeiro_nome" value="carlos"
        size="10" maxlength="15">
      Informe sua senha de acesso (8 caracteres):
      <input type="password" value="xpto" name="sua_senha"
        size="8" maxlength="8">
    </form>
  </body>
</html>
```

FIGURA 7 - Exemplo de código contendo um formulário.

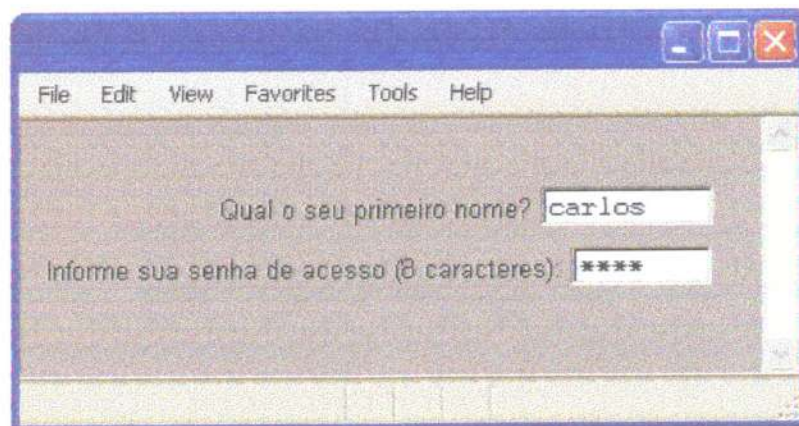


FIGURA 8 - Visualização do código interpretado pelo navegador contendo um formulário.

2.8 SQL (STRUCTURED QUERY LANGUAGE)

O Sistema Acadêmico usará um banco de dados relacional onde, para realizar algumas consultas e demais rotinas, se fará o uso da linguagem SQL.

Quando os Bancos de Dados Relacionais estavam sendo desenvolvidos, foram criadas linguagens destinadas à sua manipulação. O Departamento de Pesquisas da IBM desenvolveu

a SQL como forma de interface para o sistema de BD relacional denominado SYSTEM R, início dos anos 70. Em 1986 o *American National Standard Institute* (ANSI), publicou um padrão SQL. A SQL estabeleceu-se como linguagem padrão de Banco de Dados Relacional.

A Linguagem SQL tem como grandes virtudes a capacidade de gerenciar índices, sem a necessidade de controle individualizado de índice corrente, algo muito comum nas linguagens de manipulação de dados do tipo registro a registro. Outra característica muito importante disponível em SQL é sua capacidade de construção de visões, que são formas de visualizarmos os dados na forma de listagens independente das tabelas e organização lógica dos dados.

Outra característica interessante na linguagem SQL é a capacidade de cancelar uma série de atualizações ou de gravarmos essas atualizações, depois de iniciarmos uma sequência de atualizações. Os comandos *Commit* e *Rollback* são responsáveis por estas facilidades.

Devemos notar que a linguagem SQL consegue implementar estas soluções, somente pelo fato de estar baseada em Banco de Dados, que garantem por si mesmo a integridade das relações existentes entre as tabelas e seus índices.

Com a linguagem SQL, podem-se realizar tarefas básicas num banco de dados como consultas, inserções, alterações e exclusão de registros.

2.8.1 Instrução SELECT

Instrui o programa principal do banco de dados para retornar a informação como um conjunto de registros.

Sintaxe:

```
SELECT [predicado { * | tabela.* | [tabela.]campo1 [AS alias1] [, [tabela.]campo2 [AS alias2]
[ ...] ]}
FROM expressão_tabela [, ...] [IN bancodedadosexterno]
[WHERE... ]
[GROUP BY... ]
[HAVING... ]
[ORDER BY... ]
[WITH OWNERACCESS OPTION]
```

A instrução **SELECT** tem as partes abaixo:

Parte	Descrição
predicado	Um dos seguintes predicados: ALL , DISTINCT , DISTINCTROW ou TOP . Você usa o predicado para restringir o número de registros que retornam. Se nenhum for especificado, o padrão será ALL .
*	Especifica que todos os campos da tabela ou tabelas especificadas são selecionados.
tabela	O nome da tabela que contém os campos dos quais os registros são selecionados.
campo1, campo2	Os nomes dos campos dos quais os dados serão recuperados. Se você incluir mais de um campo, eles serão recuperados na ordem listada.
alias1, alias2	Os nomes que serão usados como títulos de colunas em vez dos nomes originais das colunas na tabela.
expressãotabela	O nome da tabela ou tabelas contendo os dados que você quer recuperar.
bancodedadosexterno	O Nome do banco de dados que contém as tabelas em expressão tabela se não estiver no banco de dados atual.

Para executar esta operação, o programa principal de banco de dados procura a tabela ou tabelas especificadas, extrai as colunas escolhidas, seleciona as linhas que satisfazem o critério e classifica ou agrupa as linhas resultantes na ordem especificada.

A instrução **SELECT** não muda os dados no banco de dados.

SELECT é normalmente a primeira palavra em uma instrução **SQL**. A maior parte das instruções **SQL** são instruções **SELECT**.

A sintaxe mínima da instrução **SELECT** é:

SELECT campos **FROM** tabela

Você pode usar um asterisco (*) para selecionar todos os campos na tabela. O exemplo abaixo seleciona todos os campos na tabela **Funcionários**:

SELECT * FROM Funcionários;

2.8.2 Instrução INSERT

Adiciona um ou vários registros a uma tabela. Isto é referido como consulta anexação.

Sintaxe:

Consulta anexação de vários registros:

```
INSERT INTO destino [IN bancodedadosexterno] [(campo1[, campo2[, ...]])]
SELECT [origem.]campo1[, campo2[, ...]]
FROM expressãodetabela
```

Consulta anexação de um único registro:

```
INSERT INTO destino [(campo1[, campo2[, ...]])]
VALUES (valor1[, valor2[, ...]])
```

A instrução *INSERT INTO* tem as partes abaixo:

Parte	Descrição
<i>destino</i>	O nome da tabela ou consulta em que os registros devem ser anexados.
<i>bancodedadosexterno</i>	O caminho para um banco de dados externo. Para uma descrição do caminho, consulte a cláusula IN.
<i>origem</i>	O nome da tabela ou consulta de onde os dados devem ser copiados.
<i>campo1, campo2</i>	Os nomes dos campos aos quais os dados devem ser anexados, se estiverem após um argumento destino ou os nomes dos campos dos quais se deve obter os dados, se estiverem após um argumento origem.
<i>expressãodetabela</i>	O nome da tabela ou tabelas das quais registros são inseridos. Este argumento pode ser um único nome de tabela ou uma combinação resultante de uma operação INNER JOIN, LEFT JOIN ou RIGHT JOIN ou de uma consulta gravada.
<i>valor1, valor2</i>	Os valores para inserir em campos específicos do novo registro. Cada valor é inserido no campo que corresponde à posição do valor na lista: Valor1 é inserido no campo1 do novo registro, valor2 no campo2 e assim por diante. Você deve separar os valores com uma vírgula e colocar os campos de textos entre aspas (" ").

Pode-se usar a instrução *INSERT INTO* para adicionar um único registro a uma tabela usando a sintaxe de consulta anexação de um único registro como mostrado acima. Neste caso, o código especifica o nome e o valor de cada campo do registro. É preciso especificar cada um dos campos do registro para os quais um valor deve ser designado e um valor para este campo. Quando não se especifica cada campo, o valor padrão ou *Null* é inserido nas colunas omitidas. Os registros são adicionados no final da tabela.

Pode-se também usar *INSERT INTO* para anexar um conjunto de registros de outra tabela ou consulta usando a cláusula *SELECT ... FROM* como é mostrado acima na sintaxe consulta anexação de vários registros. Neste caso, a cláusula *SELECT* especifica os campos para acrescentar à tabela destino especificada.

A tabela de origem ou de destino pode especificar uma tabela ou uma consulta. Se uma consulta for especificada, o programa principal de banco de dados anexa a qualquer e a todas as tabelas especificadas pela consulta.

INSERT INTO é opcional, mas quando incluída, precede a instrução *SELECT*.

Se a tabela de destino contém uma chave primária, deve-se acrescentar valores únicos, não *Null* ao campo ou campos da chave primária. Caso contrário, o programa principal de banco de dados do *Jet* não anexará os registros.

Se anexar registros a uma tabela com um campo *Counter* e quiser numerar novamente os registros anexados, não é feita a inclusão do campo *Counter* em sua consulta. Inclui-se o campo *Counter* na consulta se quiser manter os valores originais do campo.

Use a cláusula *IN* para anexar registros a uma tabela de outro banco de dados. Para achar quais registros serão anexados, antes de executar a consulta anexação, primeiro se executa e vê os resultados de uma consulta seleção que use o mesmo critério de seleção.

Uma operação de consulta anexação copia os registros de uma ou mais tabelas em outra. As tabelas que contêm os registros que você anexa não são afetadas pela operação de consulta anexação.

Em lugar de acrescentar registros existentes de outra tabela, você pode especificar o valor de cada campo em um único registro novo usando a cláusula *VALUES*. Se você omitir a lista de campo, a cláusula *VALUES* deve incluir um valor para cada campo na tabela; caso contrário, um erro ocorrerá em *INSERT*. Use uma instrução adicional *INSERT INTO* com uma cláusula *VALUES* para cada registro adicional que você quiser criar.

Exemplo de instrução *INSERT INTO*:

Esse exemplo seleciona todos os registros de uma tabela hipotética "Novos Clientes" e os adiciona à tabela "Clientes" (quando não são designadas colunas individuais, os nomes das colunas das tabelas *SELECT* devem corresponder exatamente aos da tabela *INSERT INTO*).

```
INSERT INTO Clientes SELECT [Novos Clientes].* FROM [Novos Clientes];
```

2.8.3 Instrução UPDATE

Cria uma consulta atualização que altera os valores dos campos em uma tabela especificada com base em critérios específicos.

Sintaxe:

```
UPDATE tabela SET valornovo WHERE critério;
```

A instrução *UPDATE* tem as partes abaixo:

Parte	Descrição
tabela	O nome da tabela cujos os dados você quer modificar.
valornovo	Uma expressão que determina o valor a ser inserido em um campo específico nos registros atualizados.
critério	Uma expressão que determina quais registros devem ser atualizados. Só os registros que satisfazem a expressão são atualizados.

UPDATE é especialmente útil quando se quer alterar muitos registros ou quando os registros que queira alterar estão em várias tabelas. Pode-se alterar vários campos ao mesmo tempo. O exemplo abaixo aumenta o Valor do Pedido em 10 por cento e o valor do Frete em 3 por cento para embarques do Reino Unido:

```
UPDATE Pedidos SET ValorPedido = ValorPedido * 1.1, Frete = Frete * 1.03 WHERE PaísEmbarque = 'RU';
```

2.8.4 Instrução DELETE

Cria uma consulta exclusão que remove registros de uma ou mais tabelas listadas na cláusula *FROM* que satisfaz a cláusula *WHERE*.

Sintaxe:

DELETE [tabela.*] **FROM** tabela **WHERE** critério

A instrução *DELETE* tem as partes abaixo:

Parte	Descrição
tabela.*	O nome opcional da tabela da qual os registros são excluídos.
tabela	O nome da tabela da qual os registros são excluídos.
critério	Uma expressão que determina qual registro deve ser excluído.

DELETE é especialmente útil quando se quer excluir muitos registros. Para eliminar uma tabela inteira do banco de dados, pode-se usar o método *Execute* com uma instrução *DROP*.

Entretanto, se eliminar a tabela, a estrutura é perdida. Por outro lado, quando se usa *DELETE*, apenas os dados são excluídos. A estrutura da tabela e todas as propriedades da tabela, como atributos de campo e índices, permanecem intactos.

Pode-se usar *DELETE* para remover registros de tabelas que estão em uma relação um por vários com outras tabelas. Operações de exclusão em cascata fazem com que os registros das tabelas que estão no lado "vários" da relação sejam excluídos quando os registros correspondentes do lado "um" da relação são excluídos na consulta. Por exemplo, nas relações entre as tabelas Clientes e Pedidos, a tabela Clientes está do lado "um" e a tabela Pedidos está no lado "vários" da relação. Excluir um registro em Clientes faz com que os registros correspondentes em Pedidos sejam excluídos se a opção de exclusão em cascata for especificada.

Uma consulta de exclusão exclui registros inteiros e não apenas dados em campos específicos. Se você quiser excluir valores de um campo específico, crie uma consulta atualização que mude os valores para *Null*.

Exemplo de instrução *DELETE*:

Esse exemplo exclui todos os registros de funcionários cujo título seja Estagiário. Quando a cláusula *FROM* inclui apenas uma tabela, não é necessário indicar o nome da tabela na instrução *DELETE*.

DELETE *FROM Funcionários WHERE Título = 'Estagiário';

2.9 FERRAMENTAS UTILIZADAS PARA A IMPLEMENTAÇÃO

2.9.1 Macromedia Dreamweaver e Fireworks MX

Para a edição de códigos fonte nas linguagens HTML, Java e JSP, se fez o uso do software *Macromedia Dreamweaver MX*, este sendo um editor (WYSIWYG) do termo *What you see is what you get*. Ou seja, é um editor visual onde o desenvolvedor verá como o produto final ficará na área de design.

O *Dreamweaver MX* é versátil oferecendo uma biblioteca de tags em HTML e JSP, possibilitando que o desenvolvedor possa reutilizá-las. Sendo este o principal motivo da escolha.

O design (imagens e figuras) do sistema foi concebido utilizando o software *Macromedia Fireworks MX*, ferramenta que possibilita a integração com o *Dreamweaver* e apresenta uma interface fácil possibilitando a rapidez na criação gráfica.

Ambos softwares são de distribuição livre, mas com o prazo de utilização de trinta dias.

2.9.2 JDK 1.4(Java Developer Kit)

O JDK versão 1.4 é um kit de ferramentas da *Sun Microsystems* para compilar, depurar, executar e documentar programas em Java. É de distribuição livre e encontra-se no seguinte endereço: (<http://java.sun.com>).

2.9.3 Servidor Resin

O servidor *Resin* encontra-se na versão 2.1.2 e é desenvolvido pela Empresa Caucho (<http://www.caucho.com>), sendo de distribuição livre. O *Resin* é simples de instalar e operar. Para inicializá-lo, basta acessar a pasta *bin* do *Resin* e executar o arquivo (*httpd.exe*). Para acessar às páginas JSP, basta digitar na barra de endereços do *browser* o seguinte endereço (<http://host:8080/diretório/arquivo>).

2.9.4 Microsoft Access 2000

A escolha do banco de dados Access veio da necessidade de se ter um protótipo do sistema em curto espaço de tempo, devido à facilidade de se criar um banco de dados no mesmo, acessá-lo ou portá-lo para outros computadores.

Sabe-se que este produto da *Microsoft* não oferece a segurança e o porte desejado para o sistema, portanto, deverá se usar uma outra solução que se adapte ao cliente, a Instituição FARN. Poderemos portar para o SGBD *Mysql* que apresenta as características aspiradas para o sistema.

3. IMPLEMENTAÇÃO DO SISTEMA ACADÊMICO

3.1 RESUMO DA ANÁLISE DO SISTEMA ACADÊMICO

Como primeira etapa da análise, as entrevistas foram realizadas com funcionários da FARN, em específico, lotados na Secretaria e CPD. Requisitos foram coletados que serviram para se modelar diagramas tais como, casos de uso, colaboração e classes seguidas de cenários apresentando ações desses funcionários.

3.2 BANCO DE DADOS

Foi elaborado o modelo relacional do banco de dados, desenvolvido no Access 2000, após análise de requisitos com usuários observando o que seria armazenado. Futuramente, a base de dados será migrada para um outro SGBD, oferecendo mais segurança e robustez.

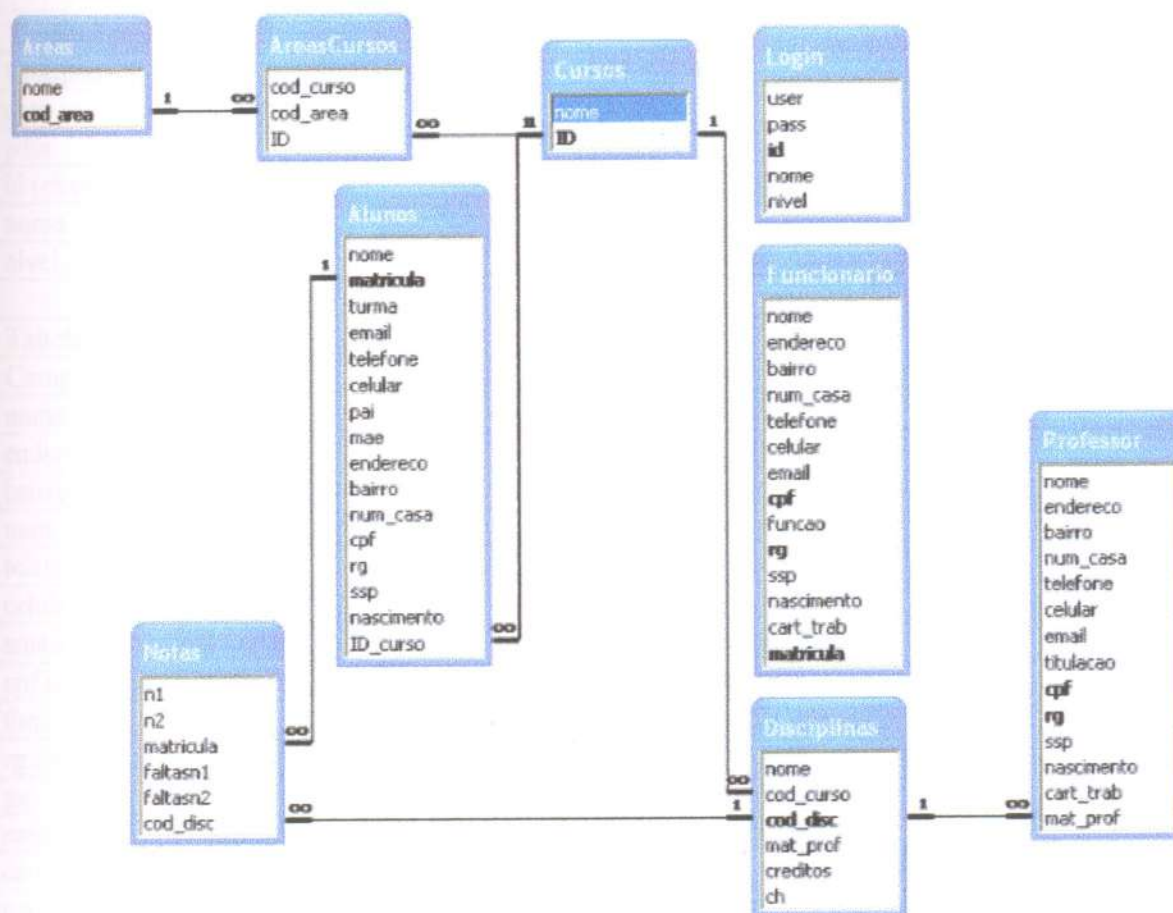


FIGURA 9 - Diagrama entidade-relacionamento da base de dados.

Segue uma descrição dos campos das tabelas.

Tabela Areas	
Campos	Descrição
nome	Nome da área (Humanas ou Tecnológica)
cod_area (chave primária)	Identificador da área

Tabela AreasCursos	
Campos	Descrição
cod_curso	Identificador do curso independente de área
cod_area (chave estrangeira)	Identificador da área
ID (chave estrangeira)	Elo identificando qual curso do aluno

Tabela Cursos	
Campos	Descrição
nome	Nome do Curso
ID (chave primária)	Elo identificando qual curso do aluno

Tabela Login	
Campos	Descrição
user	Usuário
pass	Senha
id (chave primária)	Identificador
nome	Nome do usuário
nível	Nível de Restrições

Tabela Funcionario	
Campos	Descrição
nome	Nome do Funcionário
endereço	Endereço onde o Funcionário reside
bairro	Bairro onde o Funcionário reside
num_casa	Número da casa
telefone	Telefone Residencial
celular	Telefone Celular
email	Correio Eletrônico
cpf (chave primária)	Número do CPF
funcao	Função na Empresa
rg (chave primária)	Número do RG
ssp	Estado de registro
nascimento	Data de nascimento
cart_trab	Número da carteira profissional
matricula (chave primária)	Matrícula na instituição

Tabela Alunos	
Campos	Descrição
nome	Nome completo do aluno
matricula (chave primária)	Matricula na instituição
turma	Turma "A" ou "B"
email	Correio eletrônico
telefone	Telefone residencial
celular	Telefone celular
pai	Nome completo do pai
mae	Nome completo da mãe
endereço	Endereço onde o aluno reside
bairro	Bairro onde o aluno reside
num_casa	Número da casa
cpf	Número do CPF
rg	Número do RG
ssp	Estado de registro
nascimento	Data de nascimento
ID_curso (chave estrangeira)	Elo identificando qual curso do aluno

Tabela Notas	
Campos	Descrição
n1	Primeira nota
n2	Segunda nota
matricula (chave estrangeira)	Matricula do aluno
faltasn1	Número de faltas da primeira etapa
faltasn2	Número de faltas da segunda etapa
cod_disc (chave estrangeira)	Identificador da disciplina

Tabela Disciplinas	
Campos	Descrição
nome	Nome da disciplina
cod_curso (chave estrangeira)	Identificador do curso
cod_disc (chave primária)	Identificador da disciplina
mat_prof (chave primária)	Matricula do professor
creditos	Créditos da disciplina
ch	Carga horária

Tabela Professor	
Campos	Descrição
Nome	Nome do professor
Endereço	Endereço onde o professor reside
Bairro	Bairro onde o professor reside
num_casa	Número da casa
Telefone	Número do telefone residencial
Celular	Número do telefone celular
Email	Correio eletrônico
Titulação	Grau de titulação do professor
cpf (chave primária)	Número do CPF
rg (chave primária)	Número do RG
Ssp	Estado de registro
Nascimento	Data de nascimento
cart_trab	Número da carteira profissional
mat_prof (chave estrangeira)	Matrícula na instituição

Relacionamentos:

- Áreas → AreasCursos ← Cursos:

Determina qual é a área e curso que um aluno está matriculado. Atualmente na FARN, temos os seguintes cursos e áreas com seus respectivos códigos.

Área A: Humanas

A01 - Administração

A02 - Ciências Contábeis

A03 - Direito

A04 - Administração com Habilitação em Marketing

Área B: Tecnológica

B01 - Bacharelado em Sistemas de Informação

B02 - Licenciatura em Computação

- Alunos → Cursos:

Determina uma coleção de alunos pertencentes a um determinado curso.

- Notas → Alunos:

Coleção de notas de todas as disciplinas pertencentes a um aluno.

- Notas → Disciplinas:

Amarração indicando notas de uma disciplina específica cursada por um determinado aluno.

- Disciplinas → Cursos:

Coleção de disciplinas pertencentes a um curso.

- Professor → Disciplinas:

Relação de professores associados a disciplinas.

3.3 DIAGRAMA DE CLASSES

O diagrama abaixo apresenta as classes do sistema onde dão indícios de quais métodos serão implementados. Nas classes *Disciplina*, *Nota*, *Aluno*, *Prof* e *Funcionario*, apresentam atributos e métodos usados nos *servlets* que farão a apresentação dos resultados nas telas.

A classe *SistemaAcademico*, é a principal onde todas as funcionalidade que envolvem comunicação com o banco de dados serão implementados com seus métodos. Esta classe será utilizada por todos os *servlets*, sendo caracterizada como um *bean*. Veja (fig. 10).

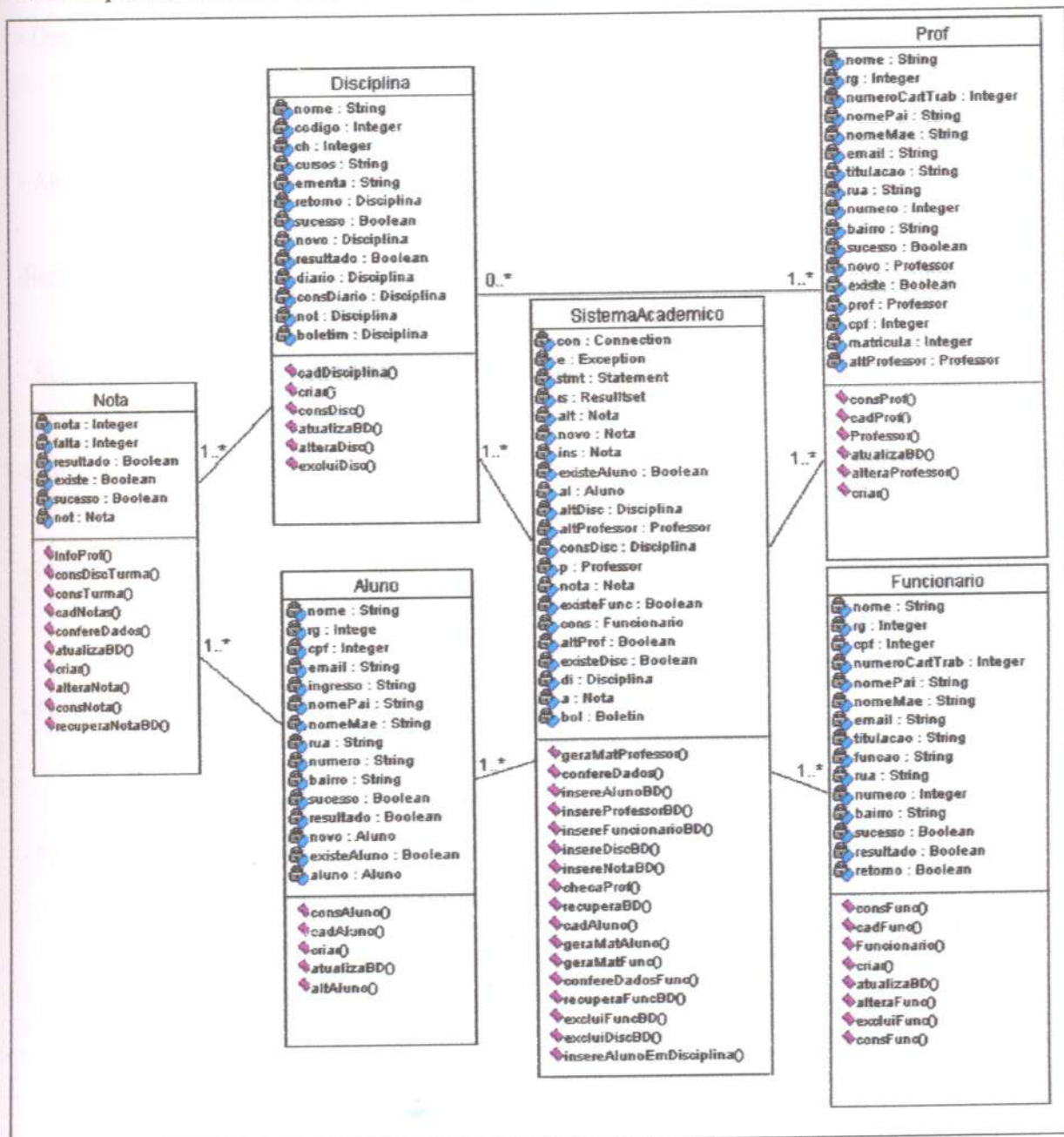


FIGURA 10 - Diagrama de classes do sistema.

Relacionamentos:

- Disciplina → Prof:

Podemos ter uma situação em que um professor não esteja ministrando uma disciplina, mas o mesmo possui vínculo com a Instituição, isso explica a cardinalidade "0..*". Cada Disciplina está associada a um ou mais professor.

- Disciplina → Nota:

Podemos ter uma coleção de notas associadas a uma disciplina específica.

- Aluno → Nota:

Temos a situação em que um aluno possui várias notas e essas associadas a disciplinas.

- SistemaAcademico → Disciplina:

No sistema acadêmico, teremos várias disciplinas ligadas, as mesmas associadas a um curso com seu identificador único.

- SistemaAcademico → Aluno:

No sistema acadêmico, teremos vários alunos associados.

- Prof → SistemaAcademico:

No sistema acadêmico, teremos vários professores associados.

- Funcionário → SistemaAcademico

No sistema acadêmico, teremos vários funcionários associados.

3.4 CARACTERÍSTICAS DA IMPLEMENTAÇÃO

Nesta seção, serão apresentadas telas do sistema integradas às funcionalidades propostas por cada tarefa. Na sequência, temos uma descrição sucinta do serviço em ordem cronológica das ações do usuário; respectivos diagramas de colaboração e atividades.

Cadastrar Aluno:

A função dessa tarefa é cadastrar novos alunos da Instituição onde teremos uma tela de *logon*, um formulário de cadastro, um *servlet* para intercambio de dados com o *bean* sistemaAcademico (fig. 11). Veja também diagramas de colaboração (fig. 12) e atividades (fig. 13).

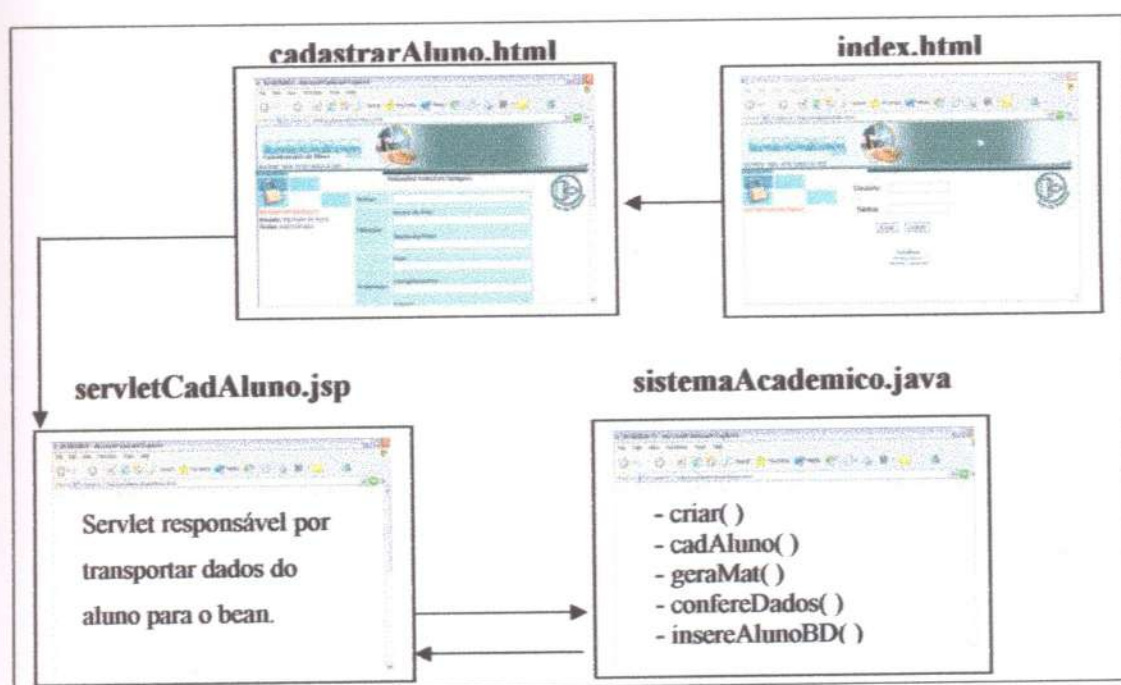


FIGURA 11- Sequência de operações para o cadastramento de alunos.

O usuário do sistema, administrador ou secretária, acessará via *browser*, a tela inicial do sistema (*index.html*). Em seguida, escolherá a opção "cadastrar aluno", onde carregará um formulário (*cadastrarAluno.html*) com campos como nome, endereço, filiação, entre outros. A matrícula será criada automaticamente seguindo a junção de parâmetros como ano de ingresso e outros. Preenchendo-os, o passo seguinte é submeter esses dados com um clique no botão "enviar".

O *servlet* (`servletCadAluno.jsp`) irá receber esses dados e organizá-lo em um objeto "Aluno". O *bean* (`sistemaAcademico.java`) se encarregará de inserir no banco de dados e retornar ao *servlet* uma resposta de erro ou sucesso.

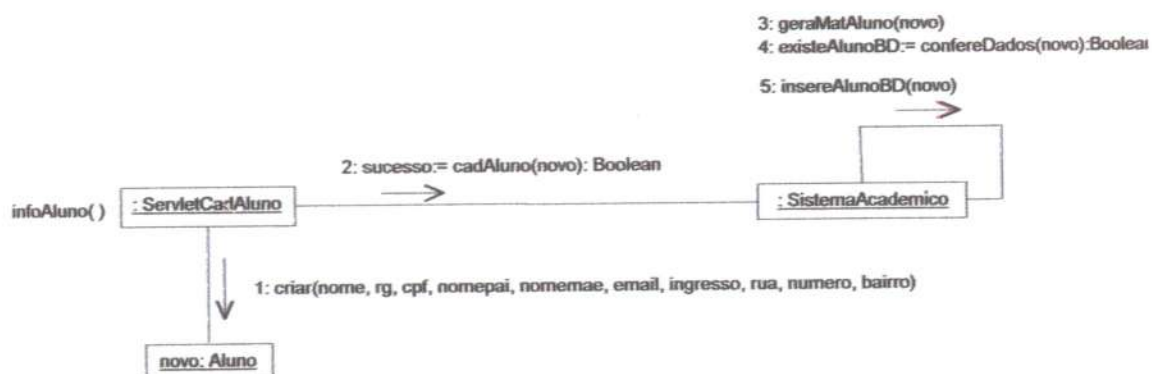


FIGURA 12 - Diagrama de colaboração cadastramento de aluno.



FIGURA 13 - Diagrama de atividades do servletCadAluno.

Alterar Aluno:

A função dessa tarefa é alterar dados de alunos da Instituição onde teremos uma tela de *logon*, um formulário de busca, outro de recadastro, um *servlet* para intercambio de dados com o *bean* sistemaAcademico (fig. 14). Veja também diagramas de colaboração (fig. 15) e atividades (fig. 16).

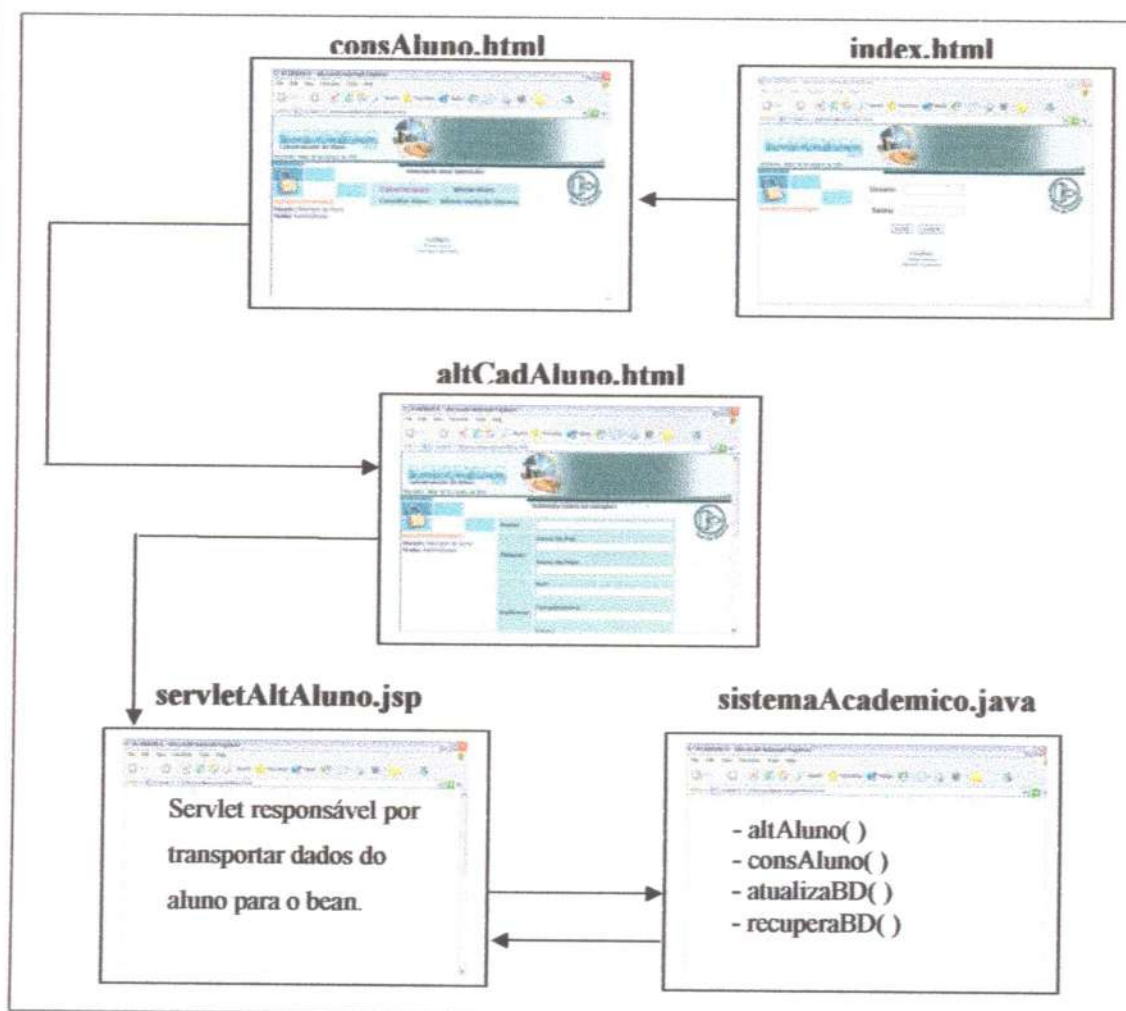


FIGURA 14 - Sequência de operações para a alteração do cadastro de alunos.

O usuário do sistema, administrador ou secretária, acessará via *browser* a tela inicial do sistema (**index.html**). Em seguida, surgirá uma tela de consulta onde poderá digitar a matrícula do aluno. Clicando na opção "alterar aluno", carregará um formulário (**altCadAluno.html**) com campos como nome, endereço, filiação, entre outros. Preenchendo o passo seguinte é submeter esses dados com um clique no botão "enviar".

O *servlet* (`servletAltAluno.jsp`) irá receber esses dados e organizá-lo em um objeto "Aluno". O *bean* (`sistemaAcademico.java`) se encarregará de alterar no banco de dados e retornar ao *servlet* uma resposta de erro ou sucesso.

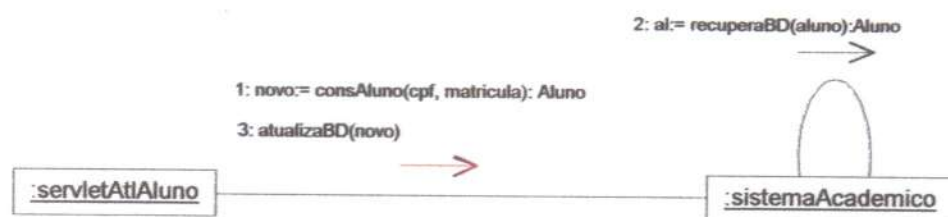


FIGURA 15 - Diagrama de colaboração alterar aluno.



FIGURA 16 - Diagrama de atividades do `servletAltAluno`.

Consultar Aluno:

Essa tarefa tem a função de realizar uma consulta de dados de alunos da Instituição onde teremos uma tela inicial de logon, formulário de busca, um *servlet* para intercambio de dados com o *bean* sistemaAcademico (fig. 17). Veja também diagramas de colaboração (fig. 18) e atividades (fig. 19).

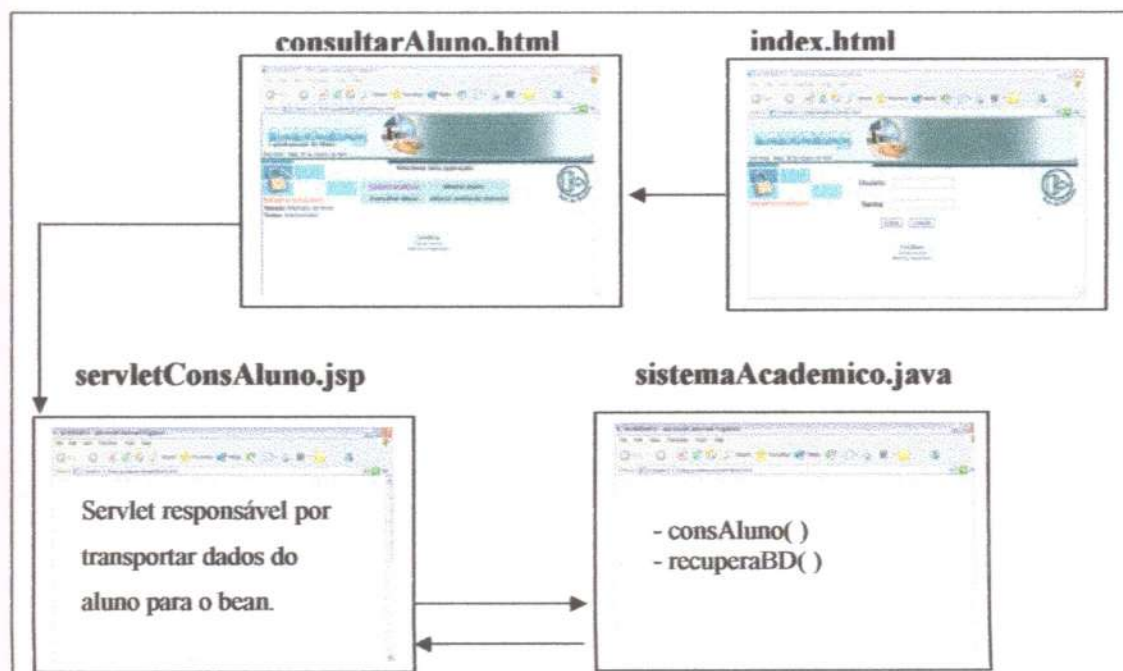


FIGURA 17 - Seqüência de operações para a consulta do cadastro de alunos.

O usuário do sistema, administrador ou secretária, acessará via browser a tela inicial do sistema (**index.html**). Em seguida, surgirá uma tela de consulta (**consultarAluno.html**) onde poderá escolher as opções de busca por curso, período, aluno e matrícula. Ao preencher o critério de busca, o usuário clicará no botão "buscar".

O *servlet* (**servletConsAluno.jsp**) irá receber esses dados do formulário e organizá-lo. Em seguida, acionará o *bean* (**sistemaAcademico.java**) que se encarregará de buscar os dados no banco de dados e retornar ao *servlet* uma resposta. O *servlet* exibirá o registro do aluno, ou caso não esteja cadastrado, uma mensagem se referindo a esse fato.

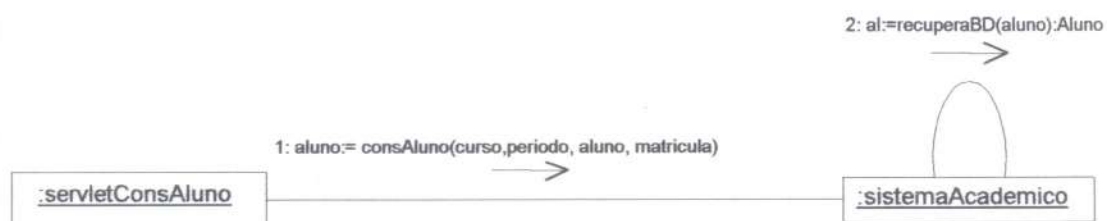


FIGURA 18 - Diagrama de colaboração consultar aluno.

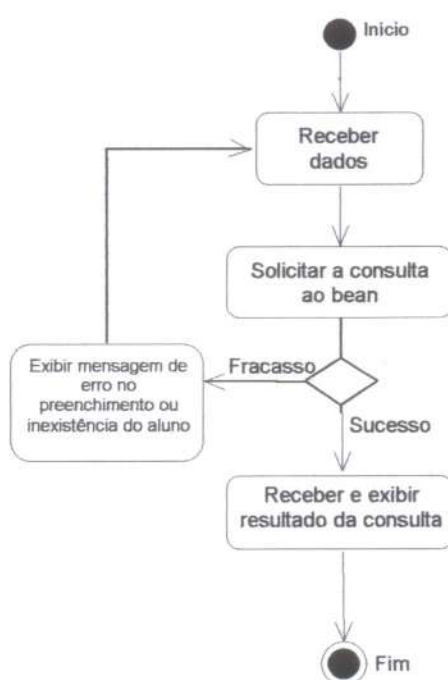


FIGURA 19 - Diagrama de atividades do servletConsAluno.

Cadastrar Professor:

A função dessa tarefa é cadastrar novos professores da Instituição onde teremos uma tela de login, um formulário de cadastro, um *servlet* para intercambio de dados com o *bean* sistemaAcademico (fig. 20). Veja também diagramas de colaboração (fig. 21) e atividades (fig. 22).

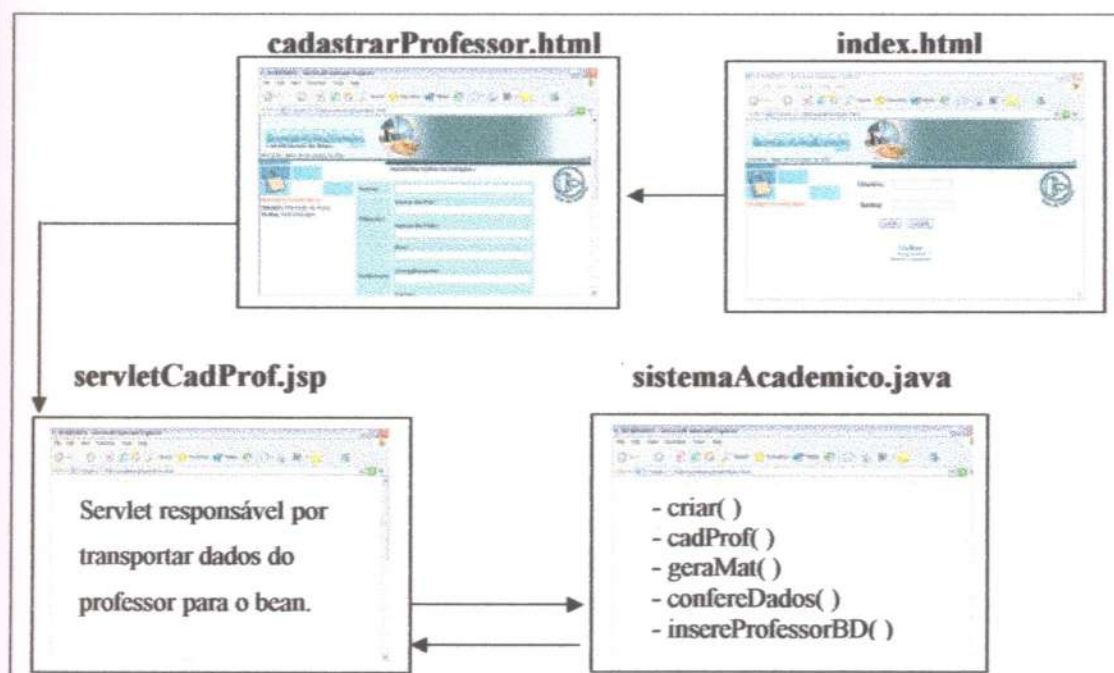


FIGURA 20 - Sequência de operações para o cadastramento de professor.

O usuário do sistema, administrador ou secretária, acessará via browser a tela inicial do sistema (index.html). Em seguida, escolherá a opção "cadastrar professor", onde carregará um formulário (cadastrarProfessor.html) com campos como nome, endereço, filiação, titulação, entre outros. Preenchendo-os, o passo seguinte é submeter esses dados com um clique no botão "enviar".

O *servlet* (servletCadProf.jsp) irá receber esses dados e organizá-lo em um objeto "Professor". O *bean* (sistemaAcademico.java) se encarregará de inserir no banco de dados e retornar ao *servlet* uma resposta de erro ou sucesso.

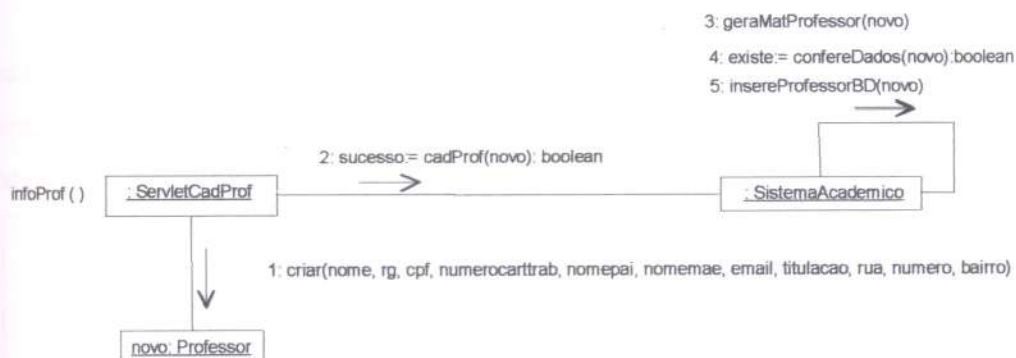


FIGURA 21 - Diagrama de colaboração cadastramento de professor.

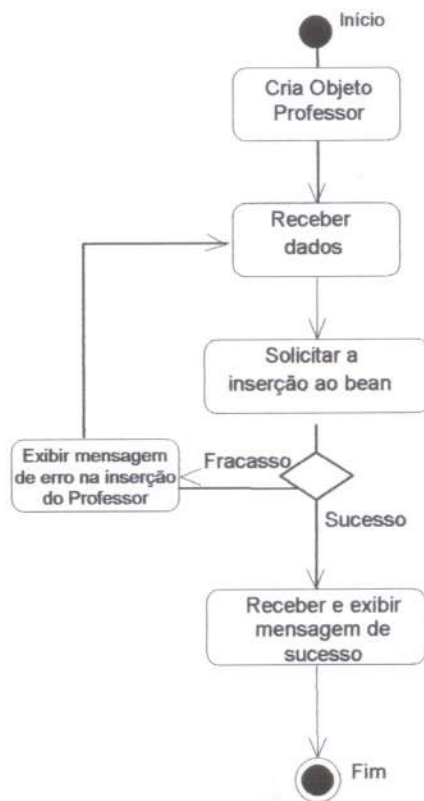


FIGURA 22 - Diagrama de atividades do servletCadProf.

Alterar Professor:

Alterar dados de professores também está previsto para o sistema. Logo teremos uma tela de logon, um formulário de busca, outro de recadastro e um *servlet* para intercâmbio de dados com o *bean* sistemaAcademico (fig. 23). Veja também diagramas de colaboração (fig. 24) e atividades (fig. 25).

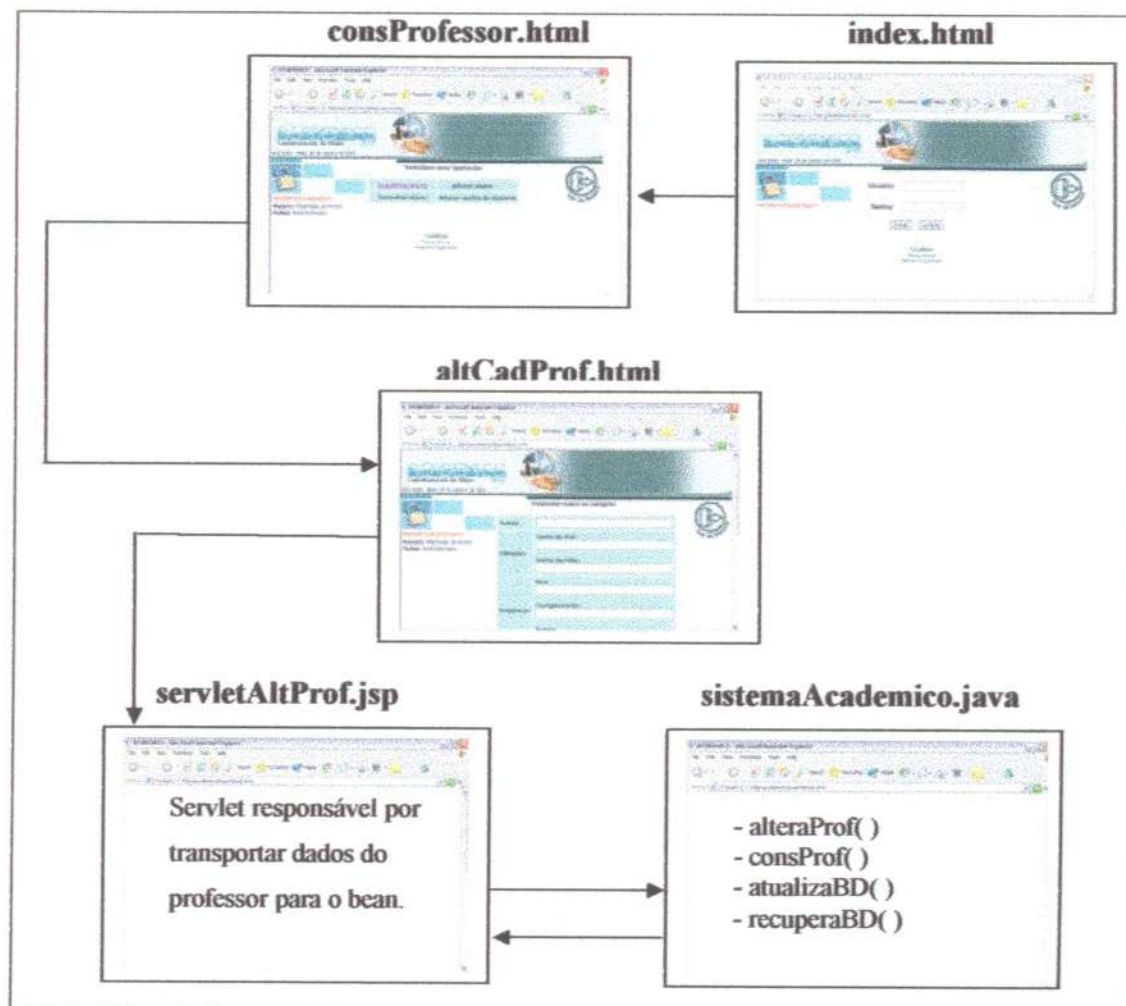


FIGURA 23 - Sequência de operações para a alteração do cadastro de professor.

O usuário do sistema, administrador ou secretária, acessará via browser a tela inicial do sistema (index.html). Em seguida, selecionando a opção "Gerenciar Professor" surgirá uma tela de consulta onde poderá digitar o número do CPF do professor. Clicando na opção "alterar professor", carregará um formulário (altCadProf.html) com campos como nome, endereço, filiação, entre outros. Preenchendo-os, o passo seguinte é submeter esses dados com um clique no botão "enviar".

O servlet (`servletAltProf.jsp`) irá receber esses dados e organizá-lo em um objeto "Professor". O bean (`sistemaAcademico.java`) se encarregará de inserir no banco de dados e retornar ao servlet uma resposta de erro ou sucesso.

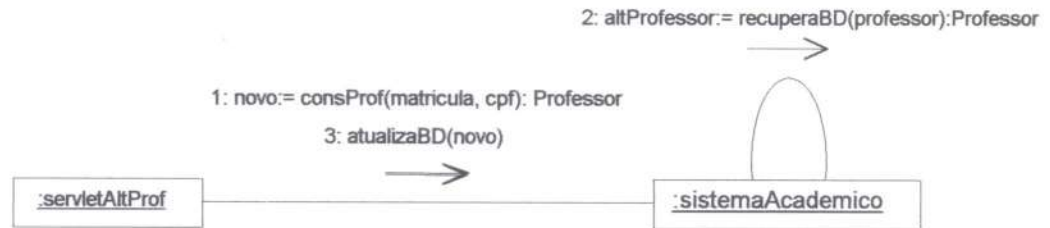


FIGURA 24 - Diagrama de colaboração alterar professor.



FIGURA 25 - Diagrama de atividades do servletAltProf.

Consultar Professor:

Essa tarefa tem a função de realizar uma consulta de dados de professores da Instituição onde teremos uma tela inicial de login, formulário de busca, um *servlet* para intercambio de dados com o *bean* sistemaAcademico (fig. 26). Veja também diagramas de colaboração (fig. 27) e atividades (fig. 28).

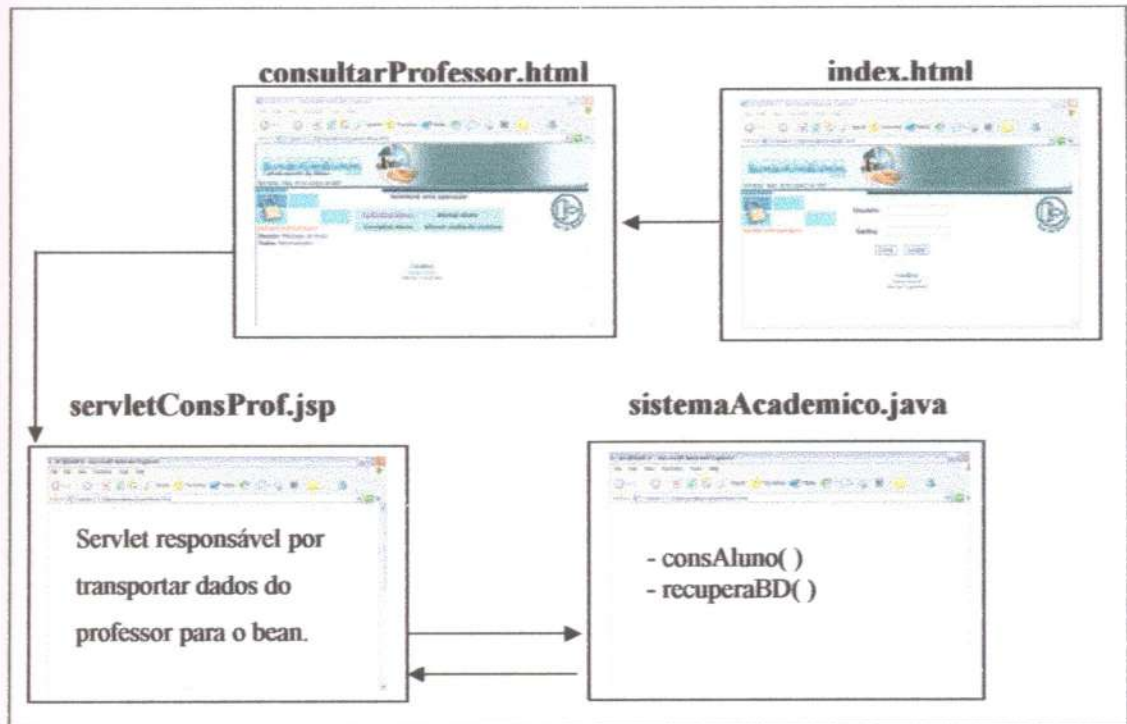


FIGURA 26 - Seqüência de operações para a consulta do cadastro de professores.

O usuário do sistema, administrador ou secretária, acessará via *browser* a tela inicial do sistema (**index.html**). Em seguida, surgirá uma tela de consulta (**consultarProfessor.html**) onde poderá escolher as opções de busca por CPF. Ao preencher o critério de busca, o usuário clicará no botão "buscar".

O *servlet* (**servletConsProf.jsp**) irá receber esses dados do formulário e organizá-lo. Em seguida, acionará o *bean* (**sistemaAcademico.java**) que se encarregará de buscar no banco de dados e retornar ao *servlet* uma resposta. O *servlet* exibirá o registro do professor ou caso não esteja cadastrado, uma mensagem se referindo a esse fato.

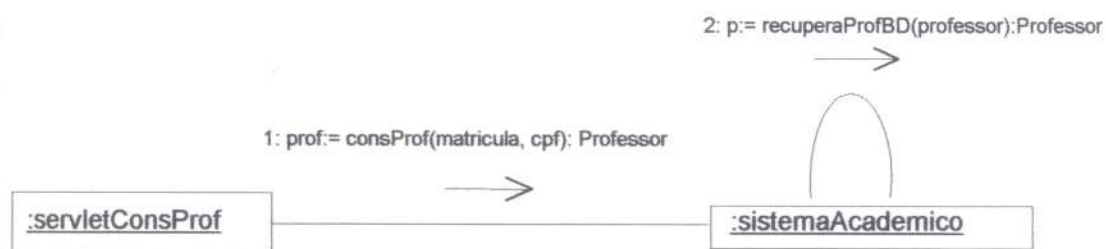


FIGURA 27 - Diagrama de colaboração consultar professor.



FIGURA 28 - Diagrama de atividades do servletConsProf.

Cadastrar Notas:

A função dessa tarefa é cadastrar notas de alunos, onde teremos uma tela de login, uma página de escolha de disciplinas e um *servlet* para intercambio de dados com o *bean* sistemaAcademico (fig. 29). Veja também diagramas de colaboração (fig. 30) e atividades (fig. 31).

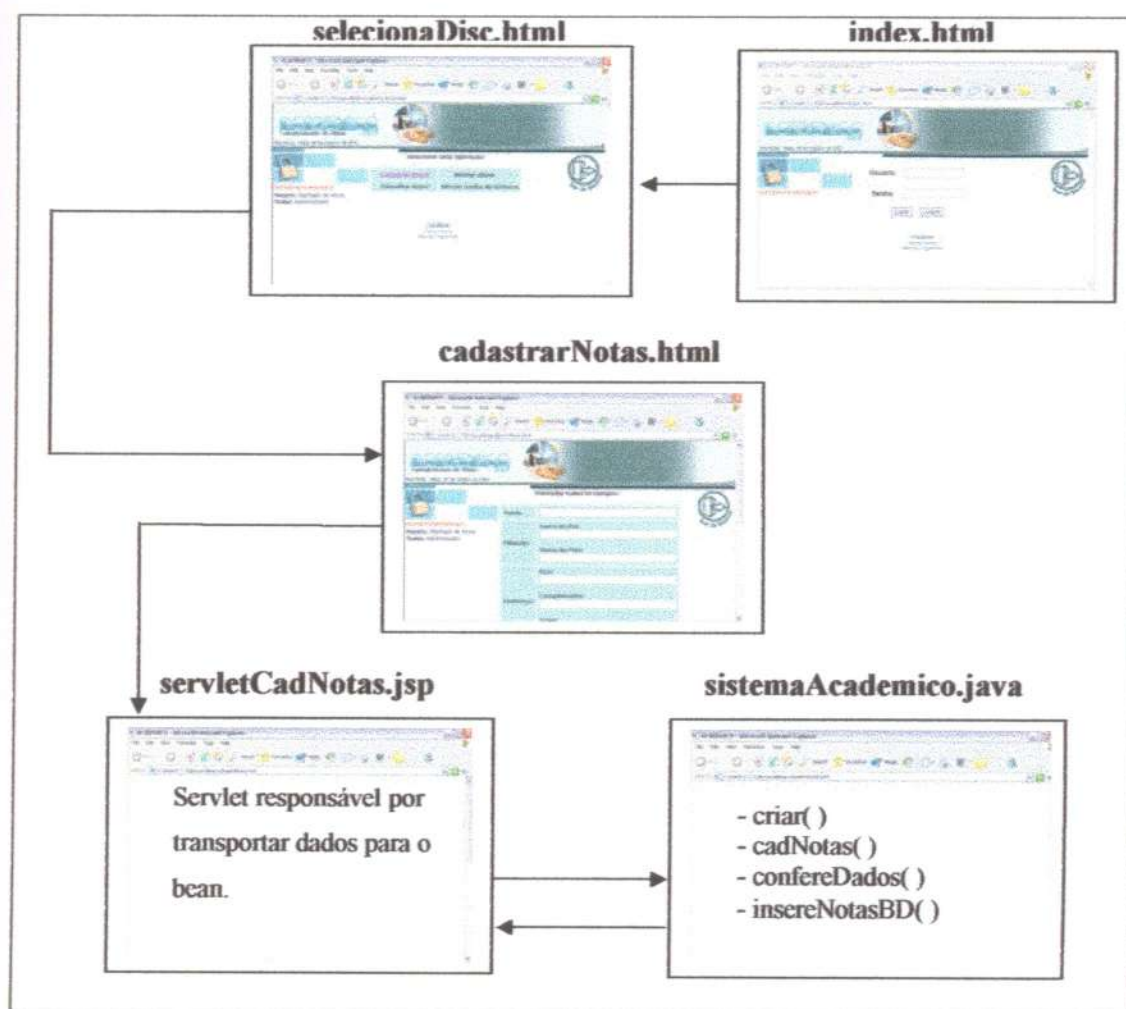


FIGURA 29 - Sequência de operações para o cadastramento de notas.

O professor acessará via browser a tela inicial do sistema (**index.html**). Em seguida, escolherá a opção "cadastrar notas", onde carregará uma página, onde o professor seleciona a disciplina em (**selecionaDisciplina.html**). Carregará um formulário semelhante a um diário de classe, onde ele poderá digitar as notas e faltas dos alunos.

O servlet (servletCadNotas.jsp) irá receber esses dados e organizá-lo em um objeto "novo". O bean (sistemaAcademico.java) se encarregará de inserir no banco de dados e retornar ao servlet uma resposta de erro ou sucesso.

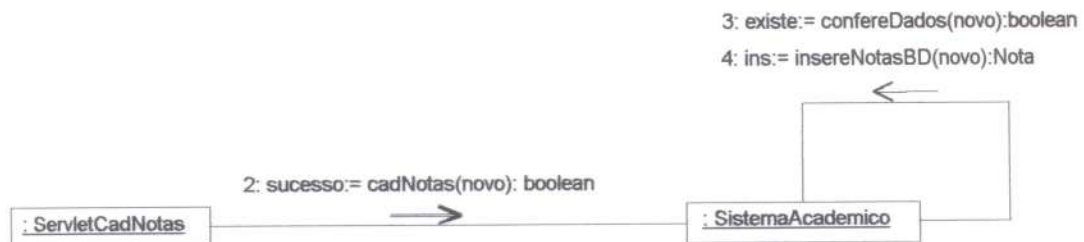


FIGURA 30 - Diagrama de colaboração cadastramento de notas.



FIGURA 31 - Diagrama de atividades do servletCadNotas.

Alterar Notas:

Alterar notas também está previsto para o sistema. Logo teremos uma tela de *logon*, um formulário de busca, outro de recadastro, um *servlet* para intercambio de dados com o *bean* sistemaAcademico (fig. 32). Veja também diagramas de colaboração (fig. 33) e atividades (fig. 34).

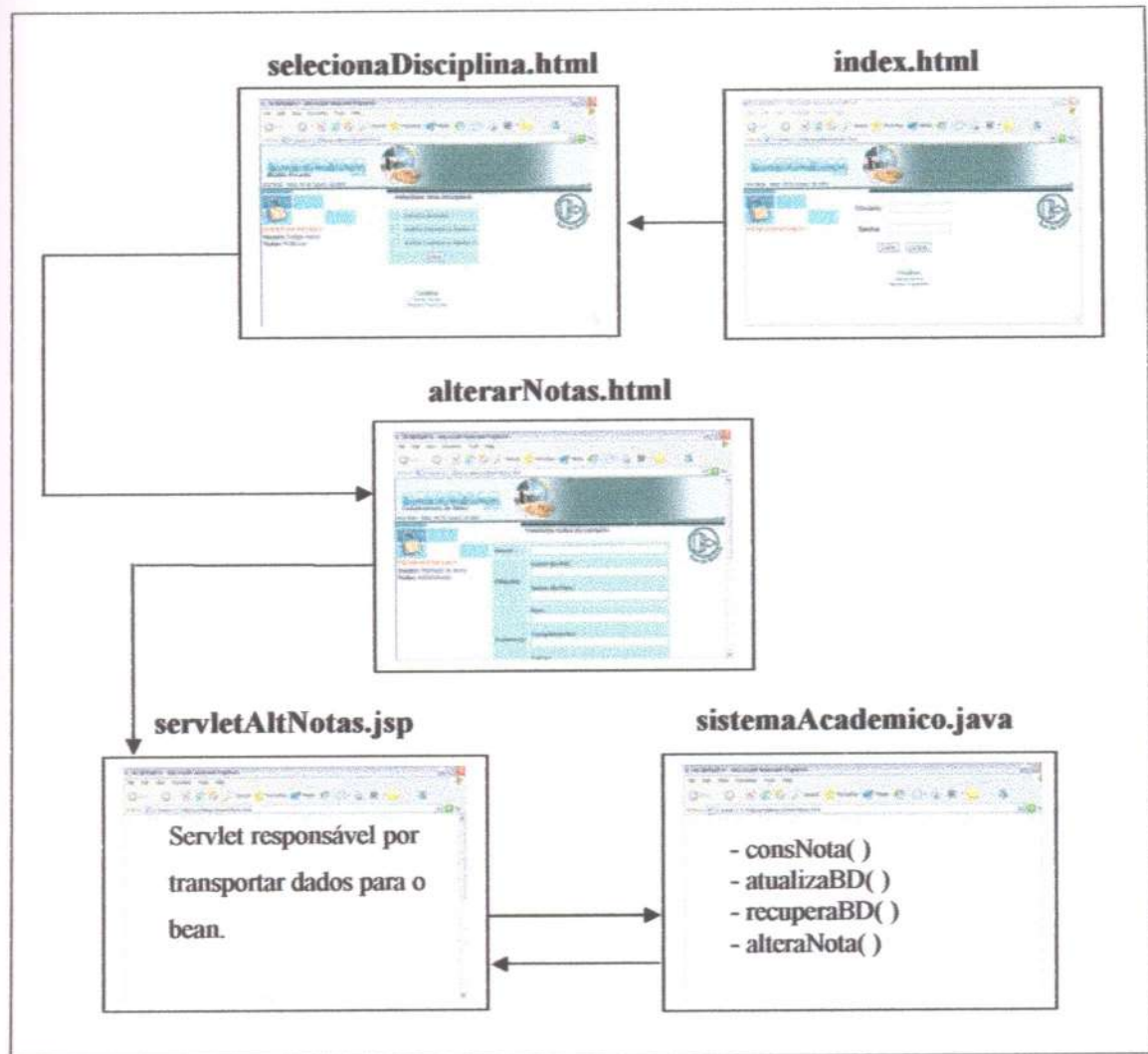


FIGURA 32 - Sequência de operações para a alteração de notas.

O professor acessará, via *browser*, a tela inicial do sistema (**index.html**). Em seguida, escolherá a opção "Alterar Notas", que carregará uma página, onde o professor seleciona a disciplina em (**selecionaDisciplina.html**). Um formulário semelhante a um diário de classe surgirá, onde ele poderá alterar as notas e faltas dos alunos.

O *servlet* (servletAltNotas.jsp) irá receber esses dados e organizá-lo em um objeto "novo". O *bean* (sistemaAcademico.java) se encarregará de inserir no banco de dados e retornar ao *servlet* uma resposta de erro ou sucesso.

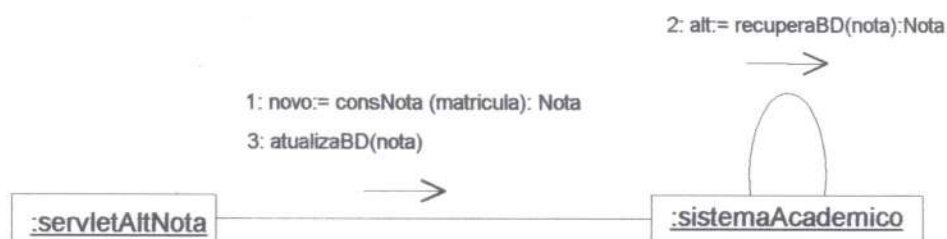


FIGURA 33 - Diagrama de colaboração alteração de notas.

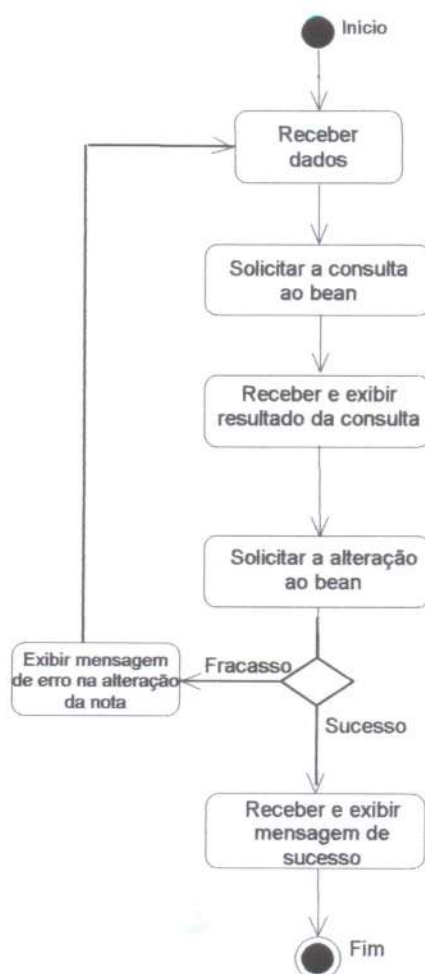


FIGURA 34 - Diagrama de atividades do servletAltNotas.

Consultar Notas:

Essa tarefa tem a função de realizar uma consulta de notas, onde teremos uma tela inicial de *logon*, formulário de busca e um *servlet* para intercâmbio de dados com o *bean* sistemaAcademico (fig. 35). Veja também diagramas de colaboração (fig. 36) e atividades (fig. 37).

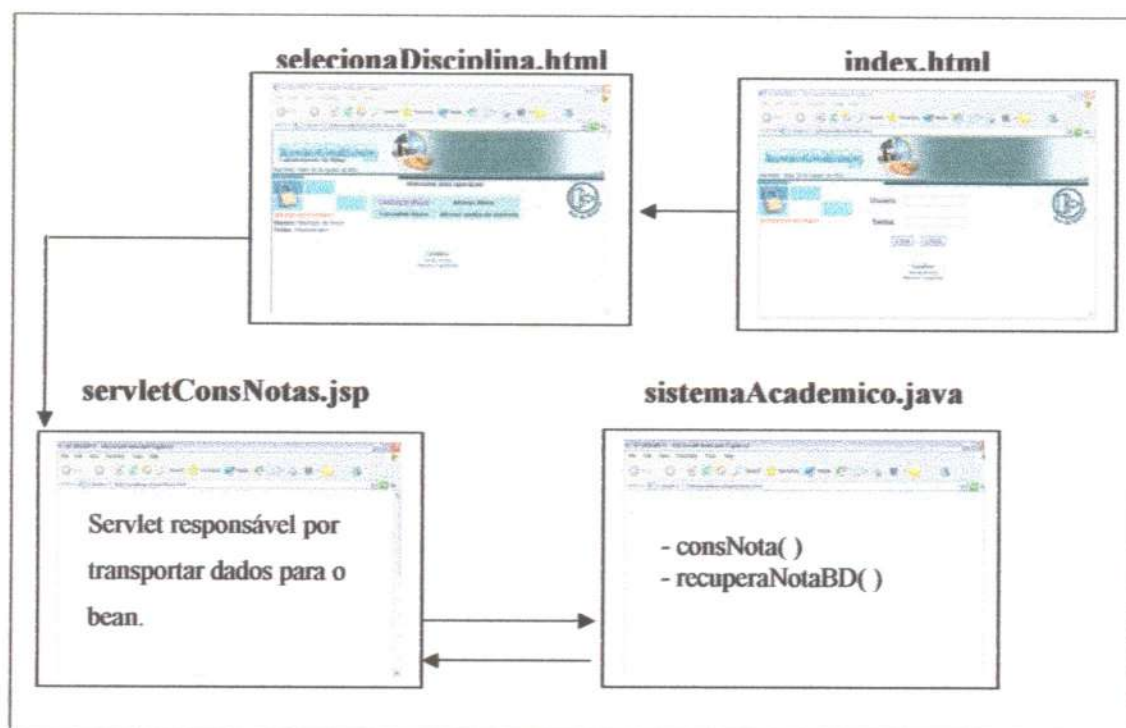


FIGURA 35 - Seqüência de operações para a consulta de notas.

O professor ou administrador do sistema acessará, via *browser*, a tela inicial do sistema (**index.html**). Em seguida, escolherá a opção "Consultar Notas" que carregará uma página onde o professor seleciona a disciplina em (**selecionaDisciplina.html**).

O *servlet* (**servletConsNotas.jsp**) irá receber esses dados e organizá-lo em um objeto "Nota".

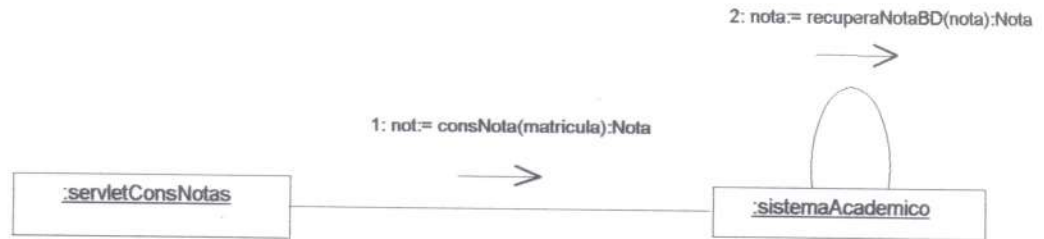


FIGURA 36 - Diagrama de colaboração consulta de notas.

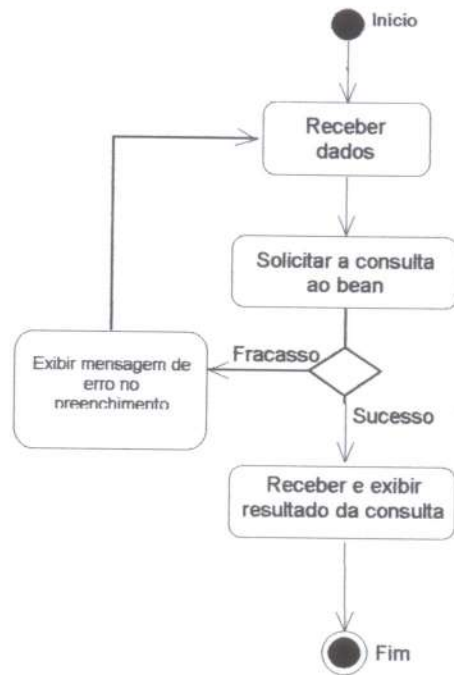


FIGURA 37 - Diagrama de atividades do servletConsNotas.

Cadastrar Funcionário:

A função dessa tarefa é cadastrar novos funcionários da Instituição, onde teremos uma tela de login, um formulário de cadastro e um *servlet* para intercâmbio de dados com o *bean* sistemaAcademico (fig. 38). Veja também diagramas de colaboração (fig. 39) e atividades (fig. 40).

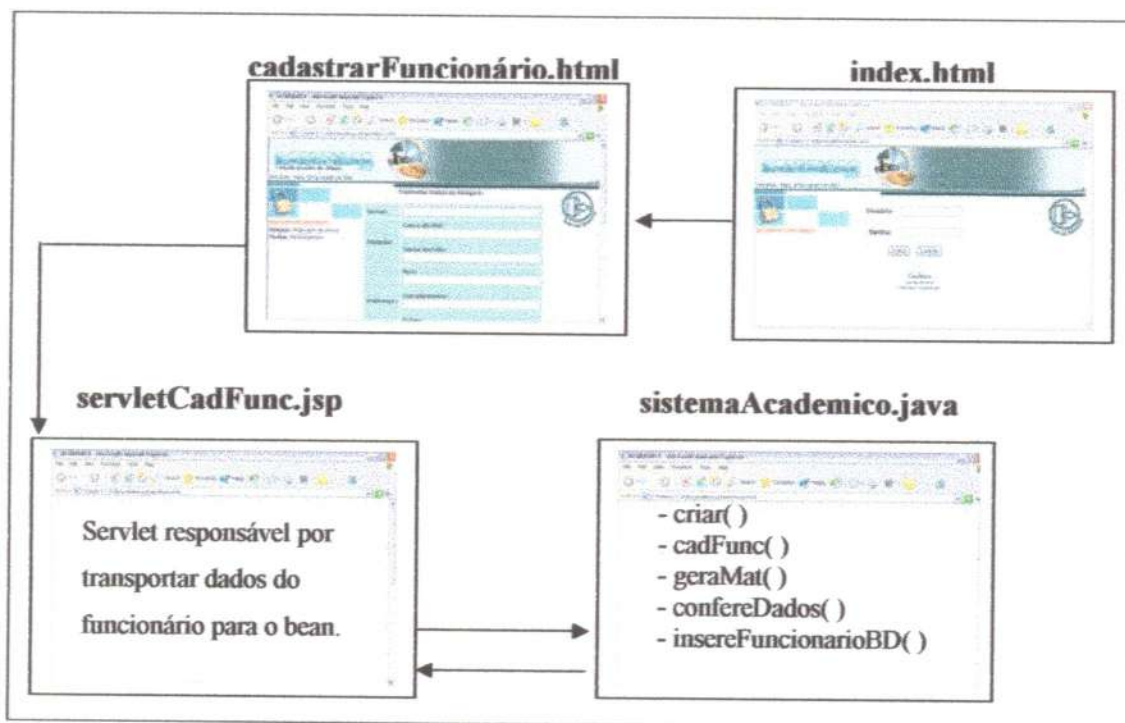


FIGURA 38 - Seqüência de operações para o cadastramento de funcionário.

O administrador do sistema acessará, via *browser*, a tela inicial do sistema (**index.html**). Em seguida, escolherá a opção "cadastrar funcionário", onde carregará um formulário (**cadastrarFuncionario.html**) com campos como nome, nº carteira de trabalho, CPF, RG, filiação, endereço, e-mail, titulação, função, entre outros. Preenchendo-os, o passo seguinte é submeter esses dados com um clique no botão "enviar".

O *servlet* (**servletCadFunc.jsp**) irá receber esses dados e organizá-lo em um objeto "Professor". O *bean* (**sistemaAcademico.java**) se encarregará de inserir no banco de dados e retornar ao *servlet* uma resposta de erro ou sucesso.

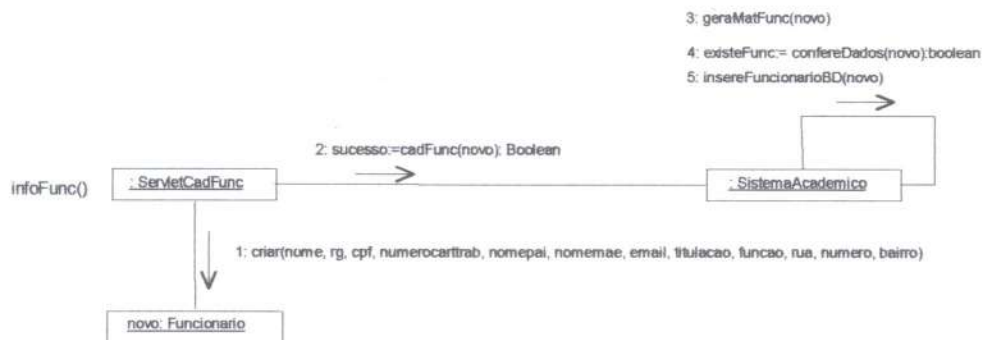


FIGURA 39 - Diagrama de colaboração cadastramento de funcionário.



FIGURA 40 - Diagrama de atividades do servletCadFunc.

Alterar Funcionário:

Alterar dados de funcionário também está previsto para o sistema. Logo teremos uma tela de *logon*, um formulário de busca, outro de recadastro e um *servlet* para intercâmbio de dados com o *bean* sistemaAcademico (fig. 41). Veja também diagramas de colaboração (fig. 42) e atividades (fig. 43).

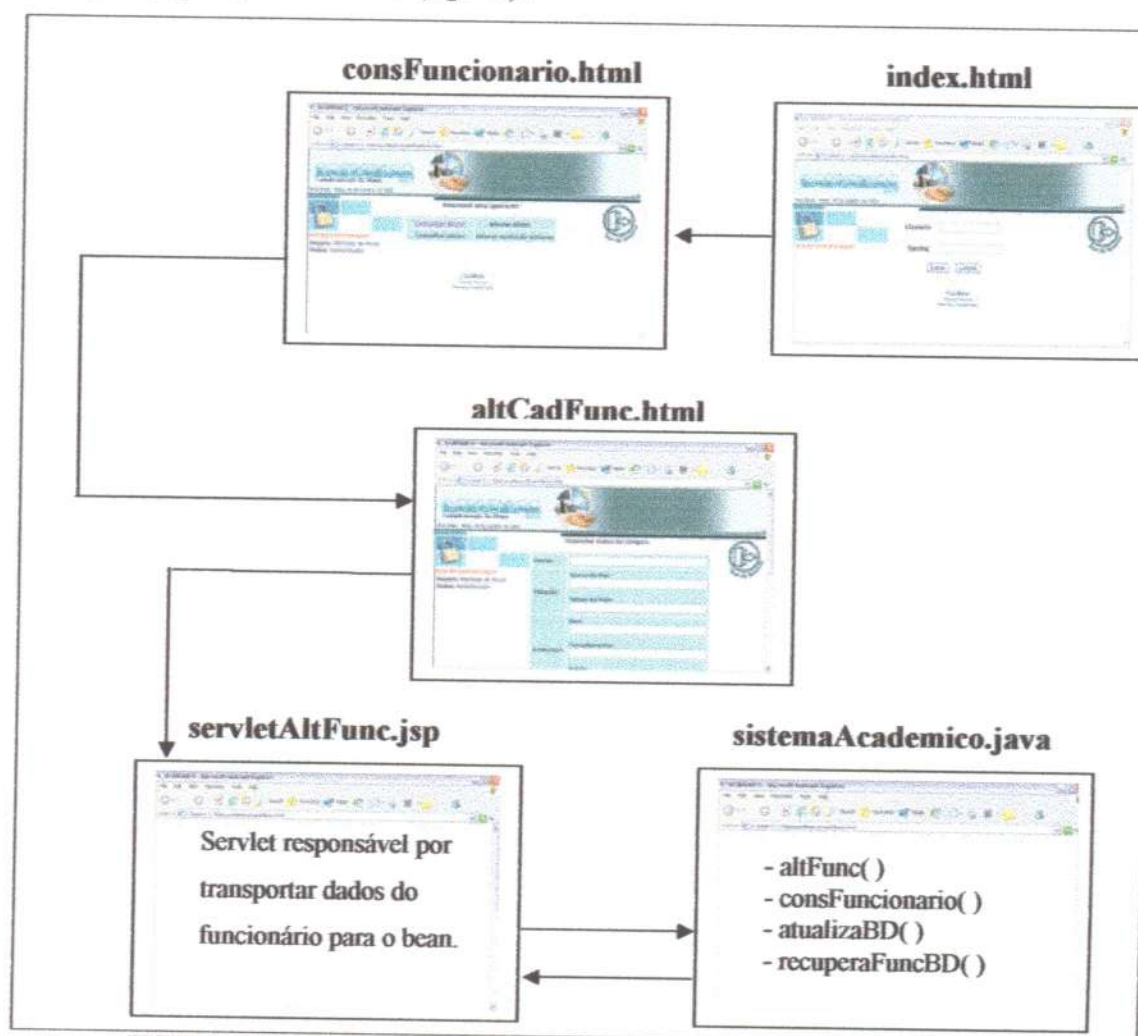


FIGURA 41 - Sequência de operações para a alteração do cadastro de funcionário.

O usuário do sistema, administrador ou secretária, acessará, via *browser*, a tela inicial do sistema (*index.html*). Em seguida, selecionando a opção "Gerenciar Funcionário" surgirá uma tela de consulta onde poderá digitar número do CPF do funcionário. Clicando na opção "alterar funcionário", carregará um formulário (*altCadFunc.html*) com campos como nome, endereço, filiação, entre outros. Preenchendo-os, o passo seguinte é submeter esses dados com um clique no botão "enviar".

O servlet (servletAltFunc.jsp) irá receber esses dados e organizá-lo em um objeto "Funcionario". O bean (sistemaAcademico.java) se encarregará de inserir no banco de dados e retornar ao servlet uma resposta de erro ou sucesso.

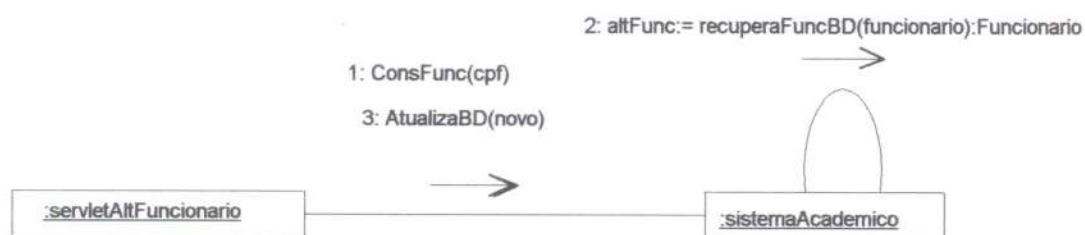


FIGURA 42 - Diagrama de colaboração alterar funcionário.

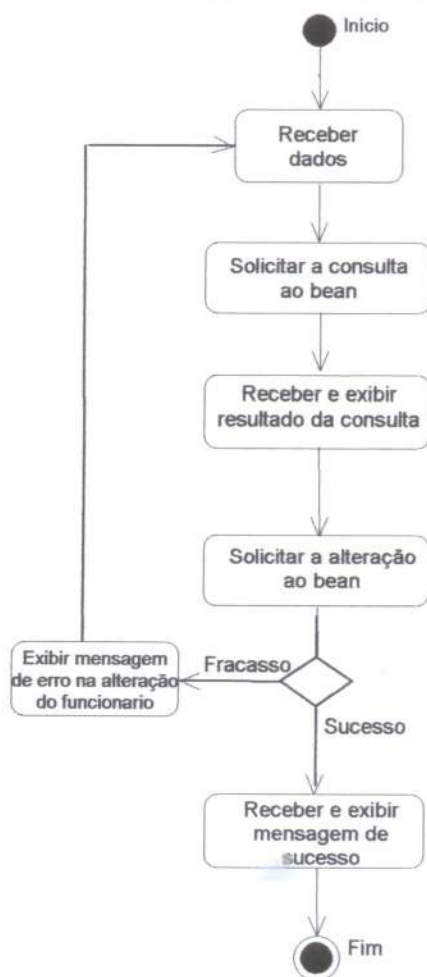


FIGURA 43 - Diagrama de atividades do servletAltFunc.

Excluir Funcionário:

A função dessa tarefa é excluir um ou mais funcionários da Instituição, onde teremos uma tela de *login*, um formulário, um *servlet* para intercâmbio de dados com o *bean* sistemaAcademico (fig. 44). Veja também diagramas de colaboração (fig. 45) e atividades (fig. 46).

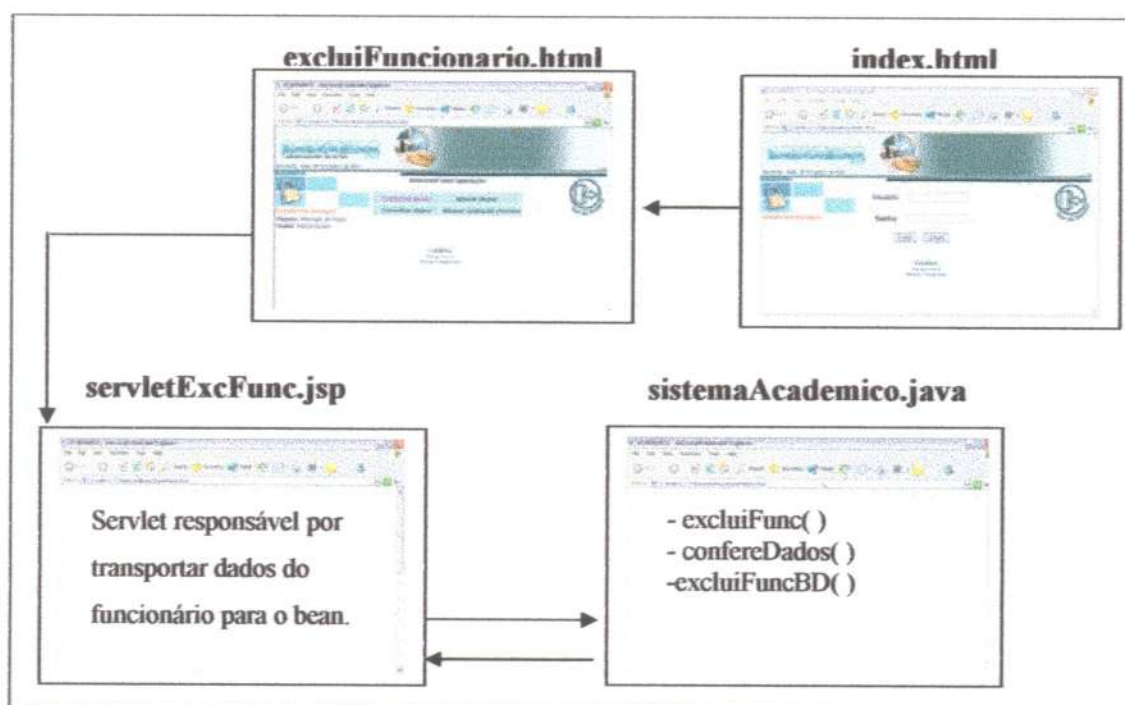


FIGURA 44 - Seqüência de operações para a exclusão de funcionário.

O usuário do sistema, administrador ou secretária, acessará, via *browser*, a tela inicial do sistema (`index.html`). Em seguida, selecionando a opção "Gerenciar Funcionário" surgirá opções. Seleciona-se "Excluir Funcionário". Uma tela de consulta (`excluiFuncionario.html`) aparecerá onde poderá digitar número do CPF do funcionário. Basta clicar no botão "excluir".

O *bean* (`sistemaAcademico.java`) se encarregará de excluir no banco de dados e retornar ao *servlet* uma resposta de erro ou sucesso.

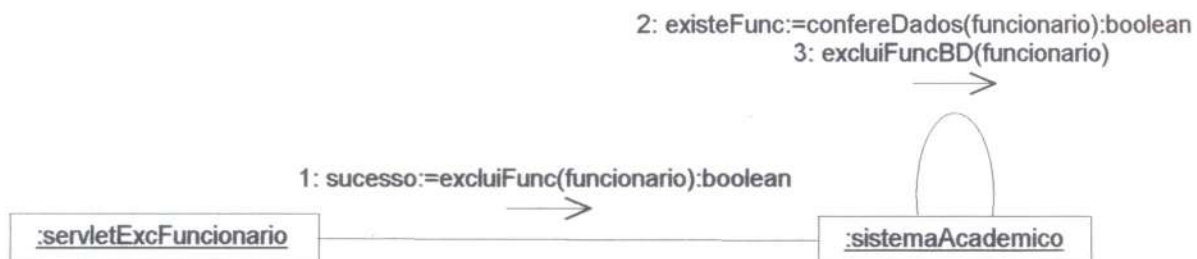


FIGURA 45 - Diagrama de colaboração excluir funcionário.

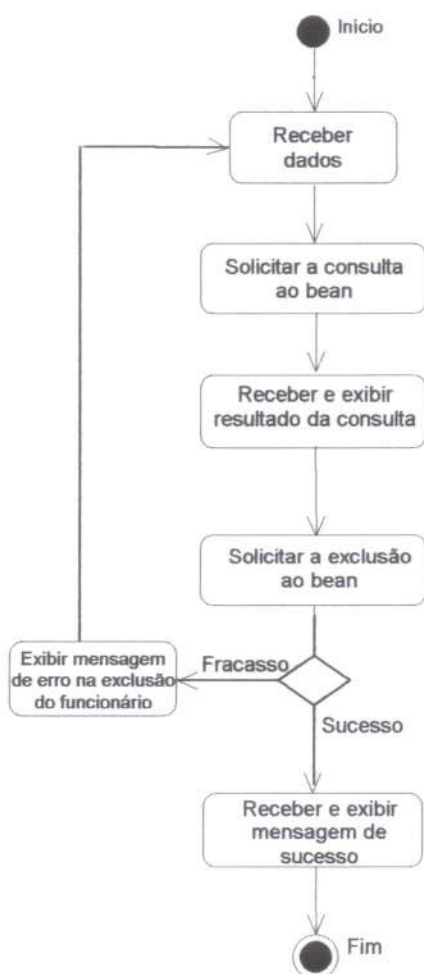


FIGURA 46 - Diagrama de atividades do servletExcFunc.

Consultar Funcionário:

Essa tarefa tem a função de realizar uma consulta de dados de funcionário da Instituição, onde teremos uma tela inicial de login, formulário de busca e um *servlet* para intercâmbio de dados com o *bean* sistemaAcademico (fig. 47). Veja também diagramas de colaboração (fig. 48) e atividades (fig. 49).

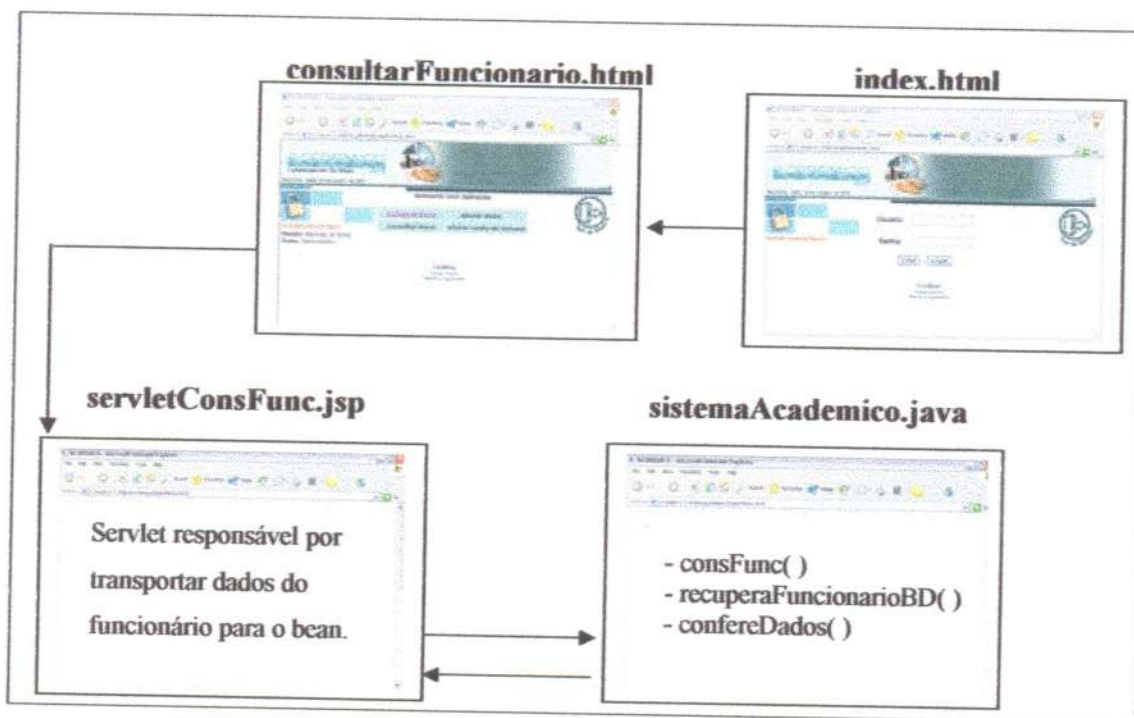


FIGURA 47 - Sequência de operações para a consulta do cadastro de funcionários.

O usuário do sistema, administrador ou secretária, acessará via browser a tela inicial do sistema (**index.html**). Em seguida, surgirá uma tela de consulta (**consultarFuncionario.html**) onde poderá escolher as opções de busca por CPF. Ao preencher o critério de busca, o usuário clicará no botão "buscar".

O *servlet* (**servletConsFunc.jsp**) irá receber esses dados do formulário e organizá-lo. Em seguida, acionará o *bean* (**sistemaAcademico.java**) que se encarregará de buscar no banco de dados e retornar ao *servlet* uma resposta. O *servlet* exibirá o registro do funcionário ou caso não esteja cadastrado, uma mensagem se referindo a esse fato.

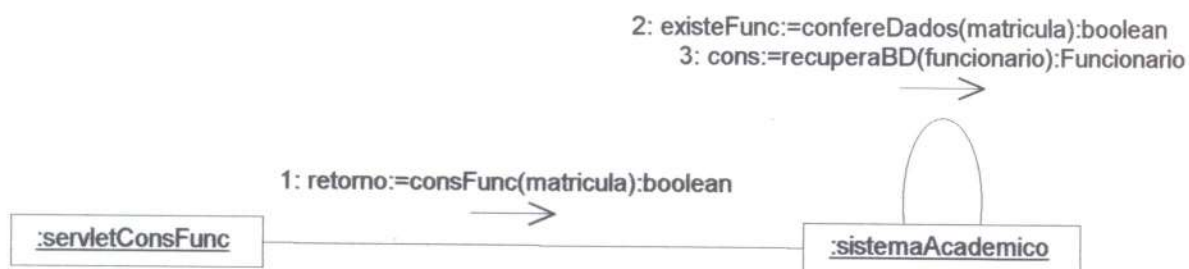


FIGURA 48 - Diagrama de colaboração consultar funcionário.

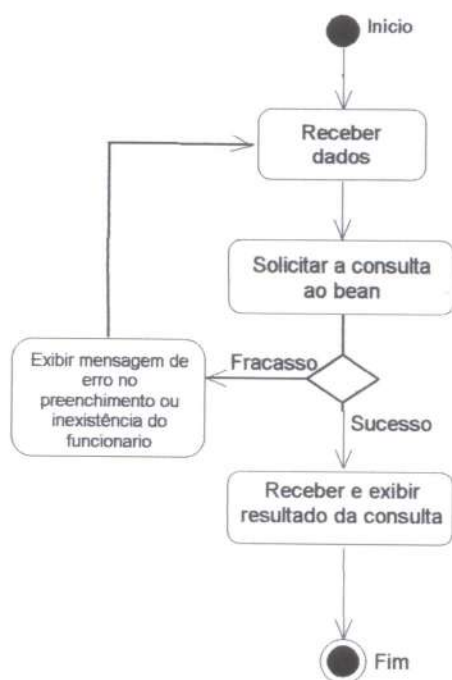


FIGURA 49 - Diagrama de atividades do servletConsFunc.

Alterar Professor em Disciplina:

Alterar dados de professores em disciplina, também está previsto para o sistema. Logo teremos uma tela de *logon*, um formulário de busca, outro de recadastro e um *servlet* para intercâmbio de dados com o *bean* sistemaAcademico (fig. 50). Veja também diagramas de colaboração (fig. 51) e atividades (fig. 52).

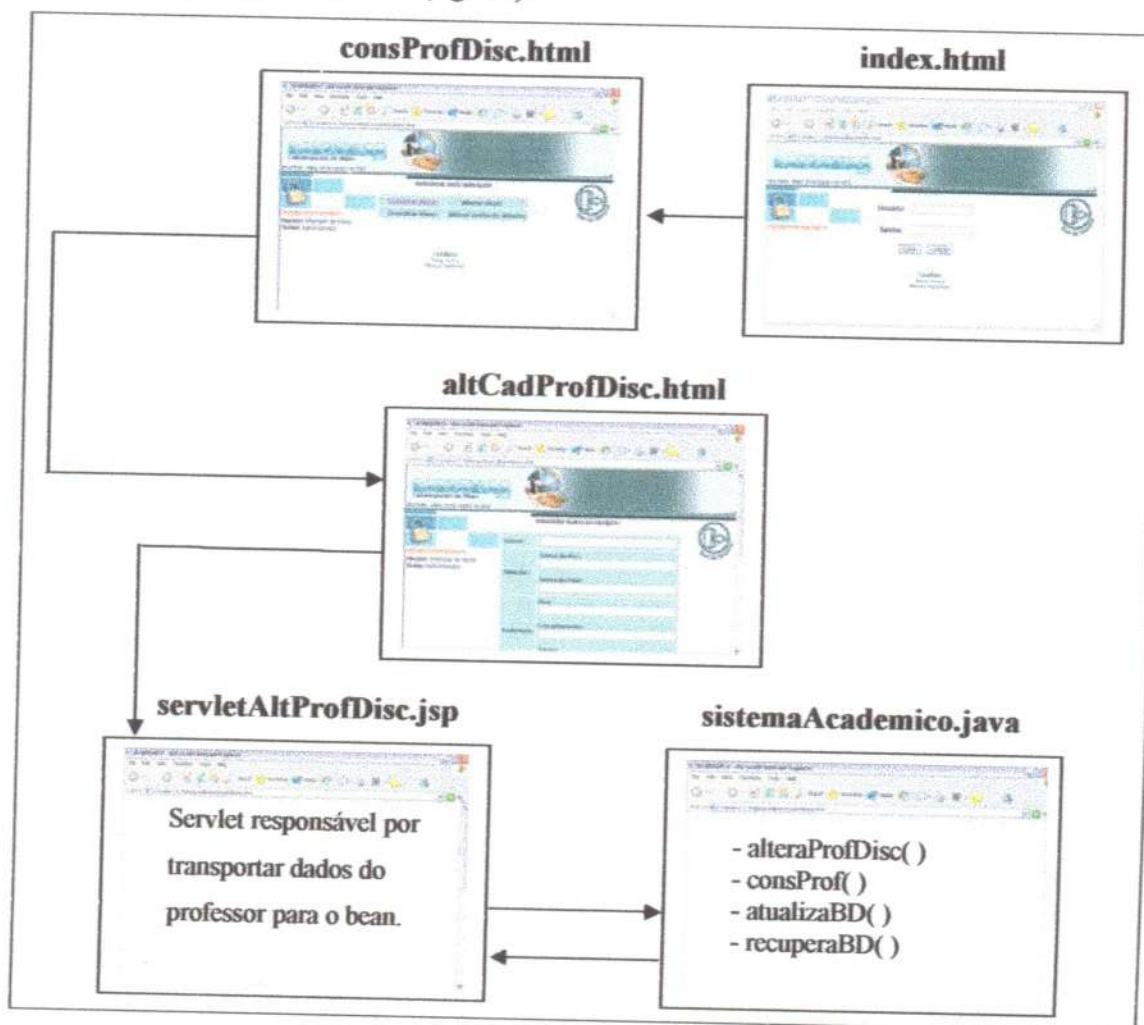


FIGURA 50 - Sequência de operações para a alteração do cadastro de professor em disciplina.

O usuário do sistema, administrador ou secretária, acessará via browser a tela inicial do sistema (*index.html*). Em seguida, selecionando a opção "Gerenciar Disciplina" surgirá uma tela de consulta (*consProfDisc.html*) onde poderá digitar número do CPF do professor. Clicando na opção "alterar professor em disciplina", carregará um formulário (*altCadProfDisc.html*) com campos como nome, endereço, filiação, entre outros. Preenchendo-os o passo seguinte é submeter esses dados com um clique no botão "enviar".

O servlet (servletAltProf.jsp) irá receber esses dados e organizá-lo em um objeto "Professor". O bean (sistemaAcademico.java) se encarregará de inserir no banco de dados e retornar ao servlet uma resposta de erro ou sucesso.

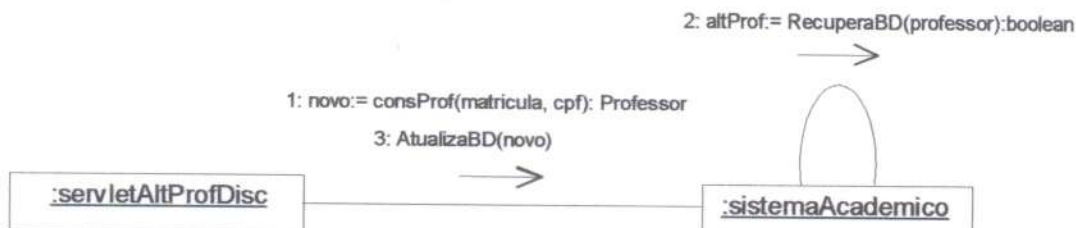


FIGURA 51 - Diagrama de colaboração alterar professor em disciplina.

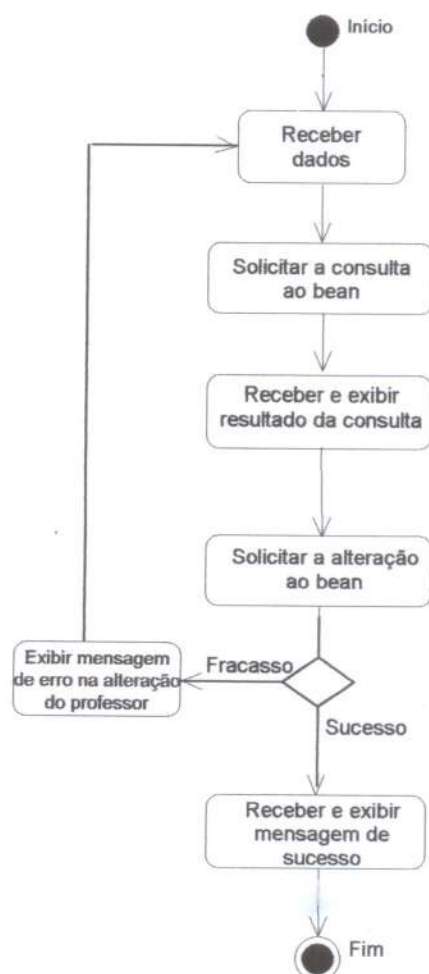


FIGURA 52 - Diagrama de atividades do servletAltProfDisc.

Cadastrar Disciplina:

A função dessa tarefa é cadastrar disciplinas correspondentes aos cursos da Instituição, onde teremos uma tela de *logon*, um formulário de cadastro e um *servlet* para intercâmbio de dados com o *bean* sistemaAcademico (fig. 53). Veja também diagramas de colaboração (fig. 54) e atividades (fig. 55).

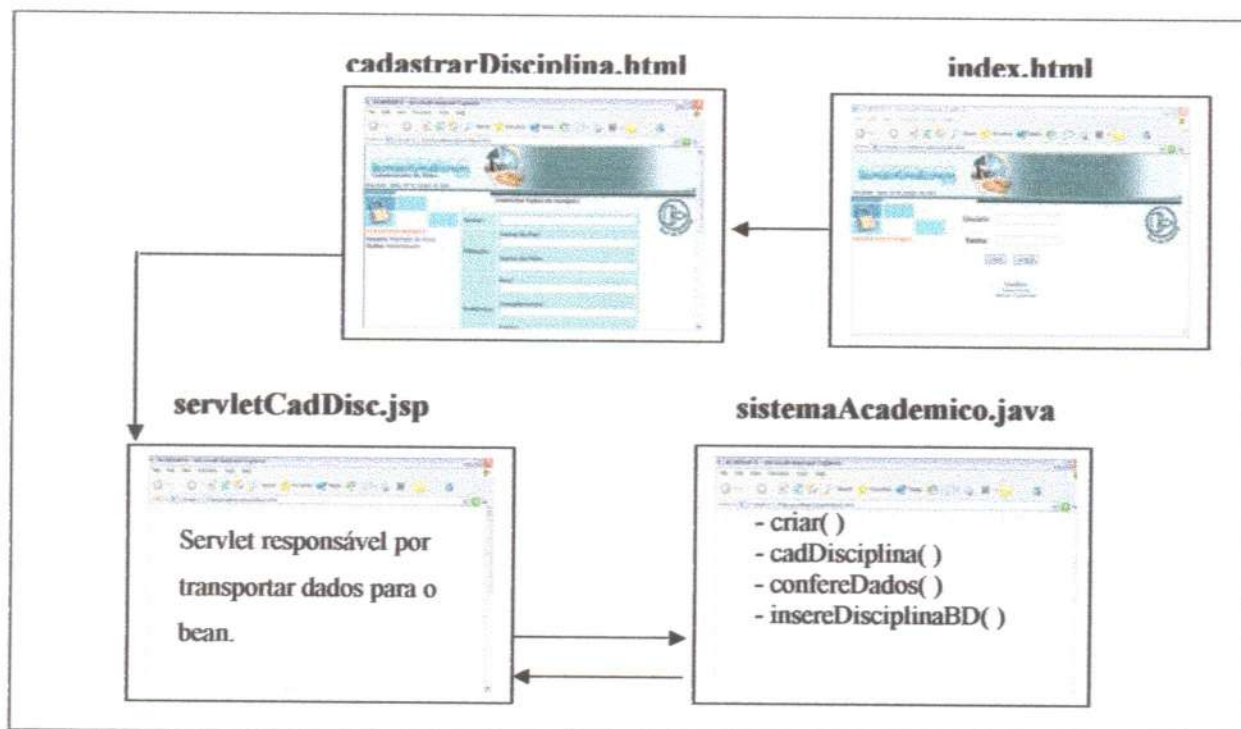


FIGURA 53 - Sequência de operações para o cadastramento de disciplina.

O usuário do sistema acessará, via *browser*, a tela inicial do sistema (*index.html*). Em seguida, escolherá a opção "cadastrar disciplina", onde carregará um formulário (*cadastrarDisciplina.html*) com campos como nome, carga horária, entre outros. Preenchendo-os, o passo seguinte é submeter esses dados com um clique no botão "enviar".

O servlet (*servletCadDisc.jsp*) irá receber esses dados e organizá-lo em um objeto "Disciplina". O bean (*sistemaAcademico.java*) se encarregará de inserir no banco de dados e retornar ao servlet uma resposta de erro ou sucesso.

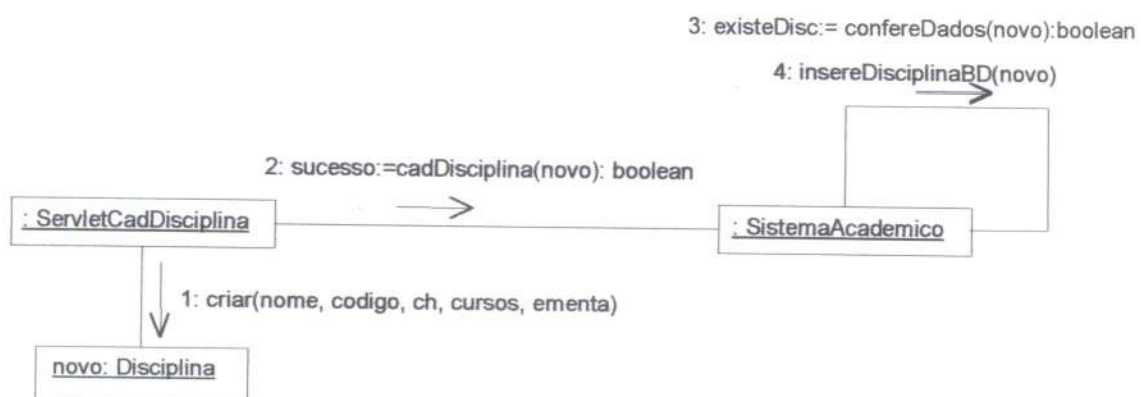


FIGURA 54 - Diagrama de colaboração cadastramento de disciplina.

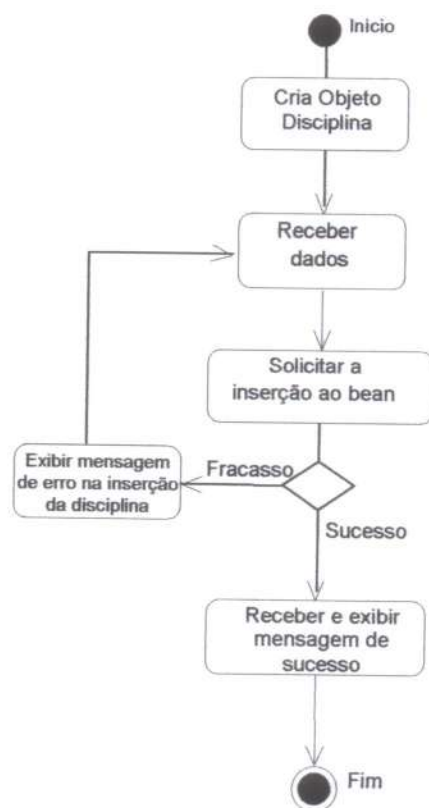


FIGURA 55 - Diagrama de atividades do servletCadDisc.

Alterar Disciplina:

Alterar dados de disciplinas também está previsto para o sistema. Logo, teremos uma tela de *logon*, um formulário de busca, outro de recadastro e um *servlet* para intercâmbio de dados com o *bean* sistemaAcademico (fig. 56). Veja também diagramas de colaboração (fig. 57) e atividades (fig. 58).

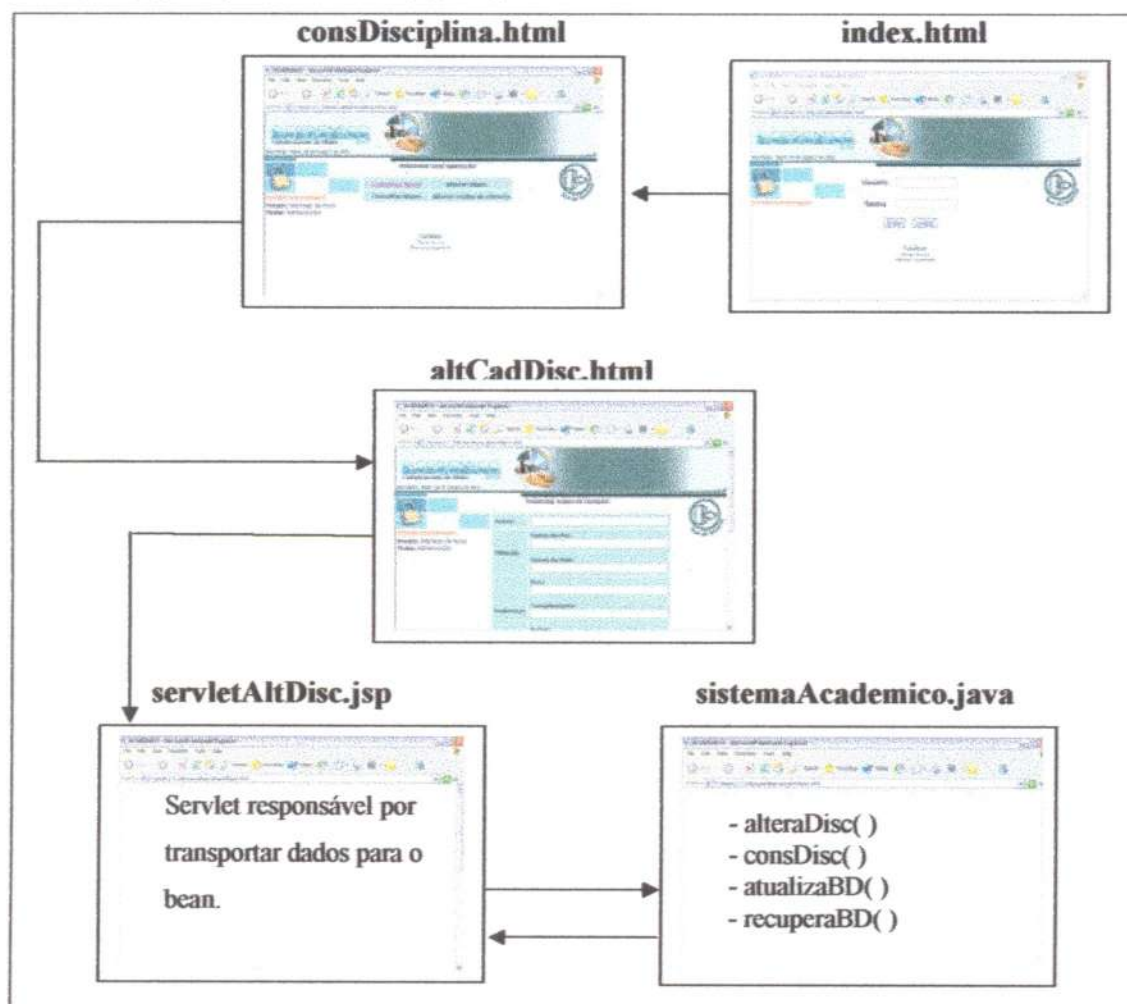


FIGURA 56 - Sequência de operações para a alteração do cadastro de disciplinas.

O usuário do sistema, administrador ou secretária, acessará, via *browser*, a tela inicial do sistema (**index.html**). Em seguida, selecionando a opção "Gerenciar Disciplina". Clicando na opção "alterar disciplina", carregará um formulário (**altCadDisc.html**) com campos pertencentes a uma determinada disciplina. Preenchendo-os, o passo seguinte é submeter esses dados com um clique no botão "enviar".

O servlet (`servletAltDisc.jsp`) irá receber esses dados e organizá-lo em um objeto "Disciplina". O *bean* (`sistemaAcademico.java`) se encarregará de alterar no banco de dados e retornar ao *servlet* uma resposta de erro ou sucesso.

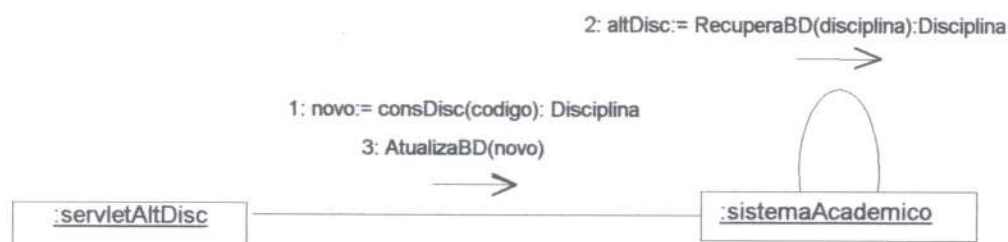


FIGURA 57 - Diagrama de colaboração alterar disciplina.

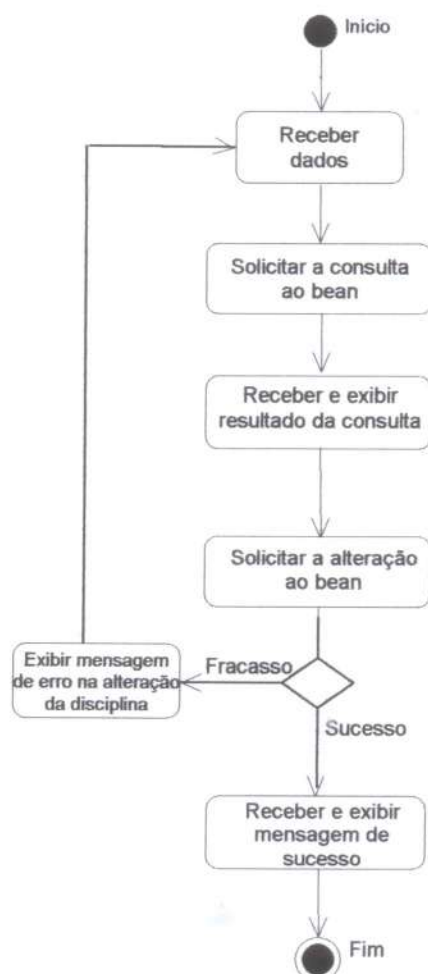


FIGURA 58 - Diagrama de atividades do servletAltDisc.

Excluir Disciplina:

A função dessa tarefa é excluir uma ou mais Disciplinas, onde teremos uma tela de *login*, um formulário e um *servlet* para intercâmbio de dados com o *bean* sistemaAcademico (fig. 60). Veja também diagramas de colaboração (fig. 61) e atividades (fig. 62).

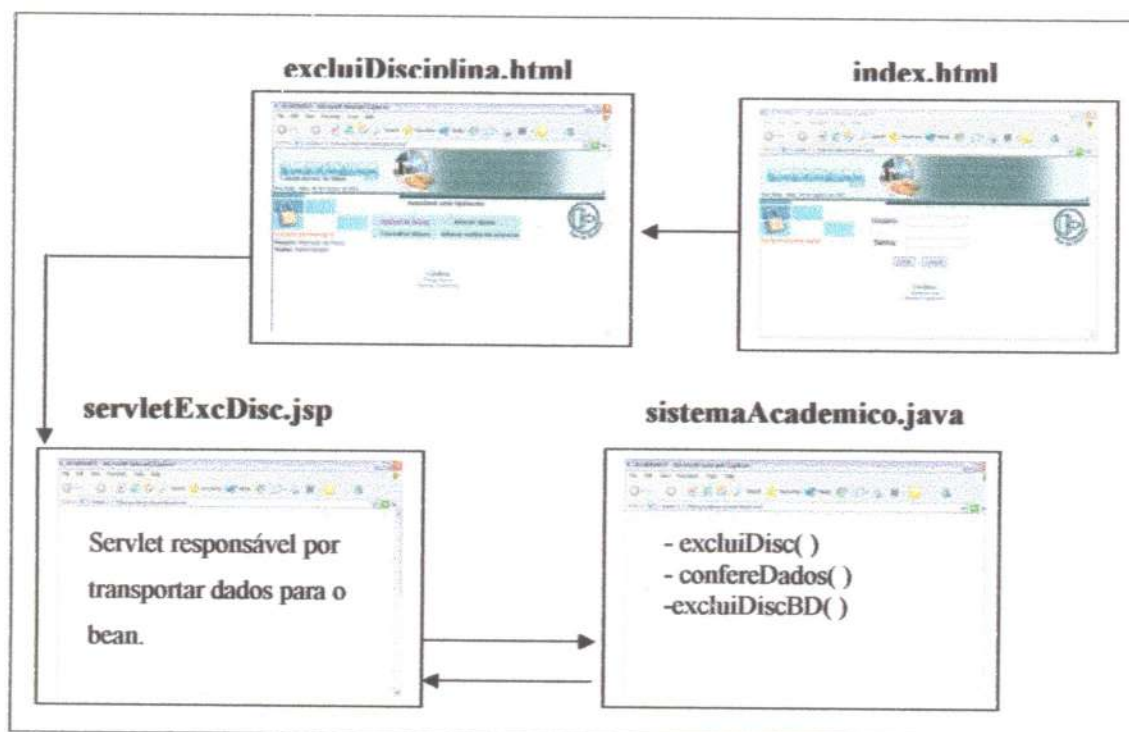


FIGURA 60 - Sequência de operações para a exclusão de disciplina.

O usuário do sistema, administrador ou secretária, acessará, via *browser*, a tela inicial do sistema (**index.html**). Em seguida, selecionando a opção "Gerenciar Disciplina" surgirá opções, seleciona-se "Excluir Disciplina", uma tela de consulta (**excluiDisciplina.html**). Basta clicar no botão "excluir".

O *bean* (**sistemaAcademico.java**) se encarregará de excluir no banco de dados e retornar ao *servlet* uma resposta de erro ou sucesso.

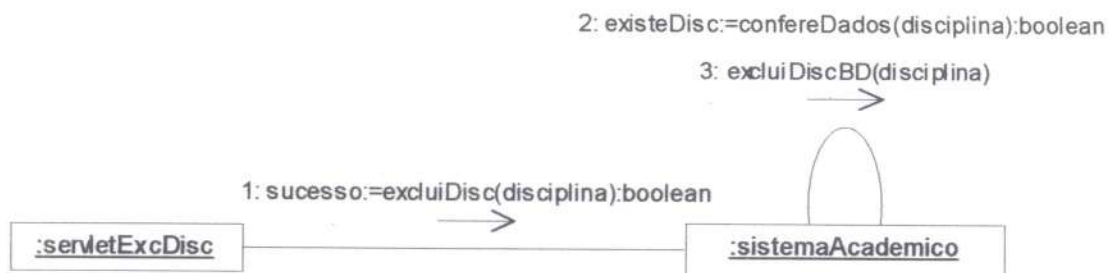


FIGURA 61 - Diagrama de colaboração excluir disciplina.

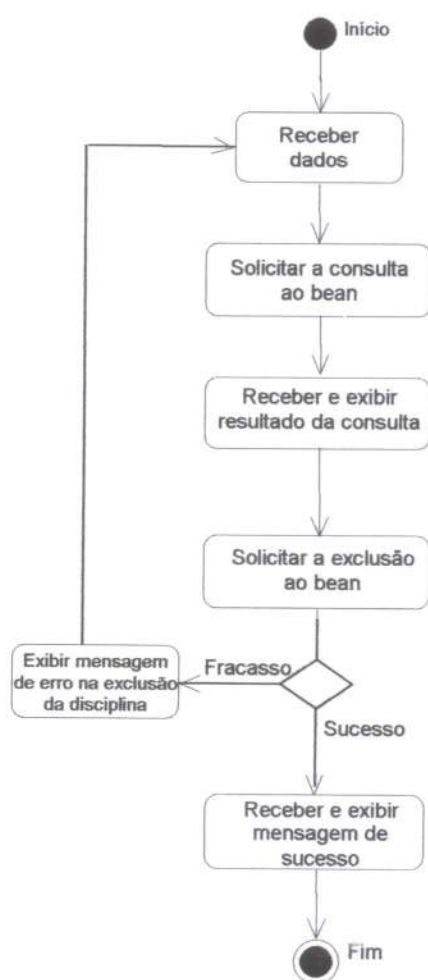


FIGURA 62 - Diagrama de atividades do servletExcDisc.

Consultar Disciplina:

Essa tarefa tem a função de realizar uma consulta de disciplinas da Instituição, onde teremos uma tela inicial de *logon*, formulário de busca e um *servlet* para intercâmbio de dados com o *bean* sistemaAcademico (fig. 63). Veja também diagramas de colaboração (fig. 64) e atividades (fig. 65).

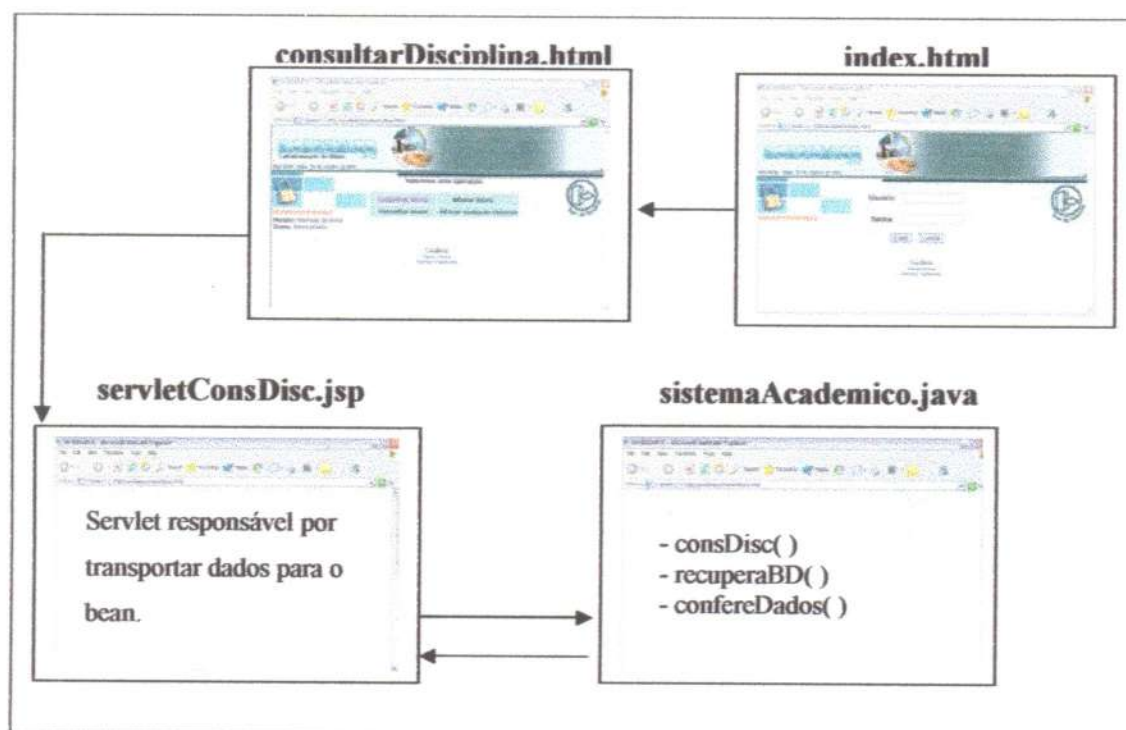


FIGURA 63 - Sequência de operações para a consulta do cadastro de disciplina.

O usuário do sistema, administrador ou secretária, acessará, via *browser*, a tela inicial do sistema (**index.html**). Em seguida, surgirá uma tela de consulta (**consultarDsciplina.html**). Ao preencher o critério de busca, o usuário clicará no botão "buscar".

O *servlet* (**servletConsDisc.jsp**) irá receber esses dados do formulário e organizá-lo. Em seguida, acionará o *bean* (**sistemaAcademico.java**) que se encarregará de buscar no banco de dados e retornar ao *servlet* uma resposta.

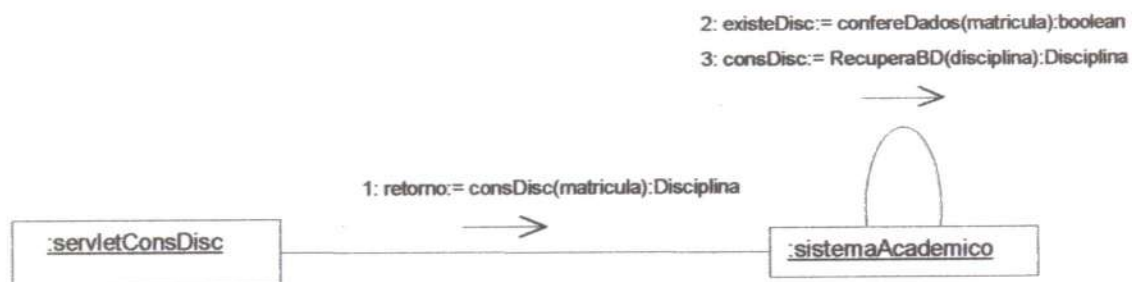


FIGURA 64 - Diagrama de colaboração consultar disciplinas.



FIGURA 65 - Diagrama de atividades do servletConsDisc.

4. CONCLUSÃO

Sistemas concebidos com a utilização da tecnologia de orientação a objetos, são desenvolvido amplamente. Isso deve-se ao fato dos mesmos serem portáteis, ágeis e fáceis de se manter. Em Instituições acadêmicas há uma grande necessidade de se implantar sistemas desse tipo, pois possibilitará a integração de elementos acadêmicos tais como professor, coordenador e aluno.

A implantação do sistema trará bastantes benefícios para a Instituição. Basta a mesma atentar e certificar-se que a solução atual é deficitária, comparada e apresentada neste trabalho.

A implementação está em fase inicial, tendo somente os módulos de cadastro e consulta de alunos desenvolvidos. Espera-se que os demais sejam concluídos por um outro docente futuro integrante, ou até mesmo por nós caso continuarmos no projeto.

Destaca-se como ponto fraco deste trabalho, a não completude da implementação e pontos fortes a descentralização de tarefas realizadas pela secretaria e CPD e total integração de informações com a Internet, a respeito de notas, para docentes e discentes.

As perspectivas futuras são de continuidade do trabalho, partindo de que esse protótipo apresenta um excelente proposta para a Instituição e assim pretende-se a continuidade do desenvolvimento. Projeta-se incubar este projeto na Instituição FARN ou até mesmo comercializar para outras.

REFERÊNCIAS BIBLIOGRÁFICAS

BATISTA, Galileu. **Conteúdo dinâmico, CGI, Servlets e JSP.**

Disponível em <<http://www.jspbrasil.com.br>>

BONFIM JÚNIOR, Francisco Tarcizo. **JSP: a tecnologia na Internet.** São Paulo: Érica, 2002.

FIELDS, Duane K. e KOLB, Mark A. **Desenvolvendo na web com JavaServer Pages.** Rio de Janeiro: Editora Ciência Moderna Ltda, 2000.

FOWLER, Martin; SCOTT, Kendal. **UML essencial.** 2 ed. Trad. Vera Pezerico e Christian Thomas Price. Porto Alegre: Bookman, 2000.

FURLAN, José Davi. **Modelagem de Objetos Através da UML.** São Paulo: Makron Books, 1998.

HAGGAR, Peter. **Java: guia prático de programação.** Trad. Daniel Vieira. Rio de Janeiro: Campus, 2000.

LARMAN, Craig. **Utilizando UML e padrões.** Trad. Luiz Augusto Meireles Salgado. Porto Alegre: Bookman, 2000.

LIESSE, Edison. **SQL: guia de consulta rápida.** São Paulo: Novatec, [s.d.].

LIMA, Gleydson. **JavaBeans.**

Disponível em <<http://www.jspbrasil.com.br>>

MAYER, Roberto Carlos. **Otimizando a Performance de Banco de Dados Relacionais.** Rio de Janeiro: Axcel, 2001.

RAMON, Fábio. **Java: guia de consulta rápida.** São Paulo: Novatec, [s.d.].

TELEMACO NETO, Ulisses. **O que é JSP.**

Disponível em <<http://www.jspbrasil.com.br>>

ANEXO A - EXEMPLO DE CÓDIGO HTML (index.html)

```

<html>
<head><title>ACADEMICO</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body bgcolor="#ffffff" leftmargin="0" topmargin="0" marginwidth="0" marginheight="0">
<table width="100" border="0" cellpadding="0" cellspacing="0">
  <tr>
    <td></td>
    <td></td>
    <td></td>
  </tr>
  <tr>
    <td colspan="3"></td>
  </tr>
  <tr>
    <td colspan="3"><form action="login.jsp" method="post"><table width="500" border="0" align="left"
cellpadding="0" cellspacing="0">
      <tr>
        <td></td>
        <td><div align="center"><strong><font size="3" face="Arial, Helvetica, sans-
serif">Usuário:</font></strong></div></td>
        <td><div align="center">
          <input name="user" type="text" id="user">
        </div></td>
      </tr>
      <tr>
        <td><div align="center"><strong><font size="3" face="Arial, Helvetica, sans-
serif">Senha:</font></strong></div></td>
        <td><div align="center">
          <input name="pass" type="password" id="pass">
        </div></td>
      </tr>
      <tr>
        <td>&nbsp;</td>
        <td colspan="2"><div align="center"><br>
          <font size="2" face="Verdana, Arial, Helvetica, sans-serif">
            <input name="entrar" type="submit" id="entrar" value="Entrar"> &nbsp;  
            <input type="reset" name="Submit2" value="Limpar">
          </font></div></td>
      </tr>
    </table>
    </form>
    </td>
    <td align="right" valign="top">&nbsp;</td>
  </tr>
</table>
</body>
</html>

```

ANEXO B - EXEMPLO DE CÓDIGO JSP (servletConsultarAluno.jsp)

```

<html>
<head><title>ACADEMICO</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"></head>
<body bgcolor="#ffffff" leftmargin="0" topmargin="0" marginwidth="0" marginheight="0">
<%@ page import = "Academico.*" %>
<jsp:useBean id="acesso" class="Academico.SistemaAcademico"/>
<jsp:setProperty name="acesso" property="tarefa" param="tarefa"/>
<jsp:setProperty name="acesso" property="nome" param="nome"/>
<jsp:setProperty name="acesso" property="nascimento" param="nascimento"/>
<jsp:setProperty name="acesso" property="pai" param="pai"/>
<jsp:setProperty name="acesso" property="mae" param="mae"/>
<jsp:setProperty name="acesso" property="endereco" param="endereco"/>
<jsp:setProperty name="acesso" property="num_casa" param="num_casa"/>
<jsp:setProperty name="acesso" property="bairro" param="bairro"/>
<jsp:setProperty name="acesso" property="email" param="email"/>
<jsp:setProperty name="acesso" property="celular" param="celular"/>
<jsp:setProperty name="acesso" property="telefone" param="telefone"/>
<jsp:setProperty name="acesso" property="cpf" param="cpf"/>
<jsp:setProperty name="acesso" property="rg" param="rg"/>
<jsp:setProperty name="acesso" property="ssp" param="ssp"/>
<jsp:setProperty name="acesso" property="curso" param="curso"/>
<jsp:setProperty name="acesso" property="turma" param="turma"/>
<jsp:setProperty name="acesso" property="matricula" param="matricula"/>
<table width="100" border="0" cellpadding="0" cellspacing="0">
  <tr>
    <td></td>
    <td></td>
    <td></td>
  </tr>
  <tr>
    <td colspan="3"></td>
  </tr>
  <tr>
    <td colspan="3">
      <table width="517" border="0" align="left" cellpadding="0" cellspacing="0">
        <tr>
          <td width="233" valign="top"><font color="#FF0000" size="2" face="Verdana, Arial, Helvetica, sans-
          serif"><b><br><% out.println(msg); %> </b></font></td>
          <td width="267" colspan="2" valign="top"><center>
            <font face="Arial" size="3"><b><br>
            Consultar aluno por matr&iacute;cula:</b></font>
            <form name="cons" method="post" action="servletResultadoConsultaAluno.jsp">
              <font face="Arial" size="2"><b>Matr&iacute;cula:</b></font>
              <input name="matricula" type="text" id="matricula">
              <br>
              <input type="hidden" name="tarefa" value="1">
              <br>
              <input type="submit" name="Submit" value="Consultar">
            </form>
            <font size="1" face="Arial, Helvetica, sans-serif"><strong>Formato:
            1999B010938</strong></font> </center></td>
        </tr>
      </table>
    </td>
  </tr>
</table>

```

```

        </td>
        <td align="right" valign="top">
            <%      if (ac != 0) {
                %>
                <a href="logout.jsp"></a>
                <%
                    }
                %>
            </td>
        </tr>
        <tr>
            <td colspan="3"><div align="center">
                <p>&nbsp;</p>
                <p><font color="#003333" size="1" face="Arial, Helvetica, sans-serif"></font></p>
            </div></td>
        </tr>
    </table>
    <div id="Layer1" style="position:absolute; width:191px; height:13px; z-index:1; left: 3px; top: 198px;"><font
color="#FF0000" size="1" face="Arial, Helvetica, sans-serif"><strong>Voc&ecirc;
est&aacute; numa &aacute;rea segura.</strong></font></div>
</body>
</html>

```

ANEXO C - EXEMPLO DE CÓDIGO JAVA (SistemaAcademico.java)

```
package Academico;
import java.sql.*;
public class SistemaAcademico extends java.lang.Object {
    // Declaração das Variáveis
    static Connection con = null;
    static private String blank = "";
    private String nome = blank;
    private String nascimento = blank;
    private String pai = blank;
    private String mae = blank;
    private String endereco = blank;
    private String num_casa = blank;
    private String bairro = blank;
    private String email = blank;
    private String celular = blank;
    private String telefone = blank;
    private String cpf = blank;
    private String rg = blank;
    private String ssp = blank;
    private String curso = blank;
    private String turma = blank;
    private String ano_ingresso = blank;
    private String area_curso = blank;
    private String cod_curso = blank;
    private String classificacao = blank;
    private String m = blank;
    private String matricula;
    private int tarefa = 0;
    // Inicia a Conexão
    public void conectar() throws SQLException {
        if (con == null) {
            try {
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                con = DriverManager.getConnection("jdbc:odbc:academico","1234","1234");
            }
            catch (Exception e) {
                System.err.println ("Erro no comando Conexão");
            }
        }
    }
    // Método que retorna o campo 'nome'
    public String getNome() {
        return nome;
    }
    // Método que define o campo 'nome'
    public void setNome(String nome) {
        this.nome = nome;
    }
    // Método que retorna o campo 'nascimento'
    public String getNascimento() {
        return nascimento;
    }
    // Método que define o campo 'nascimento'
    public void setNascimento(String nascimento) {
```

```

        this.nascimento = nascimento;
    }
    // Método que retorna o campo 'pai'
    public String getPai() {
        return pai;
    }
    // Método que define o campo 'pai'
    public void setPai(String pai) {
        this.pai = pai;
    }
    // Método que retorna o campo 'mae'
    public String getMae() {
        return mae;
    }
    // Método que define o campo 'mae'
    public void setMae(String mae) {
        this.mae = mae;
    }
    // Método que retorna o campo 'endereço'
    public String getEndereco() {
        return endereco;
    }
    // Método que define o campo 'endereço'
    public void setEndereco(String endereco) {
        this.endereco = endereco;
    }
    // Método que retorna o campo 'num_casa'
    public String getNum_casa() {
        return num_casa;
    }
    // Método que define o campo 'num_casa'
    public void setNum_casa(String num_casa) {
        this.num_casa = num_casa;
    }
    // Método que retorna o campo 'bairro'
    public String getBairro() {
        return bairro;
    }
    // Método que define o campo 'bairro'
    public void setBairro(String bairro) {
        this.bairro = bairro;
    }
    // Método que retorna o campo 'email'
    public String getEmail() {
        return email;
    }
    // Método que define o campo 'email'
    public void setEmail(String email) {
        this.email = email;
    }
    // Método que retorna o campo 'celular'
    public String getCelular() {
        return celular;
    }
    // Método que define o campo 'celular'
    public void setCelular(String celular) {
        this.celular = celular;
    }
}

```

```

// Método que retorna o campo 'telefone'
public String getTelefone() {
    return telefone;
}
// Método que define o campo 'telefone'
public void setTelefone(String telefone) {
    this.telefone = telefone;
}
// Método que retorna o campo 'cpf'
public String getCpf() {
    return cpf;
}
// Método que define o campo 'cpf'
public void setCpf(String cpf) {
    this.cpf = cpf;
}
// Método que retorna o campo 'rg'
public String getRg() {
    return rg;
}
// Método que define o campo 'rg'
public void setRg(String rg) {
    this.rg = rg;
}
// Método que retorna o campo 'ssp'
public String getSsp() {
    return ssp;
}
// Método que define o campo 'ssp'
public void setSsp(String ssp) {
    this.ssp = ssp;
}

// Método que retorna o campo 'curso'
public String getCurso() {
    return curso;
}
// Método que define o campo 'curso'
public void setCurso(String curso) {
    this.curso = curso;
}

// Método que retorna o campo 'turma'
public String getTurma() {
    return turma;
}
// Método que define o campo 'turma'
public void setTurma(String turma) {
    this.turma = turma;
}
// Método que retorna o campo 'ano_ingresso'
public String getAno_ingresso() {
    return ano_ingresso;
}
// Método que define o campo 'ano_ingresso'
public void setAno_ingresso(String ano_ingresso) {
    this.ano_ingresso = ano_ingresso;
}

```

```

// Método que retorna o campo 'area_curso'
public String getArea_curso() {
    return area_curso;
}

// Método que define o campo 'area_curso'
public void setArea_curso(String area_curso) {
    this.area_curso = area_curso;
}

// Método que retorna o campo 'cod_curso'
public String getCod_curso() {
    return cod_curso;
}

// Método que define o campo 'cod_curso'
public void setCod_curso(String cod_curso) {
    this.cod_curso = cod_curso;
}

// Método que retorna o campo 'classificacao'
public String getClassificacao() {
    return classificacao;
}

// Método que define o campo 'classificacao'
public void setClassificacao(String classificacao) {
    this.classificacao = classificacao;
}

// Método que retorna o campo 'tarefa'
public int getTarefa() {
    return tarefa;
}

// Método que define o campo 'tarefa'
public void setTarefa(int tarefa) {
    this.tarefa = tarefa;
}

// Método para resetar os campos
public void limpaTudo() {
    tarefa = 0;
}

// Método que retorna o campo 'matricula'
public String getMatricula() {
    return matricula;
}

// Método que define o campo 'tarefa'
public void setMatricula(String matricula) {
    this.matricula = matricula;
}

// Gera matricula
public String geraMat() {
    //String numero = getAno_ingresso() + getCod_curso() + getClassificacao();
    String dv;
    String letra = getArea_curso();
    String mat;
    /*String areacurso = "A";
    int let;
    int soma = 0;
    int resto;
    int dv;
    int divisao;

```

```

String mat;
let = Integer.parseInt(letra);
Long.parseLong(numero);

soma
=
((numero.charAt(0)*3)+(numero.charAt(1)*2)+(numero.charAt(2)*9)+(numero.charAt(3)*8)+(numero.charAt(4)
)*6)+(numero.charAt(5)*5)+(numero.charAt(6)*4)+(let*7)+(numero.charAt(7)*3)+(numero.charAt(8)*2));
divisao = soma/11;
resto = soma - (11*divisao);

if(getArea_curso() != "11") {
    areacurso = "B";
}

if (resto == 0 || resto == 1) {
    dv = 0;
}
else {
    dv = 11-resto;
}*/
dv = "8";
mat = getAno_ingresso() + letra + getCod_curso() + getClassificacao() + dv;
return mat;
}

// Método para gravar e pesquisar no banco de dados
public boolean fazTarefa() {
    // Incluir registro
    if (tarefa == 2) {
        try {
            // Inclui no banco de dados
            matricula = geraMat();
            conectar();
            Statement stmt = con.createStatement();
            stmt.execute ("INSERT INTO Alunos (nome, matricula, turma, email,
telefone, celular, pai, mae, endereco, bairro, num_casa, cpf, rg, ssp, nascimento, ID_curso) VALUES ('" + nome
+ "','" + matricula + "','" + turma + "','" + email + "','" + telefone + "','" + celular + "','" + pai + "','" + mae + "','" +
endereco + "','" + bairro + "','" + num_casa + "','" + cpf + "','" + rg + "','" + ssp + "','" + nascimento + "','" + curso
+ "')");

            stmt.close();
        }
        catch (Exception e) {
            // Verificar erros
            System.err.println("Erro no comando SQL de Insert");
            limpaTudo();
            return false;
        }
    }

    // Pesquisar por Matricula
    else if (tarefa == 1) {
        try {
            // Faz a pesquisa e retorna um recordset
            conectar();
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Alunos WHERE
matricula = '" + matricula + "'");
            if (rs.next()) {

```

```

        // Se o recordset contiver dados,
        // guarda-os nas respectivas variáveis
        nome = rs.getString("nome");
        nascimento = rs.getString("nascimento");
        pai = rs.getString("pai");
        mae = rs.getString("mae");
        endereco = rs.getString("endereco");
        num_casa = rs.getString("num_casa");
        bairro = rs.getString("bairro");
        email = rs.getString("email");
        telefone = rs.getString("telefone");
        celular = rs.getString("celular");
        cpf = rs.getString("cpf");
        ssp = rs.getString("ssp");
        rg = rs.getString("rg");

    }
    else {
        // Recordset vazio
        System.err.println ("Nenhum Registro encontrado!!");
        limpaTudo();
        return false;
    }
    rs.close();
    stmt.close();
}
catch (Exception e) {
    // Verificar erros
    System.err.println ("Erro no comando SQL de Select");
    limpaTudo();
    return false;
}
}
else {
    // Nenhuma Tarefa
    limpaTudo();
    return false;
}
return true;
}
}

```