UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE UM SISTEMA DE PRÉ-MATRÍCULA VIA INTERNET UTILIZANDO AGENTES COM ACESSO A BANCO DE DADOS

TRABALHO DE ESTÁGIO SUPERVISIONADO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

MAURI FERRANDIN

BLUMENAU, JUNHO/1999

1999/1-46

PROTÓTIPO DE UM SISTEMA DE PRÉ-MATRÍCULA VIA INTERNET UTILIZANDO AGENTES COM ACESSO A BANCO DE DADOS

MAURI FERRANDIN

ESTE TRABALHO DE ESTÁGIO SUPERVISIONADO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE ESTÁGIO SUPERVISIONADO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Mauricio Capobianco Lopes — Supervisor na FURB

_____

Prof. Achilles Santos Silva Junior — Orientador na Empresa

_____

Prof. José Roque Voltolini da Silva — Coordenador na FURB do Estágio Supervisionado

BANCA EXAMINADORA

_____

Prof. Mauricio Capobianco Lopes

_____

Prof. Paulo M. de Tarso Luna

_____

Prof. Dalton Solano dos Reis

Dedico este trabalho a Deus, e a todos que acreditam na ciência e enxergam na educação e na pesquisa a solução para os problemas do nosso planeta.

AGRADECIMENTOS

Agradeço a meus amigos, colegas e professores pelo companheirismo, troca de experiências e pela convivência acadêmica.

Em especial agradeço a meus familiares principalmente meus pais pela vida, exemplo de trabalho e esforços, e pelos incentivos frente as dificuldades encontradas ao longo da jornada.

Também agradeço a FERJ que possibilitou-me a realização deste estágio, bem como aos professores Mauricio e Achilles que me orientaram durante a realização deste trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS

API - *Application Program Interface*;

ASDK – *Aglets Software Development Kit*;

ATP - *Agent Transfer Protocol*;

AWB – *Aglets WorkBench*;

AWT – *Abstract Windowing Toolkit;*

CGI – *Common Gateway Interface*;

GUI - *Graphical User Interface*;

HTML – *HiperText Markup Language*;

HTTP – *HiperText Transfer Protocol*;

IP – Internet *Protocol;*

J-AAPI - *Java Aglet API*;

JDBC - *Java Database Connectivity*;

JDK – *Java Development Kit*;

RCT – Rede de Ciência e Tecnologia*;*

SGBD – Sistema Gerenciador de Banco de Dados;

SQL – *Structured Query Language;*

TCP – *Tranfer Control Protocol;*

URL - *Uniform Resource Locator*;

WWW - *World Wide Web;*

RESUMO

Este estágio ilustra por meio de um protótipo o uso de agentes móveis para implementar sistemas que necessitam efetuar transações envolvendo banco de dados através da Internet. Para a implementação foi usada a linguagem Java que é distribuída através do JDK 1.1.6, os *Aglets* que são agentes baseados na linguagem de programação Java, permitindo fazer consultas a bancos de dados utilizando o JDBC, e a tecnologia Fiji que permite o uso de agentes em páginas *Web* através de um *browser* que suporte a linguagem Java. O caso estudado foi o processo de pré-matrícula na FERJ onde foi implementada uma interface disponível via *Web* que utiliza agentes móveis para realizar as transações necessárias ao processo no banco de dados.

ABSTRACT

This work considers the use of mobile agents to effect transactions involving databases through the Internet. For the implementation was used Java language which is distributed through JDK 1.1.6, the *Aglets* that are agents based on Java programming language which allows to make consultations on databases using the JDBC, and the Fiji technology which allows to use agents through the *Web* sites in a *browser* with support for Java language. The studied case was the process academics registration in the FERJ where an *Web* interface was implemented using mobile agents to effect transactions necessary for this process in a databases system.

1 INTRODUÇÃO

A popularidade da Internet como um mecanismo universal de acesso a informações criou a necessidade do desenvolvimento de aplicações com acesso a banco de dados que suportem esta tecnologia. Isto sugere que os navegadores *Web* possam oferecer ao usuário final uma interface que lhe possibilite uma grande quantidade de aplicações [PAP98].

Atualmente dois dos meios mais usados para transações via Internet são a *Commom Gateway Interface* (CGI) e os *applets* (programas Java). Usando CGI, cada solicitação faz com que um programa seja executado no servidor *Web.* Assim, quanto mais consultas estiverem sendo solicitadas ao mesmo tempo, via CGI, mais instâncias do programa estarão sendo executadas. Já os *applets*, são programas que executam na máquina cliente, possuem uma grande portabilidade e geralmente são grandes e pesados, o que pode acarretar num alto tempo de resposta [DAR97].

A tecnologia Java tornou-se respeitada nos centros de computação pelo fato de ser portável e orientada para a Internet, o que a leva a ser a melhor linguagem para implementações Cliente/Servidor e computação móvel. Mais ainda, com a criação do JDBC (*Java Database Connectivity*) que permite o acesso a bancos de dados, tem despertado o interesse dos mais diversos fabricantes de Sistemas Gerenciadores de Banco de Dados (SGBD).

A união destas duas tecnologias, Java e *Web*, pode fornecer soluções para a recuperação de informações em bancos de dados. Atualmente, existem metodologias que possibilitam a implementação de *applet*s que acessem a bancos de dados através do JDBC, mas que possuem grandes limitações como, por exemplo, o fato de que o provedor de serviços *Web* precisa estar no mesmo servidor em que está o banco de dados, além do *browser* cliente precisar descarregar as classes do JDBC do servidor antes de inicializar seu contexto, ou seja, essas tecnologias convencionais são bastante limitadas principalmente em termos de performance.

Este estágio propõe o uso de agentes móveis para facilitar a implementação de aplicações Cliente/Servidor na *Web*. Esta nova abordagem consiste em permitir que um *applet*

descarregado por uma página *Web* a partir de um *browser* cliente possa criar um agente que se deslocará até o servidor de banco de dados, fará as consultas necessárias e retornará os resultados ao *browser* do cliente através de mensagens. O ambiente usado para execução dos agentes móveis que será utilizado é o *Aglet Technology*, desenvolvido pela IBM em Tóquio, que consiste em um conjunto de classes em Java para a criação de agentes móveis.

1.1 MOTIVAÇÃO

A combinação das tecnologias Java e *Web* é um campo promissor, tanto na área de agentes móveis e inteligentes, quanto na computação móvel e sistemas distribuídos. Ela promete revolucionar estas áreas, quebrando paradigmas e conceitos anteriores e proporcionando maiores facilidades para desenvolvimento de aplicações.

1.2 OBJETIVOS

O principal objetivo deste trabalho é implementar um protótipo de um sistema de pré-matrícula através da Internet, utilizando agentes móveis, propondo uma solução segura para desenvolvimento de aplicações na Internet que envolvam bancos de dados.

1.3 ORGANIZAÇÃO DO TEXTO

O capítulo 2 trata da revisão bibliográfica dos conceitos básicos sobre Internet e banco de dados.

O capítulo 3 será dedicado a linguagem Java, suas características e componentes de acesso a dados em bases relacionais, bem como *applet*s e questões de segurança da linguagem na *Web*.

O capítulo 4 explanará definições de agentes, bem como, suas características e seu funcionamento. Também será abordado o uso de agentes na Internet.

O capítulo 5 apresenta um conjunto de classes e interfaces implementadas em Java que possuem o nome de *aglets*. Os *aglets* foram projetados pela IBM Corporation e possuem atributos e conceitos de agentes, como, por exemplo, mobilidade. Serão abordados também seus possíveis estados e características.

O capítulo 6 é uma continuação do capítulo *aglets*, descrevendo o funcionamento da arquitetura Fiji.

O capítulo 7 é o protótipo proposto, onde serão demonstrados os ambientes de desenvolvimento e o funcionamento do protótipo.

No capítulo 8 está a conclusão do trabalho juntamente com suas deficiências e sugestões para novas pesquisas.

No capítulo 9 estão os anexos contendo os fontes com as implementações feitas na realização deste estágio.

1.4 A EMPRESA

A Fundação Educacional Regional Jaraguaense (FERJ) foi instituída por Lei Municipal nº 439 em 31 de agosto de 1973, pelo então prefeito Eugênio Strebe e tendo como seu principal idealizador o Padre Elemar Scheid.

O Padre Elemar Scheid foi o fundador e seu primeiro presidente, ficando no cargo até 1978, assumindo, então, a Professora Carla Schreiner que foi reeleita sucessivamente até os dias atuais. Seu último mandato vigorará até 01 de março do ano 2000.

A história da FERJ possui um início e meio, porém, não tem fim. O resultado até aqui alcançado é fruto de muito esforço e dedicação dos seus fundadores, dirigentes, conselheiros e funcionários e para continuar esta história de sucesso, cada vez mais, todos deverão continuar com este espírito de dedicação e comprometimento com os objetivos da instituição.

A FERJ é uma entidade de caráter comunitário e regional, sem fins lucrativos, de finalidade filantrópica, pessoa jurídica de direito privado, gozando de autonomia administrativa, financeira e disciplinar e as instituições por ela mantidas, gozarão de autonomia didático-científica, disciplinar, administrativa e econômico-financeira, nos termos da legislação federal e estadual.

1.4.1 CURSOS OFERECIDOS

A FERJ atua oferecendo hoje cursos na área de graduação, pós graduação e extensão.

Na Graduação oferece os cursos de :

- Administração;
- Ciências Contábeis;
- Pedagogia;
- Letras com habilitação em Licenciatura em Português e Inglês;
- Bacharelado em Secretário Executivo Bilíngüe em Português e Inglês;
- Tecnologia em Mecânica;
- Arquitetura e Urbanismo;
- Matemática e Alemão - Projeto Magister.

Na especialização *Lato Sensu* possui cursos na área de :

- Moda;
- Gestão Empresarial;
- Engenharia do Produto e Processo;
- Contabilidade Gerencial e Controladoria;

- Direito e Saúde;
- Informática na Educação.

Na área de mestrado existem os cursos de Marketing e Engenharia do Produto e Processo.

Também possui um curso de doutorado em andamento na área de educação.

Ao todo são 1632 alunos em cursos de graduação, 205 em especialização e 35 em doutorado.

1.4.2 ESTRUTURA FÍSICA E FUNCIONAL

O Campus onde está instalada ocupa um terreno de aproximadamente 124 mil metros quadrados, dos quais 8.713 são de área construída, englobando cinco prédios com 41 salas de aula, um auditório, área administrativa, biblioteca, laboratórios de informática e um prédio especialmente construído para abrigar os laboratórios de Arquitetura e Urbanismo.

A FERJ conta com 214 funcionários, sendo 160 docentes e 54 técnico-administrativos.

1.4.3 ASSESSORIA DE INFORMÁTICA

A Assessoria de Informática é o setor da instituição responsável pelos sistemas e pelo parque de equipamentos de informática instalado na mesma, sendo composta por quatro funcionários e está hierarquicamente subordinada a Direção Geral.

Além dos sistema de controle acadêmico, folha de pagamento, contabilidade, compras e biblioteca que são adquiridos de terceiros, o setor coordena todas as atividades nos três laboratórios de informática da instituição.

Também são atribuições do setor :

- administrar os servidores ligados a Internet através da Rede de Ciência e Tecnologia (RCT);
- planejar os investimentos na área de informática;
- proporcionar apoio e suporte a usuários;
- análise de *softwares* e *hardwares*.

1.4.4 BACKBONE DA REDE INTERNA DA FERJ

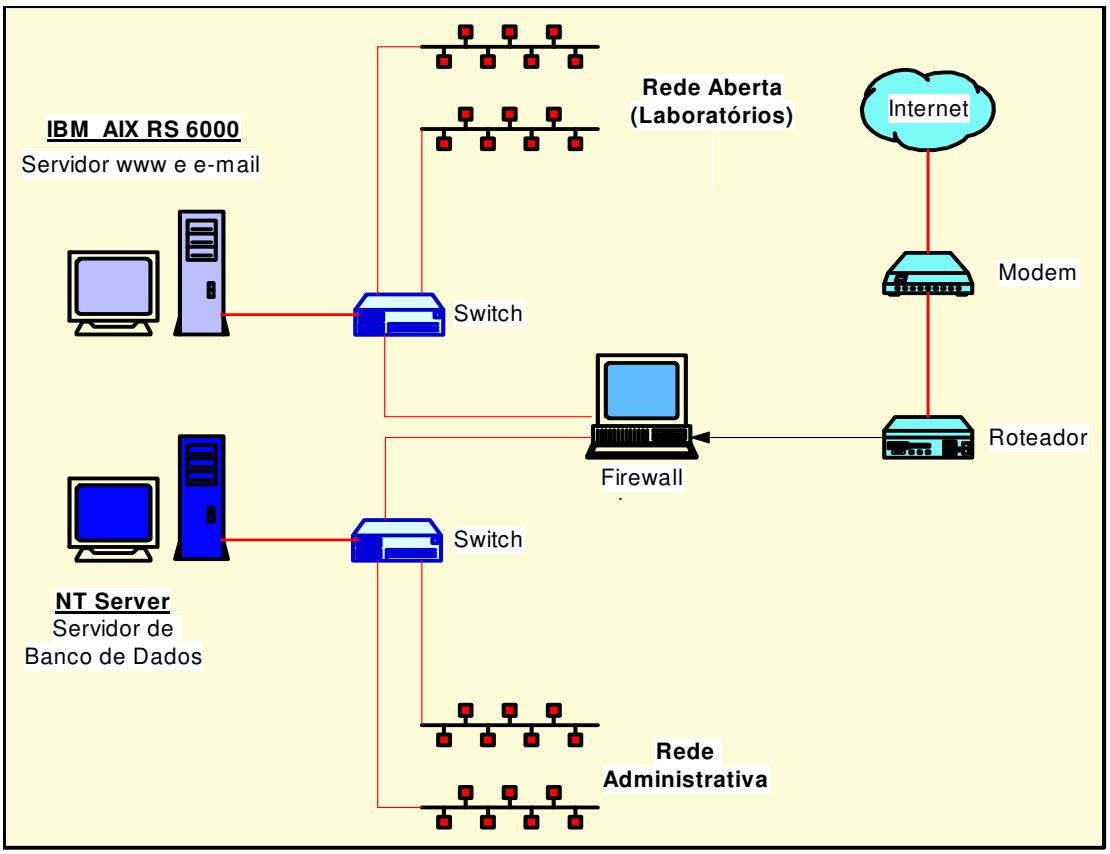

FIGURA 1 : BACKBONE DA FERJ

Conforme mostra a Figura 1, a rede interna da FERJ está com sua estrutura dividida nas seguintes sub-redes :

- **Rede administrativa ou segura** que é composta pelos equipamentos que usam os sistemas de controle acadêmico e outras aplicações da área administrativa;
- **Rede aberta ou insegura** que é composta pelos equipamentos que não precisam de uma política rígida de segurança como os laboratórios de informática e setores que utilizam apenas aplicativos comuns como o MSOffice e não precisam acessar a dados dos registros de controle acadêmico.

É importante salientar que ambas as sub-redes possuem acesso a Internet através da conexão dedicada com o *backbone* da RCT, e que, atualmente, a conexão é feita com a FEJ (Faculdade de Engenharia de Joinville.).

2 INTERNET

A Internet tem sido a novidade mais útil e atraente dos últimos anos na área de informática, mas na verdade ela já é uma tecnologia que surgiu a aproximadamente 30 anos, quando os pesquisadores do *Defense Advanced Research Project Agency* (DARPA) e da Universidade da Califórnia juntamente com outros pesquisadores desenvolveram um novo conjunto de padrões de comunicação de dados, com o objetivo de garantir a comunicação entre plataformas sob condições adversas, como por exemplo, uma guerra nuclear.

Até 1989, a Internet estava restrita para uso em ambientes acadêmicos e militares, e seu uso com fins comerciais estava restritamente proibido. Sua liberação para fins comerciais deveu-se principalmente a redução dos orçamentos militares com o fim da Guerra Fria [BEN97].

Segundo [BEN97], a Internet é um conjunto heterogêneo de computadores que se comunicam através de protocolos de rede. Já segundo [TIE97], a Internet é uma rede de computadores, uma coleção de informações e uma comunidade de pessoas.

A quantidade de conhecimento e informações disponíveis hoje na rede é enorme, contudo, o fato de existir uma certa informação disponível não significa que ela vai ser encontrada. Essa falta de um controle centralizado ocasiona muitas vezes dificuldades de recuperação das informações na rede.

2.1 FUNCIONAMENTO BÁSICO DA INTERNET

Uma das únicas informações que são catalogadas de maneira centralizada na Internet são os endereços dos computadores e sub-redes. Cada máquina ou sub-rede na Internet possui um número chamado IP (*Internet Protocol)* que a identifica unicamente e que é composto por quatro números que vão de 1 a 254 separados por pontos, por exemplo “200.135.231.17”.

Existem organismos nacionais e internacionais que controlam a distribuição de endereços IP, os quais, são classificados por classes descritas pelas letras A, B, C. Por exemplo, se uma instituição recebeu o direito de uso da classe C 123.123.123.x, onde x poderá variar entre 1 e 254 dando-lhe a possibilidade de 254 endereços. Se recebesse uma classe B 123.123.y.x ela teria a possibilidade de 64.516 endereços, variando o x e o y de 1 a 254. Por último então, se ela recebesse uma classe A como 123.x.y.z, a mesma conseguiria 16.387.064 endereços.

Para melhor exemplificar, suponhamos a grosso modo que um país como o Brasil recebesse o direito de uso de uma classe A, como por exemplo 200.x.y.z. Este pais concederia cada uma das classes B que serão subclasses da sua classe A para cada estado, por exemplo:

- Santa Catarina 200.001.x.y ;
- Paraná 200.002.x.y ;
- São Paulo 200.003.x.y.

Cada estado por sua vez distribuiria cada uma das classes C que são subclasses da sua classe B para um município por exemplo:

- Blumenau 200.001.001.x;
- Jaraguá do Sul 200.001.002.x;
- Joaçaba 200.001.003.x;
- Herval d'Oeste 200.001.004.x;
- Jaborá 200.001.005.x.

Cada município, então, será responsável pelos 254 endereços IP de sua classe C, os quais ele poderia distribuir entre suas estações conectadas a rede.

Como temos maior facilidade de memorizar nomes do que números cada endereço IP pode ser associado a um nome, assim existem em cada sub-rede uma máquina servidora que armazena o IP e o respectivo nome para a estação da sua rede, este serviço é chamado de *Domain Name System* (DNS).

Vale salientar que cada um desses endereços pode atender a centenas de usuários, uma vez que os mesmos não precisam ter um IP para acessar a Internet, mas sim acesso a um computador com um IP ligado na rede.

A Figura 2 mostra como várias instituições dos mais variados ramos estão conectadas a grande rede.

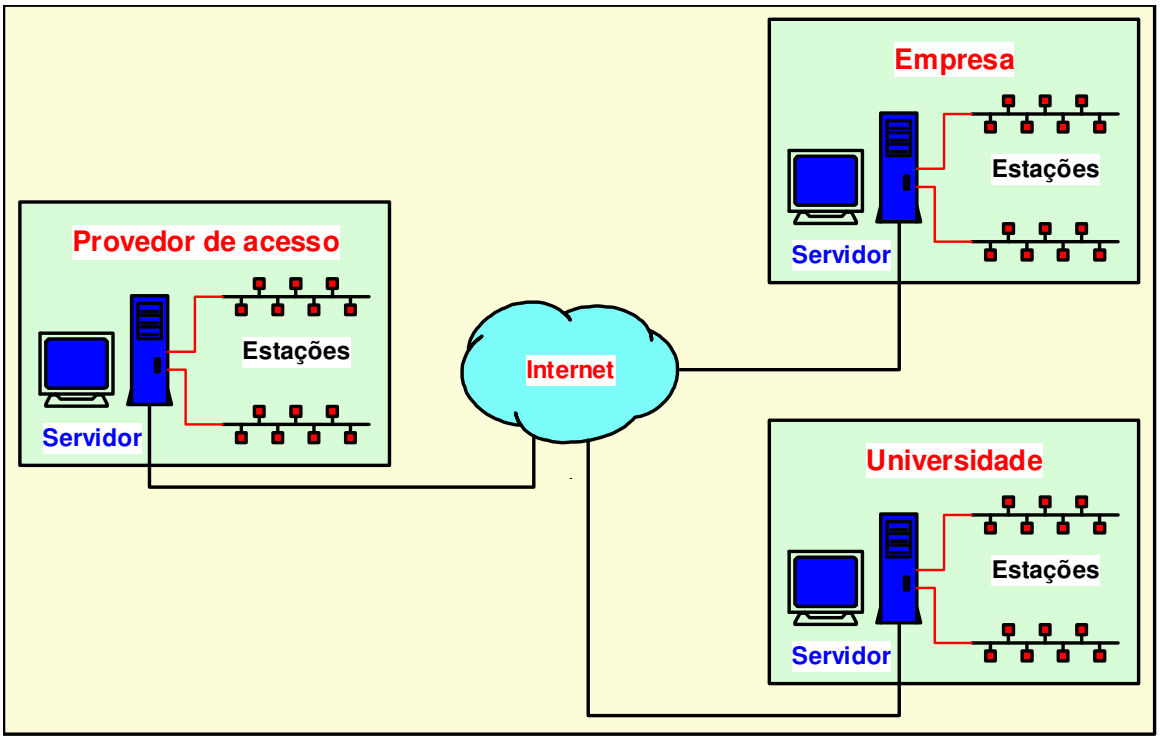

FIGURA 2 : ESCOPO DA INTERNET

2.2 *BROWSER*

Browser ou Navegador Internet é o programa cliente utilizado para acessar informações em servidores pela *Web*. Em outras palavras é o programa que possibilita a visualização de páginas com informações distribuídas por meio de texto, sons e imagens na Internet.

Em 1993, Mark Andreesen, um estudante do Centro Nacional de Aplicações para Supercomputação (NCSA), da Universidade de Illinois, apresentou o Mosaic. No ano seguinte se associou com Jim Clark para fundar a Netscape Corporation, e lançar a versão comercial do NCSA Mosaic, o Netscape Navigator.

Para aumentar a capacidade e a variedade de aplicações que o *browser* pode suportar foram criados softwares capazes de estender suas funcionalidades, sendo que os mesmos foram chamados de *plug-ins.* Um grande número deles já vêm incorporados às versões atuais dos *browsers*, mas se um *browser* acessar um site que exija o *plug-in* que não está instalado o usuário terá que descarrega-lo da *Web* e instalá-lo para conseguir acessar as informações deste site.

A maioria dos navegadores atuais já possuem também suporte a linguagem Java, permitindo a execução de um determinado tipo de programa em seu contexto.

2.3 RECURSOS DA INTERNET

Sob o ponto de vista físico a Internet é uma conexão entre redes, mas para o usuário ela é um meio que pode ser usado para fornecer um grupo de serviços disponíveis para intercâmbio de informações entre computadores ou indivíduos que fazem parte destas redes.

Os principais serviços são :

- correio eletrônico (*e-mail*): permite que um usuário envie mensagens (texto) a outros usuários na rede.
- *File Transfer Protocol* (FTP): permite a transferência eficiente de arquivos entre computadores.
- *Gopher*: sistema de obtenção de informações orientado por menus, no qual os arquivos disponíveis são indexados, por exemplo, pelo seu tema.
- *Telnet*: sistema que permite que uma máquina possa ser um terminal de outra máquina na Internet. Para isso o usuário deve possuir uma conta na máquina da qual quer ser terminal.
- *World Wide Web* (WWW): permite disponibilizar através da *Hipertext Markup Language* (HTML) não só texto, mas também recursos multimídia como imagens, sons, vídeos, entre outros, de maneira transparente para o usuário.

3 JAVA

Java é uma linguagem de programação orientada a objetos desenvolvida pela Sun MicroSystems. Modelada após o C++, foi projetada para ser pequena, simples e compatível com diversas plataformas e sistemas operacionais, tanto no código fonte como também no nível binário [LEM96], conforme a Figura 3.

Também pode ser definida como uma linguagem de desenvolvimento robusta, neutra, segura, simples, orientada a objetos, distribuída e dinâmica [JEP97].

A Figura 3 mostra a arquitetura multiplataforma da linguagem Java, onde os programas Java rodam em qualquer plataforma desde que esta possua um interpretador Java.

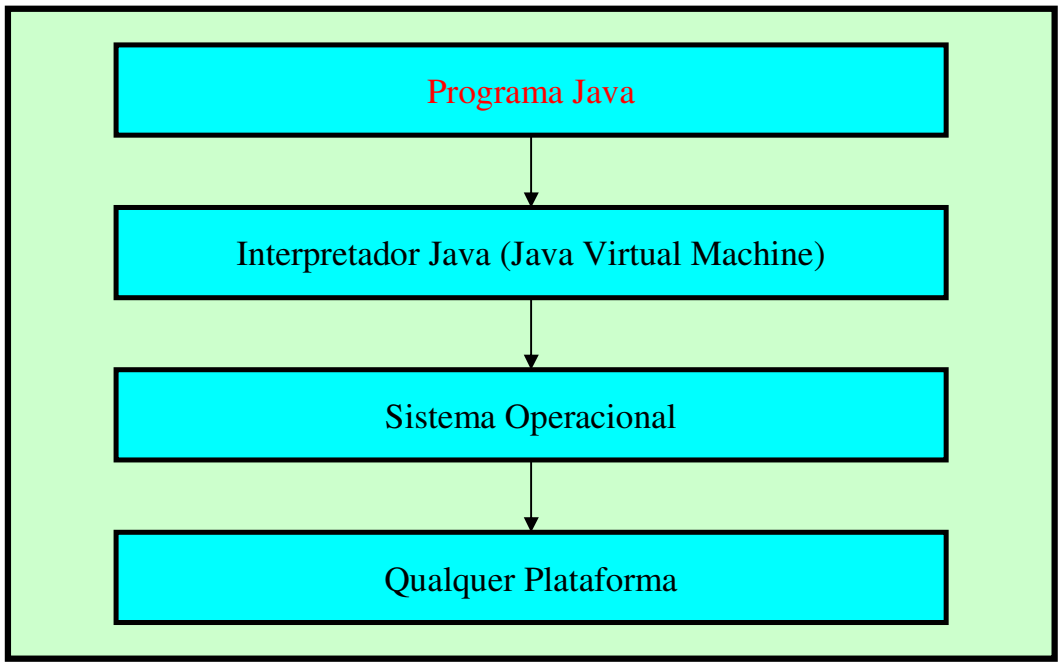

FIGURA 3 : ARQUITETURA MULTIPLATAFORMA DO JAVA

A orientação a objetos da linguagem Java combinada com a facilidade de manipulação na rede e bibliotecas de interface possibilitam o desenvolvimento não somente de uma simples animação em uma página *Web*, através de um *applet*, mas também a criação de aplicações dos mais variados tipos totalmente voltadas a Internet.

A linguagem Java é preferida para desenvolvimento de aplicações envolvendo agentes móveis e sistemas distribuídos pela facilidade com a qual se pode usar o protocolo TCP/IP para fazer conexões entre os vários clientes de uma rede [PAP98].

Existem dois tipos de programas Java : os aplicativos e os *applets*.

Aplicativos em Java são programas que podem ser executados utilizando-se apenas o interpretador Java, ou seja, podem ser executados a partir da linha de comando, como um aplicativo convencional [LEM96].

*Applet*s são executados dentro de um navegador *Web*. Para isso, uma referência ao *applet* é incorporada à página HTML usando um identificador especial da HTML. Quando esta página é descarregada em um navegador que possua suporte a linguagem Java o *applet* é descarregado do servidor *Web* e executado no sistema cliente local.

3.1 APPLETS

Os *applet*s, que podem ser definidos como um programa que pode ser carregado e executado como uma página *Web* exibida por um navegador (*browser*) que possua suporte a liguagem Java [LEM96].

É possível a criação de *applet*s que podem ser descarregados por *browsers* e que através do *driver* JDBC realizam transações em SGBD, mas isso é possível somente se o servidor HTTP estiver rodando na mesma máquina que o SGBD. Esta é uma restrição de segurança da classe *applet*. Outro meio seria o uso de soluções em camadas usando *midleware* que consiste em um programa que roda na máquina onde se encontra o servidor *Web* e funciona com *gateway* entre o *browser* e o SGBD.

3.1.1 RESTRIÇÕES IMPOSTAS AOS *APPLET*S

Partindo do princípio de que um *applet* pode ser descarregado de um servidor *Web* para outros pontos da rede e serem executados em sistemas clientes, são necessárias algumas restrições para impedir que causem danos nos sistemas em que irão ser executados. Sem estas restrições, um *applet* descarregado em um navegador poderia por exemplo executar programas do sistema local apagando todos os arquivos do disco da máquina cliente.

As principais restrições impostas aos *applet*s segundo [LAM96] são :

a) os *applets* não podem ler ou gravar no sistema de arquivos no sistema cliente, exceto diretórios especiais que são definidos pelos usuários do navegador;
b) os *applets* não podem se comunicar com outros servidores exceto aquele no qual ele está armazenado;
c) os *applets* não podem executar qualquer programa no sistema do cliente;
d) os *applets* não podem carregar programas nativos na plataforma local, incluindo bibliotecas compartilhadas tais como DLLs.

3.1.2 EXEMPLO BÁSICO DE UM *APPLET*

```
// Exemplo basico de um applet
// Autor Mauri Ferrandin
package com.ibm.fiji.mauri.exemplos;
import java.awt.*;
import java.applet.*;

public class SampleApplet extends Applet
{

  TextArea saida;
  Button botao;
```

```
    public void init()
    {
        this.setLayout (new BorderLayout ());
        saida = new TextArea (10, 60);
        botao =  new Button("Teste");
        this.add ("North", saida);
        this.add ("South", botao);
    }

    public boolean action (Event ev, Object arg)
    {
        if (ev.target == botao)
        {
            saida.append("voce clicou no botao teste !!! \n");
        }
        return true;
    }
}
```

QUADRO 1 : EXEMPLO DE *APPLET*

O exemplo no Quadro 1 mostra um *applet* que usa bibliotecas gráficas do Java. Ela é composta de um botão com texto “Teste” que é instanciado através da linha de programa “botao = new Button(“Teste”);”, e de um campo TextArea para mostrar as saídas, que corresponde a “saida = new TextArea (10, 60)”, e os dois são adicionados à interface nas linhas “this.add ("Center", saida)” e “this.add ("Center", botao)” onde o termo “this” eqüivale ao *applet* corrente.

A linha “class SampleApplet extends *Applet*” declara que a classe SampleApplet que está sendo instanciada herda as propriedades e métodos da classe *applet* que é carregada através da linha “import java.*applet*.*”.

A Figura 4 nos mostra a execução do *applet* exemplo que quando clicado no botão “Teste” ele mostrará a mensagem “você clicou no botão teste !!!“.

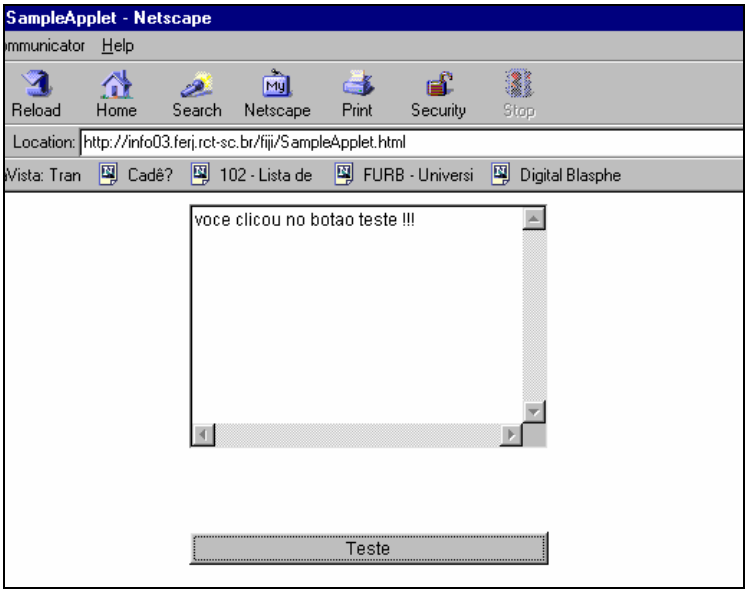

FIGURA 4 : EXECUÇÃO DO SAMPLEAPPLET

3.2 ACESSANDO A DADOS COM JAVA

Pouco depois do surgimento da linguagem Java surgiram os softwares de conectividade com SGBD denominados "*drivers*" que implementam uma tecnologia denominada *Java Database Connectivity* (JDBC), a qual é constituída de um conjunto de objetos e classes para fazer conexão com o SGBD e efetuar transações usando a SQL.

O objetivo principal de um SGBD é prover um ambiente que seja adequado e eficiente para recuperar e armazenar informações e dados [KOR95].

Com o advento do conceito Cliente/Servidor, o SGBD tornou-se atributo indispensável na confecção de sistemas, pois ele facilita, agiliza e centraliza o acesso às informações, além de contribuir para o aumento de segurança e integridade dos sistemas desenvolvidos.

O JDBC é dividido em duas em partes: JDBC *Application Programing Interface* (API), a qual é responsável pela interface com o banco de dados e JDBC *driver* API a qual executa/implementa esta interface.

A Figura 5 mostra como a linguagem Java pode ser usada para acessar a um SGBD através do uso do JDBC como camada intermediária entre a aplicação ou *applet* e o SGBD.

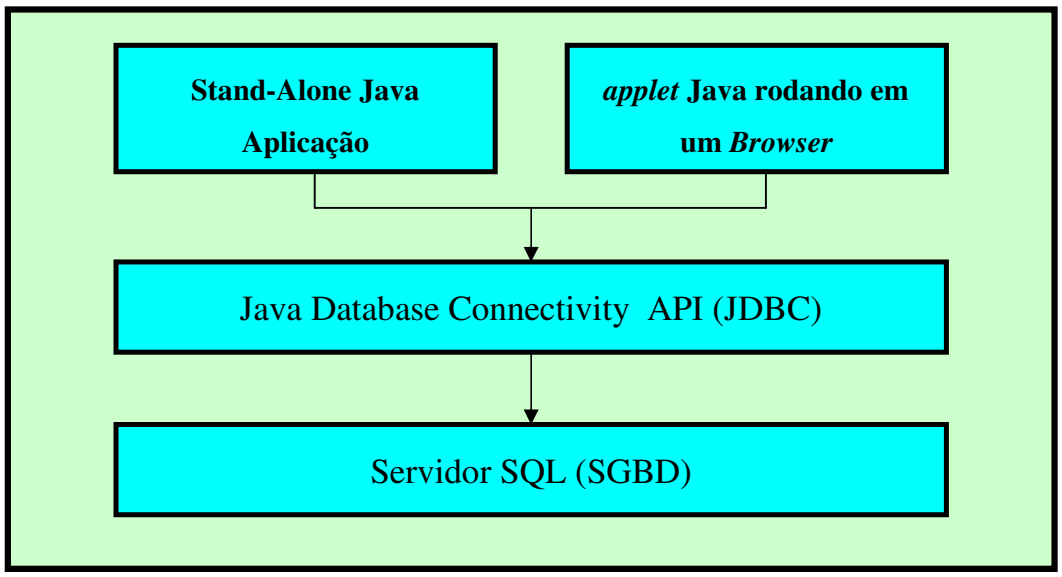

FIGURA 5 : ACESSANDO A DADOS COM JAVA

A especificação inicial da JDBC API foi em 8 de março de 1996 e desde então é uma parte da linguagem Java incluída no *Java Development Kit* (JDK) distribuído pela Sun. No geral ela possui as seguintes classes e programas Java que permitem realizar transações em SGBD:

- a *Connection class-interface* que representa uma seção com uma base de dados específica e que no seu contexto podem ser executados várias transações (*statments*) SQL retornando os resultados;
- a *DatabaseMetaData class-interface* que é usada para obter informações sobre a base de dados, como tabelas, índices, tipos de dados entre outros;
- a *Statement class-interface* usada para executar transações estáticas em SQL e obter o resultado produzido por elas;
- a *PreparedStatement class-interface* usada para executar várias transações SQL e também para consultas parametrizadas;

- a *CallableStatement class-interface* usada para executar procedimentos locais no banco de dados (*Stored Procedures);*
- a *ResultSet class-interface* usada para retornar os dados de um *Statement* SQL executado;
- a *ResultSetMetaData class-interface* usada para obter informações sobre o *ResultSet,* como o número de linhas e colunas contendo os dados obtidos na consulta.

O Quadro 2 mostra um exemplo de como acessar a um SGBD através de um aplicativo Java e a Figura 6 mostra a execução do mesmo.

```
// Importa a package java.sql que contem as classes para uso  JDBC
import java.sql.*;
class ExemploJdbc {
  public static void main (String args [])   throws SQLException  {
    // Carrega o Driver JDBC da Oracle
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    // Conecta o base de dados
   //  sintaxe  <servidor>:<porta>:<serviço>.
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:thin:@netserver.ferj.rct-sc.br:1521:orc1",
                                   "scott", "tiger");
    // Cria uma instância do objeto Statement
    Statement stmt = conn.createStatement ();
    // Executa a instrução SQL e armazena o resultado em uma instância do objeto ResultSet
    ResultSet rset = stmt.executeQuery ("select EMPNO, ENAME from EMP");
    // Mostra os dados da consulta
    while (rset.next ())
      System.out.println (rset.getString (1)+"  "+rset.getString (2) );
  }
}
```

QUADRO 2 : EXEMPLO DE ACESSO A SGBD USANDO JDBC

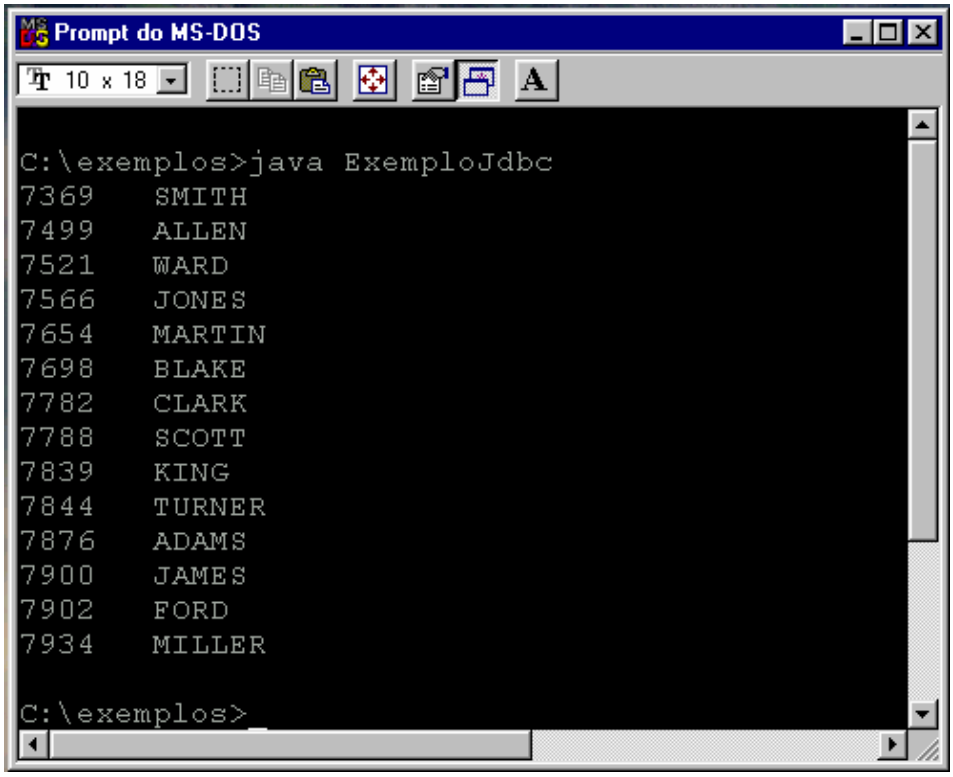

FIGURA 6 : EXECUÇÃO DO APLICATIVO EXEMPLOJDBC.JAVA

4 AGENTES

Pode-se definir um agente como qualquer pessoa ou qualquer coisa cujos atos representam alguma coisa para um grupo, com o propósito expresso de executar atos específicos que geralmente são vistos como benéficos ao grupo representado [DAR97].

No entanto, um agente de software é um software que executa tarefas para seu usuário dentro de um ambiente computacional. Os agentes de software são diferenciados basicamente de outros aplicativos pois devem possuir dimensões de mobilidade, autonomia, e a habilidade interativa, independente da presença de seu usuário [HEI97].

Um agente de software é um programa que pode parar sozinho, remeter-se para outro computador na rede e continuar executando nesse novo computador. Um agente não reinicia a execução no novo computador, ele continua de onde parou.

Agentes são autônomos porque decidem onde irão e o que farão de "suas vidas". Eles podem receber requisições externas, como de outros agentes, mas cada agente decide individualmente se vai atender ou não ao pedido. Os agentes também podem decidir suas ações, como uma viagem através da rede para um novo computador, independente de qualquer solicitação externa [VEN97].

4.1 CARACTERÍSTICAS DOS AGENTES

São características chaves de agentes inteligentes que os diferenciam de outros tipos de aplicações de software:

a) **autonomia**: o agente inteligente deve possuir a capacidade de executar ações que concluam alguma tarefa ou objetivo, sem interferências do usuário final, recebendo uma instrução inicial e indo executar as tarefas esperadas. Lenny Foner [FON94] e Michael Wooldridge [WOO95] indicam que softwares de agentes inteligentes devem ter o controle sobre seu estado interno e comportamento. Isto implica que

um agente deve ter o acesso à rede e deve ter mobilidade para viajar por ela, no caso dos agentes móveis. No contexto da *World Wide Web*, a autonomia de agentes está na habilidade de operar no domínio da Internet, enquanto o usuário final está desconectado ou longe da interação com a rede.

b) **habilidade de comunicação**: os agentes inteligentes, para conseguir alcançar seus objetivos, acessam informações sobre o estado do ambiente externo. Essa informação pode ser obtida de outros agentes ou em repositórios que as contenham armazenadas. Esta comunicação pode ser na forma de uma simples requisição/pesquisa com uma única e concisa série de possíveis respostas ou poderia ser uma comunicação complexa com respostas variáveis.

c) **capacidade para cooperação**: uma extensão natural do atributo de comunicação são os agentes de cooperação. Agentes inteligentes devem ter um espírito "colaborativo" para existir. A visão é de que agentes inteligentes trabalhem juntos para benefício mútuo na resolução de tarefas complexas.

d) **capacidade de raciocinar**: a habilidade de raciocinar durante a execução é um dos aspectos chave da inteligência que distingue os agentes inteligentes de outros agentes. Raciocinar implica que um agente pode possuir habilidade para deduzir e extrapolar, baseado em conhecimento atual e experiências anteriores. Existem 3 tipos básicos de cenários de raciocínio:
   - baseado em regras: onde os agentes usam um conjunto de pré-condições do usuário para avaliar as condições no ambiente externo;
   - baseado em conhecimento: onde são mandadas para os agentes grandes quantidades de dados sobre situações anteriores e as ações resultantes, das quais eles deduzem os movimentos futuros;
   - baseado na evolução artificial: utiliza técnicas de inteligência artificial para sua evolução.

e) **comportamento adaptável**: para manter a argumentação autônoma, o agente deve ter alguns mecanismos para poder avaliar o estado atual de seu domínio externo. Agentes devem poder examinar o ambiente externo, por exemplo a WWW, e o sucesso de ações anteriormente executadas sob condições semelhantes, e adaptar as ações para melhorar a probabilidade de alcançar as metas prosperamente.

f) **confiável**: O usuário deve estar altamente confiante que seus agentes agirão de

acordo com a sua vontade [HEI97].

4.2 AGENTES MÓVEIS

Agentes móveis são programas que tem a capacidade de migrar entre as máquinas de uma rede durante a sua execução, sempre carregando consigo o seu estado de execução. Devido a características como o carregamento dinâmico de código (serialização), e a independência de plataforma, a linguagem Java deu um novo impulso para as pesquisas nesta área [END98].

Os agentes móveis são uma tecnologia que promete tornar muito mais fácil o desenvolvimento, implementação e manutenção de sistemas distribuídos [LAN98]

4.2.1 VANTAGENS DO USO DE AGENTES MÓVEIS

Os agentes móveis podem segundo Marcos Endler [END98] :

a) reduzir o tráfego na rede;
b) executar tarefas complexas ou tediosas;
c) representar pessoas ou organizações, incorporando suas autoridades;
d) executar autonomamente durante longos períodos de tempo;
e) ser ativado por um computador móvel e retornar a ele na próxima execução;
f) ser usados em redes com conexões instáveis ou com pequena largura de banda;
g) interagir com agentes de outros usuários;
h) acessar recursos ou dados em máquinas remotas;
i) monitorar estados de sistemas ou bases de dados.

4.3 AGENTES NA INTERNET

Agentes móveis na Internet são programas que podem ser iniciados em um computador e podem ser transportados a um outro computador remoto para executar, carregando consigo sua memória. Chegando ao computador remoto, eles apresentam suas credenciais e obtêm acesso para serviços locais e dados. O computador distante também pode servir como um corretor, reunindo os agentes com interesses semelhantes e metas compatíveis e pode prover assim, um lugar, no qual os agentes podem interagir [OSH97].

Um modelo básico de funcionamento de agente inteligente é mostrado na Figura 7.

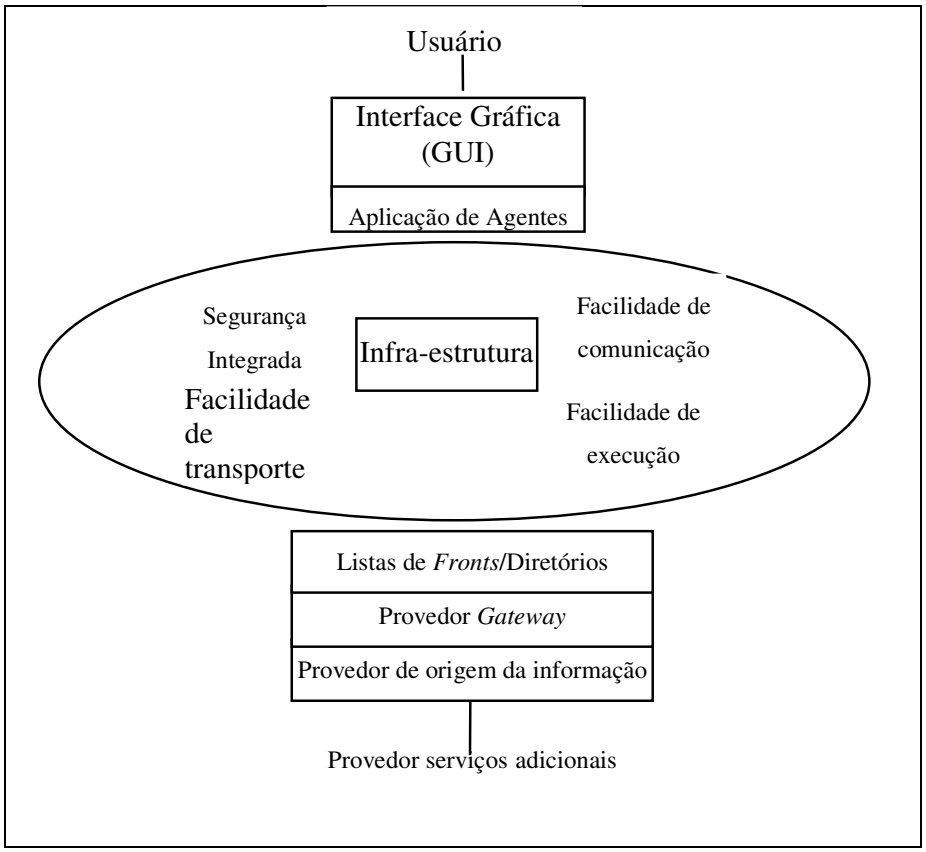

FIGURA 7 : FUNCIONAMENTO DE UM AGENTE [HEI97]

Um usuário que possua uma tarefa para realizar usa uma interface gráfica para usuário (GUI - *Graphical User Interface*), que tem acesso a uma aplicação de agente. A aplicação de agente inteligente viaja pela Internet ou por outra infra-estrutura da rede para chegar ao provedor. Finalmente, o agente pode ter acesso à fonte de informação em bancos de dados ou outro método de armazenamento de dados (informação), encontrando a informação pedida pelo usuário, completando qualquer transação necessária, e retornando uma resposta para o usuário.

O conceito de agentes móveis mudou inúmeras vezes com base em um exame de como os computadores têm se comunicado desde 1970. Agentes móveis tornaram a comunicação entre computadores e os paradigmas de interação muito mais próximos [HEI97] e isso acontece em parte devido à mobilidade dos agentes que agora executarão as chamadas localmente e não mais remotamente.

Os preceitos da tecnologia de agentes existem em muitas das aplicações que muitas pessoas utilizam diariamente. Por exemplo, o cliente de *e-mail* é um tipo de agente. Ao ser acionado, o cliente de *e-mail*, faz um apanhado nos *e-mails* não lidos no servidor de e-*mail.* Clientes de *e-mail* contemporâneos pré-classificam as mensagens recebidas em pastas específicas, baseadas em critérios pré-definidos. Desta maneira, o software torna-se uma extensão do usuário, realizando tarefas em seu favor. Assim, o computador pode ser considerado um agente, sua tarefa primária é aumentar a produtividade através da automação [SOM97].

Agentes encontram-se nas vidas das pessoas diariamente, lembram-nas de compromissos, indicam erros de ortografia ou periodicamente buscam seus *e-mails.* Enquanto alguns agentes na Internet podem executar tarefas complexas de reunião de informação autonomamente, outros agentes na Internet podem reunir informação discretamente de um conjunto limitado de *sites*, montando um agrupamento de temas interessantes.

Agentes, então, executam um serviço por qualquer reação, em resposta mudam seu ambiente ou ativam-se, procurando cumprir metas. Além do mais, agentes podem permanecer estacionários, filtrando a informação que chega ou movem-se a procura de informação específica através da Internet. Um agente na Internet segue o primeiro princípio de bom comportamento de Internet, "não causa nenhum dano". Finalmente, um elemento chave do

comportamento do agente, sua autonomia, sugere que uma vez que as metas estejam estabelecidas, seu comportamento seja guiado por suas próprias capacidades para ações, independente de seu usuário [COC98].

Usando da mobilidade oferecida pelo meio de transmissão utilizado na Internet e a tecnologia de agentes móveis, surgiram através de uma pesquisa do laboratório da IBM de Tóquio os *Aglets* (*Agents applets*) [GIL97].

5 *AGLET* API

A tecnologia usada para o desenvolvimento de *aglets* é nova e emergente. Ainda não existe muita bibliografia sobre o assunto, bem como pessoas com as quais se possa trocar experiências.

Um *aglet* é um objeto móvel em Java que pode visitar um *host* (equipamento ligado a uma rede de computadores) disponível em uma rede de computadores. Um *aglet* que executa em um *host*, pode parar sua execução de repente, pode ser despachado para um outro *host* e retomar sua execução lá. Quando um *aglet* se move leva seu código de programa, bem como seu estado (dados) [OSH97]. Conforme definição de Bret Sommers [SOM97], um *aglet* é considerado um agente móvel, porque é um objeto que possui comportamento, estado e localização.

5.1 INFRAESTRUTURA

Os tópicos abaixo são um apanhado da própria infraestrutura dos componentes que tornam capaz a transferência e comunicação entre *aglets* na Internet:

a) ***Aglet Box*** - o *aglet box* é um servidor que provê poucas facilidades para *aglets*. Seu principal propósito é o melhoramento da usabilidade. Isto é, quando *aglets* são despachados para servidores *aglets* remotos que momentaneamente não estão disponíveis, eles podem ser temporariamente armazenados numa *aglet box* privada até que eles finalmente possam ser transferidos para estes servidores, quando eles se tornarem disponíveis. O servidor *aglet box* é implementado como um *servlet* – pequenos programas em Java usados para adicionar funcionalidades ao servidor *Web* que se diferenciam dos *applet*s por não possuir interface gráfica e não rodar em um *browser* e que são uma extensão dos Java-*based Web servers* disponíveis; ou seja, um servidor *Web* baseado em Java [LAN97].

b) ***Aglet Server*** **(Tahiti)** - o a*glet server* é composto por uma combinação de um *aglet*

server standalone com um ATP (*Agent Transfer Protocol*) *daemon* (processo que fica sempre rodando no servidor), um contexto e, opcionalmente, visualizador. O contexto *aglet* roda *aglets* hospedados por este servidor. O ATP *daemon* manuseia mensagens ATP que chegam para o envio e a comunicação com os *aglets* correntemente hospedados. O *aglet server* (*Tahiti*) é um GUI para gerenciamento dos *aglets* [LAN97].

c) **HTTP Message Handling** - a*glets* podem receber mensagens HTTP diretamente. O principal cenário de uso são é o gerenciamento de *aglets* de *browsers* remotos, recebimento de requisições HTTP/CGI de um *browser* e envio de páginas HTML de volta para aquele *browser* (similar a *servlets* SSI (*Server Side Include*)) [LAN97].

d) **Fiji** - Fiji é um *plugin* mais uma biblioteca capaz de criar *applets* que rodam contextos *aglet* e podem criar, despachar (recuperar) *aglets* de (volta para) página *Web* [LAN97].

e) **HTTP Tunneling** - mensagens ATP (*Aglet Transfer Protocol*) podem ser enviadas encapsuladas no corpo de mensagens HTTP. O principal uso é encaminhando mensagens ATP (*aglets*) através de *firewalls* (barreiras de segurança) usando o servidor HTTP *proxy* (intermediário entre duas conexões) padrão [LAN97].

5.2 PRINCIPAIS CLASSES

Abaixo estão relacionadas as principais classes e interfaces da Java *Aglet* API (J-AAPI) definidas por Mitsuru Oshima [OSH98]:

a) **com.ibm.aglet.Aglet**: a classe *Aglet* é uma classe chave a J-AAPI. É uma classe abstrata que define os métodos fundamentais (por exemplo, *dispatch*(URL)) para um agente móvel controlar a sua mobilidade e ciclo de vida. Todos os agentes móveis definidos devem ser extensões desta classe abstrata *aglet*. A classe *aglet* também é usada para se ter acesso aos atributos associados ao *aglet*.

b) **com.ibm.aglet.AgletID**: cada instância de um *aglet* tem sua própria e única identidade que é imutável durante o seu ciclo de vida. Essa identidade consiste em

alguns atributos como a identificação do usuário, o host de sua origem entre outros. A classe AgletID garante a identificação única ao agente.

c) **com.ibm.aglet.AgletProxy**: a interface *AgletProxy* funciona como um manipulador de um *aglet* e provê um modo comum de outros *aglets* terem acesso a um *aglet*. Caso uma classe *aglet* tenha vários métodos públicos que não deveriam ser acessados diretamente por outros *aglets* por questões de segurança, a maneira para efetuar essa comunicação é através da interface *AgletProxy*. Para a comunicação com outro *aglet*, inicialmente deve-se obter o objeto *proxy* do primeiro *aglet* e então interagir utilizando esta interface. Assim, o objeto proxy do *aglet* age como um objeto de proteção contra outros agentes maliciosos. Quando chamado, o objeto de procuração consulta o *SecurityManager* para determinar se no contexto de execução atual é permitido executar o método, para somente então liberar o acesso. Outro papel importante da interface *AgletProxy* é proporcionar ao *aglet* transparência de localização.

d) **com.ibm.aglet.AgletContext**: um *aglet* gasta a maior parte de sua vida num contexto *aglet*. Ele "nasce", "dorme", realiza tarefas e "morre" lá. Quando em viagem pela rede, a movimentação se dá de contexto para contexto. Qualquer *aglet* pode obter uma referência para o objeto *AgletContext* pela primitiva *Aglet.getAgletContext()* e usa para obter informações como o endereço do contexto do *host* e a enumeração dos outros *aglets* do contexto para criar um *aglet* novo no contexto.

e) **com.ibm.aglet.Message:** os agentes podem se comunicar através da troca de objetos da classe Message. As mensagens contém um campo *string* (alfanumérico) para indicar o seu tipo e um segundo parâmetro no qual podem ser passados objetos como argumentos da mensagem. Por Exemplo: *Message MeuNome ("meu nome", "João").*

f) **com.ibm.aglet.Ticket** : esta classe é usada para especificar o destino e a qualidade da transferência. Em outras palavras, define o modo como o *aglet* será transferido. Ele deve incluir o destino, o protocolo a ser usado, bem como tempo de espera e requisitos de segurança. Este objeto é usado quando usa-se a *Uniform Resource Locator* (URL) como meio de especificar o destino.

g) **aglet.FutureReplay**: o objeto definido pela interface de *FutureReplay* é devolvido

pelo envio assíncrono da mensagem e usado como um manipulador para receber o último resultado assíncrono. Com esta interface, o receptor pode determinar se uma resposta está disponível e se pode esperar pelo resultado com um valor de intervalo especificado de forma que possa continuar sua execução se uma resposta não for devolvida dentro de um tempo especificado.

A *Aglet* API define a funcionalidade fundamental dos agentes móveis. A Figura 8 mostra as principais classes e interfaces na *Aglet* API e o relacionamentos entre estas classes.

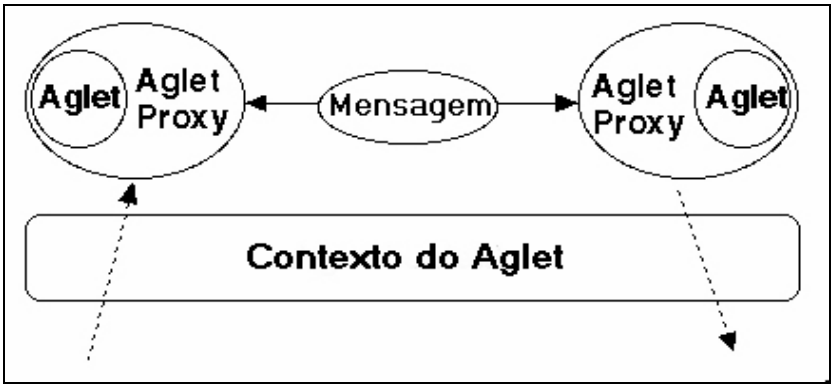

FIGURA 8 : INTERFACES E CLASSES DE UM AGLET [OSH97].

5.3 CICLO DE VIDA DE UM *AGLET*

Muitos eventos podem ocorrer durante o ciclo de vida de um *aglet*, e cada um deles é associado a um método da classe *Aglet* implementada em Java. Os principais eventos que podem ocorrer segundo Danny Lange [LAN98] são:

a) criação (*creation*) : a criação de um *Aglet* ocorre em um contexto. O novo *aglet* é associado a um identificador, inserido no contexto e inicializado. Sua execução ocorre logo após obter sucesso em sua inicialização.

b) clonagem (*cloning*) : a clonagem de um *aglet* produz uma cópia idêntica de um

aglet original no mesmo contexto. As únicas diferenças são a identidade e o fato de que o *aglet* clonado reinicia a sua execução.

c) **despacho (*dispatch*)** : despachando um *aglet* de um contexto para outro irá retirá-lo do seu contexto atual e inserí-lo em um contexto de destino onde ira reiniciar sua execução.

d) **recuperação (*retract*)** : a recuperação de um *aglet* irá retirá-lo de seu contexto e inserí-lo no contexto o qual solicitou sua recuperação.

e) **desativação (*deactivation*)** : a desativação de um *aglet* é uma capacidade que ele possui de parar sua execução temporariamente e armazenar seu estado em um local secundário.

f) **ativação (*activation*)** : a ativação de um *aglet* restaura a execução de um *aglet* que foi desativado em um contexto.

g) **desalocação (*dispose*)** : a desalocação de um *aglet* irá encerrar sua execução e retirá-lo de seu contexto.

h) **envio de Mensagens (*messaging*)** : o envio de mensagens entre *aglets* involve o envio (*sending*), o recebimento (*receiving*) e o tratamento das mensagens (*handling*), sejam elas síncronas ou assíncronas.

A Figura 9 representa o ciclo de vida dos *aglets*.

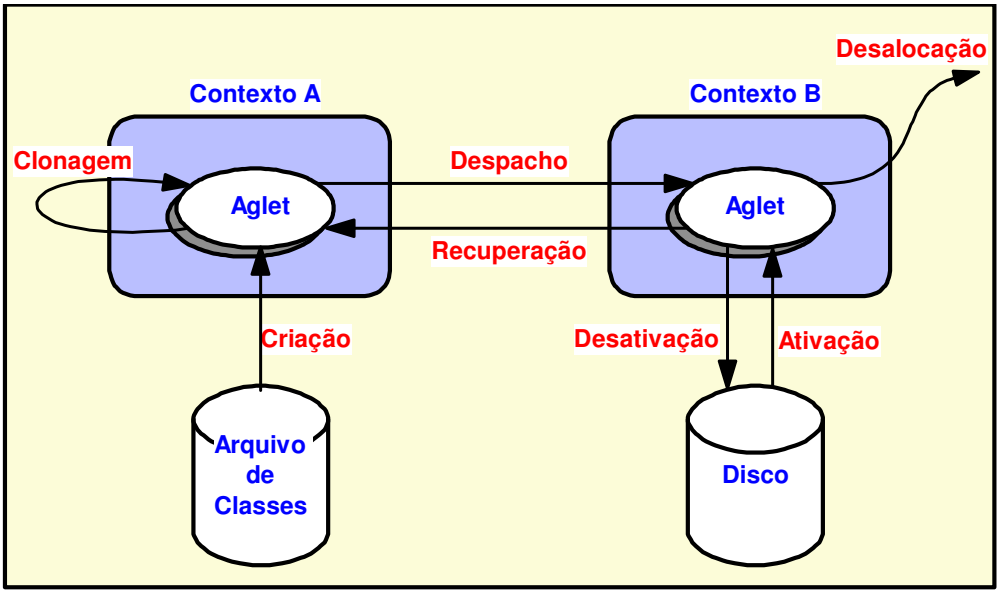

FIGURA 9 : CICLO DE VIDA DOS *AGLETS*

5.3.1 CRIANDO UM *AGLET*

Para se fazer alguma coisa com um *aglet*, obviamente primeiro tem-se que criá-lo. Existem essencialmente dois modos de se criar um *aglet*, conforme demonstrado na Figura 9, instanciá-lo de uma classe de *aglet* ou clonando-o de um *aglet* existente.

O *aglet* é criado em um contexto, no qual passará a maior parte de sua vida. Quando move-se de uma máquina a outra, move-se de contexto para contexto. Pode-se considerar um contexto como um ambiente uniforme para o *aglet*. Não importa se o *aglet* está executando em um PC ou em uma estação de trabalho Unix, pois estará garantido o ambiente do *aglet*. Para se instanciar um novo *aglet*, necessita-se ter acesso ao contexto atual. Então, pode-se fazer isso pela chamada do método *getAgletContext()*, na própria interface.

Durante o processo de criação três métodos são chamados:

a) protected Aglet (): Cria um *aglet* não inicializado. Este método é chamado uma vez na vida de um *aglet*. Como regra, não se pode sobrescrever esta criação. Logo, pode-se sobrescrever a onCreation(), para inicializar um *aglet* após sua criação.
b) public void onCreation (): Inicializa o *aglet* novo. Este método só é chamado uma vez no ciclo de vida de um *aglet*. Sobrescreve-se este método para customizar a inicialização do *aglet*.
c) public void run (): Este método é invocado depois de uma criação com sucesso, despacho, recuperação ou ativação do *aglet*.

Estes métodos são invocados automaticamente pelo contexto durante a criação do *aglet*. Dois deles, *onCreation()* e *run()*, provêem um modo elegante de customização na criação de um *aglet* [LAN97].

5.3.2 CLONANDO UM *AGLET*

Um modo alternativo de criar novos *aglets* é a clonagem. Se já se tem um *aglet* em seu contexto, pode-se criar um gêmeo usando o método clone (). Uma chamada bem sucedida do método *onClone()* em um determinado *aglet* criará uma cópia idêntica deste, dentro do contexto corrente do *aglet*. Neste método não é envolvido o clone diretamente, para proteger seus métodos públicos de acesso direto, mas em seu lugar retorna o *AgletProxy*, que é o manipulador para o *aglet* [LAN97].

5.3.3 DESPACHANDO UM *AGLET*

Para que um *aglet* possa visitar um *host* remoto, é necessário o método *dispatch*() e a URL da máquina remota. Assim como o método *dispose()*, o método *dispatch()* não pode ser anulado; ou seja, quando se executar o *dispose*, o *aglet* morre e quando se executar o *dispatch* o *aglet* vai para outro *host*.

A URL é um modo padrão de especificação de um servidor remoto na rede. Além do *host* e nome do domínio, a URL inclui informações a respeito do protocolo particular usado para a comunicação com o servidor remoto. É usado o protocolo ATP como um protocolo de rede para despachar *aglets*.

O método *dispatch*() não força a especificação do protocolo. A escolha de protocolo depende inteiramente do conjunto de protocolos suportados por uma implementação particular do J-AAPI que se está usando e dos protocolos suportados pelo servidor remoto [LAN97].

5.3.4 RECUPERANDO UM *AGLET*

Às vezes pode-se não querer esperar por um retorno do *aglet*, mas que isso possa ser feito de uma maneira assíncrona. O método *retractAglet* no contexto do *aglet*, permite fazer isso. O método pega como argumento uma URL que especifica o *host* remoto e especifica o *aglet* que vai ser retornado.

Para executar a solicitação de retorno, é necessário a especificação da URL em que está localizado e a identificação do *aglet*. Para todo *aglet* existe um identificador único, tal que todo *aglet* dentro de uma rede pode ser acessado exclusivamente combinando seu identificador com seu URL no *host*. Como segue a sintaxe: *atp://algum.host.com#indentificação_do_aglet*.

Uma tentativa para retornar um *aglet* distante conduzirá à execução do método *onReverting()*. Esta chamada de método pode ser usada como uma advertência que alguém está tentando retornar o *aglet.* Sobrescrever este método permite que o *aglet* se prepare para o retorno. O método também pode ser usado para prevenir o retorno [LAN97].

5.3.5 DESATIVANDO O *AGLET*

O *aglet* permite armazenamento temporário em um meio secundário. Para desativar o *aglet* é necessário solicitar que armazene seu estado atual e "vá dormir". Quando o *aglet* retorna para seu contexto, diz-se que ele foi ativado.

Ao ser desativado, o *aglet* necessita saber o tempo que deve permanecer desativado. O período especificado é apenas uma sugestão para o sistema. Quando o *aglet* é ativado de fato depende muito da implementação atual que está sendo usada e está a cargo do *host*.

Quando o método *deactive*() é chamado, invocará o método *onDeactivating*() imediatamente. Usualmente este método é sobrescrito para permitir ao *aglet* terminar sua tarefa atual antes de "ir dormir". A duração do sono do *aglet* é indicado em milisegundos através do argumento *duration*. O *aglet* será ativado pelo contexto depois que o período especificado decorreu [LAN97].

5.3.6 DESALOCANDO UM *AGLET*

O *aglet* aloca vários recursos enquanto está em um contexto de *aglets* e depois de cumprir sua tarefa ele libera estes recursos. Um *aglet*, na maioria das vezes, executa suas tarefas em um servidor distante como convidado, assim, é obrigado a minimizar o consumo dos recursos do *host*. O método para desalocar um *aglet* é o *dispose()*.

O sistema eliminará todas as linhas que pertencem ao *aglet* que está executando em caso de desalocação. Fazendo isso, eliminará todas as referências entre o contexto e o *aglet* a ser desalocado. Contudo, não há garantia de que a memória alocada pelo *aglet* será desalocada imediatamente. O *Java Garbage Collector* limpará o *aglet*, quando todas as referências tiverem sido eliminadas, incluindo referências de outros *aglets*.

Chamando a função *dispose()*, ela conduzirá imediatamente a chamada do método *onDisposing()*, que pode ser usado para customizar o processo de desalocação. Pode-se sobrescrever este método com algumas ações que precedem a desalocação atual do *aglet*. Por exemplo, pode-se usar o método para permitir o *aglet* preparar sua própria desalocação, pela associação do fechamento de arquivos e janelas [LAN97].

5.4 EXEMPLO DE AGLET

O Quadro 3 apresenta um *Aglet* que mostra a mensagem “Alo Mundo” e o nome da sua classe.

```
/* Exemplo de aglet
   Autor Mauri Ferrandin*/
package examples.mauri;
import com.ibm.aglet.*;
public class AgletSimples extends Aglet {
  public void run() {
    System.out.println("Alo mundo! Eu sou " + getClass().getName()+ ".");
  }
}
```

QUADRO 3 : EXEMPLO DE AGLET

5.5 MENSAGENS

Um *aglet* pode comunicar-se com outro através do envio e recebimento de mensagens, para isto, o *aglet* que quiser se comunicar precisa instanciar um novo objeto do tipo mensagem que na J-AAPI está definida em com.ibm.aglet.Message e enviá-la ao outro *aglet* com o qual deseja estabelecer comunicação. Um objeto da classe mensagem tem como seu argumento, um atributo determinando o tipo da mensagem e um objeto arbitrário que é passado como argumento. O *aglet* receptor pode tratar a mensagem através do método *Aglet.handleMessage()* conforme mostra o Quadro 4.

```
/* Exemplo de tratamento de mensagem em aglets */

class ExemploMensagem extends Aglet {
        public boolean handleMessage(Message msg) {
            if (msg.sameKind("FaçaAlgumaCoisa")) {
               FaçaAlgumaCoisa();
            } else if (msg.sameKind("NaoFaçaNada")) {
               NaoFaçaNada ();
            }
         }
    }
```

QUADRO 4 : EXEMPLO DE TRATAMENTO DE MENSAGEM

5.5.1 TIPOS DE MENSAGENS

A J-AAPI possue os seguintes tipos de mensagens : *Now-type*, *Future-type* e *Oneway-type*.

5.5.1.1 MENSAGEM *NOW-TYPE*

As mensagens *now-type* são síncronas e bloqueiam o *aglet* emissor até que o recipiente complete o tratamento da mensagem.

Para enviar uma mensagem *Now-type* é invocado o método AgletProxy.sendMessage(Message msg), conforme exemplo no Quadro 5 :

```
/* Exemplo de mensagem now-type */

String resposta = proxy.sendMessage(new Message("pergunta"));
        System.out.println(resposta);
```

QUADRO 5 : MENSAGEM NOW-TYPE

5.5.1.2 MENSAGEM *FUTURE-TYPE*

As mensagens *future-type* são assíncronas e não bloqueiam a execução do *aglet* emissor. O método de envio retorna um objeto do tipo *FutureReply* que pode ser usado para obter as respostas ou esperar para mais tarde.

Para enviar uma mensagem *future-type* é invocado o método *AgletProxy.sendAsyncMessage(Message msg)*, conforme o exemplo no Quadro 6.

```
/* Exemplo de mensagem future-type */
FutureReply resposta = proxy.sendAsyncMessage(new Message("question"));
 int tarefas = 10;
 while(resposta.isAvailable() == false && tarefas-- >0) {
     ExecuteOutrasRotinas();
  }
  System.out.println( (String)resposta.getReply() );
```

QUADRO 6 : MENSAGEM DO TIPO FUTURE-TYPE

5.5.1.3 MENSAGEM *ONEWAY-TYPE*

As mensagens *oneway-type* são assíncronas e não bloqueiam a execução corrente no *aglet* emissor. O que a difere das mensagens *Future-type* é que o *aglet* receptor não precisa enviar resposta.

Para enviar uma mensagem *Oneway-type* é invocado o método *AgletProxy.sendOnewayMessage(Message msg)*, conforme o exemplo no Quadro 7:

```
/* Exemplo de mensagem oneway-type */

    Msg = new Message("pergunta")
    AgletProxy.sendOnewayMessage(Msg).
```

QUADRO 7 : MENSAGEM ONEWAY-TYPE

5.5.2 MENSAGENS REMOTAS

Os *Aglets* suportam passagem de mensagens remotas, e os objetos da classe *aglet* podem se comunicar remotamente com a mesma eficiência que em comunicações locais.

O parâmetro do objeto mensagem a ser enviado remotamente pode ser qualquer tipo que implemente o método Java.io.Serializable.

O envio de mensagens remotas é diferente do despacho de um *aglet*, pois uma mensagem remota não causa transferência de *bytecode*, uma vez que as classes usadas para transmitir as mensagens estejam instaladas em ambos os servidores.

Mensagens remotas podem ser largamente usadas para comunicação entre *aglets* que residem em servidores diferentes, eles podem reduzir o tráfego na rede além do custo para definir novas classes para serem despachadas entre os servidores.

5.5.3 EXEMPLO DE MENSAGEM

O exemplo abaixo é composto de um *aglet* pai – descrito no Quadro 8, que criará um *aglet* filho - descrito no Quadro 9, no mesmo *host* e armazenará o proxy do filho na variável proxy através da qual o despachará este mesmo *aglet* filho para outro *host* e enviará a ele a mensagem do tipo “adição” com o parâmetro “30”.

```
/* Aglet Pai */
package examples.mauri.testes02;
import com.ibm.aglet.*;
import Java.net.*;
import com.ibm.aglet.Message.*;
class  Pai extends Aglet {
   public void run()    {
     try  {
          System.out.println("Aglet Pai criando aglet Filho e pegando o proxy");
            AgletProxy proxy;
            Proxy=getAgletContext().createAglet(this.getcodeBase(),"examples.testes02.Filho",
          null);
           // Dispachando o aglet para outro server
            proxy.dispatch(new URL(“host2.com.br:434”);
           Message msg = new Message("adição",30);
            System.out.println("Aglet Master enviando mensagem para o slave");
            Integer retorno = (Integer)proxy.sendMessage(msg);
            System.out.println("Aglet Slave retornou "+retorno);
     }   catch(Exception e)   {
             System.out.println(e.getMessage());
     }
  }
  public void onCreate( ) {
     System.out.println("Aglet Master criado");
  }
}
```

QUADRO 8 : EXEMPLO DE MENSAGEM - *AGLET* PAI

O *aglet* filho após criado recebe a mensagem do pai, verifica o tipo da mensagem e executa a adicao, adicionando 10 ao valor recebido e envia a soma total como resposta a mensagem enviada pelo *aglet* pai.

```
/*Aglet Filho */
package examples.testes02;
import com.ibm.aglet.Aglet;
import ava.net.*;
import com.ibm.aglet.Message;
public class  Filho extends Aglet {
    public void onCreate(){
          System.out.println("Aglet filho criado");
    }
   public boolean handleMessage(Message msg)  {
          int k = 0;
          try{
            if (msg.sameKind("adicionar")) {
               k = ((Integer)msg.getArg()).intValue();
               System.out.println("O valor recebido "+k);
               k = k + 10;
              msg.sendReply(k);
              System.out.println("A resposta foi enviada !");
           }
       }catch(Exception e)   {
            System.out.println(e.getMessage());
        }
       return true;
  }
}
```

QUADRO 9 : EXEMPLO DE MENSAGEM – *AGLET* FILHO

5.6 O PROTOCOLO ATP

O *Agent Transfer Protocol* (ATP) é um protocolo usado em sistemas distribuídos baseados em agentes para transportar os agentes móveis entre os computadores da rede. Orientado para a Internet, o ATP oferece uma plataforma simples e independente para transferência de agentes entre plataformas ao longo da rede. O ATP pode ser usado para transporte de agentes programados em qualquer ambiente não sendo restrito aos *aglets*, tratando assim a mobilidade dos agentes de maneira genérica e uniforme.

A Figura 10 mostra um exemplo de dois ambientes de agentes em um mesmo computador usando o protocolo ATP para acessar a rede como meio de comunicação e deslocamento dos agentes.

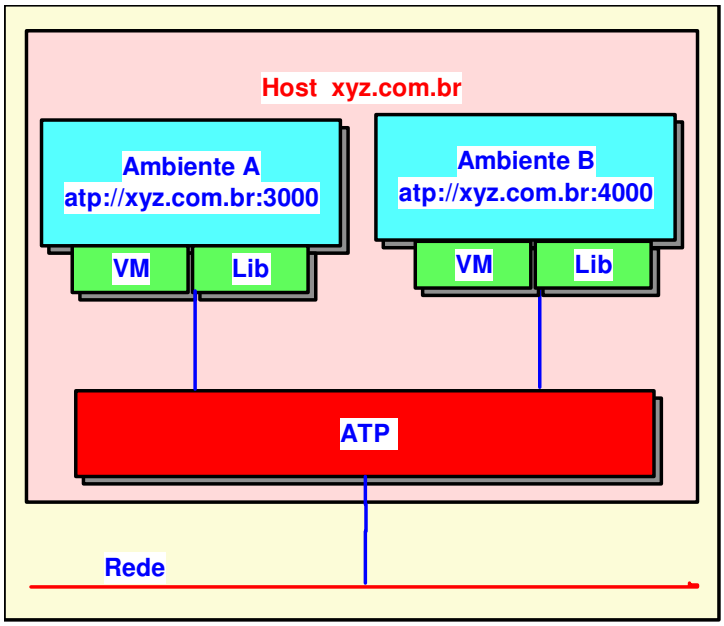

FIGURA 10 : USO DO ATP COM AMBIENTES DIFERENTES

6 FIJI *AGLET* API

O Fiji é um complemento do *aglet*. Contudo, será estudado neste capítulo o roteador Fiji, sua arquitetura, bem como o tema segurança.

6.1 INTRODUÇÃO

Fiji é um *plugin* mais uma biblioteca capaz de criar *applets* que rodam contextos *aglet* e podem criar, despachar (recuperar) *aglets* de (volta para) página *Web*. A biblioteca de *aglets* Fiji é uma biblioteca de classes AWB (*Aglet Workbench*) selecionadas que é uma extensão de um *browser Web,* semelhante a um *plugin*. Já o Kit Fiji é uma biblioteca de classes para desenvolvimento de *applets* Fiji, e um módulo roteador, que deve ser instalado no mesmo *host* no qual esta sendo usado servidor *Web* [LAN97].

6.2 ROTEADOR

Observe na Figura 11 que a política de segurança do *applet* permite *applets* estabilizarem uma comunicação remota (por exemplo, *sockets*) somente com o *host* de onde vieram. Assim, para que *applets* se transfiram e comuniquem-se em diferentes servidores *aglets* remotos, (por exemplo, enviando mensagens ATP remotas), num caminho transparente, o Kit Fiji provê um *proxy* (chamado de roteador) que roda em código *applets* e encaminha todas as mensagens ATP originadas por um *applet* para seu destino.

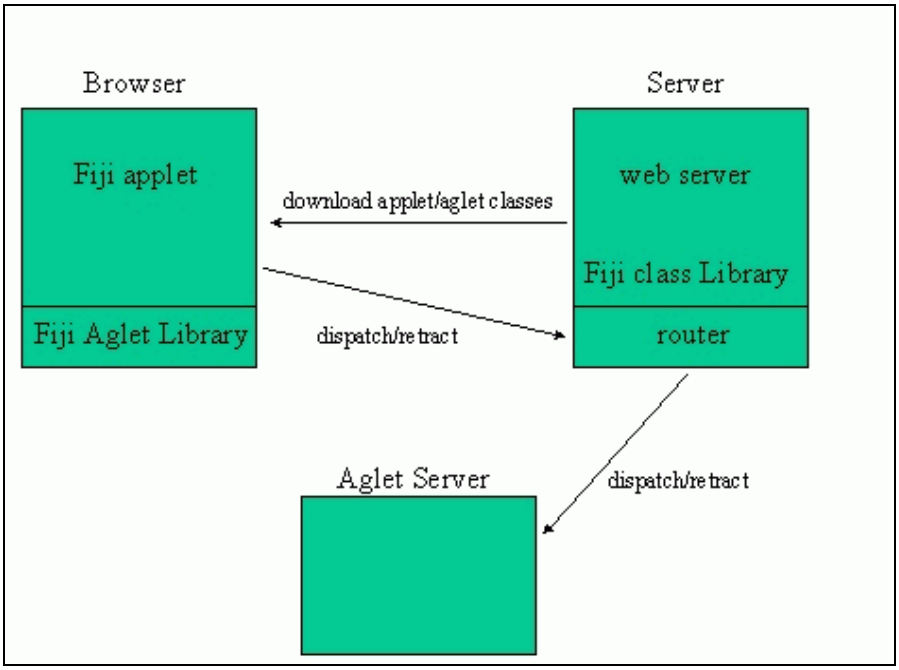

FIGURA 11 : ARQUITETURA DO FIJI

6.3 A ARQUITETURA DO ROTEADOR

Em vez de direcionar mensagens ATP recebidas pelo roteador, usa-se técnicas de HTTP *Tunneling* que encapsulam mensagens ATP dentro de mensagens HTTP como mensagens introduzidas no roteador. O roteador recebe mensagens HTTP (contendo mensagens ATP) e as envia para seu destino final (especificado na URL absoluto da linha de requisição de HTTP). Usando as técnicas de HTTP *Tunneling* pode-se usar qualquer servidor (comercial) de *proxy* disponível (rodando numa base de códigos), como roteador e usar um proxy HTTP para atravessar *firewalls* (não como o roteador que pode basicamente encaminhar mensagens) e conectando o roteador a componentes baseados em HTTP padrão de uma infraestrutura de rede (por exemplo, conectando o roteador dentro de um ambiente Intranet e um *proxy* HTTP para conectar fora com um ambiente Internet). Consequentemente, usando o roteador, o destino final de mensagens ATP (por exemplo, Tahiti) capacita o HTTP a aceitar mensagens HTTP, contendo mensagens ATP [LAN97].

6.4 MODELO DE SEGURANÇA PARA AGLETS EM *BROWSER*

Nos tópicos seguintes são descritos dois modelos de segurança para *aglets* em *browsers*: política de segurança do *aglet* e concessão de privilégios.

6.4.1 POLÍTICA DE SEGURANÇA DO *AGLET*

Em *applets* Fiji, a política de segurança de um *aglet* é uma extensão de um *applet*. Assim, uma operação que é inicializada por um *aglet* está automaticamente checada pelo gerenciador de segurança do *applet*. Se permitido, é também checado por um gerenciador de segurança separado do *aglet* (não referenciado da classe Java.*lang.System*) que implementa uma política básica de segurança padrão para *aglets* maliciosos [LAN97].

6.4.2 CONCESSÃO DE PRIVILÉGIOS

Para administrar *aglets* e usar o protocolo ATP pela biblioteca de classes *aglet*, é necessário permitir que *applets* possam:

a) manipular *threads* e grupos de *threads* (outros grupos de *threads* do *applet*);
b) modificar a propriedade do sistema Java.*handler.protocol.pkgs* para incluir o pacote manipulador do protocolo ATP.

Para que estas operações sejam executadas é necessário que o usuário conceda privilégios para tais operações. A concessão destes privilégios é questionada na execução do *applet.*

Na prática, quando um *applet* Fiji é carregado, um painel de segurança do Java é exibido uma vez para cada privilégio necessário, num total de 4 privilégios necessários. É necessário que o usuário conceda estes privilégios. Se qualquer um dos privilégios requeridos é negado, a execução do *applet* é imediatamente interrompida. Atualmente, usuários só podem conceder privilégios baseado na confiança do *site* da IBM, de onde o Fiji é adquirido [LAN97].

6.5 RESTRIÇÕES DA PROGRAMAÇÃO *AGLET*

Os *aglets* devem possuir exatamente a mesma base de códigos que os *applets* Fiji criados. Na prática, usar o *createAglet ()* com uma base de códigos diferente que o *applet* Fiji, levantará uma *AgletException* com a mensagem "*invalid codebase*" (base de códigos inválida).

Os *aglets* não podem ser despachados para páginas *Web* (rodando em *applets* Fiji). Só *aglets* cuja base de códigos é a mesma que um *applet* Fiji podem ser transferidos para seu contexto *aglet* por operações de recuperação (*retract*).

6.6 EXEMPLO BÁSICO DE UM APPLET FIJI

O Quadro 10 mostra o exemplo de um *applet* Fiji simples que usa componentes gráficos do Java para que o agente possa ser executado no *browser*, assim, toda a funcionalidade dos *aglets* pode ser incorporada a um *applet*, o qual será descendente da classe com.ibm.fiji.lib.FijApplet que está declarada na cláusula *import* usada para carregar outras classes em um programa Java.

A Figura 12 mostra a execução do exemplo de *Aglet* Fiji através do *browser* Netscape, no mesmo uma vez clicado o botão “Teste” ele mostrará a mensagem “voce clicou no botão Teste” e também mostrará o nome de sua classe.

```
/* Exemplo de applet Fiji
* Autor Mauri Ferrandin
*/
package com.ibm.fiji.mauri.exemplos;
import com.ibm.fiji.lib.FijiApplet;
import com.ibm.aglet.*;
import java.awt.*;

public class SampleAgletFiji extends FijiApplet {
  TextArea saida;
  Button botao;

  public void init()  {
     super.init();
     this.setLayout (new BorderLayout ());
     saida = new TextArea (10, 180);
     botao =  new Button("Teste");
     this.add("North", saida);
     this.add("South", botao);
   }

  public boolean action (Event ev, Object arg)  {
    if (ev.target == botao)    {
       saida.append("voce clicou no botao teste !!! \n");
       saida.appendText("Alo mundo! Eu sou " + getClass().getName()+ ".\n");
     }
    return true;
  }
}
```

QUADRO 10 : EXEMPLO DE *APPLET* FIJI

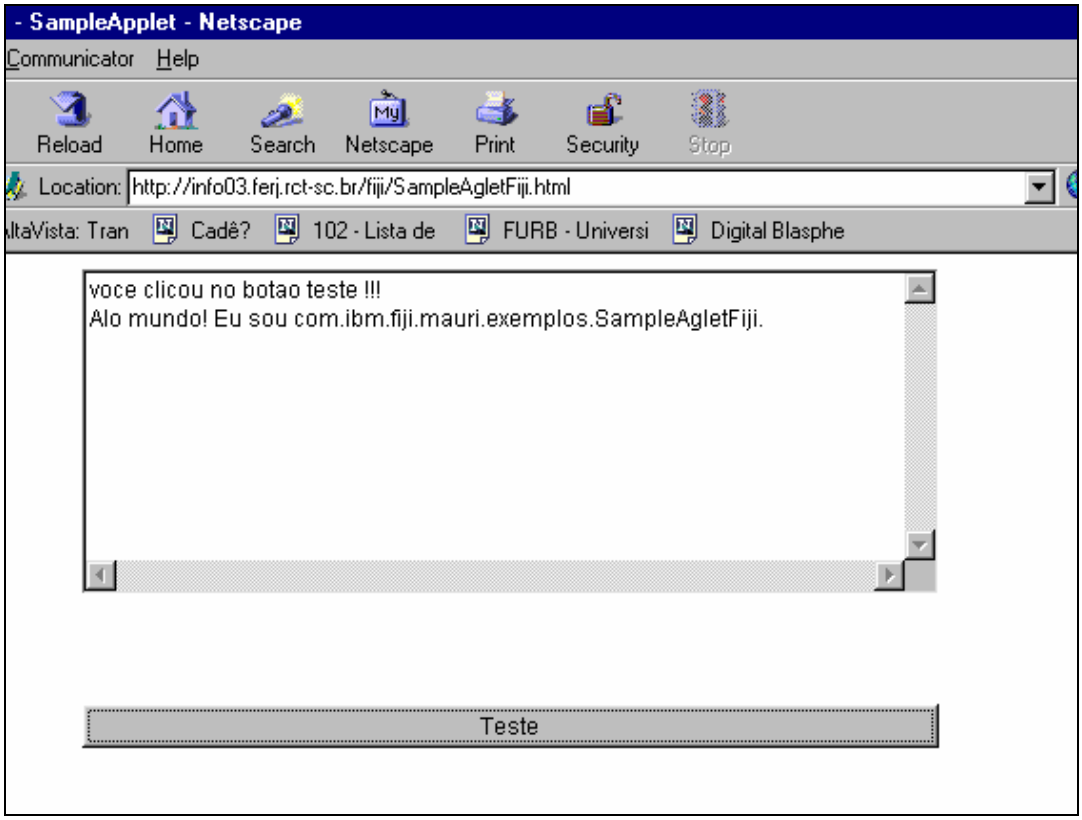

FIGURA 12 : EXECUÇÃO DO EXEMPLO DE *AGLET* FIJI

6.7 LIMITAÇÕES DE USO

A tecnologia Fiji bem como a tecnologia aglet possuem algumas limitações de uso :

- são tecnologias proprietárias da IBM, e para serem usadas é necessária a aquisição das licensas para uso das mesmas;
- seu funcionamento é restrito de acordo com o *browser* que será usado para carregar/executar as aplicações;
- o *plug-in* durante a realização deste trabalho deixou de ser distribuído e a última versão liberada pela IBM expirou, assim foi necessário o uso da última versão liberada atrasando a data do sistema local. A IBM já se comprometeu a disponibilizar uma nova versão do *plug-in* tão logo ela esteja estável.

7 DESENVOLVIMENTO DO TRABALHO

Este trabalho é uma continuação do TCC (Trabalho de Conclusão de Curso) Desenvolvimento de aplicações para Internet através de agentes com acesso a bancos de dados de Roberto Carlos Dariva [DAR97].

Para o seu desenvolvimento será utilizada a metodologia da prototipação.

Segundo Melendez [MEL90], uma metodologia sempre será utilizada no desenvolvimento de sistemas de informação, e a prototipação representa hoje uma boa solução para a maioria dos problemas desta área.

Existem várias ramificações da metodologia da prototipação dentre as quais pode-se citar as mais importantes que são a metodologia da prototipação descartável e a metodologia da prototipação fundamental, sendo que a grande diferença entre as duas está no fato que na descartável o protótipo será descartado após a aprovação do sistema, ao passo que na fundamental o protótipo, após a aprovação do sistema, passará por refinamentos até se tornar o sistema final.

Para a especificação do sistema proposto neste trabalho será usada a metodologia da prototipação fundamental que é composta de oito fases conforme Figura 13, sendo que as fases 5,6,7 e 8 não serão implementadas.

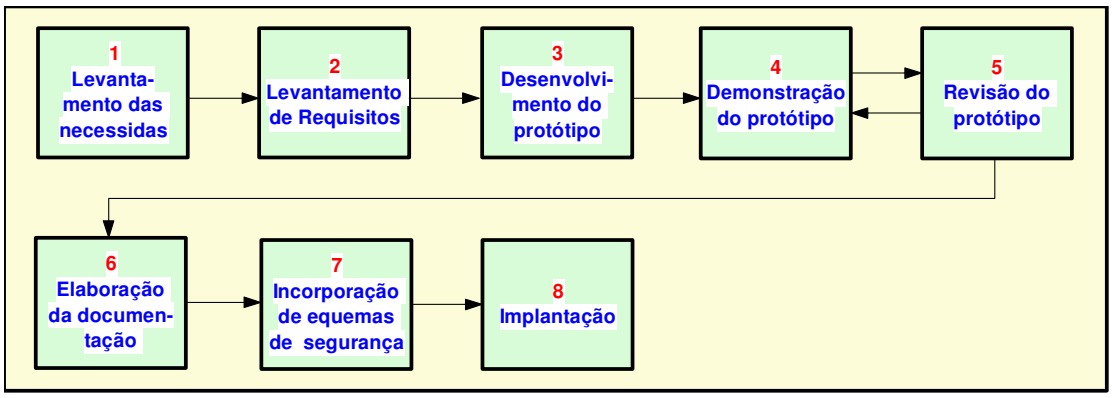

FIGURA 13 : METODOLOGIA DA PROTOTIPAÇÃO

7.1 LEVANTAMENTO DAS NECESSIDADES

O processo de pré-matrícula ou reserva de vaga como é conhecido em outras instituições de ensino superior sempre foi um processo capaz de acabar com a paciência de qualquer um, principalmente quando o aluno, por cultura ou falta de interesse, acaba deixando para fazê-lo na última hora.

Na FERJ este processo é bastante burocrático, e o aluno muitas vezes precisa ir a até três setores diferentes : biblioteca, secretaria e tesouraria para concluí-lo.

Este trabalho propõe a disponibilização deste serviço através da Internet, o que irá facilitar as coisas para os alunos, uma vez que quem está associado a um provedor comercial poderá fazê-lo de sua própria casa ou empresa, e quem não possui nenhuma outra maneira de acessar a rede pode dirigir-se ao *campus* da instituição e fazê-lo através dos laboratórios, e em última instância, o aluno poderia recorrer aos procedimentos normais de execução deste processo.

7.1.1 OUTRAS INSTITUIÇÕES

As intituições educacionais estão cadas vez mais utilizando a Internet como um meio para divulgação de informações para seus alunos e também para possibilitar ao aluno a realização de certos procedimentos que podem ser realizados através da rede e que agilizam os processos internos e aumentam a qualidade no atendimento.

Existem vários exemplos do uso da Internet para divulgar informações acadêmicas, dentre elas vale destacar a própria FERJ, que disponibiliza as notas semestrais dos alunos, a Universidade Regional de Blumenau (FURB), que possue inúmeros serviços dentre os quais merecem destaque a consulta de notas, situação financeira, e a reserva de vaga (equivalente a pré-matrícula e outras instituições), a Universidade para Desenvolvimento do Alto Vale do Itajaí (UNIDAVI), que disponibiliza serviços de consulta de notas, solicitação de declarações,

e muitos colégios, dentre os quais merece destaque o Colégio Catarinense que diponibiliza os boletins dos alunos.

As técnicas e ferramentas utilizadas para disponibilizar os serviços são as mais variadas, sendo que os meios mais usados são os *scripts* CGI e *applets* escritos em Java. Nota-se aqui uma tendência que parte principalmente das intituições de ensino superior a cada vez mais disponibilizar novos serviços para facilitar e aumentar a qualidade no atendimento e no ensino.

7.2 LEVANTAMENTO DE REQUISITOS

Na identificação de requisitos será esboçado o objetivo do sistema, o Diagrama de Fluxo de Dados (DFD) dos níveis 0 e 1, lista de eventos e respostas do sistema e o Modelo Entidade Relacionamento (MER).

O objetivo deste sistema é permitir aos alunos a realização da pré-matrícula via Internet.

A Figura 14 e a Figura 15 representam o DFD de nível 0 e 1 respectivamente.

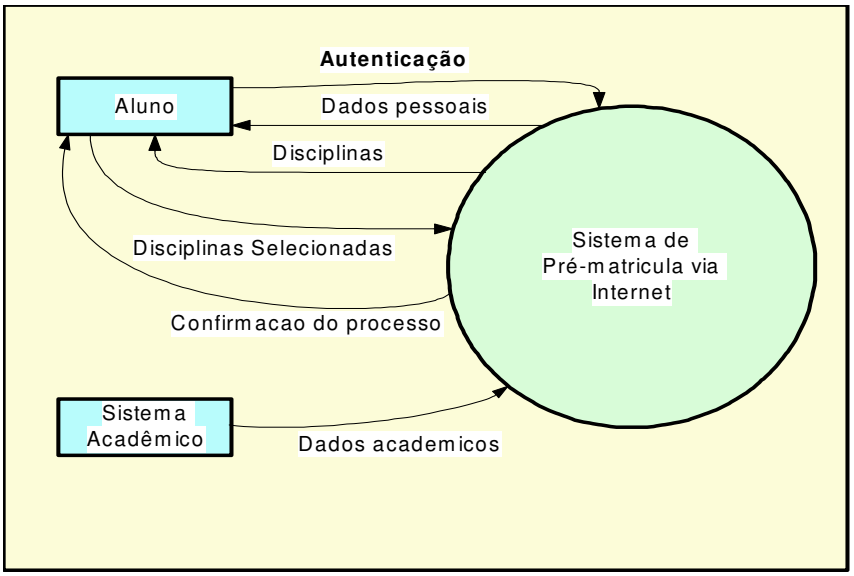

FIGURA 14 : DFD NÍVEL 0

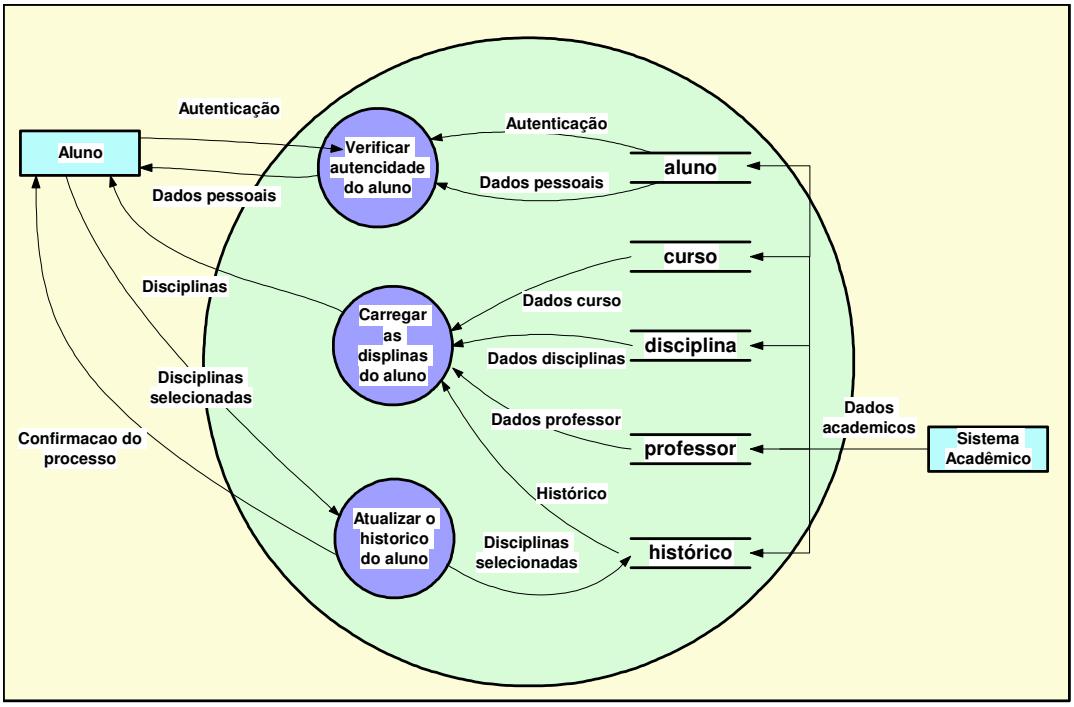

FIGURA 15 : DFD NÍVEL 1

Os principais processos do sistema são :

a) **Verificar autenticidade do aluno.** Neste processo, que é disparado pelo aluno, o sistema verifica a autenticidade do aluno através de seu código de matrícula e da sua senha que já estão previamente cadastrados no sistema acadêmico;
b) **Carregar as disciplinas do aluno.** Este processo é disparado automaticamente após o usuário ser considerado válido no processo de validação;
c) **Atualizar o histórico do aluno.** Este processo é disparado pelo aluno após este selecionar as disciplinas que o mesmo pretende cursar no semestre;

A Figura 16 apresenta o MER do sistema. É importante ressaltar que este modelo já está implantado pois o mesmo faz parte do sistema de controle academico da FERJ que utiliza o Oracle versão 8.0 como SGBD, e que no na mesmo estão descritos apenas os dados principais que serão utilizados na confeccão deste protótipo.

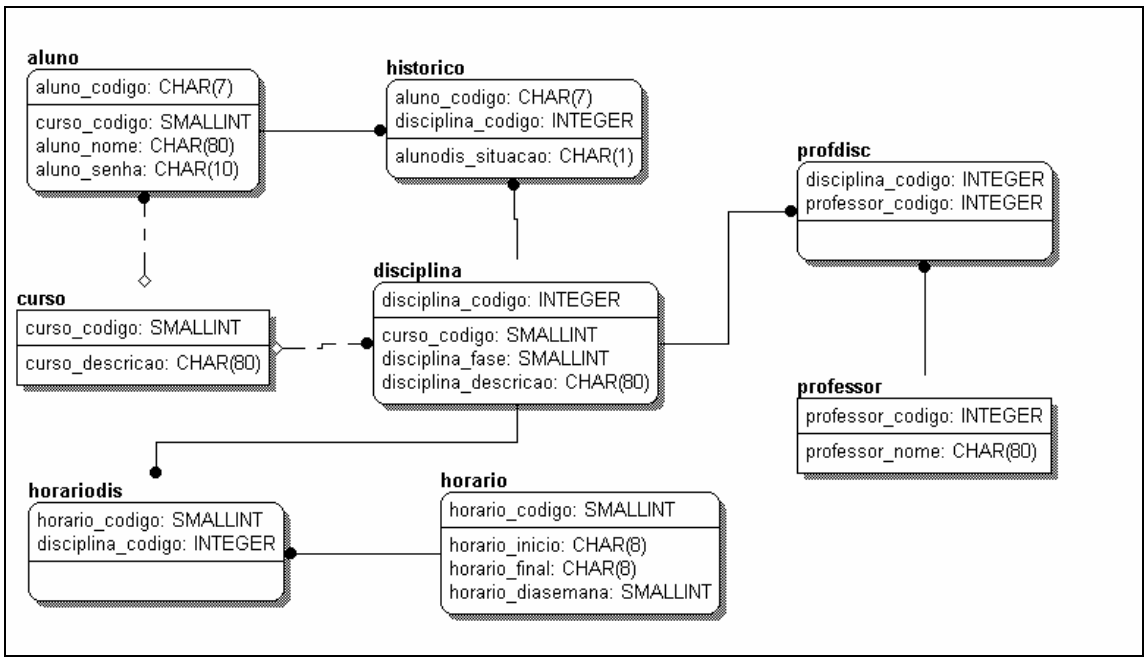

FIGURA 16 : MODELO ENTIDADE RELACIONAMENTO

O Quadro 11 apresenta o dicionário de dados do sistema, o qual só representa os dados principais do sistema.

Autenticação = CódigoAluno + SenhaAluno;
Dados Pessoais = CódigoAluno + Nome + CódigoCurso + DescriçãoCurso;
Disciplinas = {CódigoDisc + Status + Descrição + Fase + {Horário}};
Horário = CódigoHorário + DiaSemana + HorarioInicio + HorarioFim;
Disc. Selecionadas = {CódigoDisc + Status};
Confirmação Processo = Disciplinas + Ok;

QUADRO 11 : DICIONÁRIO DE DADOS

7.3 DESENVOLVIMENTO DO PROTÓTIPO

Neste tópico será demostrado o desenvolvimento do protótipo, desde a instalação do software necessário, bem como o projeto, implementação e a execução do sistema.

7.3.1 INSTALAÇÃO DO SOFTWARE NECESSÁRIO

Os softwares necessários para a instalação e funcionamento do sistema são:

a) Java : para instalar a linguagem Java basta instalar o JDK que é composto pelo compilador Java, o interpretador, bibliotecas de classes, e documentação, sendo que o mesmo é gratuitamente distribuído em vários sites na Internet. Para a implementação será utilizado o pacote JDK versão 1.1.6 que já vem com programa de auto instalação no caso da plataforma Windows 95. Após a instalação deve-se atentar quanto às variáveis de ambiente como PATH e CLASSPATH. Este último serve para indicar onde se encontram as classes que podem ser carregas nos programas. Exemplo "SET CLASSPATH=C:\JDK1.1.6\LIB\CLASSES.ZIP". Este comando adicionado aos arquivos de inicialização do sistema faz com que toda vez que o compilador ou a máquina virtual Java for usado, os mesmos procurem pelas classes que necessitarem no arquivo CLASSES.ZIP, que contém um conjunto de classes zipadas.

b) Netscape : para a instalação do *browser* Netscape basta o usuário executar o assistente e seguir as instruções. A versão 4.6 será utilizada sendo que a mesma já vem com suporte a linguagem Java.

c) *Aglets* Software Development Kit (ASDK) : o pacote ASDK possui um programa de instalação que é executado através do interpretador Java no Windows. No Unix ele usa um *shell script* para a instalação. Deve-se dedicar uma atenção especial para as variáveis ambientes utilizadas como PATH e CLASSPATH que são fundamentais para o funcionamento do software. O ASDK é composto pelo Tahiti que consiste em uma aplicação Java que funciona como um servidor de *aglets*, e

uma biblioteca de classes usadas para se criar os *aglets* que serão executados através do Tahiti. A Figura 17 mostra a interface do servidor de *aglets* Tahiti.

FIGURA 17 : TAHITI

d) Fiji Library, *plug-in* e *router* : o Fiji *Library* possui uma biblioteca de classes que serão usadas para a criação de *applet*s Fiji. Bem como o *router* Fiji que permitirá a comunicação entre os *applet*s Fiji e os *aglets* rodando em servidores de *aglets*. A instalação é bastante facilitada pelo arquivo de instalação.

7.3.2 FUNCIONAMENTO DOS AGENTES

Este tópico trata do funcionamento dos agentes implementados, mostrando suas ações e comportamento ao longo de sua execução.

O protótipo é composto por dois agentes:

a) **Agente Interface** : disponibiliza uma interface gráfica para o usuário que possibilitando-lhe uma interface amigável rodando sobre um *browser.* Para obter os dados que serão recuperados de um SGBD remoto ele comunica-se com o agente SGBD. Para sua implementação serão criadas classes com componentes para auxiliar o desenvolvimento da interface gráfica, como a classe DlgMensagem que *server* para mostrar um caixa de diálogo com mensagens para o usuário, a classe

Disciplina que mostrará os dados de uma determinada disciplina e a classe Cabeçalho que será usada para colocar os títulos nos campos de cada disciplina.

b) **Agente SGBD** : é o agente móvel que é criado pelo agente interface e tem como objetivos, se deslocar ate o servidor de banco de dados para executar transações e enviar o resultado das mesmas através de mensagens remotas ao agente interface.

Ambos os agentes serão implementados na linguagem Java. No quadro 12 é demonstrado como os agentes procederão ao longo do processo, comunicando-se entre eles e interagindo com o usuário.

Quem ?	O que ?
Usuário	Descarrega página HTML contendo a chamada para o agente de interface.
Agente Interface	Inicializa componentes gráficos de interface com o usuário e solicita a aprovação dos certificados.
Usuário	Informa código de matrícula e senha.
Agente Interface	Cria o agente SGBD.
Agente SGBD	É inicializado.
Agente Interface	Despacha agente SGBD para a máquina onde se encontra o SGBD.
Agente SGBD	É deslocado para a máquina que contém o SGBD.
Agente Interface	Envia mensagem para o agente SGBD com o código e a senha do usuário.
Agente SGBD	Verifica se o código e a senha do usuário são válidos e envia resposta para o agente interface.
Agente Interface	Recebe resposta do agente SGBD e se o usuário é valido envia mensagem ao SGBD solicitando o histórico do usuário.
Agente SGBD	Seleciona as disciplinas do histórico do aluno e as envia respondendo a mensagem para o agente interface.
Agente Interface	Recebe o histórico e disponibiliza para o aluno.
Usuário	Seleciona as disciplinas desejadas.
Agente Interface	Envia mensagem com as disciplinas selecionadas pelo usuário para o agente SGBD.
Agente SGBD	Atualiza o histórico do aluno.
Agente Interface	Mostra ao usuário apenas as disciplinas selecionadas.

Usuário	Confere as disciplinas escolhidas e finaliza as tarefas.
Agente Interface	Finaliza o agente SGBD.
Agente SGBD	É finalizado.
Agente Interface	Termina sua execução.

QUADRO 12 : FUNCIONAMENTO DOS AGENTES

7.3.3 IMPLEMENTAÇÃO DO PROTÓTIPO

A implementação esta dividida em duas partes, a implementação do agent SGBD e do agente Interface. Para demonstrar a implementação do protótipo serão demonstrados vários quadros com partes principais da implementação final. Os códigos fontes de todas as implementações feitas neste trabalho estão disponíveis no capítulo 9, em anexo.

7.3.3.1 IMPLEMENTANDO O AGENTE SGBD

O Quadro 13 mostra as *packages* que precisam ser carregadas para a criação do agente SGBD, dentre elas vale destacar a "com.ibm.aglet.*" e a "com.ibm.aglet.event.*" que importa as classes para a criação do *aglet* e tratamento dos eventos que ocorrem nos mesmos, e a *package* "java.sql.*" que carrega os *drivers* JDBC.

```
package com.ibm.fiji.mauri.teste01;
import com.ibm.aglet.*;
import com.ibm.aglet.event.*;
import java.io.*;
import java.net.*;
import java.sql.*;
```

QUADRO 13 : CLASSES NECESSÁRIAS PARA A CRIAÇÃO DO AGENTE SGBD

O Quadro 14 mostra como o agente SGBD irá proceder no tratamento das mensagens. Para a comunicação com o agente interface foram definidos dois tipos de mensagens que podem ser recebidas, o tipo “SELECT”, que quando recebida chamará o método transacao_select(String exp) que é demonstrado no Quadro 15, e o tipo “UPDATE” que quando recebida chamará o método transacao_update(String exp) que é demonstrado no Quadro 16.

```
public boolean handleMessage(Message msg) {
  try  {
       System.out.println((String)msg.getKind());
       if (msg.sameKind("SELECT"))     {
          System.out.println("recebeu msg select");
          msg.sendReply(transacao_select((String)msg.getArg()));
       }  else if (msg.sameKind("UPDATE"))  {
            msg.sendReply(transacao_update((String)msg.getArg()));
        }
      }     catch(Exception e)  {
          System.out.println("Aglet Slave Falhou");
          System.out.println(e.getMessage());
    }
    return true;
  }
```

QUADRO 14 : TRATAMENTO DE MENSAGENS NO *AGLET* SGBD

```
// Metodo usado para selecionar dados do banco
public static String transacao_select(String exp) throws SQLException
{
 try
 {
   DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
   Connection conn = DriverManager.getConnection
("jdbc:oracle:thin:@200.135.232.1:1521:orc1","tccmauri","mauri1607");
   Statement stmt = conn.createStatement ();
   ResultSet rset = stmt.executeQuery(exp);
   ResultSetMetaData rsetm = rset.getMetaData();
```

```
            int i;
          String cc = "*";
          int ncols = rsetm.getColumnCount();
          for (i = 1; i <= ncols ;i++ )
       {
        cc = cc + rsetm.getColumnName(i);
        if (i < ncols)
        {
         cc = cc + ",";
        }
       }
       cc = cc + "*";
       while (rset.next ())
       {
        for (i = 1; i <= ncols ;i++ )
        {
         cc = cc  + rset.getString(i);
         if (i < ncols)
         {
          cc = cc + ",";
         }
        }
        cc = cc + "*";
                    }
       System.out.println("Teste de acesso ao oracle - Escreveu results");
       System.out.println(" mandou o par "+ cc+ "para a var cc");
       conn.close();
       return cc;
      }
     catch(Exception e)
     {
       System.out.println("Erro ao consultar o SGBD");
       return "ERRO_TRANSACAO_BANCO";
     }
    }
```

QUADRO 15 : MÉTODO TRANSACAO_SELECT( ) DO AGENTE SGBD

```
   // Executa alteracoes no banco de dados
   public static String transacao_update(String exp) throws SQLException
   {
     try
     {
       DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
       Connection conn = DriverManager.getConnection
("jdbc:oracle:thin:@200.135.232.1:1521:orc1","tccmauri","mauri1607");
       Statement stmt = conn.createStatement ();
       stmt.executeUpdate(exp);
       conn.close();
     }
     catch(Exception e)
     {
       System.out.println(e.getMessage());
       return "ERRO_TRANSACAO_BANCO";
     }
     return "OK";
    }
```

QUADRO 16 : MÉTODO TRANSACAO_UPDATE( ) DO AGENTE SGBD

A diferença entre o método transacao_select( ) implementado conforme o Quadro 15 e o transacao_update( ) conforme o Quadro 16 consiste no fato de que o método transacao_update( ) é usado para atualizar campos em tabelas no SGBD e retorna a *String* "ERRO_TRANSACAO_BANCO" se ocorrer alguma exceção ou a *String* “OK” quando a transação for realizada com êxito. Já o método transacao_select( ) será usado para executar apenas consultas no SGBD e retornará os dados da mesma concatenados em uma *String*, onde as linhas da consulta SQL serão separadas pelo caracter “*” e cada campo da linha será delimitado pelo caracter “,”, sendo que na primeira linha serão colocados os nomes das colunas consultadas. Por exemplo, se uma consulta SQL retornasse os dados do Quadro 17, a *string* que seria devolvida pelo método seria :

“ * Codigo, Nome, Idade, Sexo * 1,Raul Seixas, 50, M * 2, Jimi Hendrix, 54, M * ”.

Codigo	Nome	Idade	Sexo
1	Raul Seixas	50	M
2	Jimi Hendrix	54	M

QUADRO 17 : EXEMPLO DE DADOS DE UMA CONSULTA

Em ambos os métodos é utilizado o *driver* JDBC onde a linha "DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());" carrega uma nova instância do *driver* que é fornecido pelo fabricante do SGBD, que neste caso é a Oracle. A linha Connection "conn = DriverManager.getConnection ("jdbc:oracle:thin:@200.135.232.1:1521:orc1","tccmauri","mauri1607");" cria uma nova conexão com o banco e a linha "Statement stmt = conn.createStatement ();" cria uma novo *statement* usando o método conn.createStatement () do objeto Connection que é usado para executar as transações SQL através da chamada do método "stmt.executeQuery(exp)" no método transacao_select( ) ou stmt.executeUpdate(exp) no método transacao_update( ), onde stmt é uma instância do objeto Statement e "exp" é a expressão SQL a ser executada como por exemplo "select * from table1".

7.3.3.2 IMPLEMENTANDO O AGENTE INTERFACE

O agente interface será descrito neste tópico juntamente com os diversos quadros contendo as principais partes da implementação final.

O *layout* inicial da tela do sistema foi idealizado conforme a Figura 18, sendo que o mesmo será inicialmente composto de dois campos para digitação de código e senha por parte do usuário, uma área onde lhe serão mostradas as mensagens do sistema, e a área onde estarão as disciplinas na qual será usado uma linha para cabeçalho e uma para cada disciplina que será mostrada. A figura também mostra as classes que serão criadas para implementar a interface e qual a funcionalidade de cada uma.

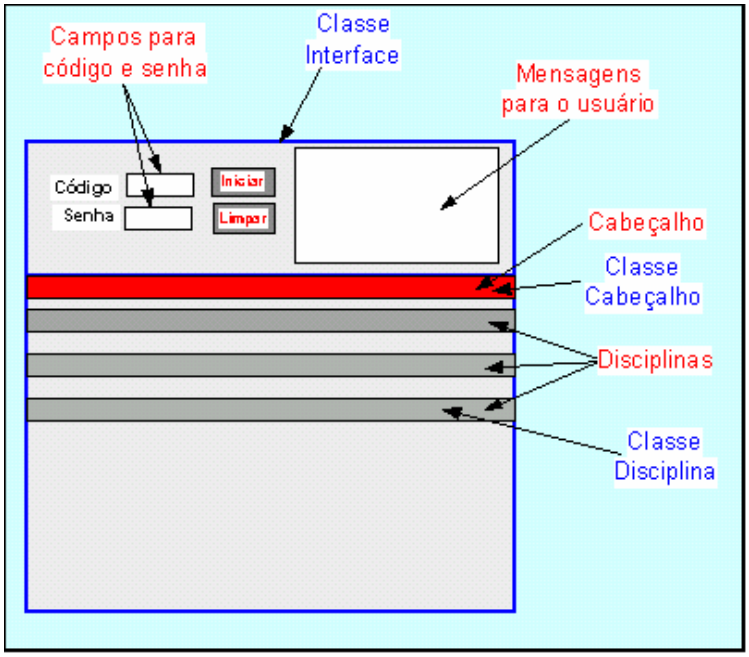

FIGURA 18 : *LAYOUT* DA TELA INICIAL DO SISTEMA

O Quadro 18 mostra as *packages* contendo as classes necessárias para o agente interface. Entre elas merecem maior destaque a "com.ibm.fiji.lib.FijiApplet" que contém as classes para criação de *Applets* Fiji, a *package* "com.ibm.aglet.*", que contém as classes para implementação de *Aglets* e o "java.awt.*" que contém as classes com componentes para desenvolvimento de interfaces gráficas, tais como botões, menus e outros.

```
package com.ibm.fiji.mauri.teste01;
import com.ibm.fiji.lib.FijiApplet;
import java.net.*;
import com.ibm.aglet.*;
import com.ibm.aglets.util.*;
import java.awt.*;
```

QUADRO 18 : CLASSES NECESSÁRIAS PARA A CRIAÇÃO DO AGENTE INTERFACE

O Quadro 19 contém a declaração das principais variáveis usadas no sistema. A linha "final String StrSgbd = "atp://netserver.ferj.rct-sc.br:434"" declara uma variável que irá

armazenar o endereço do Tahiti rodando no SGBD. A seguinte “final String ClassSgbd = "com.ibm.fiji.mauri.teste01.AgletSgbd"“ contém o nome da classe que corresponde ao agente SGBD. A declaração “AgletProxy ProxySgbd = null“ inicializa com “null” uma instância do objeto AgletProxy que irá posteriormente armazenar o Proxy do agente SGBD para que o agente interface possa enviá-lo a SGBD e também enviar-lhe mensagens. Na seqüência temos o uso da classe Disciplina “Disciplina[] Disc “ declara um array de objetos da classe Disciplina que será descrita no Quadro 25.

```
// Armazena URL do SGBD
 final String StrSgbd = "atp://info03.ferj.rct-sc.br:434";
 // Classe que corresponde ao agente SGBD
 final String ClassSgbd = "com.ibm.fiji.mauri.teste01.AgletSgbd";
 // Variavel para armazenar o proxy do SGBD quando este for criado
 AgletProxy ProxySgbd = null;
 // Variavel que armazena o retorno das mensagens enviadas
 String retorno;
 // Variavel que armazenará o numero de disciplinas do aluno
 int ndis;
 // Objeto que cria o cabecalho das disciplinas
 Cabecalho cab = new Cabecalho();
 // Array de objetos do tipo disciplinas
 Disciplina[] Disc;
 // Variavel que armazenará o numero de disciplinas selecionadas pelo usuario
 int nselected;
```

QUADRO 19 : PRINCIPAIS VARIÁVEIS DO AGENTE INTERFACE

No Quadro 20 é mostrado o método criaAgenteSgbd() do agente Interface que executa a criação do agente SGBD através do método “getAgletContext().createAglet(getCodeBase(), ClassSgbd ,null)“ sendo que ClassSgbd é a variável com o nome da classe definida no Quadro 19. O Proxy do agente criado é armazenado no objeto ProxySgbd, através do qual ele será despachado ao SGDB através do método “ProxySgbd.dispatch(new URL(StrSgbd))“. O método criaAgenteSgbd() retorna true se a criacao e o despacho foram normais e false se ocorreu alguma exceção.

```
private boolean criaAgenteSgbd()
 {
  AgletProxy Proxy_velho = null;
  try
  {
   ProxySgbd = getAgletContext().createAglet(getCodeBase(), ClassSgbd ,null);
   Proxy_velho = ProxySgbd;
   ProxySgbd = ProxySgbd.dispatch(new URL(StrSgbd));
  }
  catch (Exception e)
  {
    if (Proxy_velho != null)
    {
      try
      {
        Proxy_velho.dispose();
      }
      catch (Exception x)
      {
       ProxySgbd = null;
       return false;
      }
    }
    ProxySgbd = null;
    mostraexcecao(e);
    return false;
  }
  mostraestado("Agente Sgbd despachado para "+StrSgbd);
  return true;
 }
```

QUADRO 20 : CRIANDO O AGENTE SGBD E DESPACHANDO PARA O SERVIDOR SGBD

O Quadro 21 mostra o método consultaAgenteSgbd() usado para enviar mensagens para o agente SGBD, que retorna uma *string* com o resultado da consulta se tudo correu bem ou "ERRO_ENVIAR_MENSAGEM" se ocorrer alguma exceção.

```
private String consultaAgenteSgbd(String str1,String str2)
 {
   String ret;
   try
   {
         Message msg = new Message(str1,str2);
         ret = (String)ProxySgbd.sendMessage(msg);
   }
   catch (Exception e)
   {
    mostraexcecao(e);
    return "ERRO_ENVIAR_MENSAGEM";
   }
   return ret;
 }
```

QUADRO 21 : MÉTODO PARA COMUNICAÇÃO COM O AGENTE SGBD

No Quadro 22 é demonstrado a primeira troca de mensagens entre os agentes, na qual o agente interface através do método consultaAgenteSgbd() envia uma mensagem do tipo “SELECT” com a expressão SQL a ser executada no SGBD pelo agente SGBD e o mesmo recebe o retorno da mensagem armazenando na variável retorno, através da qual avaliará se o usuário é válido e se o mesmo for inválido ele mostrará mensagem de erro e permitirá a repetição da identificação. Da mesma forma se não for possível estabelecer comunicação com o TAHITI que esta rodando na máquina do SGBD, ele mostrará a mensagem informando a ocorrência do erro de comunicação.

```
retorno = this.consultaAgenteSgbd("SELECT","SELECT ALUNO.ALUNO_CODIGO,
ALUNO.ALUNO_NOME,ALUNO.CURSO_CODIGO,CURSO.CURSO_DESCRICAO
FROM ALUNO ALUNO,CURSO CURSO WHERE ALUNO.CURSO_CODIGO =
CURSO.CURSO_CODIGO AND ALUNO.ALUNO_CODIGO = '"+textField1.getText()+"'
AND ALUNO.ALUNO_SENHA = '"+textField2.getText()+"'");
      // Se a consulta retornar 0 linhas -> usuario invalido
      // senao mostra os dados do usuario
      if (retorno == "ERRO_ENVIAR_MENSAGEM")
      {
        DlgMensagem dialog = new DlgMensagem("Erro de comunicação ","Não foi possível
estabelecer comunicação !!!");
        dialog.show();
        return false;
      }
      if (retorno == "ERRO_TRANSACAO_BANCO")
      {
       DlgMensagem dialog = new DlgMensagem("Erro no SGBD ","Não foi possível
executar a transacao  !!!");
       dialog.show();
       return false;
      }
      if (nlinhas(retorno) == 0)
      {
        DlgMensagem dialog = new DlgMensagem("Usuário inválido !!!","O código ou a
senha informados é inválidos !");
        dialog.show();
        mostraestado("Codigo do aluno ou senha inválida !!!");
        return false;
      }
```

QUADRO 22 : MENSAGEM SOLICITANDO OS DADOS PESSOAIS DO ALUNO

O código do Quadro 23 só é executado se o usuário for considerado válido. Nele é enviado uma nova mensagem do tipo “SELECT” ao agente SGBD solicitando os dados do

histórico do aluno através da instrução SQL e montado a interface com o usuário através do uso do objeto Disc que é um *array* de instâncias do objeto Disciplina descrito no Quadro 25.

```
retorno = this.consultaAgenteSgbd("SELECT","SELECT AD.DISCIPLINA_CODIGO,
D.DISCIPLINA_DESCRICAO,D.DISCIPLINA_FASE,AD.ALUNODIS_SITUACAO,P.PR
OFESSOR_CODIGO,P.PROFESSOR_NOME,H.HORARIO_CODIGO,H.HORARIO_INICI
O,H.HORARIO_FINAL,H.HORARIO_DIASEMANA FROM ALUNODIS AD,
DISCIPLINA D, PROFESSOR P, PROFDISC PD, HORARIO H, HORARIODIS HD
WHERE AD.DISCIPLINA_CODIGO = D.DISCIPLINA_CODIGO AND
P.PROFESSOR_CODIGO=PD.PROFESSOR_CODIGO AND
D.DISCIPLINA_CODIGO=PD.DISCIPLINA_CODIGO AND H.HORARIO_CODIGO =
HD.HORARIO_CODIGO AND HD.DISCIPLINA_CODIGO = AD.DISCIPLINA_CODIGO
AND AD.ALUNO_CODIGO='"+textField1.getText()+"' ORDER BY
D.DISCIPLINA_FASE,D.DISCIPLINA_CODIGO, H.HORARIO_INICIO,
H.HORARIO_DIASEMANA");

        if (retorno == "ERRO_ENVIAR_MENSAGEM")
        {
          DlgMensagem dialog = new DlgMensagem("Erro de comunicação ","Não foi
possível estabelecer comunicação !!!");
           dialog.show();
           return false;
        }
        if (retorno == "ERRO_TRANSACAO_BANCO")
        {
          DlgMensagem dialog = new DlgMensagem("Erro no SGBD ","Não foi possível
executar a transacao  !!!");
           dialog.show();
           return false;
        }
        if (nlinhas(retorno) == 0)
        {
          DlgMensagem dialog = new DlgMensagem("Erro !!!","Nenhuma disciplina foi
encontrada !");
           dialog.show();
           return false;
```

```
}
this.setSize(750,1440);
int i = 1;
   cab.setVisible(true);
   Integer cod1, cod2;
      cod2 =  new Integer(0);
   ndis = 0;

   // Mostra as disciplinas
   for (i = 1; i <= nlinhas(retorno) ; i++ )
   {
    cod1 =  new Integer(getCampos(retorno,i,1));
    if (cod1.intValue() != cod2.intValue())
    {
     ndis ++;
     Disc[ndis].mudaTexto(getCampos(retorno,i,1),1);
     Disc[ndis].mudaTexto(getCampos(retorno,i,2),2);
     Disc[ndis].mudaTexto(getCampos(retorno,i,3),3);

     // Altera o estado da disciplina
     Disc[ndis].setChecked(true);
     if (getCampos(retorno,i,4).trim().charAt(0) == 'C')
     {
     Disc[ndis].setChecked(true);
        Disc[ndis].setEnabledcb(false);
     }else if (getCampos(retorno,i,4).trim().charAt(0) == 'M')
     {
      Disc[ndis].setChecked(true);

     }else if (getCampos(retorno,i,4).trim().charAt(0) == 'A')
     {
       Disc[ndis].setChecked(false);
     }

     Disc[ndis].setVisible(true);
     Disc[ndis].setStateInicial();
```

```
            }else if (cod1.intValue() == cod2.intValue())
            {
              switch (getCampos(retorno,i,10).trim().charAt(0))
              {
               case '1' :
                  Disc[ndis].mudaTexto(getCampos(retorno,i,8).substring(0,5)+"-
"+getCampos(retorno,i,9).substring(0,5),4);
                  break;

                    case '2' :
                  Disc[ndis].mudaTexto(getCampos(retorno,i,8).substring(0,5)+"-
"+getCampos(retorno,i,9).substring(0,5),5);
                  break;

               case '3' :
                  Disc[ndis].mudaTexto(getCampos(retorno,i,8).substring(0,5)+"-
"+getCampos(retorno,i,9).substring(0,5),6);
                  break;

                  case '4' :
                  Disc[ndis].mudaTexto(getCampos(retorno,i,8).substring(0,5)+"-
"+getCampos(retorno,i,9).substring(0,5),7);
                  break;

               case '5' :
                  Disc[ndis].mudaTexto(getCampos(retorno,i,8).substring(0,5)+"-
"+getCampos(retorno,i,9).substring(0,5),8);
                  break;
                }
              }
              cod2 = cod1;
            }
        }
```

QUADRO 23 : MENSAGEM SOLICITANDO AS DISCIPLINAS E MONTAGEM DA INTERFACE

No Quadro 24 é mostrado como o agente interface envia várias mensagens do tipo "UPDATE" (uma para cada disciplina) para o agente SGBD atualizando o seu histórico.

```
    if (button.getLabel() == "Enviar")
    {
            mostraestado("Clicado o botao Enviar");
            mostraestado("Atualizando as disciplinas");

            nselected = 0;
           // Esconde as disciplinas
           for (int i = 1; i <= ndis ; i++ )
                 Disc[i].setVisible(false);

          // Verifica se o aluno alterou e executa a alteracao no banco
          // Prepara para mostrar as disciplinas selecionadas
          for (int i = 1;i <= ndis ;i++ )
          {
            if (Disc[i].isEnabledcb() == true)
            {
              try
             {
               if (Disc[i].getStatecb() == true)
               {
                  nselected++;
                  Disc[nselected].Copia(Disc[i]);
                  Disc[nselected].setEnabled(false);

                  if ((Disc[i].getStatecb() != Disc[i].getStateInicial()))
                  {
                   retorno = this.consultaAgenteSgbd("UPDATE","UPDATE  ALUNODIS AD
SET AD.ALUNODIS_SITUACAO = 'M' WHERE AD.DISCIPLINA_CODIGO =
"+Disc[i].getLabelcb()+" AND AD.ALUNO_CODIGO ='"+textField1.getText()+"'");
                 mostraestado("Disciplina atualizada retornada  "+Disc[i].getLabelcb()+"
"+retorno);
             }
           }else if (Disc[i].getStatecb() == false)
```

```
                        {
                         if ((Disc[i].getStatecb() != Disc[i].getStateInicial()))
                         {
                          retorno = this.consultaAgenteSgbd("UPDATE","UPDATE
ALUNODIS AD SET AD.ALUNODIS_SITUACAO = 'A' WHERE
AD.DISCIPLINA_CODIGO = "+Disc[i].getLabelcb()+" AND
AD.ALUNO_CODIGO='"+textField1.getText()+"'");
                      mostraestado("Disciplina atualizada retornada  "+Disc[i].getLabelcb()+"
"+retorno);
                    }
                  }
                 }
                catch (Exception e)
                {
                  mostraexcecao(e);
                }
     }
  }
```

QUADRO 24 : MENSAGENS ATUALIZANDO AS SELEÇÕES DO USUÁRIO

```
String[] str;
 public Disciplina()   {
   n = str.length;
   l = new TextField[str.length -1];
   cb = new Checkbox(String.valueOf(str[0]),true);
   this.setLayout(new FlowLayout());
   this.add(cb);
   for (int i = 0; i < str.length -1; i++) {
    aux = String.valueOf(str[i + 1]);
    l[i] = new TextField(String.valueOf(str[i + 1]),aux.length() -1);
    l[i].setFont(new Font("Serif", 0, 10));
    if (i > 2)      {
     l[i].resize(10,10);
    }
    this.add(l[i]);
   }
```

```
}
```

QUADRO 25 : CLASSE DISCIPLINA – PARTE PRINCIPAL

Nos quadros anteriores foram usados alguns métodos não descritros até o momento e que estão no Quadro 26 como o “mostraestado()“ e “mostraexececao()“ que é usado para mostrar uma mensagem em um campo TextArea usado como meio de mostrar mensagens aos usuários. Também são de grande importância os métodos “nlinhas()“ e “getCampos()“, sendo que o primeiro retorna o número de linhas de uma consulta recebida através de mensagem e o segundo recupera um determinado campo na *String*, lembrando que os dados da consulta SQL são retornados em uma *String* obedecendo algumas regras de formatação descritas no tópico 7.3.3.1.

```
public void mostraestado(String str){
   textArea1.appendText(str+ "\n");
}

public void mostraexcecao(Exception excecao){
   mostraestado(excecao.getMessage());
}

public int nlinhas(String str)  {
   int count = 0;
   for (int i = 0; i < str.length(); i++ )    {
     if (str.charAt(i)=='*')      {
       count ++;
     }
   }
   if (count < 3)    {
     return 0;
   }  else    {
     return count -2;
   }
  }
```

```
public String getCampos(String str, int lin, int col)  {
   int last = 1;
   int i;
  String ret = "";
   for (i = 0 ;i < lin ;i++) {
       last = str.indexOf("*",last + 1);
    }
   for (i = 1 ;i < col ;i++)    {
      last = str.indexOf(",",last + 1);
   }
  for (i = last ; (i < str.length()) && (str.charAt(i+1) != ',') && (str.charAt(i+1) != '*') ;i++) {
      ret = ret + str.charAt(i +1);
   }
   return ret;
}
```

QUADRO 26 : MÉTODOS ESPECIAIS DO *AGLET* INTERFACE

O Quadro 27 mostra o método finalizaAgenteSgbd() do agente Interface que finaliza o agente SGBD, sendo que o mesmo nem chega a retornar até a estação cliente onde foi criado, ele é finalizado pelo agente Interface que executa esta tarefa através do método “dispose()” chamado através do *proxy* do agente SGBD.

```
private boolean finalizaAgenteSgbd()
  {
     try
     {
       mostraestado("Enviando mensagem para finalizar agente remoto");
       ProxySgbd.dispose();
       ProxySgbd = null;
     }
     catch (Exception e)
     {
       mostraexcecao(e);
       return false;
     }
```

```
    mostraestado("Agente Remoto finalizado...Tenha um bom dia !!!!");
    return true;
  }
```

QUADRO 27 : FINALIZAÇÃO DO AGENTE SGBD

7.3.3.3 CRIANDO A PÁGINA HTML

Depois dos agentes é necessário criar uma página HTML que iniciará a execução do *aglet* no *browser*. O Quadro 28 mostra esta implementação que compõe o arquivo projeto01.html, sendo que a principal parte do mesmo esta na *tag* <APPLET code=com.ibm.fiji.mauri.teste01.Interface.class width=750 height=1440> onde é indicado onde esta a classe do agente interface, bem como os parâmetros passados referentes a localização do *router* que deve estar sendo executando no mesmo servidor que esta executando o servidor HTTP, que permitirá a comunicação do agente interface com servidores diferentes do qual ele é descarregado para o *browser*.

```
<HTML>
<HEAD>
  <META HTTP-EQUIV="Content-Type"
    CONTENT="text/html; charset=iso-8859-1">
  <META NAME="GENERATOR"
    CONTENT="Mozilla/4.01 [en] (Win95; I) [Netscape]">
  <TITLE>FERJ – Sistema de Pré-matrícula</TITLE>
</HEAD>
<BODY>
<!—Inicio da tag applet  -->
<APPLET code=com.ibm.fiji.mauri.teste01.Projeto01.class
        width=750 height=1440>
 <!—Parâmetros passados ao applet  -->
 <PARAM name=verbose value="true">
 <param name=routerHost value="info03.ferj.rct-sc.br">
```

```
 <param name=routerPort value=4340>
  
</APPLET>
</CENTER>
</BODY>
</HTML>
```

QUADRO 28 : CÓDIGO HTML PARA EXECUÇÃO DOS AGENTES

7.4 DEMONSTRAÇÃO E USO DO PROTÓTIPO

Para iniciar a execução do protótipo depois de compilados os códigos fontes dos agentes são necessários alguns passos iniciais.

7.4.1 INICIANDO A EXECUÇÃO DO *ROUTER*

O *router* é uma *package* fornecida pela IBM implementada em Java que possue como função principal permitir que um *aglet* rodando em um *browser* possa estabelecer comunicação com outros servidores além daquele do qual ele foi descarregado para o *browser*. Por ser implementado em Java, ele é independente de plataforma e pode ser iniciado tanto no Windows em uma janela MS-DOS como em um terminal Unix através da linha de comando, mas, para facilitar, durante a fase de desenvolvimento foi criado um arquivo de lote para inicialização do mesmo. O exemplo do Quadro 29 mostra o arquivo routerdmauri.bat que carrega o router em ambiente Windows. Para sua execução em plataforma Unix basta alterar os caminhos para os arquivos e diretórios, sendo o resto muito semelhante.

```
echo off
REM ================================================
REM routerdmauri.bat
REM ================================================
SET ROUTER_HOME=C:\Fiji1.0.2\Kit\router
SET JDK_HOME=c:\jdk1.1.6
SET java_vm=c:\jdk1.1.6\bin\java
java_vm –classpath %ROUTER_HOME%\router.jar;%JDK_HOME%
      \lib\classes.zip;    %CLASSPATH% -Dprogram-name=routerd
      com.ibm.fiji.router.Router %1 %2 %3 %4 %5 %6 %7 %8 %9
```

QUADRO 29 : ARQUIVO DE LOTE PARA EXECUÇÃO DO ROUTER NO WINDOWS 95

A porta na qual o *router* deve executar pode ser passada como parâmetro, e se nenhuma porta for informada ele iniciará na porta padrão 4340. Veja na Figura 19 a inicialização do router.

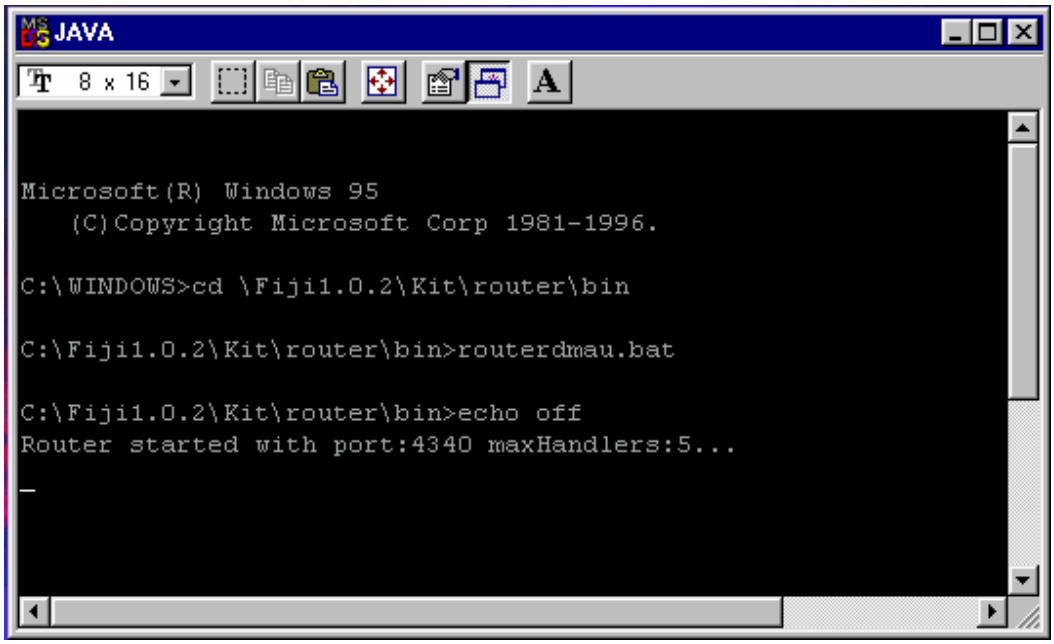

FIGURA 19 : INICIALIZAÇÃO DO ROUTER

7.4.2 INICIANDO A EXECUÇÃO DO TAHITI

O Tahiti, que irá rodar na máquina em que se encontra o SGBD é mais fácil de ser inicializado, uma vez que o programa de instalação do mesmo já coloca um atalho no menu iniciar. Deve-se atentar é claro para as variáveis de ambiente como por exemplo o CLASSPATH, no qual é necessário incluir as classes do JDBC fornecidas pela Oracle que serão utilizadas pelo agente SGBD para acessar o SGBD. As variáveis de ambiente do Tahiti são definidas no arquivo agletsrv.ini. O Quadro 30 mostra o conteúdo deste arquivo no qual foi incluída a expressão "C:\jdk1.1.6\lib\classes111.zip" para carregar as classes do JDBC através do CLASSPATH.

```
[Launch]
COMMAND LINE="C:\JDK1.1.6\BIN\..\bin\java.exe" "-Dinstall.root=C:\Aglets1.0.3"
–classpath "C:\Aglets1.0.3\lib\aglets.jar;C:\jdk1.1.6\lib\classes.zip;
.;C:\jdk1.1.6\lib\classes111.zip;.;C:\Aglets1.1b\lib\aglets.jar;.;C:\Fiji1.0.2\Kit\fiji.zip;C:\JDK1
.1.6\BIN\..\classes;C:\JDK1.1.6\BIN\..\lib\classes.zip;C:\JDK1.1.6\BIN\..\lib\classes.jar;C:\JD
K1.1.6\BIN\..\lib\rt.jar;C:\JDK1.1.6\BIN\..\lib\i18n.jar" com.ibm.awb.launcher.Agletsd
WORKING DIRECTORY=C:\Aglets1.0.3\
```

QUADRO 30 : ARQUIVO AGLETSRV.INI USADO NA INICIALIZAÇÃO DO TAHITI

Também pode ser indicado neste arquivo a porta em que o Tahiti deve rodar através do parâmetro "-port <porta> " o qual deve ser colocado após o nome da classe pincipal do Tahiti que corresponde a "com.ibm.awb.launcher.Agletsd". Se nenhum parâmetro especificando a porta for informado o Tahiti executará utilizando a porta padrão 434.

A Figura 20 mostra o Tahiti rodando na porta 434 no servidor info03.ferj.rct-sc.br:434.

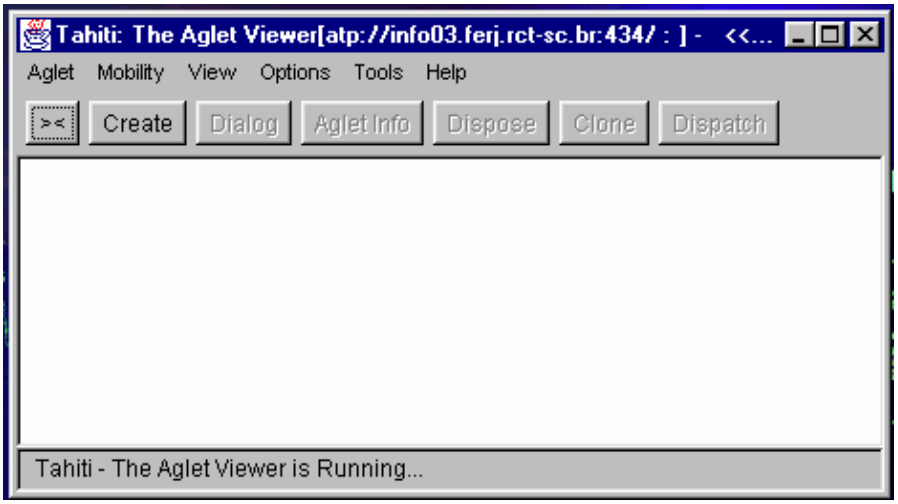

FIGURA 20 : EXECUTANDO TAHITI

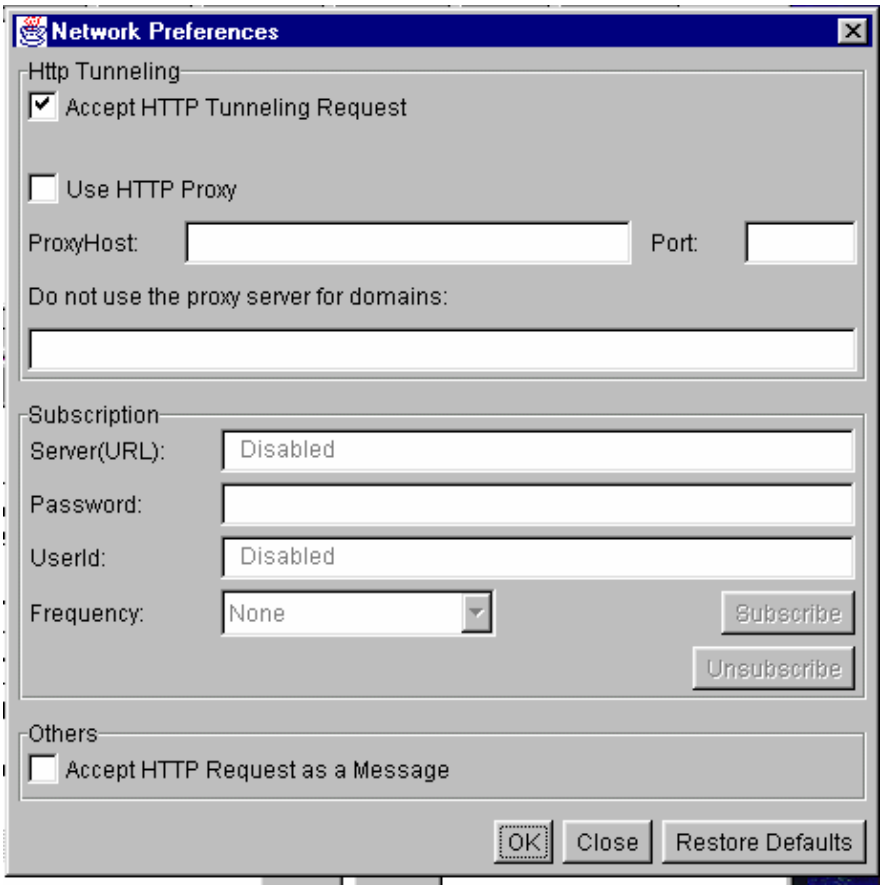

FIGURA 21 : CONFIGURANDO O TAHITI

Deve-se também configurar o Tahiti para aceitar chamadas HTTP através da opção no menu Options – Network Preferences habilitando o checkbox "*Accept HTTP Tuneling Request*" conforme a Figura 21 para que o mesmo aceite mensagens e outros *aglets* despachados de ou para *browsers*.

7.4.3 CONFIGURANDO O NETSCAPE

O navegador Netscape já possue todo o suporte à execução dos *Aglets*, sendo que a única configuração necessária pelo usuário,é colocar o arquivo fiji.zip que contém o *plug-in* no diretório "NETSCAPE_HOME\plug-ins\".

Quanto a versão do netscape que foi utilizada, o protótipo foi testado nas versões 4.5 e 4.6 do Netscape Communicator.

O protótipo não funciona em nenhuma das versões do Internet Explorer da Microsoft por causa de diferenças na especificação do seu suporte a linguagem Java.

7.4.4 CARREGANDO O PROTÓTIPO

Uma vez feita a configuração deve-se iniciar o Netscape e acessar o endereço no qual esta o documento HTML que inicia o agente Interface, que no caso deste protótipo se encontra em http://info03.ferj.rct-sc.br/fiji/projeto01.html.

Será necessário a concessão de quatro certificados conforme a Figura 22, que é feita clicando no botão "*Grant"*, ao contrário ele pode negar os privilégios através do botão "*Deny"* o que fará com que a execução do agente Interface seja interrompida. Estes certificados já foram explicados no item 6.4.

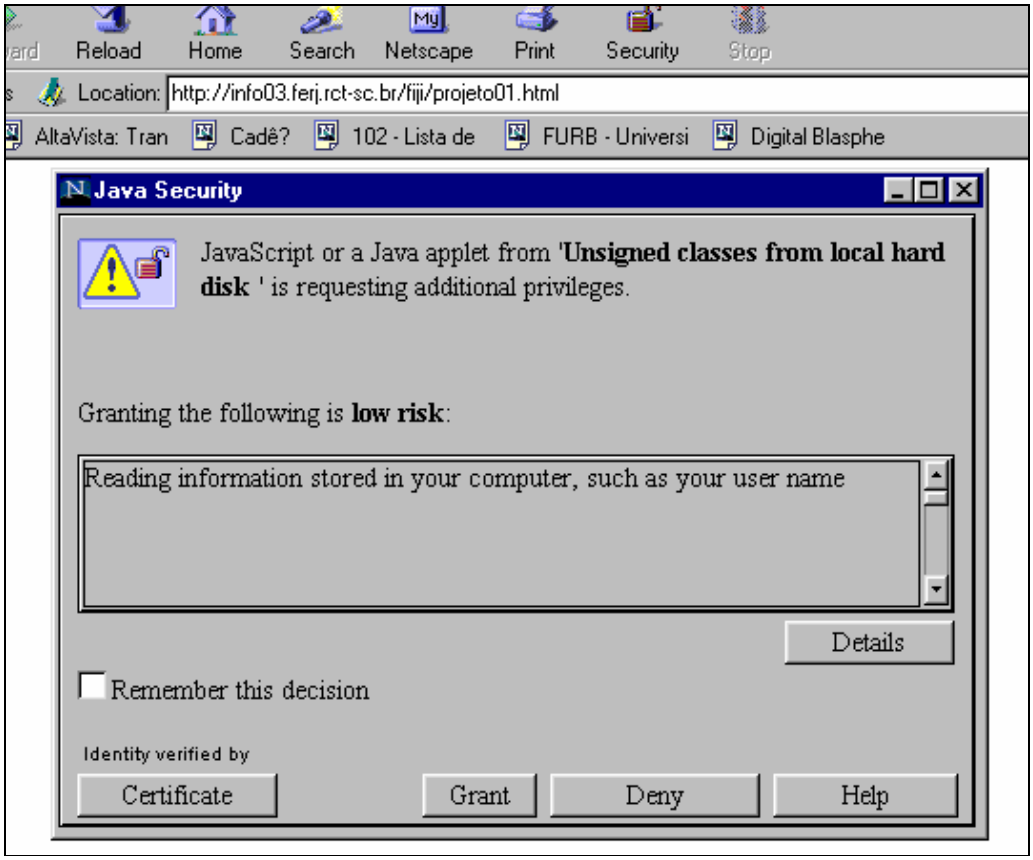

FIGURA 22 : CERTIFICADOS A SEREM CONCEDIDOS

Finalmente, então tem-se na Figura 23 a tela inicial do protótipo rodando no navegador Netscape. A interface inicial é composta de um campo para o usuário informar sua matrícula e outro para a sua senha, um botão para iniciar o processo e outro para limpar os campos, bem como um campo TextArea no canto superior direito onde serão mostradas as mensagens para o usuário.

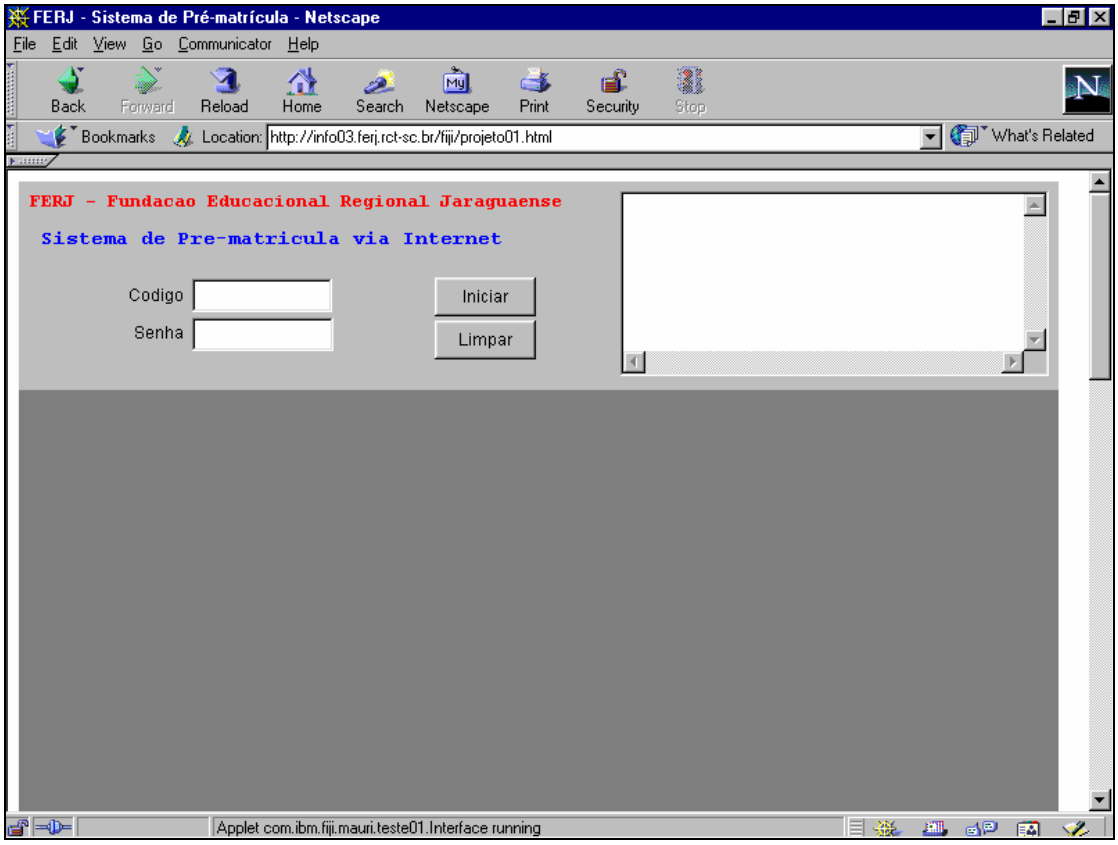

FIGURA 23 : TELA INICIAL DO PROTÓTIPO

7.4.5 USANDO O PROTÓTIPO

Na primeira parte da utilização do protótipo o usuário passará pela identificação do mesmo junto ao sistema para verificar sua autenticidade que será confirmada através da informação de uma senha por parte do mesmo, senha esta que esta cadastrada junto ao sistema acadêmico da FERJ.

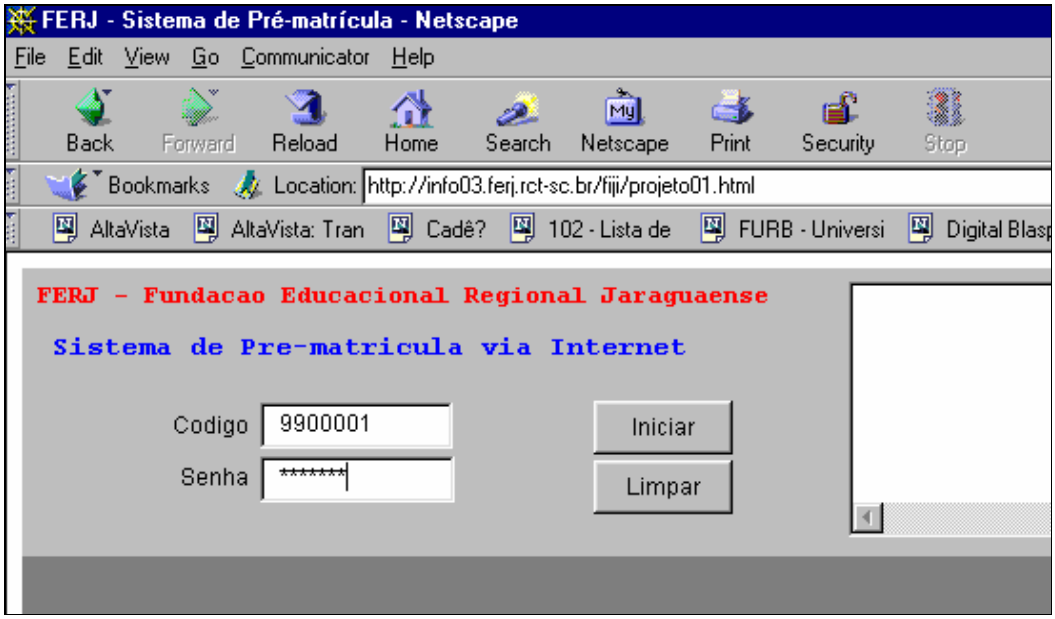

FIGURA 24 : INFORMANDO A SENHA PARA AUTENTICAÇÃO

De posse do código de matrícula e a senha, o agente Interface cria o agente SGBD e o despacha até a máquina onde está o SGBD que esta rodando o Tahiti, e inicia sua comunicação com ele através de mensagens para fazer a validação do usuário, consultar seu histórico e posteriormente mostrar para o usuário as disciplinas de seu curso conforme podemos ver na Figura 25. Se o usuário for inválido, o agente SGBD continua no servidor SGBD, sendo que se o usuário fechar o *browser* será acionado o método *stop()* da classe applet que é usado para desalocar o mesmo, evitando assim que vários agentes criados e enviados ao servidor de banco de dados fiquem perdidos pelo fato do usuário finalizar a aplicação antes de executar todos os passos da mesma.

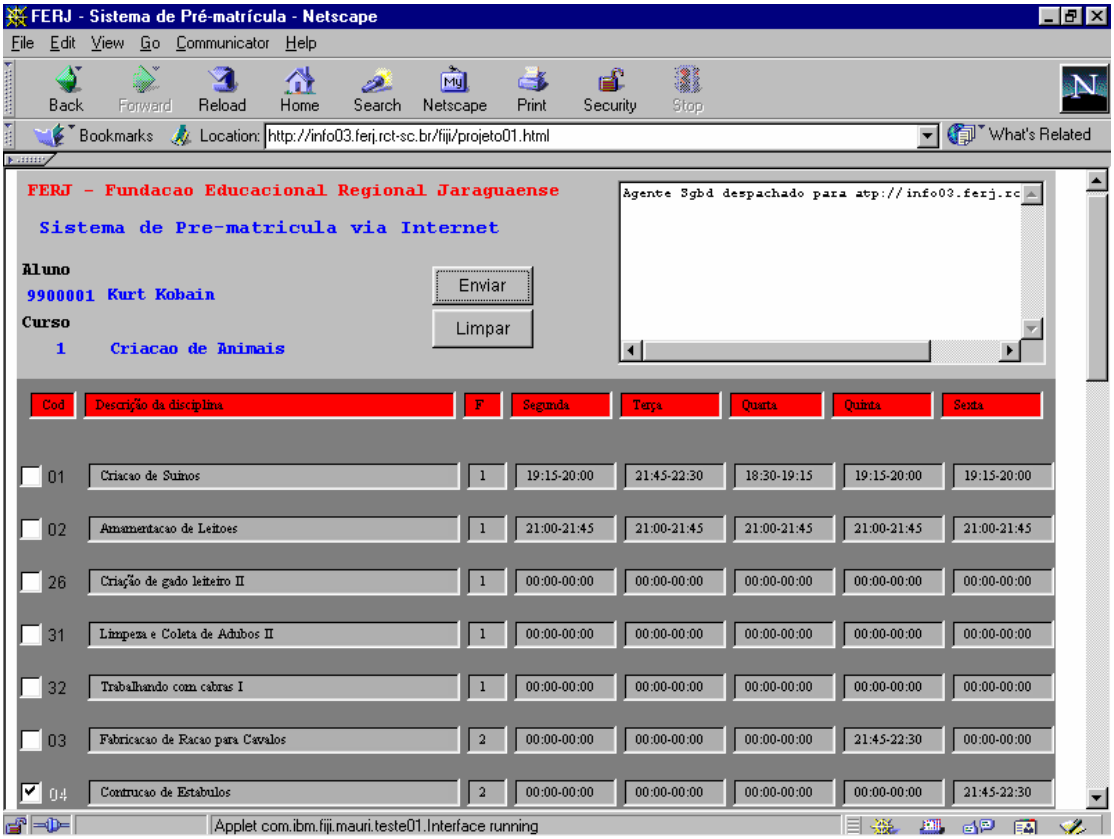

FIGURA 25 : MOSTRANDO AS DISCIPLINAS PARA O USUÁRIO

A Figura 26 mostra o Tahiti após receber o agente despachado pelo agente Interface .

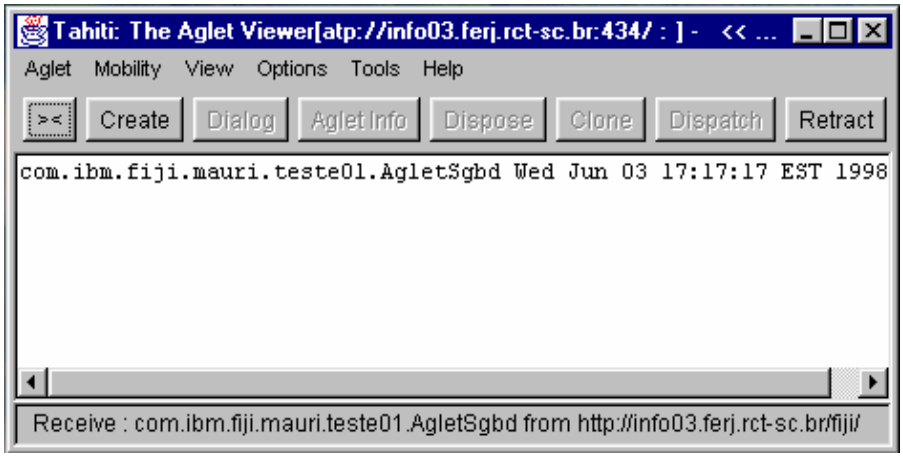

FIGURA 26 : TAHITI APÓS RECEBER O AGENTE SGBD

O usuário pode então selecionar as disciplinas desejadas através do checkbox que lhe é mostrado junto a cada disciplina, sendo que as disciplinas que ele já concluiu aparecem com este checkbox setado e sem possibilidades do usuário desabilitá-lo (disciplina 04). Veja na Figura 27 que o usuário selecionou as disciplinas 01,02,26,32.

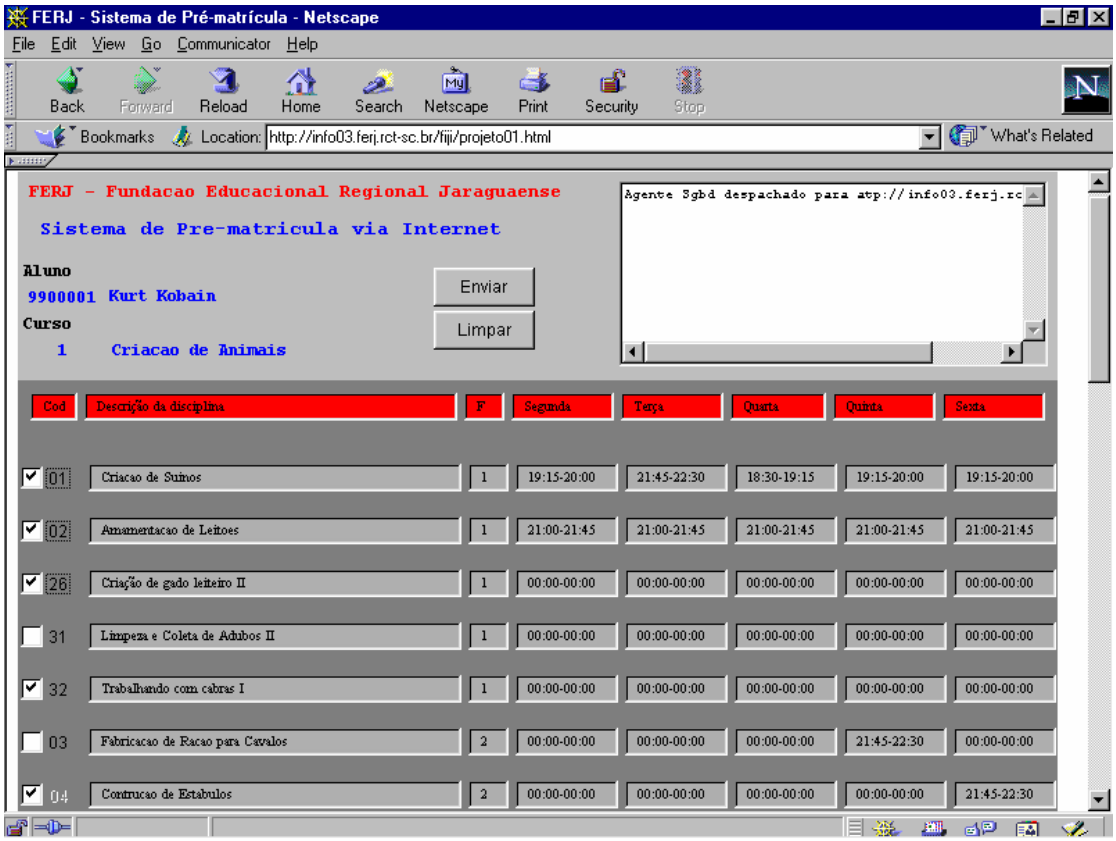

FIGURA 27 : USUÁRIO SELECIONA AS DISCIPLINAS

Quando o usuário terminar a sua seleção ele clicará no botão “Enviar”, evento este que fará com que o agente Interface entre novamente em comunicação com o agente SGBD para atualizar o histórico do aluno de acordo com suas seleções e depois lhe mostre apenas um diálogo de mensagem indicando a finalização do agente remoto e as disciplinas selecionadas para que o mesmo confira as disciplinas nas quais se matriculou. A Figura 28 representa esta etapa, note que foram mostradas ao usuário apenas as disciplinas 01,02,26,32 que ele irá cursar no semestre.

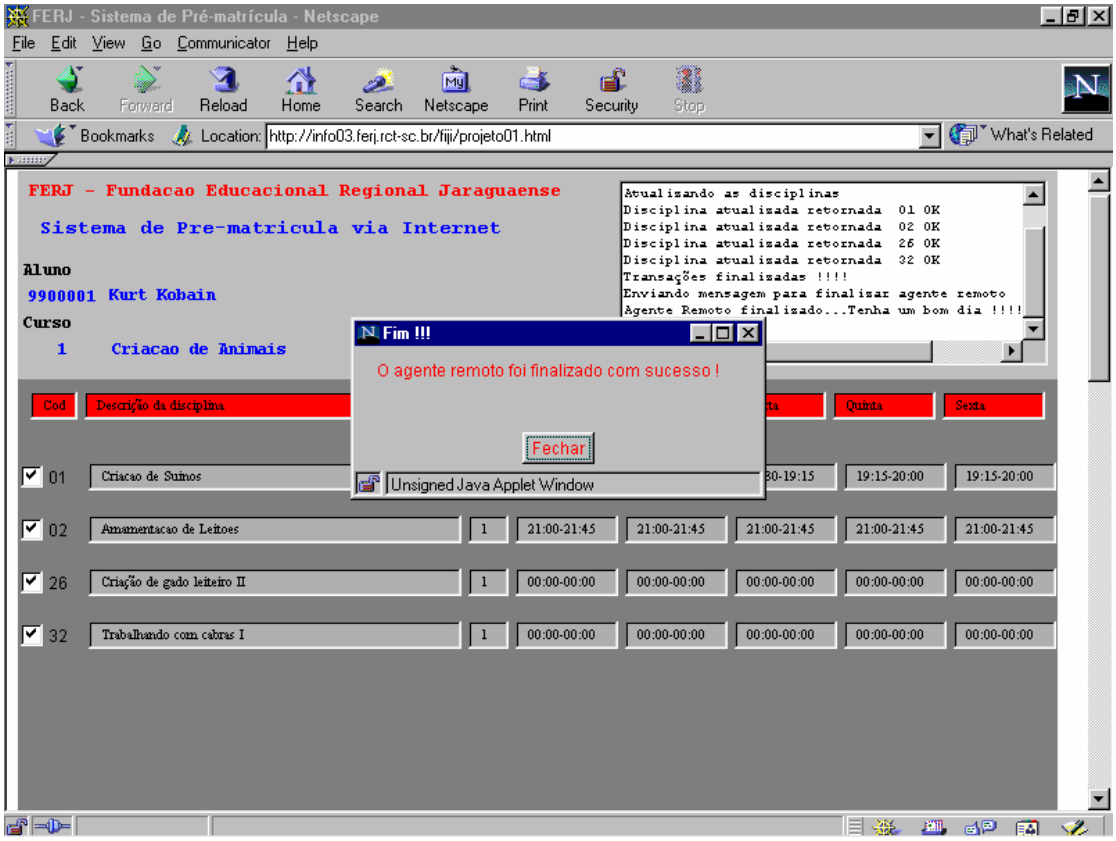

FIGURA 28 : DISCIPLINAS NAS QUAIS O USUÁRIO MATRICULOU-SE

8 CONCLUSÕES E SUGESTÕES

Os objetivos deste trabalho foram alcançados, uma vez que foi implementado o protótipo de uma interface, disponibilizada via *web,* para execução do processo de pré-matrícula na FERJ.

Com este estágio pode-se concluir que o uso de agentes móveis na Internet são uma tecnologia promissora, não somente para efetuar transações em SGBD, mas também em aplicações mais dinâmicas e distribuídas, nas quais é necessário o uso de código móvel através da rede.

Os *aglets* se mostraram como uma tecnologia eficaz na recuperação e alteração de informações em bancos de dados e bastante confiável, bem como a tecnologia *Fiji* que possibilita que os *applets* possam ser usados para criar e executar agentes em seu contexto. Os *aglets* também contribuem para a redução do tráfego na rede. Como no caso estudado, o usuário descarrega para seu *browser* apenas elementos de interface, sendo que os *drivers* JDBC e as classes necessárias para se estabelecer conexão com o banco de dados só serão carregadas pelo agente SGBD depois que ele já se deslocou para a máquina em que se encontra o SGBD, o que supera uma das grandes limitações de um *applet* convencional que precisa descarregar todos estes *drivers* para depois conectar-se ao banco de dados a partir da estação do cliente.

A linguagem Java, juntamente com o JDBC, é a tecnologia que faltava para alavancar o desenvolvimento de sistemas totalmente voltados para a *web* e que possam ser executados pelo usuário de qualquer estação, independente de plataforma, é robusta e confiável, deixando um pouco a desejar em termos de performance, o que promete ser melhorado com as próximas versões da linguagem e com o advento de novas tecnologias de *hardware*. A combinação destas tecnologias com os agente móveis na Internet pode trazer soluções simples e baratas para os problemas encontrados no desenvolvimento de sistemas na Internet que acessem a banco de dados.

O alunos e a própria FERJ serão os grandes beneficiados com o uso de sistemas via Internet para se efetuar processos que só podem ser realizados na instituição. O primeiro só

precisará acessar o site da instituição e poderá ele próprio executar os processos, e o segundo, reduzirá os gastos operacionais com formulários, bem como irá proporcionar melhor qualidade no atendimento .

8.1 DEFICIÊNCIAS

A grande deficiência do protótipo está no fato de o mesmo rodar apenas no Netscape, acima da versão 4.0, não podendo ser executado através do Internet Explorer que possue hoje grande fatia do mercado de *browsers*, isso se deve principalmente a algumas diferenças na especificação do suporte a linguagem Java impostas pelo seus desenvolvedores.

Outra deficiência está no fato de que a última *release* do Fiji *plug-in* está expirada e a IBM ainda não disponibilizou novas versões da mesma.

8.2 MELHORAMENTOS E NOVOS TRABALHOS

Como melhoramentos e continuação para este trabalho sugere-se :

- incorporação de esquemas de segurança e encriptação de dados eficazes;
- aplicação de ferramentas e técnicas para desenvolvimento de interfaces;
- melhoramento do protocolo estabelecido para troca de mensagens entre os agentes.

Outro trabalho possível seria um estudo para verificar em que proporção os agentes reduzem o tráfego na rede em relação às tecnologias tradicionais.

Novos trabalhos podem ser desenvolvidos neste campo, como por exemplo o desenvolvimento de agentes para recuperar informações em sites na *web*, ou seja, implementar agentes que pudessem migrar entre os servidores de páginas *web* até encontrar informações desejadas pelo usuário final. Assim cada provedor deixaria um servidor de

agentes rodando (Tahiti) para que os agentes pudessem se hospedar em sua máquina e recuperar as informações através de algoritmos de pesquisa em páginas HTML.

Outros trabalhos buscando alternativas para disponibilizar serviços via Internet usando agentes poderiam ser desenvolvidos como uma forma de dar continuidade a este protótipo.

Também pode-se fazer um estudo comparativo entre os outros tipos de ambientes diponíveis para a criação de agentes na Internet como o Voyager da ObjectSpace, o Odyssey da General Magic, e outros mais.

9 ANEXOS

9.1 AGENTE SGBD

```
/* Agente SGBD
   Autor Mauri Ferrandin
*/

package com.ibm.fiji.mauri.teste01;

import com.ibm.aglet.*;
import com.ibm.aglet.event.*;
import java.io.*;
import java.net.*;
import java.sql.*;

public class AgletSgbd extends Aglet implements Externalizable,
       MobilityListener, PersistencyListener, CloneListener
{

     static final String SgbdServer = "200.135.24.69:1521";
     // Metodo usado para selecionar dados do banco
     public static String transacao_select(String exp) throws SQLException
     {
      try
      {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn = DriverManager.getConnection
("jdbc:oracle:thin:@"+SgbdServer+":orc1","tccmauri","mauri1607");
        Statement stmt = conn.createStatement ();
        ResultSet rset = stmt.executeQuery(exp);
        ResultSetMetaData rsetm = rset.getMetaData();
              int i;
           String cc = "*";
```

```
          int ncols = rsetm.getColumnCount();
          for (i = 1; i <= ncols ;i++ )
      {
        cc = cc + rsetm.getColumnName(i);
        if (i < ncols)
        {
          cc = cc + ",";
        }
      }
      cc = cc + "*";
      while (rset.next ())
      {
        for (i = 1; i <= ncols ;i++ )
        {
          cc = cc  + rset.getString(i);
          if (i < ncols)
          {
            cc = cc + ",";
          }
        }
        cc = cc + "*";
                    }
      System.out.println("Teste de acesso ao oracle - Escreveu results");
      System.out.println(" mandou o par "+ cc+ "para a var cc");
      conn.close();
      return cc;
    }
    catch(Exception e)
    {
      System.out.println("Erro ao consultar o SGBD");
      return "ERRO_TRANSACAO_BANCO";
    }
  }


// Executa alteracoes no banco de dados
        public static String transacao_update(String exp) throws SQLException
```

```
   {
     try
     {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection conn = DriverManager.getConnection
("jdbc:oracle:thin:@"+SgbdServer+":orc1","tccmauri","mauri1607");
      Statement stmt = conn.createStatement ();
      stmt.executeUpdate(exp);
      conn.close();
     }
    catch(Exception e)
     {
      System.out.println(e.getMessage());
      return "ERRO_TRANSACAO_BANCO";
     }
    return "OK";
   }

  public AgletSgbd() {
        print("Inicializando");
  }

  // Faz o tratamento das mensagens recebidas
  public boolean handleMessage(Message msg) {

   System.out.println("Recebeu msg ");
         try
         {
    // Verifica o tipo da mensagem recebida e executa a procedure correspondente
           if (msg.sameKind("SELECT"))
           {
        System.out.println("recebeu msg select");
             msg.sendReply(transacao_select((String)msg.getArg()));
        }
        else if (msg.sameKind("UPDATE"))
           {
             msg.sendReply(transacao_update((String)msg.getArg()));
```

```
        }
    }
        catch(Exception e)
        {
                System.out.println("Erro no Aglet Slave");
                        System.out.println(e.getMessage());
        }
                return true;
}

public void onCreation(Object o) {
        getMessageManager().setPriority("dialog",MessageManager.ACTIVATE_AGLET);
        print("Creation.");
        addMobilityListener(this);
        addPersistencyListener(this);
        addCloneListener(this);
        print("teste.");
       }

public void onClone(CloneEvent ev) {
        print("Clone.");
}

public void onCloned(CloneEvent ev) {
        print("Clonado.");
}


public void onArrival(MobilityEvent ev) {
        print("Chegando.");
}

public void onActivation(PersistencyEvent ev) {
        print("Ativando.");
}

public void onCloning(CloneEvent ev) {
```

```
        print("Clonando.");
    }

    public void onDispatching(MobilityEvent ev) {
        print("Despachando para " + ev.getLocation() + ".");
    }

    public void onDisposing() {
        print("Desalocando.");
    }

    public void onDeactivating(PersistencyEvent ev) {
        print("Desativando.");
    }

    public void onReverting(MobilityEvent ev) {
        print("Recuperando.");
    }

    private void print(String m) {
        System.out.println("SampleAglet: " + m);
    }

    public void writeExternal(ObjectOutput out) throws IOException {
        print("ReadExternal");
       }

    public void readExternal(ObjectInput in) throws IOException{
        print("WriteExternal");
    }
}
```

QUADRO 31 : FONTES DO AGENTE SGBD

9.2 AGENTE INTERFACE

9.2.1 CLASSE INTERFACE

```
/* Agente Interface
   Autor Mauri Ferrandin
*/

package com.ibm.fiji.mauri.teste01;

import com.ibm.fiji.lib.FijiApplet;
import java.net.*;
import com.ibm.aglet.*;
import com.ibm.aglet.event.*;
import com.ibm.aglets.util.*;
import java.awt.*;
import java.io.*;



public class Interface extends FijiApplet {

  // Armazena URL do SGBD
  final String StrSgbd = "atp://200.135.24.69:434";
  // Classe que corresponde ao agente SGBD
  final String ClassSgbd = "com.ibm.fiji.mauri.teste01.AgletSgbd";
  // Variavel para armazenar o proxy do SGBD quando este for criado
  AgletProxy ProxySgbd = null;
  // Variavel que armazena o retorno das mensagens enviadas
  String retorno;
  // Variavel que armazenará o numero de disciplinas do aluno
  int ndis;
  // Objeto que cria o cabecalho das disciplinas
  Cabecalho cab = new Cabecalho();
  // Array de objetos do tipo disciplinas
  Disciplina[] Disc;
```

```
// Variavel que armazenará o numero de disciplinas selecionadas pelo usuario
int nselected;

// Variavel que armazenará o numero de disciplinas selecionadas pelo usuario
boolean terminou = false;



// Objetos que compoe a interface grafica
Panel pnusuario = new Panel();
Panel pndisciplinas = new Panel();
TextArea textArea1 = new TextArea();
Label label1 = new Label();
Label label2 = new Label();
Label label3 = new Label();
TextField textField1 = new TextField();
TextField textField2 = new TextField();
Button btLimpar = new Button();
Button btExecutar = new Button();
Label lbNomeSistema = new Label();
Label lbFerj1 = new Label();
Label lbTextoAluno = new Label();
Label lbTextoCurso = new Label();
Label lbCodAluno = new Label();
Label lbNomeAluno = new Label();
Label lbCodCurso = new Label();
Label lbNomeCurso = new Label();


// inicializa os componentes graficos
public void init ()
{
  super.init();

        lbNomeSistema.setVisible(true);
  lbNomeSistema.setForeground(Color.blue);
  lbNomeSistema.setFont(new Font("Courier", 1, 15));
```

```
lbNomeSistema.setBounds(new Rectangle(7, 32, 351, 19));
lbNomeSistema.setText("Sistema de Pre-matricula via Internet");
lbNomeSistema.setAlignment(1);
lbFerj1.setVisible(true);
lbFerj1.setForeground(Color.red);
lbFerj1.setFont(new Font("Courier", 1, 13));
lbFerj1.setBounds(new Rectangle(8, 5, 410, 19));
lbFerj1.setText("FERJ - Fundacao Educacional Regional Jaraguaense");
textArea1.setBackground(Color.white);
textArea1.setFont(new Font("Courier", 0, 10));
textArea1.setBounds(new Rectangle(434, 7, 309, 133));
textArea1.setEditable(false);
label1.setAlignment(2);
label1.setText("Codigo");
label2.setText("Senha");
label2.setAlignment(2);
label1.setBounds(new Rectangle(74, 70, 45, 23));
label2.setBounds(new Rectangle(77, 97, 42, 23));
      textField1.setBounds(new Rectangle(125, 70, 101, 24));
textField2.setBounds(new Rectangle(125, 98, 102, 24));
textField2.setEchoChar('*');
btLimpar.setBounds(new Rectangle(300, 100, 73, 28));
btLimpar.setLabel("Limpar");
btExecutar.setBounds(new Rectangle(300, 69, 73, 28));
btExecutar.setLabel("Iniciar");
lbTextoAluno.setVisible(false);
lbTextoAluno.setForeground(Color.black);
lbTextoAluno.setFont(new Font("Courier", 1, 12));
lbTextoAluno.setBounds(new Rectangle(7, 81, 44, 19));
lbTextoAluno.setText("Aluno");
lbTextoCurso.setText("Curso");
lbNomeAluno.setText("Teste de Aluno 12345678901234567890123456789012345678901234567890");
lbCodCurso.setBounds(new Rectangle(50, 100, 100, 19));
lbCodCurso.setText("9900001");
lbNomeCurso.setText("Teste de Aluno 12345678901234567890123456789012345678901234567890");
lbNomeCurso.setBounds(new Rectangle(107, 100, 449, 19));
lbNomeCurso.setFont(new Font("Courier", 1, 12));
```

```
lbNomeCurso.setVisible(false);
lbNomeCurso.setForeground(Color.blue);
lbCodCurso.setFont(new Font("Courier", 1, 12));
lbCodCurso.setVisible(false);
lbCodCurso.setForeground(Color.blue);
lbCodCurso.setAlignment(1);
lbNomeAluno.setBounds(new Rectangle(108, 82, 449, 19));
lbNomeAluno.setFont(new Font("Courier", 1, 12));
lbNomeAluno.setVisible(false);
lbNomeAluno.setForeground(Color.blue);
lbCodAluno.setBounds(new Rectangle(50, 82, 100, 19));
lbCodAluno.setText("9900001");
lbCodAluno.setFont(new Font("Courier", 1, 12));
lbCodAluno.setVisible(false);
lbCodAluno.setForeground(Color.blue);
lbTextoCurso.setBounds(new Rectangle(4, 100, 45, 19));
lbTextoCurso.setFont(new Font("Courier", 1, 12));
lbTextoCurso.setVisible(false);
lbTextoCurso.setForeground(Color.black);
lbTextoAluno.setVisible(false);
lbTextoAluno.setForeground(Color.black);
lbTextoAluno.setFont(new Font("Courier", 1, 12));
lbTextoAluno.setBounds(new Rectangle(4, 62, 39, 19));
lbTextoAluno.setText("Aluno");
lbTextoCurso.setText("Curso");
lbNomeAluno.setText("Teste de Aluno 12345678901234567890123456789012345678901234567890");
lbCodCurso.setBounds(new Rectangle(3, 119, 57, 19));
lbCodCurso.setText("9900001");
lbNomeCurso.setText("Teste de Aluno 12345678901234567890123456789012345678901234567890");
lbNomeCurso.setBounds(new Rectangle(68, 119, 355, 19));
lbNomeCurso.setFont(new Font("Courier", 1, 12));
lbNomeCurso.setVisible(false);
lbNomeCurso.setForeground(Color.blue);
lbCodCurso.setFont(new Font("Courier", 1, 12));
lbCodCurso.setVisible(false);
lbCodCurso.setForeground(Color.blue);
lbCodCurso.setAlignment(1);
```

```
lbNomeAluno.setBounds(new Rectangle(66, 80, 358, 19));
lbNomeAluno.setFont(new Font("Courier", 1, 12));
lbNomeAluno.setVisible(false);
lbNomeAluno.setForeground(Color.blue);
lbCodAluno.setBounds(new Rectangle(3, 81, 57, 19));
lbCodAluno.setAlignment(1);
lbCodAluno.setText("9900001");
lbCodAluno.setFont(new Font("Courier", 1, 12));
lbCodAluno.setVisible(false);
lbCodAluno.setForeground(Color.blue);
lbTextoCurso.setBounds(new Rectangle(4, 100, 38, 19));
lbTextoCurso.setFont(new Font("Courier", 1, 12));
lbTextoCurso.setVisible(false);
lbTextoCurso.setForeground(Color.black);
this.setBackground(SystemColor.controlShadow);
this.setSize(new Dimension(750, 440));
pnusuario.setBackground(Color.lightGray);
pnusuario.setBounds(new Rectangle(0, 0, 750, 150));
pnusuario.setLayout(null);
pndisciplinas.setBackground(Color.gray);
pndisciplinas.setBounds(new Rectangle(0, 149, 750, 1290));
pndisciplinas.setLayout(new FlowLayout());
this.setLayout(null);
this.setSize(750,440);
this.add(pnusuario, null);
pnusuario.add(textArea1, null);
pnusuario.add(textField1, null);
pnusuario.add(textField2, null);
pnusuario.add(btExecutar, null);
pnusuario.add(btLimpar, null);
pnusuario.add(label2, null);
pnusuario.add(label1, null);
pnusuario.add(lbFerj1, null);
pnusuario.add(lbTextoCurso, null);
pnusuario.add(lbNomeCurso, null);
pnusuario.add(lbNomeAluno, null);
pnusuario.add(lbCodCurso, null);
```

```
  pnusuario.add(lbCodAluno, null);
  pnusuario.add(lbTextoAluno, null);
  pnusuario.add(lbNomeSistema, null);
  //cab.setVisible(false);
  pndisciplinas.add(cab);


 // Instacia os objetos disciplina e as adiciona ao pndisciplinas

  this.add(pndisciplinas, null);

  Disc = new Disciplina[19];

  for (int i=0; i<19 ;i++ )
  {
    Disc[i] = new Disciplina();
    Disc[i].setEnabled(false);
   // Disc[i].setVisible(false);
    pndisciplinas.add(Disc[i]);
  }
  this.add(pndisciplinas, null);

}
public void mostraestado(String str)
{
  // Mostra as mensagens no objeto TextArea
  textArea1.appendText(str+ "\n");
}
public void mostraexcecao(Exception excecao)
{
  //Mostra as mensagens de Erro no objeto TextArea
  mostraestado("Ocorreu um erro !!");
  mostraestado(excecao.getMessage());
}
public int nlinhas(String str)
{
  // Retorna o numero de linhas que foram retornadas de uma consulta SQL
```

```
  int count = 0;
  for (int i = 0; i < str.length(); i++ )
  {
    if (str.charAt(i)=='*')
    {
      count ++;
    }
  }
  if (count < 3)
  {
    return 0;
  }
  else
  {
    return count -2;
  }
}

public String getCampos(String str, int lin, int col)
{
  // Retorna um campo dentre uma consulta retornada
  int last = 1;
          int i;

  for (i = 0 ;i < lin ;i++)
  {
    last = str.indexOf("*",last + 1);
  }
  for (i = 1 ;i < col ;i++)
  {
    last = str.indexOf(",",last + 1);
  }

  String ret = "";
  for (i = last ; (i < str.length()) && (str.charAt(i+1) != ',') && (str.charAt(i+1) != '*') ;i++)
  {
    ret = ret + str.charAt(i +1);
```

```
   }
   return ret;
  }

  public boolean handleEvent(Event event)
  {
   if (event.id == Event.ACTION_EVENT) {
    if (event.target instanceof Button) {
      return handleButton((Button)event.target);
    }
   }
   return super.handleEvent(event);
  }

  protected boolean handleButton(Button button)
  {
   // Faz o tratamento de eventos ocorridos em Botoes
   if (button == btExecutar) {
    if (button.getLabel() == "Iniciar")
    {
     if (ProxySgbd == null)
     {
      // cria o agent e o despacha para o servidor de banco de dados remoto
      if (!criaAgenteSgbd())
      {
       DlgMensagem dialog = new DlgMensagem("Erro de comunicação ","Não foi possível
estabelecer comunicação !!!");
        dialog.show();
        return false;
      }

     }
     // Envia mensagens para o agente remoto solicitando os dados do aluno
     mostraestado("Recuperando os dados pessoais do aluno e verificando autenticidade !!");
     retorno = this.consultaAgenteSgbd("SELECT","SELECT
ALUNO.ALUNO_CODIGO,ALUNO.ALUNO_NOME,ALUNO.CURSO_CODIGO,CURS
O.CURSO_DESCRICAO FROM ALUNO ALUNO,CURSO CURSO WHERE
```

```
ALUNO.CURSO_CODIGO = CURSO.CURSO_CODIGO AND
ALUNO.ALUNO_CODIGO = '"+textField1.getText()+"' AND ALUNO.ALUNO_SENHA =
'"+textField2.getText()+"'");

      // Se a consulta retornar 0 linhas -> usuario invalido
      // senao mostra os dados do usuario
      if (retorno == "ERRO_ENVIAR_MENSAGEM")
      {
        mostraestado("Erro de comunicação, contate o responsavel pelo sistema");
        DlgMensagem dialog = new DlgMensagem("Erro de comunicação ","Não foi possível
estabelecer comunicação !!!");
        dialog.show();
        return false;
      }
      if (retorno == "ERRO_TRANSACAO_BANCO")
      {
       mostraestado("Erro no banco de dados, contate o responsavel pelo sistema");
       DlgMensagem dialog = new DlgMensagem("Erro no SGBD ","Não foi possível
executar a transacao  !!!");
       dialog.show();
       return false;
      }
      if (nlinhas(retorno) == 0)
      {
        mostraestado("Codigo do aluno ou senha inválida, informe um código e uma senha
válida !!!");
        DlgMensagem dialog = new DlgMensagem("Usuário inválido !!!","O código ou a
senha inválida !");
        dialog.show();
        return false;
      }
      if (nlinhas(retorno) > 0)
      {
       // Mostra os dados pessoais do aluno

       lbCodAluno.setText(getCampos(retorno,1,1));
       lbNomeAluno.setText(getCampos(retorno,1,2));
```

```
            lbCodCurso.setText(getCampos(retorno,1,3));
            lbNomeCurso.setText(getCampos(retorno,1,4));
            label1.setVisible(false);
            label2.setVisible(false);
            btExecutar.setLabel("Enviar");
            textField1.setVisible(false);
            textField2.setVisible(false);
            lbTextoCurso.setVisible(true);
            lbTextoAluno.setVisible(true);
            lbCodAluno.setVisible(true);
        lbNomeAluno.setVisible(true);
        lbCodCurso.setVisible(true);
            lbNomeCurso.setVisible(true);

        // Envia mensagens para o agente remoto solicitando as disciplinas do aluno
        mostraestado("Recuperando as disciplinas do aluno !!!");
        retorno = this.consultaAgenteSgbd("SELECT","SELECT AD.DISCIPLINA_CODIGO,
D.DISCIPLINA_DESCRICAO,D.DISCIPLINA_FASE,AD.ALUNODIS_SITUACAO,P.PR
OFESSOR_CODIGO,P.PROFESSOR_NOME,H.HORARIO_CODIGO,H.HORARIO_INICI
O,H.HORARIO_FINAL,H.HORARIO_DIASEMANA FROM ALUNODIS AD,
DISCIPLINA D, PROFESSOR P, PROFDISC PD, HORARIO H, HORARIODIS HD
WHERE AD.DISCIPLINA_CODIGO = D.DISCIPLINA_CODIGO AND
P.PROFESSOR_CODIGO=PD.PROFESSOR_CODIGO AND
D.DISCIPLINA_CODIGO=PD.DISCIPLINA_CODIGO AND H.HORARIO_CODIGO =
HD.HORARIO_CODIGO AND HD.DISCIPLINA_CODIGO = AD.DISCIPLINA_CODIGO
AND AD.ALUNO_CODIGO='"+textField1.getText()+"' ORDER BY
D.DISCIPLINA_FASE, D.DISCIPLINA_CODIGO,
H.HORARIO_DIASEMANA,HD.HORARIO_CODIGO");

        if (retorno == "ERRO_ENVIAR_MENSAGEM")
        {
          mostraestado("Erro de comunicação, contate o responsavel pelo sistema");
          DlgMensagem dialog = new DlgMensagem("Erro de comunicação ","Não foi
possível estabelecer comunicação !!!");
          dialog.show();
          return false;
        }
```

```
        if (retorno == "ERRO_TRANSACAO_BANCO")
        {
          mostraestado("Erro no banco de dados, contate o responsavel pelo sistema");
          DlgMensagem dialog = new DlgMensagem("Erro no SGBD ","Não foi possível
executar a transacao  !!!");
          dialog.show();
          return false;
        }
        if (nlinhas(retorno) == 0)
        {
          DlgMensagem dialog = new DlgMensagem("Erro !!!","Nenhuma disciplina foi
encontrada !");
          dialog.show();
          return false;
        }



           // Mostra as disciplinas para o aluno

        this.setSize(750,1440);
        int i = 1;
              Integer cod1, cod2;
              cod2 =  new Integer(0);
           ndis = -1;

           for (i = 1; i <= nlinhas(retorno) ; i++ )
           {
             cod1 =  new Integer(getCampos(retorno,i,1));
             if (cod1.intValue() != cod2.intValue())
             {
               ndis ++;
               Disc[ndis].setEnabled(true);
               Disc[ndis].mudaTexto(getCampos(retorno,i,1),1);
           Disc[ndis].mudaTexto(getCampos(retorno,i,2),2);
               Disc[ndis].mudaTexto(getCampos(retorno,i,3),3);

              // Altera o estado da disciplina
```

```
            Disc[ndis].setChecked(true);
            if (getCampos(retorno,i,4).trim().charAt(0) == 'C')
            {
            Disc[ndis].setChecked(true);
               Disc[ndis].setEnabledcb(false);
            }else if (getCampos(retorno,i,4).trim().charAt(0) == 'M')
            {
          Disc[ndis].setChecked(true);

            }else if (getCampos(retorno,i,4).trim().charAt(0) == 'A')
            {
          Disc[ndis].setChecked(false);
            }

            Disc[ndis].setVisible(true);
            Disc[ndis].setStateInicial();



          }

          // Mostra o horario de cada disciplina

          switch (new Integer(getCampos(retorno,i,10)).intValue())
          {
              case 1 :
                 Disc[ndis].mudaTexto(getCampos(retorno,i,8).substring(0,5)+"-
"+getCampos(retorno,i,9).substring(0,5),4);
                 break;

                    case 2 :
                 Disc[ndis].mudaTexto(getCampos(retorno,i,8).substring(0,5)+"-
"+getCampos(retorno,i,9).substring(0,5),5);
                 break;


              case 3 :
                 Disc[ndis].mudaTexto(getCampos(retorno,i,8).substring(0,5)+"-
```

```
"+getCampos(retorno,i,9).substring(0,5),6);
                    break;

                    case 4 :
                    Disc[ndis].mudaTexto(getCampos(retorno,i,8).substring(0,5)+"-
"+getCampos(retorno,i,9).substring(0,5),7);
                    break;

                  case 5 :
                    Disc[ndis].mudaTexto(getCampos(retorno,i,8).substring(0,5)+"-
"+getCampos(retorno,i,9).substring(0,5),8);
                    break;
             }

             cod2 = cod1;
            }

      // Esconde os objetos disciplina vazios

      for (int p = ndis + 1; p <= 19; p++ )
      {
        Disc[p].setVisible(false);
      }

      }


      }else if (button.getLabel() == "Enviar")
      {
            // Atualizan as disciplinas no banco de dados
            mostraestado("Atualizando as disciplinas...Aguarde !!!");

            nselected = -1;
                  // Esconde as disciplinas
                  for (int i = 0; i < 19 ; i++ )
                 Disc[i].setVisible(false);
```

```
        // Verifica se o aluno alterou e executa a alteracao no banco
        // Prepara para mostrar as disciplinas selecionadas
                for (int i = 0;i <= ndis ;i++ )
                {
                 if (Disc[i].isEnabledcb() == true)
                 {
                   try
                   {
                     if (Disc[i].getStatecb() == true)
                     {
                       nselected++;
                       Disc[nselected].Copia(Disc[i]);
                  Disc[nselected].setEnabled(false);

                       if ((Disc[i].getStatecb() != Disc[i].getStateInicial()))
                       {
                         retorno = this.consultaAgenteSgbd("UPDATE","UPDATE
ALUNODIS AD SET AD.ALUNODIS_SITUACAO = 'M' WHERE
AD.DISCIPLINA_CODIGO = "+Disc[i].getLabelcb()+" AND AD.ALUNO_CODIGO
="'+textField1.getText()+"'");
                mostraestado("Disciplina "+Disc[i].getLabelcb()+" atualizada com sucesso !!!
"+retorno);
              }
            }
            else if (Disc[i].getStatecb() == false)
                     {
                       if ((Disc[i].getStatecb() != Disc[i].getStateInicial()))
                       {
                         retorno = this.consultaAgenteSgbd("UPDATE","UPDATE
ALUNODIS AD SET AD.ALUNODIS_SITUACAO = 'A' WHERE
AD.DISCIPLINA_CODIGO = "+Disc[i].getLabelcb()+" AND
AD.ALUNO_CODIGO='"+textField1.getText()+"'");
                mostraestado("Disciplina "+Disc[i].getLabelcb()+" atualizada com sucesso !!!
"+retorno);
                   }
            }
                     }
```

```
          catch (Exception e)
          {
            mostraexcecao(e);
                        }
                      }
                    }

                    // Mostra as disciplinas que o usuario esta matriculado

                    for (int i = 0 ;i <= nselected ;i++ )
            Disc[i].setVisible(true);

          // Esconde os botoes

          btExecutar.setVisible(false);
             btLimpar.setVisible(false);
                    repaint();

        mostraestado("Transações finalizadas !!!!");
        terminou = true;

                    // Finaliza o Agente Remoto
                    boolean ret = this.finalizaAgenteSgbd();
                    if (!ret)
                    {
                      DlgMensagem dialog = new DlgMensagem("Erro do sistema !!!","Erro ao
finalizar o agente, repita o processo !");
          dialog.show();
                    }else if (ret)
                    {
                      DlgMensagem dialog = new DlgMensagem("Fim !!!","O agente remoto foi
finalizado com sucesso !");
          dialog.show();
                    }
       }
     }else if (button == btLimpar)
     {
```

```
    // Limpa os campos codigo e senha
    textField1.setText("");
    textField2.setText("");
  }
  return true;
}

private boolean finalizaAgenteSgbd()
{
   try
   {
     mostraestado("Enviando mensagem para finalizar agente remoto");
     ProxySgbd.dispose();
     ProxySgbd = null;
   }
   catch (Exception e)
   {
     mostraexcecao(e);
     return false;
   }
   mostraestado("Agente remoto finalizado com sucesso !!!");
   return true;
}

private boolean criaAgenteSgbd()
{
  AgletProxy Proxy_velho = null;
  try
  {
    ProxySgbd = getAgletContext().createAglet(getCodeBase(), ClassSgbd ,null);
    Proxy_velho = ProxySgbd;
    ProxySgbd = ProxySgbd.dispatch(new URL(StrSgbd));
  }
  catch (Exception e)
  {
    if (Proxy_velho != null)
    {
```

```
        try
        {
          Proxy_velho.dispose();
        }
        catch (Exception x)
        {
         ProxySgbd = null;
         return false;
        }
     }
     ProxySgbd = null;
     mostraexcecao(e);
     return false;
  }
  mostraestado("Agente Sgbd despachado para "+StrSgbd);
  return true;
}

private String consultaAgenteSgbd(String str1,String str2)
{
   String ret;
   try
   {
            Message msg = new Message(str1,str2);
          ret = (String)ProxySgbd.sendMessage(msg);
   }
   catch (Exception e)
   {
    mostraexcecao(e);
    return "ERRO_ENVIAR_MENSAGEM";
   }
   return ret;
}


public void stop()
{
```

```
    if ((ProxySgbd != null) && (!terminou))
    {
      DlgMensagem dialog = new DlgMensagem("Erro usuário","O agente foi finalizado");
      dialog.show();
      finalizaAgenteSgbd();
    }

  }


}
```

QUADRO 32 : CÓDIGO FONTE DO AGENTE INTERFACE

9.2.2 CLASSE CABEÇALHO

```
/* Classe cabecalho
   Autor Mauri Ferrandin
*/

package com.ibm.fiji.mauri.teste01;

import java.awt.*;
import java.lang.String;

public class Cabecalho extends Panel
{
 TextField[] l;
 int n;
 String aux;
 String[] str = {"Cod","Descrição da disciplina                    ","F ","Segunda    ","Terça
","Quarta    ","Quinta     ","Sexta      "};

 public Cabecalho()
 {
```

```
   n = str.length;
   l = new TextField[str.length];
   this.resize(10,750);
   this.setLayout(new FlowLayout());
   for (int i = 0; i < str.length; i++) {
     aux = String.valueOf(str[i]);
     l[i] = new TextField(aux,aux.length() -1);
     l[i].setFont(new Font("Serif", 0, 10));
     l[i].setBackground(Color.red);
     this.add(l[i]);
   }
  }

  public void mudaTexto(String str,int pos)
  {
     l[pos - 1].setText(str);
  }

  public void setVisible(boolean e)
  {
   int i;
   for (i = 0;i < n ;i++ )
   {
     if (e)
     {
       l[i].setVisible(true);
     }
     if (!e)
     {
       l[i].setVisible(false);
     }
   }
  }
}
```

QUADRO 33 : FONTES DA CLASSE CABEÇALHO

9.2.3 CLASSE DISCIPLINA

```
/* Classe Disciplina
   Autor Mauri Ferrandin
*/

package com.ibm.fiji.mauri.teste01;

import java.awt.*;
import java.lang.String;

public class Disciplina extends Panel {

 // Aramazena o estado inicial do checkbox para ver se o mesmo foi alterado
 boolean StatecbInicial;
 TextField[] l;
 Checkbox cb;
 int n;
 String aux;
//String[] str =
{"00","000000000000000000000000000000000000000000000000000","00","00:00-
00:00","00:00-00:00","00:00-00:00","00:00-00:00","00:00-00:00"};
 String[] str = {"  ","                                                ","   ","           ","           ","           ","
","           "};

 // Cria os componentes de uma disciplina
 public Disciplina()
  {
    n = str.length;
    l = new TextField[str.length -1];
    cb = new Checkbox(String.valueOf(str[0]),false);
    this.resize(10,750);
    this.setLayout(new FlowLayout());
    this.add(cb);
    for (int i = 0; i < str.length -1; i++) {
      aux = String.valueOf(str[i + 1]);
```

```
    l[i] = new TextField(String.valueOf(str[i + 1]),aux.length() -1);
    l[i].setFont(new Font("Serif", 0, 10));
    l[i].setBackground(Color.gray);

    if (i > 2)
    {
      l[i].resize(10,10);
    }
    this.add(l[i]);
  }
}

// Metodo que copia o estado inicial da disciplina para a
// variavel setStatecbInicial
public void setStateInicial()
{
   StatecbInicial = cb.getState();
}

// Metodo que retorna o estado inicial armazenado na
// variavel setStatecbInicial
public boolean getStateInicial()
{
   return StatecbInicial;
}

// Metodo para copiar o conteudo da disciplina atual
// para outro objeto do tipo disciplina
public void Copia(Disciplina Modelo)
{
  //Copia o codigo da disciplina
  this.mudaTexto(Modelo.getLabelcb(),1);
  //Copia o estado da disciiplina
  this.setChecked(Modelo.getStatecb());
  this.setEnabledcb(Modelo.isEnabledcb());
  //Copia os outros valores
  for (int i = 0; i < str.length -1 ; i++) {
```

```
    this.l[i].setText(Modelo.l[i].getText());
  }
}

// Metodo para alterar o texto do campos que
// compoe a disciplina
public void mudaTexto(String str,int pos)
{
  if (pos == 1)
  {
    switch (Integer.valueOf(str).intValue())
    {
      case 1 : str = "01";
               break;
      case 2 : str = "02";
               break;
      case 3 : str = "03";
               break;
      case 4 : str = "04";
               break;
      case 5 : str = "05";
               break;
      case 6 : str = "06";
               break;
      case 7 : str = "07";
               break;
      case 8 : str = "08";
               break;
      case 9 : str = "09";
               break;
    }

   cb.setLabel(str);
  }

  if (pos > 1)
  {
```

```
    l[pos - 2].setText(str);
  }
}

// Torna a disciplina visive ou invisivel
public void setVisible(boolean e)
{
 int i;
 for (i = 0;i < n -1;i++ )
 {
   if (e)
   {
     l[i].setVisible(true);
     cb.setVisible(true);
   }
   if (!e)
   {
     l[i].setVisible(false);
     cb.setVisible(false);
   }
 }

}

// Habilita o checkbox da disciplina
public void setChecked(boolean e)
{
 cb.setState(e);

}

// Torna o checkbox editavel ou nao
public void setEnabledcb(boolean e)
{
 cb.setEnabled(e);
}
```

```
 // Retorna o estado do checkbox
 public boolean getStatecb()
 {
  return cb.getState();
 }

 // Retorna o texto do checkbox
 public String getTextcb()
 {
  return cb.getLabel();
 }

 // retorna o true se o checkbox estiver enabled
 public boolean isEnabledcb()
 {
  return cb.isEnabled();
 }

 public String getLabelcb()
 {
  return cb.getLabel();
 }

 public void repaint()
 {
  this.repaint();
 }

 public void remove()
 {
  this.remove();
 }

}
```

QUADRO 34 : FONTES DA CLASSE DISCIPLINA

9.2.4 CLASSE DLGMENSAGEM

```
package com.ibm.fiji.mauri.teste01;

import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

public class DlgMensagem extends Frame {



   public DlgMensagem(String tpmsg, String msg) {

      this.setTitle(tpmsg);
      this.setResizable(false);
      this.setForeground(Color.red);
      this.setBackground(SystemColor.inactiveCaptionBorder);


      Panel p2 = new Panel();
      p2.setLayout(new FlowLayout());
      p2.add(new Label(msg));
      add("Center", p2);

      Panel p1 = new Panel();
      p1.setLayout(new FlowLayout());
      p1.add(new Button("Fechar"));
      add("South", p1);

      resize(300, 150);
      this.setLocation(250,225);
      init();
   }
```

```
  public void init(){

    this.addWindowListener(new java.awt.event.WindowAdapter() {
      public void windowDeactivated(WindowEvent e) {
        this_windowDeactivated(e);
      }
    });
  }

  void this_windowDeactivated(WindowEvent e) {
    this.show();
  }


  public boolean action(Event event, Object object) {
      if (object.equals("Fechar")) {
          dispose();
          return true;
      }
      return false;
  }


  public boolean handleEvent(Event event) {
    if (event.id == Event.WINDOW_DESTROY) {
          dispose();
          return true;
      }
      return super.handleEvent(event);
  }
}
```

QUADRO 35 : FONTES DA CLASSE DLGMENSAGEM

REFERÊNCIAS BIBLIOGRÁFICAS

[BEN97] BENETT, Gordon. **INTRANETS, como implantar com sucesso na sua empresa.** Rio de Janeiro: Campus, 1997.

[COC98] COCKAYNE, William R.; ZYDA, Michael. ***Mobile agents: explanations and examples*.** 1998. Endereço Eletrônico: http://www.mannaging.com/Cockayne/.

[DAR97] DARIVA, Roberto Carlos. **Desenvolvimento de aplicações para Internet através de acesso a banco de dados**. Blumenau 1997 : Monografia (Bacharelado em Ciências da Computação) Centro de Ciências Exatas e Naturais, FURB.

[END98] ENDLER, Marcos, **Novos paradigmas de interação usando agentes móveis**. 1998. Instituto de Matemática e Estatísica da USP – São Paulo – Brasil.

[FON94] FONER, Lenny. ***What's an agent, anyway? A sociological case study*.** 1994. Endereço Eletrônico: http://foner.www.media.mit.edu/people/foner/Julia/Julia.html.

[GIL97] GILBERT, Don. ***Intelligent agents: the right information at the right time*.** 1997. Endereço Eletrônico: http://www.networking.ibm.com/iag/iaghome.html.

[HEI97] HEILMANN, kathryn; KIHANYA, Dan; LIGHT, Alastair; MUSEMBWA, Paul. ***Inteligent agents: a technology and business application analisys*.** 1997. Endereço Eletrônico: http://www.mines.unancy.fr/gueniffe/CoursEMN/I31/heilmann/heilmann.html.

[JEP97] JEPSON, Brian. **Programando banco de dados em Java.** São Paulo: Makron Books, 1997.

[KOR95] KORTH, Henry F., SILBERSCHATZ, Abraham. **Sistema de banco de dados.** São Paulo: Makron Books, 1995.

[LAN97] LANGE, Danny B.; OSHIMA, Mitsuru. ***Programming mobile agents in Javatm - with the Java Aglet API***. 1997. Endereço Eletrônico: http://www.trl.ibm.co.jp/aglets.

[LAN98] LANGE, Danny B.; OSHIMA, Mitsuru. ***Mobile agents with Java – the aglet API***. 1998. Endereço Eletrônico: http://www.trl.ibm.co.jp/aglets.

[LEM96] LEMAY, Laura, PERKINS, Charles. **Aprenda em 21 dias Java.** São Paulo: Campus, 1996.

[MEL90] MELENDEZ, Rubens Filho. **Prototipação de sistemas de informação**. Rio de Janeiro: Livros Técnicos e Científicos Editora, 1990.

[OSH97] OSHIMA, Mitsuru; KARJOTH, Guenter. ***Aglets specification (Alpha5) draft*.** 1997. Endereço Eletrônico: http://www.trl.ibm.co/aglets/spec_alpha5.html.

[OSH98] OSHIMA, Mitsuru; KARJOTH, Guenter. **Aglets specification 1.1 draft.** 1997. Endereço Eletrônico: http://www.trl.ibm.co/aglets/spec_11.html.

[PAP98] PAPASTAVROU, Stavros, SAMARAS, George, EVAGGELIA, Pitoura. ***Mobile agents for WWW distributed database access***. 1998. Department of Computer Science University of Cyprus - Cyprus and Department of Computer Science University of Ioannina - Greece.

[SOM97] SOMMERS, Breat. ***Agents : not just for Bond anymore*.** 1997. Endereço Eletrônico: http://www.Javaworld.com/Javaworld/jw-04-1197/jw-04-agents.html

[TIE97] TIETTEL Ed , STEWART, James M.. INTRANET BIBLIA, **Guia completo para implementar, administrar e utilizar uma Intranet.** São Paulo: Berkley Brasil, 1997.

[VEN97] VENNERS, Bill. ***Under the hood: the architecture of aglets*****.** 1997 Endereço Eletrônico: http://Javaworld.com/Javaworld/jw-04-1997/jw-04-hood.html

[WOO95] WOOLDRIDGE, Michael. ***Intelligent agents: theory and practice***. 1995. Endereço Eletrônico: http://www.doc.mmu.ac.uk:80/STAFF/mike/ker95/ker95-html.html.