Universidade Federal da Paraíba – UFPB

Centro de Ciência e Tecnologia – CCT

Departamento Engenharia Elétrica

Relatório de Estágio

Sistema Remoto de Supervisão

Aluno: Andrey Elisio Monteiro Brito

Orientador: Péricles Rezende de Barros

Campina Grande, 20 de março de 2002.

Universidade Federal da Paraíba
Departamento de Engenharia Elétrica
Centro de Ciência e Tecnologia
Laboratório de Instrumentação e Controle

Relatório de Estágio: Sistema Remoto de Supervisão

Aluno: Andrey Elísio Monteiro Brito – Matrícula : 29611015
Email: andrey@dsc.ufpb.br

Orientador: Prof. Péricles Rezende de Barros
Email: prbarros@dee.ufpb.br

Campina Grande – 20 de março de 2002

Biblioteca Setorial do CDSA. Fevereiro de 2021.

Sumé - PB

Sumário

Introdução

Atualmente, é comum a utilização de dispositivos controladores ou supervisores para os mais diversos fins, de simples sensores a sistemas de controle automatizado. Um aspecto importante na utilização destes dispositivos é a monitoração, controle e gerenciamento dos mesmos. Freqüentemente existem funcionários responsáveis pela inspeção ou isto é feito através de redes industriais dedicadas interligadas a sistemas complexos de gerência e supervisão.

A possibilidade de se criar um sistema simples e flexível, capaz de promover acesso fácil a sistemas que possam ser conectados a computadores em rede, mostra-se então, bastante atraente, permitindo o acesso aos sistemas controladores através de redes locais, Intranets ou da própria Internet.

Objetivo

Construir uma interface baseada em Java Applets® para um sistema de supervisão e controle remoto. O sistema tem os seguintes requisitos:

1. Instalação simples no lado servidor.
2. Execução simples, sem necessidade de instalação de componentes extras (necessitar apenas da Máquina Virtual Java) sem dependência de plataforma de hardware ou sistema operacional.
3. Possa ser inserido em uma página HTML.
4. Permitam uma personalização fácil por quem disponibiliza o serviço. Permitindo apresentar mensagens e inserir entradas numéricas de valores para interatividade do cliente, além dos gráficos das variáveis observadas.
5. A integração do servidor com o processo deve ser a mais flexível possível, e por isso a comunicação entre o servidor e o sistema de controle deve ser feita através de arquivos.
6. O sistema deve permitir uma interface bem definida, para uma fácil extensão de funcionalidades e uma evolução posterior.
7. A leitura de valores deve acontecer através de uma fila, onde os valores ainda não lidos permanecem armazenados. Observar que o sistema de controle (ou o mecanismo que conecta o sistema ao microcomputador) é o responsável pelo acréscimo de informação ao arquivo, a fila é implementada, esvaziando este arquivo quando os valores nele contidos forem lidos, desta forma, o sistema de controle é quem deve possuir um limite de tamanho máximo do arquivo de comunicação.

A documentação "**Javadoc**" das classes, que fornece uma consulta rápida a assinatura e descrições de métodos e atributos, muito grande para ser inserida neste relatório, pode ser gerada diretamente executando o comando javadoc sobre os arquivos-fonte de interesse, pois os mesmos estão comentados no padrão requerido.

Tecnologias Utilizadas

O sistema proposto é implementado, no lado do cliente (o terminal que acessa o serviço), através de **Java Applets®**, permitindo a disponibilização de uma interface executada diretamente sobre uma **Máquina Virtual Java** (do inglês, JVM – Java Virtual Machine), ou integrada a uma pagina da World Wide Web e visualizada a partir de um navegador. No lado do servidor (o computador que provê o serviço e troca dados com o sistema de controle), o sistema é implementado através de Servidores Web com suporte a Java Servlets® (se necessário "upload" de parâmetros, caso não haja "uploads", não há necessidade de suporte a Servlets), ou através de um servidor autônomo implementado em Java executando sobre uma JVM no servidor.

Publicação Web

Em 1993 a World Wide Web (WWW) [1] começou a sua explosão de popularidade pela facilidade de divulgação e a capacidade de acesso a conteúdo multimídia através da Internet. Neste caso a utilização de servidores Web para a disponibilização dos serviços ocorreu por este se tratar de um serviço já disponível nos servidores de redes comuns, desta forma, não seria acrescentado serviços ao servidor original, sem custos de processamento, manutenção e segurança adicionais, haja vista, que o servidor Web já deve possuir uma manutenção própria.

É definido no protocolo HTTP (RFC 2616) [2], um método PUT, que suportaria o armazenamento de um arquivo em uma **URI** (Uniform Resource Identifier), que seria utilizador para o envio de parâmetros do usuário para o sistema de controle, entretanto, esse método não é normalmente implementado nos servidores, que prevêem o suporte através de módulos extras específicos para esta tarefa, desta forma optou-se pela utilização de um Servlet, que funcionaria tanto implementando a fila de saída (leitura de valores), com o recebimento dos parâmetros, como para o armazenamento dos parâmetros inseridos pelo usuário no arquivo de comunicação na direção Usuário → Sistema.

Java

Em todo o projeto foi utilizada a linguagem de programação Java [3], sua escolha aconteceu por esta ser uma linguagem que reúne diversos fatores de importância fundamental no desenvolvimento de aplicações modernas. Mesmo que algumas linguagens tenham sido desenvolvidas para tarefas específicas, o Java e as extensões padrão, permitem o desenvolvimento de quaisquer aplicações. Entre as características que se destacam, temos:

- Orientada a objetos: fornece um nível mais alto de abstração, trabalhando com objetos. Um objeto é uma encapsulamento de dados com o código (os métodos) que atuam sobre esses dados, as comunicações e funcionalidades são realizadas a partir de interfaces (e proteções) bem estabelecidas. Na técnica convencional, a programação procedural, o nível de abstração era bem menor, permitindo um pouco mais de flexibilidade a um custo alto de risco e um menor reaproveitamento de módulos.
- Portável: a maioria das linguagens de programação trabalha com compiladores que transformam um código-fonte em uma linguagem de máquina. Esta linguagem de máquina é específica da arquitetura do processador e do sistema operacional. Isso torna sua execução bastante rápida. Outra forma encontrada, é através de interpretadores, que traduzem o código-fonte à medida que ele é executado, desta forma, se permite uma portabilidade do código-fonte, com o custo de um baixo desempenho. Com Java, o código-fonte é traduzido em uma forma intermediária chamada "**Java bytecode**" que permite uma portabilidade para qualquer conjunto sistema operacional+processador que execute uma Máquina Virtual Java, isso permite que o desempenho seja superior ao das linguagens interpretadas e mesmo assim possua uma grande portabilidade. Futuramente o "Java bytecode" poderá ser interpretado diretamente pelos processadores Java em desenvolvimento, permitindo um grande desempenho em sistemas computacionais de pequeno porte.
- Recuperação automática de memória fora de uso (Automatic Garbage Collection), toda memória que não for utilizada é automaticamente desalocada, evitando as preocupações com a alocação e desalocação de memória.
- Segurança: Java fornece diversos mecanismos de segurança de modo a atender as crescentes preocupações com segurança e privacidade.
- Projeto voltado a Internet e redes: A linguagem Java já foi projetada para atender as necessidades crescentes de interconexão. Permitindo, em conjunto com a portabilidade, o desenvolvimento simples de aplicações que utilizem comunicação via rede.

Java Applets

Java Applets [4] são pequenos aplicativos Java que podem ser executados pela Máquina Virtual Java de navegadores. Estes aplicativos são escritos na linguagem Java e compilados em "Java bytecode", formato reconhecido pela Máquina Virtual Java de qualquer sistema operacional. Existem vantagens e desvantagens que precisam ser consideradas para aplicativos Java executando na máquina cliente: a vantagem principal é que um aplicativo Java é independente da plataforma utilizada (roda em Linux, Windows, Mac, etc) e do processador (Intel, Motorola, etc); a maior desvantagem é a necessidade de mais poder de processamento no cliente.

No nosso caso a utilização de Java Applets aconteceu por ser a única possibilidade viável de se manter uma atualização freqüente dos gráficos observados. Desta forma, o Applet, como uma aplicação executando no cliente, mantêm uma conexão com o servidor, recebendo os novos valores e atualizando por si só o gráfico exibido no cliente. Outras formas implicariam na transferência de gráficos completos do servidor (que construiria um novo gráfico para cada valor recebido) para o cliente, causando além de um menor desempenho, um grande acréscimo da quantidade de informações transferidas pela rede e uma grande carga de processamento no servidor.

Java Servlets

Java Servlets [5] são módulos que estender as capacidades de resposta de um servidor, neste caso, servidores Web com suporte a Java. Esses Servlets recebem dados de uma requisição via rede e podem compor, aplicando alguma lógica de processamento, uma resposta dinâmica, ou realizar alguma tarefa no próprio servidor. Eles funcionam então, como um programa que é executado para responder uma requisição feita ao servidor e, funcionando internamente a este, não possuem uma interface.

Devido a crescente importância dos Servlets, diversos fabricantes produzem servidores Web suportando essa tecnologia como alguns produzidos por empresas como Sybase, IBM, Apache, Macromedia, HP, Oracle e Microsoft. E como são baseados na API Servlet, uma extensão padrão do Java, que nada assume sobre o ambiente, protocolo ou arquitetura do servidor, por este motivo podem executar em diferentes tipos de servidor. É uma vantagem sobre métodos mais antigos como o CGI, pois além do melhor desempenho, não é necessário um desenvolvimento voltado a plataforma do servidor.

Arquitetura multicamadas

O número de camadas de um sistema pode ser visto como o número de diferentes módulos que uma requisição ou comando atravessa antes de ser totalmente processado, essas camadas são lógicas, podendo estar todas elas, por exemplo, executando numa mesma máquina. As arquiteturas mais simples consistem em sistemas de uma ou duas camadas, como por exemplo, uma camada sendo a interface cliente e a outra, o servidor. Desta forma, cada cliente trata da interface e da lógica de aplicação e os servidores fornecem os dados. Obviamente qualquer mudança na lógica deve ser atualizada em todos os clientes.

Um sistema mais avançado seria composto de três camadas, por exemplo, uma camada de interface HTML, um servidor Web (que trata da lógica, compondo as páginas) e um servidor de dados. Desta forma alterações na lógica aconteceriam apenas no servidor.

Com a tendência atual de permitir flexibilidade, reutilização, escalabilidade e segurança, surge a necessidade de uma melhor fatoração, de modo a isolar funcionalidade em níveis e permitir utilização de serviços e características implementadas por terceiros para o aprimoramento do sistema.

Desta forma nosso sistema possui uma camada de apresentação, uma camada de comunicação-cliente (essas duas primeiras acessíveis ao desenvolvedor de forma individual, mas constituem a aplicação que executa no lado do cliente), um servidor Web, uma interface de comunicação-controle (que são os arquivos gerenciados pelo sistema operacional da máquina que os hospeda) e o próprio sistema de controle. Desta forma, podemos ter o cliente, utilizando o serviço através de um

navegador Web em uma máquina remota, um servidor Web usufruindo todos os mecanismos de segurança, suporte a falhas e distribuição de carga que forem necessários e um microcomputador de pequeno porte compartilhando um diretório de usuário com o servidor (onde o arquivo de comunicação é armazenado) e conectado via porta serial com o sistema de controle. Quaisquer mudanças podem ser realizadas em quaisquer camadas, desde que respeitadas as interfaces, sem que o resto do sistema precise ser notificado ou atualizado.

Java Archive Tool

De modo a simplificar a manipulação de todas as classes do projeto foi utilizada a ferramenta Java Archive (JAR) [6]. Ela permite que vários arquivos possam ser armazenados (compactados ou não) em um único arquivo e mesmo assim possam ser executados. Essa ferramenta apresenta várias vantagens:

- Os arquivos podem ser transferidos em uma mesma conexão, já que agora estão reunidos em um único arquivo.
- A diminuição do tempo de transferência dos arquivos devido à compactação.
- Os arquivos podem ser selados e digitalmente assinados. Desta forma, Applets assinados podem ter maiores permissões de execução.
- Controle de versão. Em cada arquivo JAR segue um arquivo (que é automaticamente incluído) que pode carregar informações de execução e versão.
- Portabilidade, pois o mecanismo para manipulação desses arquivos está incluso no núcleo da API da plataforma Java.

Pode-se também criar associações (alguns sistemas operacionais criam automaticamente) para executar os arquivos JAR diretamente do gerenciador de arquivos. Um resumo das operações mais básicas pode ser encontrado na tabela abaixo.

Para criar um arquivo	`jar cf` *`arquivo_jar arquivos_entrada`*
Ver o conteúdo	`jar tf` *`jar-file`*
Extrair o conteúdo	`jar xf` *`jar-file`*
Executar (o primeiro executa a classe definida no arquivo de definições internas, o segundo executa a classe especificada).	`java -jar` *`app.jar`* `java -jar` *`app.jar classe_principal`*

No nosso caso o arquivo de informações interno (chamado arquivo de "manifesto"). Diz que execute a classe SelfApp no caso de execução do nosso arquivo JAR. A utilização em Applets requer parâmetros adicionais no código HTML [4].

Softwares necessários

- Servidor → Apache Tomcat: O projeto Jakarta (da Apache Software Foundation [7]) cria e mantém soluções abertas de distribuição pública e gratuita, voltados a plataforma Java. O Jakarta Tomcat é um dos subprojetos, ele é uma implementação das tecnologias Java Servlet e do JavaServer Pages (JSP). O Jakarta Tomcat 4.0.1 suporta as definições Servlet 2.3 e JSP 1.2. [8]

- Cliente → Java Runtime Enviroment (JRE) 1.3.1 (ou versões mais recentes) : consiste na Máquina Virtual Java, as classes básicas da plataforma Java e os arquivos de suporte. É a parte de execução do JDK (Java Development Kit), ou seja, sem os compiladores, depuradores ou ferramentas. É o conjunto mínimo de aplicativos para o suporte a plataforma Java. [9].

Sistema Proposto

Cliente

O cliente é um conjunto de Applets geradores de gráficos (derivados da classe PlotData), um conjunto de formulários (derivados da classe DataForm) e um conjunto de comunicadores (derivados da classe Comunicador).

Servidor composto por um servidor autônomo

O servidor autônomo foi a primeira alternativa a ser estudada, consiste em um programa, executando em um computador em rede, que pode ser o computador que está diretamente conectado ao sistema de controle, ou usufruindo algum mecanismo de compartilhamento de arquivos com o computador diretamente conectado com o sistema de controle. A principal vantagem desta solução é no caso do servidor não possuir um servidor Web em execução, pois os recursos de processamento necessários para executá-lo são muito menores que os recursos necessários para um servidor Web. (Ver Apêndice A para mais detalhes deste servidor)

Como a leitura é realizada por arquivos, então as tarefas desse servidor autônomo são:

- Um servidor simples (apenas uma requisição é atendida por vez), que irá servir os arquivos de dados (implementando também a fila de saída para os dados) e receber os dados da interface que então irá gerar os arquivos de parâmetros.
- Características desse servidor:
  - Implementado em Java, executando como aplicação independente.

- Lê os parâmetros de funcionamento para leitura ou escrita de parâmetros na mesma porta ou em portas diferentes, o que é determinado a partir dos parâmetros de execução do programa. Protocolos definidos para cada uma das duas possibilidades.

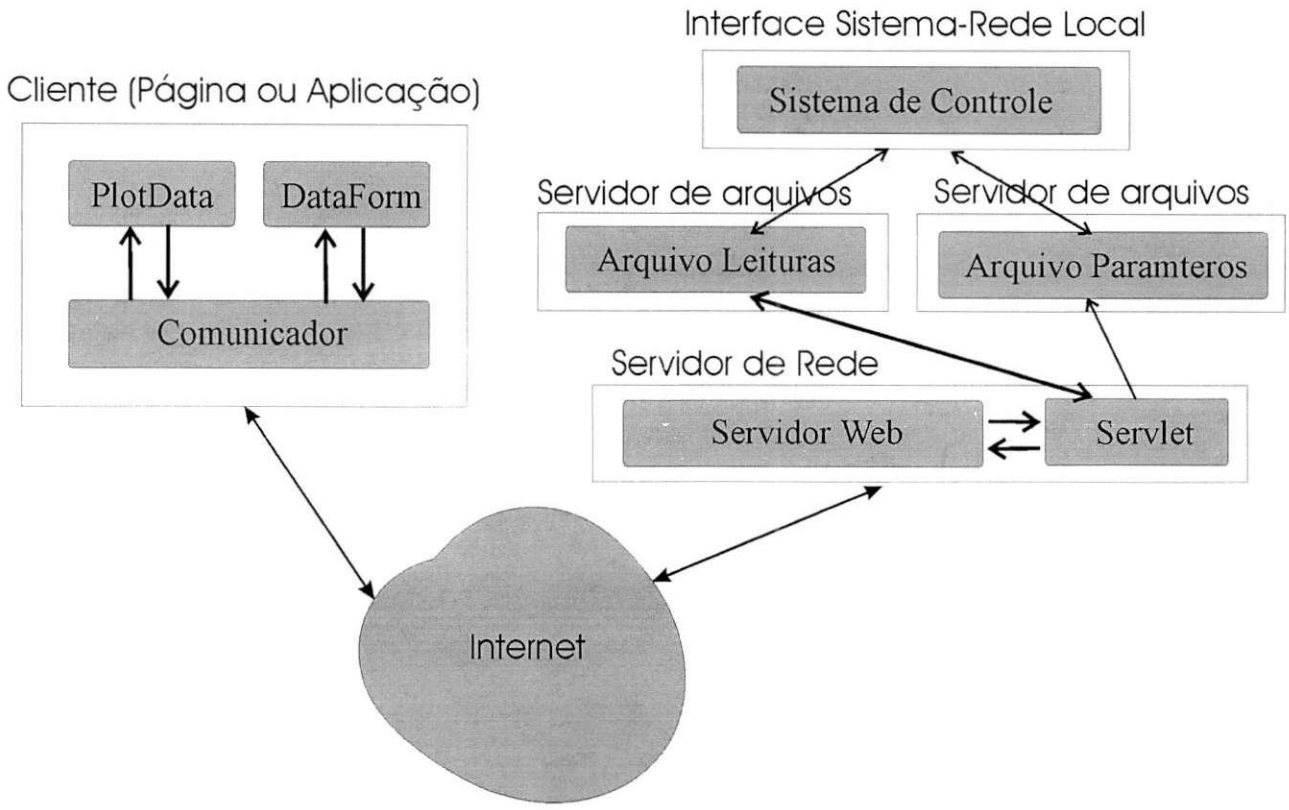
Servidor composto por um servidor Web com suporte a Java Servlets

Essa foi a solução totalmente implementada. Foi utilizado o servidor Web Apache [7] e o Jakarta Tomcat 4.0.1 [8]. O primeiro não possui suporte natural a Servlets, e foi a solução utilizada quando não eram necessários "uploads" de parâmetros. O Tomcat, pode ser utilizado em conjunto com o servidor Web Apache ou sozinho (em conjunto com o Apache, pode utilizar diversos serviços e módulos presentes no mesmo). Neste projeto ele foi utilizado sozinho. Um único recurso foi publicado no serviço de divulgação do Tomcat, um Servlet (ver Apêndice B), que realiza a leitura e escrita nos arquivos de comunicação.

Essa solução permite que o Servlet responsável seja expandido, implementando serviços de comunicação com outros aplicativos, autenticação e muitos outros, pois é implementado a partir de uma extensão padrão do Java e pode utilizar outras extensões do Java para implementar funcionalidades diversas.

Organização do Sistema

Um diagrama do sistema pode ser visto na figura abaixo, as regiões amarelas representam camadas lógicas (varias regiões amarelas podem estar na mesma máquina ou não). As caixas verdes representam módulos, ou funcionalidades, (módulos numa mesma camada devem estar na mesma máquina). No caso do "Servidor de Arquivos" tanto o servidor Web como a "Interface Sistema-Servidor" devem possuir um mecanismo de acesso compartilhado aos arquivos de comunicação (por exemplo, um disco compartilhado NFS numa rede Linux, SMB em redes Windows e Samba em redes mistas).



As classes *PlotData* e *DataForm* são filhas (herdam métodos e atributos) da classe *AppletGrafico*. Esta cuida basicamente da interface entre classes de uso específico e o objeto *Comunicador* associado. Para implementação de novas funcionalidades pode-se estender (por herança) as classes filhas, aproveitando os métodos especializados, gráficos no caso do *PlotData*, e manipulação de componentes no *DataForm*. Ou pode-se estender diretamente *AppletGrafico* (ver documentação Javadoc). Da mesma forma pode-se estender a classe *Comunicador* (ver documentação Javadoc) sem perder a compatibilidade com as classes descendentes de *AppletGrafico*.

Personalização

Um exemplo completo e comentado pode ser encontrado no Apêndice I. Aqui se destacam os aspectos mais importantes na personalização da aplicação, executando como uma aplicação independente, isto é, os Applets executam dentro de uma janela gráfica (Windows ou Linux, ou qualquer ambiente gráfico com uma JVM), e não dentro de um navegador.

O primeiro passo para a personalização é criar uma aplicação de Java gráfica padrão, a classe exemplo é descendente de *JFrame* (janela padrão) e implementa a interface *ActionListener* (tem uma função *actionPerformed*) para que possa reagir a interações em menus.

No construtor dessa classe deve-se criar um objeto Comunicador para os gráficos e um para os formulários (caso necessário). E inicializá-los com os endereços dos servidores, conforme exibido no trecho de código abaixo.

```
public SelfApp() {
.....

// Adicionando o comunicador
//--------------------------

comunicador= new Comunicador();
comunicador_up= new Comunicador();

// Conecta ao servidor, porta e recurso especificado
// um comunicador para os graficos (download apenas) e outro para os
// formularios (upload apenas)

comunicador.init("150.165.60.43",8080,"/servlet/FifoServlet?op=r&",
            "HTTP/1.0");

comunicador_up.init("150.165.60.43",8080,"/servlet/FifoServlet?op=w&",
            "HTTP/1.1");
}
```

O formato da inicialização é *init (endereço_máquina, porta, nome_do_recurso, protocolo)*. No caso do endereço pode ser utilizado o nome ou IP da máquina. A união do endereço, porta e nome do recurso, formam o nome do recurso a ser acessado, neste caso, a **URL** do Servlet, que é http://150.165.60.43:8080/servlet/FifoServlet?op=r& . A URL do Servlet já inclui o tipo de serviço, determinado pelo parâmetro "*op=r*" , outros parâmetros, se necessário, serão adicionados pelo objeto *Comunicador* (o símbolo "&" no fim da URL é o separador de parâmetros). Observar que é fornecida uma URL para "download" e uma para "upload", determinados, respectivamente, pelo *op=r* e *op=w*.

Criar duas rotinas (métodos privados e sem retorno), uma chamada "*adicionaGraficos*" e outra chamada "*adicionaForms*" (estrutura sugerida, não obrigatória, os Applets podem ser inseridos em qualquer fase do programa), em ambas, depois de inseridos os componentes desejados deve-se inicializar o Applet em questão com o método *init (nome_do_Comunicador_associado)*. Na rotina "*adicionaGrafico*" deve-se criar objetos *PlotData* (que contem um gráfico de um ou mais parâmetros) com os parâmetros desejados e depois se personaliza com os métodos:

- *getDataSet (n):* retorna o n-ésimo conjunto de dados (objeto *DataSet* [10], ver Apêndice G) associado ao objeto *PlotData* correspondente. Pode-se então utilizar os métodos e atributos da classe *DataSet* (ver Apêndice G) como por exemplo, legenda, cor de linha, entre outros.

- *getGraph ():* retorna o gráfico (objeto *Graph2D* [10], ver Apêndice H) associado a um objeto *DataPlot*. Através desse objeto podem ser ajustados eixos, bordas, etc.

```
private void adicionaGraficos() {
      PlotData ap;
      ap=new PlotData("Gráfico Seno","VALOR TEMPO");

      DataSet d= ap.getDataSet(1);
      d.legend(1,5, "Senoíde de Referência");
      d.linecolor= new Color(255,0,0);

      // Ajusta a quantidade de espaço no buffer (quantos pontos)
      ap.setTamanhoBuffer(100);
      getContentPane().add(ap.getContentPane());

      ap.init(comunicador);
}
```

A rotina "*adicionaForm*" deve criar objetos tipo *DataForm*, estes podem conter entradas numéricas (especificando-se faixa de valores permitidos e valor inicial) e textos em HTML. O que é feito invocando-se os métodos (da classe *PlotData*):

- *insereEntrada (parâmetro, título, valor mínimo, valor máximo, valor inicial)* : o argumento "parâmetro" se refere ao nome que o objeto *Comunicador* associado a esse objeto *DataForm*, reconhecerá como sendo o identificador deste componente. O título é o título da caixa que contém o componente de entrada e o "*slider*" (componente) associado.

- *insereRotulo (parâmetro, texto)*: o primeiro argumento funciona da mesma maneira que para o item anterior, e o parâmetro "texto" é um texto, que pode ser HTML, que irá constar no formulário. Podendo ser utilizado para informações dinâmicas ou explicação dos controles ou da tela.

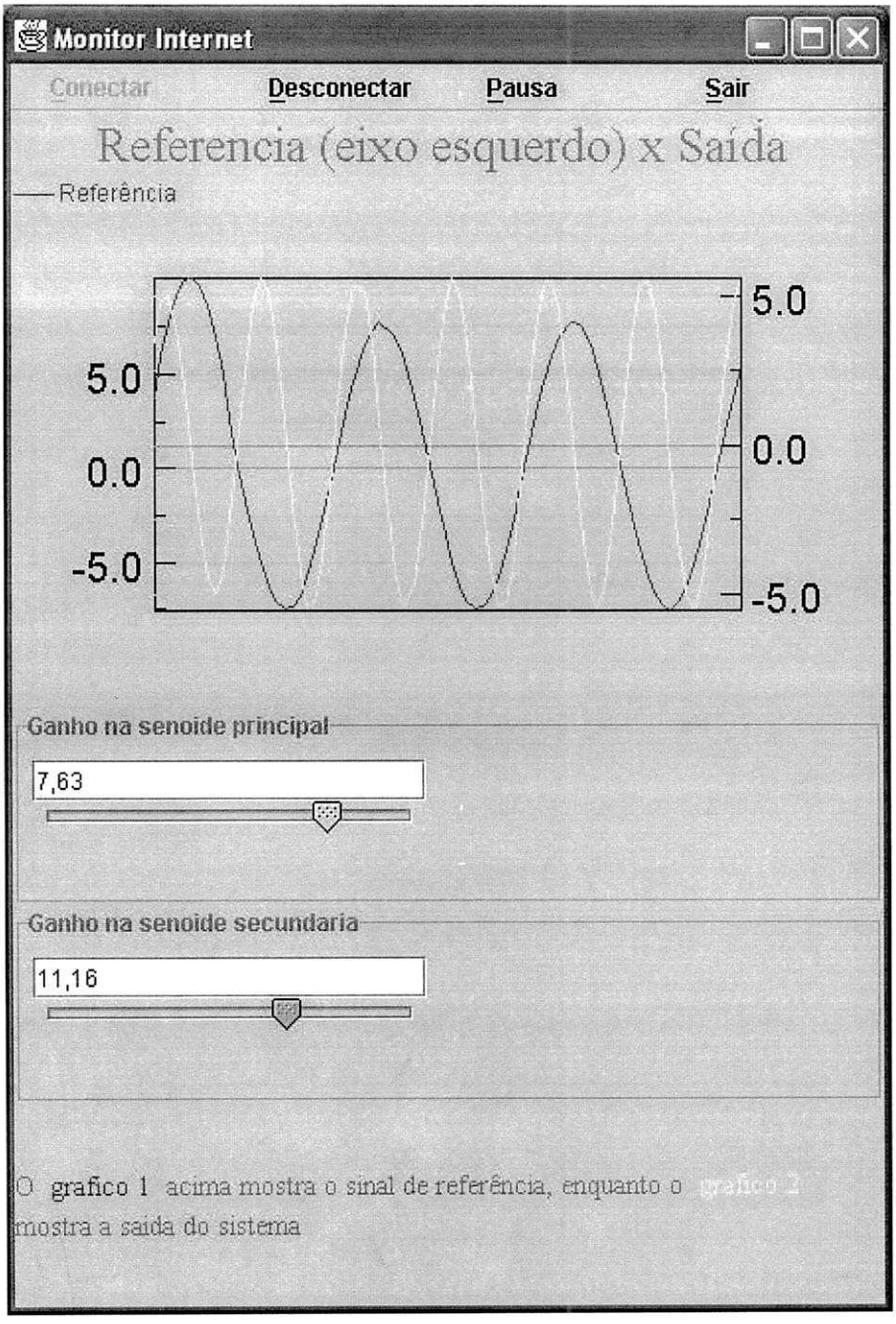
```
private void adicionaForms() {
 DataForm d;

 d=new DataForm();

 d.insereEntrada("GANHO","Ganho na senoide principal", 0, 10, 10);
 d.insereEntrada("GANHO2","Ganho na senoide secundaria", 4, 15, 10);

 d.insereRotulo("DESCRICAO","<html> A senoide principal é a de " +
      "<font color=red> vermelho <font>.");
 getContentPane().add(d.getContentPane());

 d.init(comunicador_up);
}
```

Por fim cria-se o método principal da classe (método "main"), que deverá estabelecer o título e tamanho padrão da janela. Outras coisas podem ser inseridas neste ponto, como janelas de aviso padrão ou configurações adicionais para a janela.

```
public class SelfApp extends JFrame implements ActionListener {
// Declaração de atributos
.....

public SelfApp() { ..... }

public void actionPerformed(ActionEvent e) {....}

private void adicionaGraficos {....}

private void adicionaForms {....}

public static void main(String[] args) {
        SelfApp window = new SelfApp();
        window.setTitle("Monitor Internet");
        window.setSize(450, 660);
        window.setVisible(true);
    }
}
```

Resultados

Um exemplo do programa em execução pode ser visto abaixo.

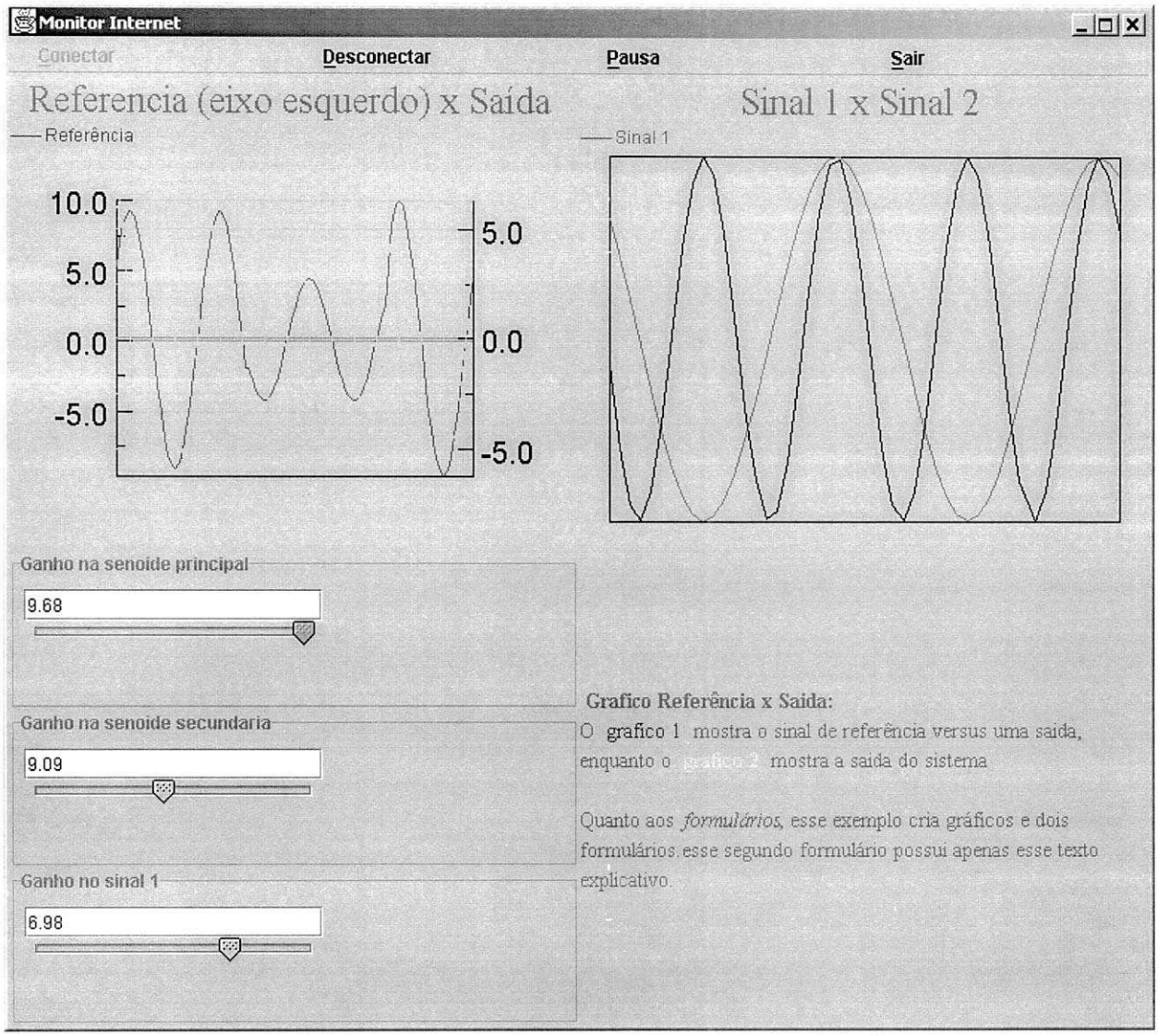


Nele podemos ver um exemplo de todos os componentes. Temos a metade superior da janela formada pelo objeto *PlotData*. Na metade inferior da tela temos duas entradas numéricas e um rótulo. Como descrito pela mensagem, os controles definem a amplitude das senóides amarela e vermelha. Esse exemplo foi executado em um sistema Windows XP.

Os objetos podem ser inseridos em qualquer ordem, e sua organização na tela é feita de conforme o gerenciador de aparência da janela do aplicativo. Isto pode ser visto no código do construtor da classe exemplo. O trecho é mostrado abaixo.

```
public SelfApp() {
….
      // 2 linhas x 1 colunas de componentes
      // Todos com o mesmo tamanho
      getContentPane().setLayout(new GridLayout(2,1));
…..
}
```

Num outro exemplo (executando em um sistema Windows 2000), mostrado acima, temos dois gráficos e dois formulários. Todos eles possuem o mesmo tamanho, pois o gerenciador de aparência ("Layout Manager") é do tipo GridLayout, outros gerenciadores podem ser utilizados, como por exemplo BoxLayout, FlowLayout, entre outros [11].



Toda a informação necessária para a utilização e das classes desenvolvidas pode ser obtida na documentação Javadoc gerada pelos arquivos documentados.

Referências Bibliográficas

1. Internet: World Wide Web Consortium – http://www.w3.org.
2. Internet: Hypertext Transfer Protocol - http://www.w3.org/Protocols/HTTP/.
3. Internet: The Source for the Java Technology – http://java.sun.com.
4. Internet: The Java Tutorial – Trail Applets.
   http://java.sun.com/docs/books/tutorial/applet/index.html.
5. Internet: The Java Tutorial – Trail Servlets.
   http://java.sun.com/docs/books/tutorial/servlets/index.html.
6. Internet: The Java Tutorial, Trail: JAR Files
   http://java.sun.com/docs/books/tutorial/jar/index.html.
7. Internet: The Apache Software Foundation – http://www.apache.org.
8. Internet: Jakarta Tomcat - http://jakarta.apache.org/tomcat/index.html.
9. Internet: The Java 2 Platform - http://java.sun.com/j2se/1.4/.
10. Internet: Java 2D Graph Package Version 2.4 - Graph Class Library -
    http://www.sci.usq.edu.au/staff/leighb/graph/Top.html
11. Internet: The Java Tutorial - Using Layout Managers.
    http://java.sun.com/docs/books/tutorial/uiswing/layout/using.html.

Apêndice A – Servidor composto por um servidor autônomo

O Protocolo (lado do servidor)

O protocolo de alteração de parâmetros

1. Conexão estabelecida.
2. Inicie o temporizador de escuta de conexão.
3. Se uma linha contendo "GET" e "op=w" em quaisquer posições for recebida, vá para o passo 5 (isso implementa compatibilidade com as outras soluções de oferta do serviço).
4. Caso uma linha que não contenha os dois trechos especificados acima tenha sido recebida (e não seja também a linha de início do protocolo para alteração de parâmetros), ou o temporizador tenha expirado, envie uma mensagem de erro pela conexão e encerre-a, adicionando com uma mensagem de erro no arquivo de registro de ocorrências relatando "Solicitacao Invalida".
5. Reinicie o temporizador para o recebimento dos dados.
6. Enquanto o temporizador não atingir o tempo máximo de espera (definido na execução do servidor) e a conexão não for finalizada pelo cliente (o Servlet só trata uma conexão por vez) adicione as linhas ao conjunto de linhas a serem adicionadas ao arquivo de transferências de parâmetros com o sistema de controle. As linhas deverão estar no formato: $PARÂMETRO$ = VALOR (observar que o nome do parâmetro não poderá conter o símbolo '$').
7. Se o tempo máximo de espera for alcançado, descarte o conjunto de linhas e não faça nenhuma alteração no arquivo de comunicação.
8. Se a linha finalizadora for recebida, insira o conjunto de linhas recebidos no arquivo de comunicação, adicione-as também no arquivo de registros e encerre a conexão.

O protocolo de leitura de valores

1. Conexão estabelecida.
2. Inicie o temporizador de escuta de conexão.
3. Se uma linha contendo "GET" ou "op=r" em quaisquer posições for recebida, vá para o passo 5 (isso implementa compatibilidade com as outras soluções de oferta do serviço).
4. Caso uma linha que não contenha os dois trechos especificados acima tenha sido recebida (e não seja também a linha de início do protocolo para alteração de parâmetros), ou o temporizador tenha expirado, envie uma mensagem de erro pela conexão e encerre-a, adicionando com uma mensagem de erro no arquivo de registro de ocorrências relatando "Solicitacao Invalida".
5. Leia o arquivo de dados (gerado pelo sistema de controle) e o transmita. As linhas deverão estar no formato: $PARÂMETRO$ = VALOR (observar que o nome do parâmetro não poderá conter o símbolo '$').

6. Mantenha a conexão aberta, e caso em algum momento se queira encerrar a transmissão, transmita as duas linhas: "Connection: Close", que mantém a compatibilidade com o protocolo HTTP utilizados nos outros casos de servidor, e "%%%FIM%%%", que informa ao cliente que a conexão será fechada e este deverá iniciar uma nova.
7. Caso a conexão seja encerrada pelo cliente, retorne ao estado inicial.

Apêndice B – Servidor composto por um servidor Web com suporte a Java Servlets

O código-fonte do Servlet utilizado nos experimentos é o visto no trecho de código abaixo.

```
/***
*
*  Servlet responsável pela implementação da Fila de saída dos valores lidos.
*
***/

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FifoServlet extends HttpServlet {

        /**
        * Repassa as requisições tipo GET para o método que trata das requisições
        * POST.
        **/
   public void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws IOException, ServletException
   {
      doPost(request, response);
   }

        /**
        * Trata as requisições POST. Dependendo do valor do parâmetro "op" recebido,
        * esse método lê o arquivo de comunicação (com os valores vindos do controlador,
        * caso op=r), ou escreve os todos os parâmetros recebidos em um arquivo predeterminado
        * (caso op=w).
        **/
   public void doPost(HttpServletRequest request, HttpServletResponse res)
                                        throws IOException, ServletException
   {
        // Conjunto de parametros recebidos
        Enumeration parametros = request.getParameterNames();

        // Respectivamente, saída para a conexão e saída para o arquivo de comunição
                PrintWriter saida_usuario = res.getWriter ();
                PrintWriter saida_arquivo = new PrintWriter(new FileOutputStream(
                                "g:\\req.dat", false)); // true=acrescenta no arquivo, false=sobreescreve
```

Código de Servlet – FifoServlet.java

```
                try {
                        // Caso op=w, grave os parametros
                        if (request.getParameter("op").compareTo("w")==0) {
                                while (parametros.hasMoreElements()) {
                String name = (String)parametros.nextElement();
                String value = request.getParameter(name);
                if (name.compareTo("op") !=0 ) {
                        saida_arquivo.println("$"+name+"$"+"= " + value);
                        saida_usuario.println(name + " = " + value); // Eco na conexao
                }
                }
            }

            // Caso op=r, leia o arquivo
                else if (request.getParameter("op").compareTo("r")==0) {
                                int c;
                                String s=new String("d:\\progra~1\\apache~1\\apache\\htdocs\\apps\\buf.dat");
                FileReader bufin = new FileReader(new File(s));

                while ((c = bufin.read()) != -1) {
                        saida_usuario.write(c);
                    }
                    bufin.close();

                                // Depois de lido, apague o conteúdo do arquivo.
                PrintWriter bufout = new PrintWriter(new FileOutputStream(s));

                    bufout.println("%%FIM%%");

                    bufout.close();
                        }

            saida_usuario.close();
            saida_arquivo.close();
          } catch(Exception ex) { // Ignore exceções
                }
        }
}
```

Código do Servlet – FifoServlet.java (cont.)

A partir do mecanismo de herança, este Servlet herda funções-padrão implementadas na classe *HttpServlet*. É necessário portanto apenas um método que construa uma resposta personalizada para este Servlet, utilizando a lógica da aplicação.

Na requisição de páginas (protocolo HTTP [2]), podem ser utilizados dois métodos, o método mais comum é o "*GET*", onde todas as informações que serão repassadas estão inseridas na URL, esse método é bastante comum em mecanismos de busca, por exemplo, e pode ser observado no campo de endereço do navegador. O outro método é o "*POST*" nesse caso, a URL exibida consiste apenas no endereço do próprio recurso e as informações são repassadas internamente na conexão. (colocar os dois exemplos). No nosso caso, caso a requisição chegue através de um método "*GET*" (o método "*doGet*" do Servlet é invocado), a informação é repassada para o método que trataria uma requisição "*POST*" (método "*doPost*"). Ambos os métodos possuem duas estruturas, uma estrutura

tipo "*HttpServletRequest*", que entre outras coisas, armazena os dados repassados junto da requisição, e uma estrutura tipo *HttpServletResponse* que abstrai o fluxo de resposta do Servlet (em direção ao usuário).

Apêndice C – Exemplo de personalização – SelfApp

Este apêndice apresenta um exemplo do um arquivo SelfApp que produziu o segundo exemplo da seção resultados. De posse deste arquivo, seriam necessários apenas as classes do resto do projeto.

Observar que algumas linhas foram quebradas devido às margens de impressão.

```
/*******

        Classe que roda os applets como uma aplicação independente

********/

import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import graph.*;


public class SelfApp extends JFrame
            implements ActionListener {


// Os dois comunicadores, um para leitura e outro para Uploads
        Comunicador comunicador;
        Comunicador comunicador_up;

        // Barra de Menus
        JMenuBar menuBar;
JMenuItem menuItemConectar, menuItemDesconectar, menuItemPausa,menuItemSair;  boolean pausado=false;

  public SelfApp() {


    // Mecanismo para fechar a janela
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {System.exit(0);}
    });


    /************************
        Definição do Menus
    ************************/


    //Cria a barra de menus
    menuBar = new JMenuBar();
    setJMenuBar(menuBar);

    //a group of JMenuItems
    menuItemConectar = new JMenuItem("Conectar",
                KeyEvent.VK_C);
    menuItemConectar.addActionListener(this);
```

```
        menuBar.add(menuItemConectar);


        menuItemDesconectar = new JMenuItem("Desconectar",
                        KeyEvent.VK_D);
        menuItemDesconectar.addActionListener(this);
        menuItemDesconectar.setEnabled(false);
        menuBar.add(menuItemDesconectar);

        menuItemPausa = new JMenuItem("Pausa",
                        KeyEvent.VK_P);
        menuItemPausa.addActionListener(this);
        menuItemPausa.setEnabled(false);
        menuBar.add(menuItemPausa);

        menuItemSair = new JMenuItem("Sair",
                        KeyEvent.VK_S);
        menuItemSair.addActionListener(this);
        menuBar.add(menuItemSair);

        /*****************
                        Adicionando os comunicadores à janela
        ******************/

                        // 2 linhas x 2 colunas de componentes
                        // Todos com o mesmo tamanho
                        getContentPane().setLayout(new GridLayout(2,2));


                        // Adicionando o comunicador
                        //--------------------------

                comunicador= new Comunicador();
                comunicador_up= new Comunicador();

                // Adicionando o contentPane (parte visível) do applet.
                // É mais eficiente que adicionar o applet todo
        // Se quizer exibir a janela de log do comunicador é só descomentar
//              getContentPane().add(comunicador.getContentPane());
//              getContentPane().add(comunicador_up.getContentPane());


                // Conecta ao servidor, porta e recurso especificado
                // um comunicador para os graficos (download apenas) e outro para os formularios
                // (upload apenas)
                comunicador.init("150.165.60.43",8080, "/servlet/FifoServlet?op=r&", "HTTP/1.0");

                comunicador_up.init("150.165.60.43", 8080, "/servlet/FifoServlet?op=w&","HTTP/1.1");

                        // Redefinir o tempo de atualização de 1 segundo (padrão)
                        // Apenas para destacar a funcionalidade
                        comunicador.periodoAtualizacao=1000; // tempo em milisegundos

                        pack();
        }
```

```
    private void adicionaGraficos() {
            // Adicionando os Gráficos
            //-----------------------
            PlotData ap;
            DataSet d;


            // Primeiro gráfico (algumas personalizações a mais)
            //----------------

            ap=new PlotData("Referencia (eixo esquerdo) x Saída ",
                    "VALOR TEMPO FUNC TFUNC2"); // Cria componente com o par de parametros
                                                                                    //
Lembrando que devem ser sempre pares


            ap.init(comunicador);  // Passa a referência do comunicador que está sendo utilizado

            d= ap.getDataSet(1);  // Conjunto de dados associado ao 1o. par de parametrps
            DataSet d2= ap.getDataSet(2);  // Conjunto de dados associado ao 2o. par de parametros

            // Ainda não entendo bem o mecanismo de legenda - Ver graph.DataSet
            d.legend(1,8, "Referência");


            d.linecolor= new Color(255,0,0);  // Cor da linha em RGB
            d2.linecolor= new Color(255,255,0);  // Cor da linha em RGB


            // Os eixos apresentam um funcionamento estranho - Ver graph.Axis
            ap.ajustaEixo(5,-5,1,"ESQUERDO"); // os dois primeiros parametros não estavam fazendo
diferenca
            ap.ajustaEixo(5,-5,2,"DIREITO"); // os dois primeiros parametros não estavam fazendo
diferenca

            Graph2D g=ap.getGraph();
    g.zerocolor = new Color(0,255,0); // Cor diferenciada para o eixo 0
    g.borderTop    = 50;          // espaçamento no topo
    g.borderBottom = 50;          // espaçamento no fim

    // Ajusta a quantidade de espaço no buffer (quantos pontos)
    ap.setTamanhoBuffer(100);

            getContentPane().add(ap.getContentPane()); // Adiciona a parte visivel à janela atual

            // mais um gráfico (algumas personalizações a mais)
            //----------------
            // Cria componente com o par de parametros
            // Lembrando que devem ser sempre pares
            ap=new PlotData("Sinal 1 x Sinal 2 ",
                    "COSSENO TEMPO FUNC2 TFUNC2");


            ap.init(comunicador);  // Passa a referência do comunicador que está sendo utilizado
```

```
                d= ap.getDataSet(1);  // Conjunto de dados associado ao 1o. par de parametros

                // Ainda não entendo bem o mecanismo de legenda - Ver graph.DataSet
                d.legend(1,8, "Sinal 1");


                d.linecolor= new Color(255,0,255);  // Cor da linha em RGB
                d2.linecolor= new Color(125,255,125);  // Cor da linha em RGB

                // Ajusta a quantidade de espaço no buffer (quantos pontos)
                ap.setTamanhoBuffer(50);

                getContentPane().add(ap.getContentPane()); // Adiciona a parte visivel à janela atual

        }


  private void adicionaForms() {
        DataForm d;

        d=new DataForm();

        // Insere duas entradas
        d.insereEntrada("GANHO","Ganho na senoide principal", 0, 10, 10);
        d.insereEntrada("GANHO2","Ganho na senoide secundaria", 4, 15, 10);
        // Insere mais uma entradas
        d.insereEntrada("GANHO3","Ganho no sinal 1", 0, 10, 6.98);

        getContentPane().add(d.getContentPane());
        d.init(comunicador_up);

        d=new DataForm();
        // Insere rótulo explicativo
    d.insereRotulo("LOG2", "<html> <b> Grafico Referência x Saida: </b> <p> O <font color=red> grafico 1 </font>
"+
                " mostra o sinal de referência versus uma saida, enquanto o <font color=yellow> grafico 2 "+
                "</font> mostra a saida do sistema"+
                "<p> <p> Quanto aos <i>formulários</i>, esse exemplo cria gráficos e dois formulários."+
                "esse segundo formulário possui apenas esse texto explicativo.</html>"
                );

        getContentPane().add(d.getContentPane());

        d.init(comunicador_up);

  }


        // Resposta aos eventos do menu principal
  public void actionPerformed(ActionEvent e) {
          JMenuItem source = (JMenuItem)(e.getSource());

          if (source.getText().compareTo("Conectar")==0) {
                comunicador.start();
                comunicador_up.start();
                menuItemDesconectar.setEnabled(true);
```

```
                menuItemConectar.setEnabled(false);
                menuItemPausa.setEnabled(true);
                adicionaGraficos();
                adicionaForms();
                validate();
        }

          if (source.getText().compareTo("Desconectar")==0) {
                comunicador.stop();
                menuItemConectar.setEnabled(true);
                menuItemDesconectar.setEnabled(false);
                menuItemPausa.setEnabled(false);
        }

          if (source.getText().compareTo("Pausa")==0) {
                        if (pausado) {
                                comunicador.start();
                                pausado=false;
                        }
                        else {
                                comunicador.stop();
                                pausado=true;
                        }
    }

          if (source.getText().compareTo("Sair")==0) {
                        System.exit(0);
    }

  }

        // Funcão principal
        public static void main(String[] args) {
    SelfApp window = new SelfApp();

    window.setTitle("Monitor Internet");
    window.setSize(450, 660);
    window.setVisible(true);
  }

}
```

Glossário

Javadoc: Ferramenta que constrói uma documentação em HTML de classes Java, desde que os programas-fonte tenham sido comentados obedecendo as definições Javadoc. Fornecem os valores de retorno, tipos e funções dos parâmetros dos métodos, detalhes sobre os atributos, métodos e atributos herdados de outras classes, entre outras informações. Provida também de mecanismos de índice e procura.

HTTP: Hypertext Transfer Protocol, é o protocolo de transferências de arquivo da Web. Abre conexão com o servidor Web geralmente através da porta 80 (padrão) e pode solicitar mais de um acesso por conexão na versão HTTP/1.0 em diante.

Java bytecode: Código gerado pelo compilador Java, é o formato entendido pela Máquina Virtual Java e futuramente pelos processadores Java.

JVM: É a Máquina Virtual Java (do inglês Java Virtual Machine), é o coração da plataforma Java, interpreta o Java bytecode em linguagem de máquina (específica do processador e do sistema operacional).

Java Plataform: Java 2 Plataform descreve a linguagem java, as bibliotecas e as ferramentas para que outros possam implementar kits de desenvolvimento Java.

URI: URI (Uniform Resource Identifier) é o termo genérico que se utiliza para uma seqüência que define a localização de um recurso. Um URL é um exemplo de um URI.

URL: O URL (Uniform Resource Locator) define a localização de um determinado recurso da Internet. O formato típico de um URL é:

protocol://host:port/path/file.ext

- "protocol" indica ao navegador (ou qualquer outro aplicativo) como ter acesso ao recurso:
  i. file:// recupera um arquivo do sistema local (o seu disco, provavelmente)
  ii. ftp:// utiliza o ftp para recuperar o arquivo
  iii. http:// recupera o arquivo de um servidor WWW utilizando o http
  iv. gopher:// utiliza a aplicação Gopher
  v. telnet:// utiliza uma aplicação Telnet
  vi. news:// utiliza o NNTP para recuperar o conteúdo de grupos de notícias

- "host" é o nome (ou IP) do computador onde se encontra o recurso.
- ":port" (opcional) apenas se inclui quando o hospedeiro (host) não utilizar a porta padrão do protocolo.
- "path" define a localização (na hierarquia de diretórios) do recurso no hospedeiro.
- "file.ext" é o nome do arquivo que se pretende recuperar.Caso não seja indicado um nome de arquivo, os navegadores procuram um arquivo INDEX.HTM ou INDEX.HTML.