INSTITUTO DE ENGENHARIA NUCLEAR

RT-IEN-03/2009

Requisitos básicos para implementação de um sistema de procedimentos computadorizados no laboratório de interfaces homem-sistema - LABIHS

por

Mauro Vitor de Oliveira
Viviane Felix de Carvalho

Novembro/2009

INSTITUTO DE ENGENHARIA NUCLEAR

Título: **Requisitos básicos para implementação de um sistema de procedimentos computadorizados no laboratório de interfaces homem-sistema - LABIHS**				
Autor(es): Mauro Vitor de Oliveira e Viviane Felix de Carvalho				e-mail: mvitor@ien.gov.br
Identificação: RT-IEN-03/2009	Nº de páginas: 69	Tipo de Divulgação: Irrestrita (x) Restrita ( )	Divulgar para:	Localização:
Publicação externa associada (congresso/periódico):				
Palavras chave: Procedimentos computadorizados, simulador, planta industrial.				

Resumo:

O Laboratório de Interfaces Homem-Sistema (LABIHS) é formado por um conjunto de equipamentos e programas de computadores que simulam os processos de uma usina nuclear com 930 MWe de potência, formando um simulador compacto de uma planta nuclear PWR (*Pressurized Water Reactor*) e constituindo uma sala de controle avançada com várias interfaces gráficas representando os vários sistemas que compõem o reator nuclear.

Uma das linhas de pesquisa do LABIHS é o desenvolvimento e avaliação de sistemas de procedimentos computadorizados (PBCs) para salas de controle avançadas de plantas industriais. Este relatório tem como principal objetivo descrever os requisitos básicos para implementação do Cliente PBC a ser implementado para a planta nuclear do simulador do LABIHS bem como descrever o estágio atual de implementação deste sistema no laboratório.

Abstract:

The Human-System Interface Laboratory (LABIHS) is composed by a set of equipment and computational programs that simulate the process of a Pressurized Water Reactor (PWR) of a Nuclear Power Plant (NPP). The nuclear simulator simulates the main circuits of the NPP and is operated by soft panels from a set of computer stations.

The development and evaluation of computerized operation procedures for advanced control rooms is one of research area of the LABIHS. The main objective of this work is to implement a computerized based procedures (CBPs) system in the LABIHS laboratory. The report describes the basic requirements for implement a CBP Client for the LABIHS simulator nuclear power plant as well as the current stage of implementation of this system in the laboratory.

Emissão		Nome	Rubrica	Data
Data: 03/11/2009	Elaboração:	Mauro Vitor de Oliveira		
Divisão: DICH	Revisão:	Paulo Victor Rodrigues de Carvalho		
Serviço: SEESC	Aprovação :	Mauro Vitor de Oliveira		

Instituto de Engenharia Nuclear:
Rua Hélio de Almeida, 75, Cidade Universitária, Ilha do Fundão, CEP 21941-972, CP 68.550, Rio de Janeiro – RJ - Brasil .
Tel.: 00 55 21 2173-3700 Internet: www.ien.gov.br

SUMÁRIO

1. INTRODUÇÃO

1.1 Motivação

Os procedimentos de operação fornecem instruções para guiar os operadores na monitoração, tomada de decisão e controle de uma planta industrial. Historicamente, os procedimentos de operação das plantas têm sido baseados/utilizados em papel (PBPs) e não são considerados como parte da interface homem/sistema (IHS). Contudo, em virtude da tendência mundial de se usar sistemas computadorizados em plantas industriais é natural, e potencialmente vantajoso, instalar procedimentos na forma computadorizada (PBCs). Além disso, após o acidente de *Three Mile Island* (TMI), a indústria de plantas nucleares de potência reconheceu a importância de se usar tecnologias sonora e de procedimentos de fácil utilização para lidar com os principais distúrbios da planta (NUREG-0578, 1979).

Tendo em vista a grande importância desses procedimentos e a complexidade e atualização dos mesmos, viu-se a necessidade da aplicação de técnicas computadorizadas para o desenvolvimento de um sistema de procedimentos baseados em computador (PBCs). No entanto, mesmo que a tendência mundial aponte cada vez mais para a utilização de sistemas computadorizados em plantas industriais, antes de tudo é de fundamental importância verificar se, ao implantar este novo sistema, será possível integrá-lo aos sistemas existentes, garantir a confiabilidade das operações e ainda a preparação da equipe de operadores para a utilização do sistema.

Após estudos realizados com diversos PBCs e uma análise completa sobre o impacto em sua utilização no simulador da planta nuclear do LABIHS, constatamos que os procedimentos de operação das plantas industriais baseados em computador trazem benefícios satisfatórios, tais como: melhor legibilidade e confiabilidade, agilização do acesso e atualização da informação, otimização do tempo da equipe de operação, além da redução de erros, desperdícios e perdas. Tendo em vista que a atualização dos procedimentos em papel são mais complexas e demoradas, qualquer alteração realizada no manual de operação implica em digitá-los novamente, reimprimi-los e redistribuir cópias para os operadores. Portanto, para que seja possível maior agilidade, eficiência e segurança na monitoração, tomada de decisão e controle de uma planta industrial, os procedimentos de operação em papel, utilizados no simulador da planta nuclear do

LABIHS, serão informatizados e a partir disso, poderão ser considerados como parte da interface homem-sistema.

1.2 Objetivos do relatório

Este relatório tem como objetivo principal descrever os passos iniciais para o desenvolvimento de um sistema de procedimentos computadorizados para informatização dos manuais de operação do LABIHS, no que se refere ao desenvolvimento do processo que trabalha a requisição dos serviços operacionais (Cliente) na utilização dos PBCs. O presente relatório também apresenta os passos subsequentes para a informatização dos PBPs, bem como, frisa a necessidade da análise dos requisitos do sistema para garantir a continuidade da implementação do procedimento computadorizado que virá substituir o atual procedimento em papel utilizado no simulador do LABIHS.

1.3 Organização do relatório

Como ponto de partida, a seção 2 deste relatório apresenta a metodologia utilizada para a continuação da implementação do sistema de procedimentos computadorizados no LABIHS. A seção 3 apresenta uma prévia descrição de modelagem e análise, para auxiliar no entendimento da elaboração de projetos computacionais e compreensão de sua inclusão neste relatório. A seção 4 traz o escopo do sistema, ou seja, o conjunto dos requerimentos ou características que definem o projeto. Este item deve ser visto como parte principal na elaboração de um produto de software. Na seção 5 é descrito e detalhado o principal requisito do sistema que está sendo desenvolvido. A seção 6 apresenta os requisitos de software que serão utilizados para o desenvolvimento e execução do sistema. A seção 7 apresenta os requisitos de hardware, citando a configuração mínima para cada plataforma. A seção 8 descreve os requisitos de interface que especifica desde a exibição até as alterações que podem ser feitas nos *layouts* das telas. A seção 9 traz as ferramentas utilizadas tanto na programação quanto na análise exigida pelos padrões de desenvolvimento de software. A seção 10 traz a descrição das telas do sistema e o código-fonte utilizado em sua implementação. Na seção 11 é apresentada a modelagem do caso de uso, ilustrando o cenário do sistema e ainda sua descrição detalhada. A seção 12 descreve o atual estágio do desenvolvimento

do sistema do LABIHS e as conclusões obtidas. O anexo I descreve como deve ser feita a instalação da ferramenta utilizada no desenvolvimento do sistema. No anexo II são apresentados os passos para abrir e executar o sistema.

2. METODOLOGIA

O desenvolvimento de sistemas PBCs é considerado como parte de uma das linhas de pesquisa do LABIHS. Este documento é apresentado para garantir a continuidade da informatização dos manuais de operação baseados em papel e servirá de auxílio no desenvolvimento do sistema de computador que substituirá os atuais PBPs.

Com base na avaliação feita da utilização de um sistema PBC implantado no simulador do LABIHS (OLIVEIRA *et al*. 2009) foram definidas as características principais do sistema PBC a ser desenvolvido.

O sistema será dividido em dois componentes básicos: um servidor e clientes, que se conectarão ao servidor. Estes componentes serão escritos em linguagem Java. Já os procedimentos de operação da planta, a serem lidos pelo servidor e distribuídos para os clientes, serão escritos em XML.

Tendo em vista que CBP apresentam, além dos procedimentos, informações do processo, eles normalmente são integrados com os computadores do processo. A figura 2.1 apresenta como será feita a interligação do sistema de PBC ao simulador do LABIHS, via protocolo de comunicação TCP/IP.

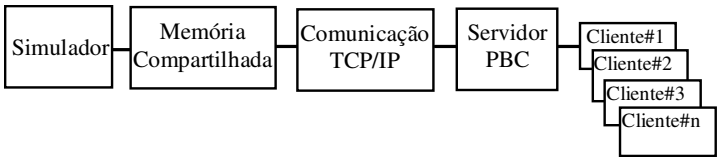


Figura 2.1 - Interligação do simulador ao PBC.

O servidor do sistema PBC gerenciará todos os arquivos que forem acessados pelos clientes. Este gerenciamento central amplia a consistência dos procedimentos. Por exemplo, os membros da equipe não terão diferentes versões dos arquivos de procedimentos. Esse gerenciamento se aplica também aos parâmetros da planta e aos símbolos de dispositivos. O servidor do PBC enviará, sob demanda, essas informações para os clientes PBC. Contudo, cada cliente tem o seu próprio arquivo de configuração informando onde o servidor está residente e o qual é o seu nome como operador.

A comunicação entre o servidor o os clientes é baseada no protocolo TCP/IP. O protocolo de comunicação é projetado especialmente para distribuir os recursos rapidamente e manter a consistência entre clientes e servidor.

O sistema PBC não terá limitações no número de clientes usuários mas eles serão separados em membros da equipe de operação e convidados, quando se conectarem ao servidor. Cada membro da equipe será nomeado conforme sua tarefa, isto é, operador do reator (RO), operador da turbina (TO), operador supervisor (SRO). Além disso, será proibido ao mesmo usuário se conectar como dois clientes ao mesmo tempo e, além disso, somente um servidor PBC poderá rodar em um computador.

Os arquivos de procedimentos serão escritos de acordo com a linguagem PML – *Procedure Markup Language*, que é uma instanciação das especificações gerais do XML 1.0. A base de dados da planta será também escrita em XML.

Finalmente, a base de dados do servidor PBC será dinamicamente atualizada pelos programas externos.

Como ponto inicial para implementação de um sistema de PBC para o simulador do LABIHS, este relatório apresenta as etapas estudadas para concepção, análise de requisitos e diagramas necessários para desenvolvimento da interface dos clientes do sistema. Estas etapas são importantes para garantir a continuidade da implementação do PBC que substituirá os atuais PBPs utilizados no simulador.

Seguindo as fases estabelecidas para o desenvolvimento adequado de um projeto baseado em computador, este relatório trará em seu escopo, juntamente com a respectiva proposta de concepção do sistema, a análise e diagramas necessários ao desenvolvimento da aplicação. Primeiramente, serão descritas as informações fundamentais para o entendimento do programador responsável pelo desenvolvimento do código onde serão definidos os parâmetros do processo de requisição dos serviços e os comandos que atenderão a descrição dos requisitos necessários para o sistema.

3. MODELAGEM E ANÁLISE

A fase referente à modelagem e análise dos requisitos do sistema PBC que vem sendo desenvolvido no LABIHS abrange tanto o desenvolvimento orientado a objetos quanto às tradicionais técnicas de desenvolvimento estruturado.

A modelagem dos diagramas de análise define quem faz o quê, isto é, as ações que serão efetuadas pelos usuários do sistema (operadores) para que seja possível produzir um software de qualidade que atenda às necessidades da equipe de operadores e responsáveis pelo simulador do laboratório.

Para representar as ações do sistema foi modelado o diagrama de *use-case* (caso de uso). O caso de uso especifica o comportamento do sistema ou parte(s) dele e descreve a funcionalidade do sistema desempenhada pelos atores, representando cada uma das principais interações entre o usuário e a máquina.

Segundo JACOBSON *et al*. (1999), podemos dizer que um caso de uso é um *"documento narrativo que descreve a seqüência de eventos de um ator que usa um sistema para completar um processo"*. No caso de uso o que definimos como ator refere-se ao usuário que interage com o sistema e a seqüência dos eventos (*use-case*) está relacionada às ações/interações entre o(s) ator(s) e o sistema.

Esta fase também abrange definir claramente o que o sistema deverá fazer, isso servirá para que o programador responsável pela implementação das funcionalidades do sistema, possua o conhecimento necessário sobre o que deverá ser desenvolvido e o resultado que terá de ser obtido no final.

4. ESCOPO DO SISTEMA

O sistema de procedimentos computadorizados desenvolvido para o LABIHS tem a mesma finalidade que os procedimentos em papel e seguirá parâmetros de programação e comandos semelhantes ao sistema ImPRO (JUNG *et al*., 2004).

O sistema de PBC do LABIHS apresentará um fluxograma posicionado verticalmente, onde será indicado o fluxo gerado a partir de cada passo dado na lógica de ação da operação realizada, para isso o sistema exibirá *"Comentários"* que deverão ser respondidos pelo operador com verdadeiro, falso ou desconhecido, antes de prosseguir com a operação.

Além disso, o sistema deverá exibir em alguns comentários um recurso de resposta automática, o qual poderá ou não ser usado e, após acionado pelo operador, as opções de verdadeiro, falso e desconhecido da *'lista suspensa'*, apresentada no formulário da lógica, serão selecionadas automaticamente pelo sistema, em seguida as respostas fornecidas, tanto pelo operador quanto pelo recurso automático, serão

capturadas e validadas pelo sistema, de acordo com cada comentário respondido na lógica, semelhantemente ao sistema ImPRO.

Paralelamente, o fluxograma indicará cada fluxo equivalente às respostas fornecidas na lógica de operação, o sistema também deverá exibir informações do processo com os valores das variáveis de operação da planta constando o nome, *status*, unidade e descrição, como o sistema estudado.

Finalmente, como diferencial estabelecido para o PBC do LABIHS, deverá ser apresentado um gráfico correspondente aos processos realizados em cada operação, este gráfico garantirá um melhor acompanhamento e aproveitamento dos procedimentos computadorizados realizado no simulador do laboratório.

5. REQUISITOS DO SISTEMA

A seguir são apresentados os requisitos do Cliente PBC a ser implantado no LABIHS.

Requisito	**Descrição detalhada**
Selecionar a opção correspondente a operação realizada (*verdadeiro, falso* ou *desconhecido*).	O operador tem acesso a escolha das respostas para a operação na lógica de ação do sistema, podendo indicar uma das opções apresentadas ou ativar a seleção automática de acordo com as necessidades do momento em que a operação é realizada.
Pós-condição	Visualizar o fluxo apresentado no fluxograma do procedimento a partir da resposta fornecida ao comentário da lógica de sucesso do sistema.
Pré-condição	Acessar o sistema e informar as respostas para cada comentário na lógica de ação.
Passo a passo da ação	O operador irá entrar no sistema e abrirá no menu superior da tela principal o fluxograma e a lógica, onde deverá escolher verdadeiro, falso ou desconhecido, de acordo com o comentário apresentado e a operação que está sendo realizada. O sistema validará as respostas que indicará o fluxo correspondente no fluxograma, as sessões das ações deverão ser guardadas para, ao final, ser gerado um gráfico de acordo com as operações realizadas.
Requisitos rastreados	Envio automático de solução para o problema da operação.
Código do requisito	R1

Requisito	Descrição detalhada
Visualizar o fluxograma de representação da visão geral do procedimento	O operador tem acesso ao fluxograma que apresentará os fluxos de cada ação realizada no procedimento onde são definidos os *checks* e *actions* na forma de árvore de lógica de sucesso
Pós-condição	Visualizar o processo onde todos os passos do procedimento são *renderizados* com o passo de foco no centro da janela
Pré-condição	Informar as ações e verificações na árvore lógica de sucesso, seguir com o procedimento através dos botões de navegação
Passo a passo da ação	O operador irá entrar no sistema e abrirá no menu superior da tela principal o fluxograma e a lógica, onde deverá informar as ações e *checks* na árvore lógica. O sistema validará as respostas que indicará o fluxo correspondente no fluxograma, o usuário pode se familiarizar com os passos do procedimento expandindo e fechando passos ou navegando no fluxograma do procedimento.
Requisitos rastreados	Envio automático de solução para o problema da operação.
Código do requisito	R2

Requisito	Descrição detalhada
Recebe, em tempo real, as informações relativas ao processo de operação	O operador, através da janela de informações do processo, recebe as informações sobre a condição de operação, componentes e de valores de variáveis da planta.
Pós-condição	Visualizar as variáveis do processo em tempo real com os componentes da planta apresentados em forma de símbolos de dispositivos
Pré-condição	Informar as ações e *checks* na árvore lógica de sucesso, seguir com o procedimento através dos botões de navegação
Passo a passo da ação	O operador irá entrar no sistema e abrirá no menu superior da tela principal o fluxograma, lógica e as variáveis, onde paralelamente as ações realizadas nos requisitos R1 e R2, visualizará as informações do processo, em tempo real, sobre a condição de operação, componentes e de valores de variáveis da planta. . Os componentes da planta: bombas, válvulas, etc. terão de ser apresentados na forma de símbolos de dispositivos.
Requisitos rastreados	Envio automático de solução para o problema da operação.
Código do requisito	R3

Requisito	Descrição detalhada
Exibir o gráfico temporal das variáreis dinâmicas	Ao clicar no item gráfico do menu, o operador poderá visualizar automaticamente o gráfico temporal das variáveis dinâmicas do processo.
Pós-condição	Visualizar as informações do processo através do gráfico temporal.
Pré-condição	Depois de realizados os procedimentos de operação, abrir o item gráfico do menu.
Passo a passo da ação	O operador visualizará, ao abrir o item gráfico do menu, uma representação temporal das variáveis obtidas em tempo real a partir dos passos dados no requisito R3. Esta representação é feita através do gráfico temporal baseado nos valores das variáveis dinâmicas.
Requisitos rastreados	Envio automático de solução para o problema da operação.
Código do requisito	R4

6. REQUISITOS DE SOFTWARE

Linguagem de programação

Para o desenvolvimento, a linguagem de programação que vem sendo utilizada é o Java J2SE, correspondente ao aplicativo construído, executado e depurado via *Desktop*.

Software de desenvolvimento

O software de desenvolvimento que vem sendo utilizado trata-se da IDE NetBeans 6.1 (descrita seção 9 deste relatório). Esta ferramenta também requer a instalação do JVM, dado que a IDE só pode ser executada em sistemas operacionais que aceitam a Máquina Virtual Java.

Máquina Virtual Java

Em todas as fases de implementação e para a execução do sistema é necessário que o Java VM (*Java Virtual Machine*) esteja instalado na máquina. Por isso, estamos utilizando o Java Development Kit na versão 7 oferecido gratuitamente pela Sun. Mais conhecido como JDK, o Java Development Kit, que em português significa kit de desenvolvimento Java, é significativamente utilizado pelos programadores de Java.

7. REQUISITOS DE HARDWARE

Os requisitos de hardware foram estabelecidos especialmente para a fase de implementação do sistema, visto que o sistema vem sendo desenvolvido no IDE NetBeans 6.1 que requer as seguintes configurações mínimas de hardware para as plataformas apresentadas abaixo:

Microsoft Windows XP Professional SP2	
Processador	800 MHz Intel Pentium III ou equivalente
Memória	512 MB
Espaço em disco	750 MB de espaço livre em disco

Microsoft Windows Vista	
Processador	800 MHz Intel Pentium III ou equivalente
Memória	512 MB
Espaço em disco	750 MB de espaço livre em disco

Ubuntu 7.x	
Processador	800 MHz Intel Pentium III ou equivalente
Memória	512 MB
Espaço em disco	650 MB de espaço livre em disco

Red Hat EL 4	
Processador	800 MHz Intel Pentium III ou equivalente
Memória	512 MB
Espaço em disco	450 MB de espaço livre em disco

Solaris OS versão 10 (SPARC)	
Processador	UltraSPARC II 450 MHz
Memória	512 MB
Espaço em disco	650 MB de espaço livre em disco

Solaris OS versão 10 (edição plataforma x86/x64)	
Processador	AMD Opteron 1200 Series 1,8 GHz
Memória	512 MB
Espaço em disco	650 MB de espaço livre em disco

Macintosh OS X 10.4.9 Intel	
Processador	Intel Dual Core
Memória	512 MB
Espaço em disco	650 MB de espaço livre em disco

Macintosh OS X 10.4.9 PPC	
Processador	PowerPC G4
Memória	512 MB
Espaço em disco	650 MB de espaço livre em disco

O sistema também pode ser executado nas seguintes plataformas:
Java Desktop System 2
Microsoft Windows 2000 Professional SP4
Mac OS X 10.4.5 Intel
Diversas outras distribuições do Linux como o Red Hat Enterprise Linux

8. REQUISITOS DE INTERFACE

Para a fase de implementação a resolução mínima de tela é 1024 X 768 pixels, visto que, estamos utilizando o IDE NetBeans 6.1. Na interface, já construída para o PBC do LABIHS, é apresenta inicialmente uma tela composta por um menu fixado em sua parte superior (*tela principal*). Este é o recurso onde o operador tem acesso, separadamente, a cada item que deseja visualizar ou com o qual precisa trabalhar no momento do procedimento. Estes itens são: *Fluxograma, Lógica, Variáveis, Gráfico* e ainda *Configuração*, neste ele poderá definir a aparência do layout da tela.

Ao abrir um item do menu o operador poderá verificar que as janelas são exibidas isoladamente na parte interna da tela principal e cada uma destas janelas aparece consecutivamente em um canto distinto da tela. As janelas são exibidas de forma alinhada, lado a lado ou uma abaixo da outra, em tamanhos adequados e inicialmente fixo . Isso foi determinado para permitir que, quando o operador desejar abrir todos os itens do menu, estes sejam visualizados por completo e de maneira apropriada.

Diferentemente de outros PBCs, a tela do sistema de procedimento desenvolvida no LABIHS permite uma total flexibilidade de tamanho, posicionamento, alinhamento e ainda a maximização, minimização e fechamento isolado das janelas internas do sistema. Isso quer dizer que as telas *fluxograma*, *lógica*, *variáveis*, *gráfico* e *configuração* podem ser visualizadas tanto individualmente quanto em conjunto e, se necessário, podem ser maximizadas, minimizadas ou até mesmo fechadas sem influenciar umas nas outras. Além disso, o tamanho e a posição de cada tela podem ser alterados (arrastando o mouse) conforme as necessidades e preferências do operador, garantido assim flexibilidade, personalização e melhor visualização dos recursos de interface do sistema.

9. FERRAMENTAS DE DESENVOLVIMENTO

9.1 Linguagem de programação – Java J2SE (Java Standard Edition)

O J2SE (Java 2 Standard Edition) ou Java SE é uma ferramenta de desenvolvimento para a plataforma Java. Ela contém todo o ambiente necessário para a criação e execução de aplicações Java, incluindo a máquina virtual Java (JVM), o compilador Java, as APIs do Java e outras ferramentas utilitárias.

O Java 2 Standard Edition vem sendo utilizado com Java Development Kit e Java Runtime Environment (JDK+JRE) para desenvolvimento de aplicações Java em *Desktop*. O JRE – Java Runtime Environment significa Ambiente de Tempo de Execução Java, e também é utilizado para executar as aplicações da plataforma Java. Ele é igualmente composto por bibliotecas (APIs) e ainda pela Máquina virtual Java.

9.2 Componentes

9.2.1 Swing

O pacote javax.swing foi criado em 1997 e inclui os componentes GUI que se tornaram padrão em Java a partir da versão 1.2 da plataforma Java 2. A maioria dos componentes Swing (assim são denominados) são escritos, manipulados e exibidos completamente em Java, sendo conhecidos como componentes Java puros. Isso oferece a eles um maior nível de portabilidade e flexibilidade. Os nomes de tais componentes recebem um "J", como, por exemplo: JLabel, JButton, JFrame, JPanel etc.

O Swing também fornece flexibilidade para personalizar a aparência e o comportamento dos componentes de acordo com o modo particular de cada plataforma, ou mesmo alterá-los enquanto o programa está sendo executado. As opções são a personalização com o estilo do Microsoft Windows, do Apple Macintosh ou do Motif (UNIX).

9.2.2 A.W.T

Os componentes GUI oriundos do pacote Abstract Windowing Toolkit (java.awt) tiveram origem na versão 1.0 da plataforma Java 2, e estão diretamente associados com os recursos da interface gráfica da plataforma do usuário. Dessa forma, a aparência dos componentes difere quando o programa é executado no Microsoft Windows e no Apple Macintosh. Podemos dizer que estes componentes considerados peso-pesados "herdam"

a aparência definida pela plataforma, pois o A.W.T. foi projetado para que cada máquina virtual Java implemente seu elemento de interface. Isso pode ser desejável, uma vez que permite aos usuários do programa utilizar os componentes GUI com que eles já estão familiarizados, porém o layout e o alinhamento dos componentes pode se alterar devido aos tamanhos diferentes dos mesmos em cada plataforma.

9.3 NetBeans IDE – versão 6.1

O NetBeans é um projeto que se dedica a prover ferramentas aplicáveis ao processo de desenvolvimento de software e que possam ser empregadas pelos desenvolvedores e demais usuários, entidades e corporações que utilizam estas ferramentas como base para seus próprios produtos.

Esta IDE é um dos melhores ambientes de desenvolvimento integrado disponíveis atualmente para produzir diferentes tipos de aplicação com Java. Ele facilita o trabalho de desenvolvimento de vários tipos de aplicações, desde aquelas que deverão rodar na web até aquelas que deverão rodar em dispositivos móveis.

Segundo informações obtidas no site do NetBeans IDE 6.1, ele é um ambiente de desenvolvimento integrado, modular e baseado em padrões, escrito na linguagem de programação Java. O NetBeans consiste de uma IDE de código aberto (open-source) escrito na linguagem de programação Java e em uma plataforma de aplicativos, que pode ser utilizada como uma estrutura genérica para construir qualquer tipo de aplicativo. Ele só pode ser executado em sistemas operacionais que aceitam a Java VM.

É importante destacar que o NetBeans precisa que o JDK (*Java Development Kit*) já esteja instalado em sua máquina para funcionar. Você sequer conseguirá instalar o NetBeans se o JDK não tiver sido previamente instalado. (Santos, 2007).

9.4 UML *(Unified Modeling Language)*

A UML (*Unified Modeling Language* - Linguagem de Modelagem Unificada) é uma linguagem visual utilizada para modelar sistemas computacionais orientados a objeto. Nos últimos anos, a UML consagrou-se como a linguagem-padrão de modelagem adotada pela indústria de engenharia de software, existindo atualmente um amplo mercado para profissionais que a dominem. (Guedes, 2004)

A UML foi nossa opção, por ser tratar de uma linguagem padrão, além disso, os elementos UML são usados para criar diagramas, que representam uma determinada parte, ou um ponto de vista do sistema.

9.5 JUDE Community – versão 5.0

JUDE - Java e UML Desenvolvedores" Ambiente - JUDE é uma IDE para Modelagem de Dados (UML) criada com Java e de uso fácil e intuitivo. Escolhemos o Jude pois, com ele é possível realizar uma modelagem de dados complexa e o JUDE apresenta os dados para o usuário de forma clara e ainda possuí a vantagem de seu *layout* ser bem intuitivo.

É possível trabalhar com vários diagramas, classes, caso de uso, desenvolvimento etc. São oito tipos diferentes de diagramas: de Classes, Caso de Uso, Estado, Atividade, Seqüência, Colaboração, Componente e Distribuição. É possível criar alguns itens do diagrama de classes apenas clicando com o botão direto do mouse nos itens dos diagramas de casos de uso e selecionando a opção desejada.

10. DESCRIÇÃO DAS TELAS

10.1 Inicial

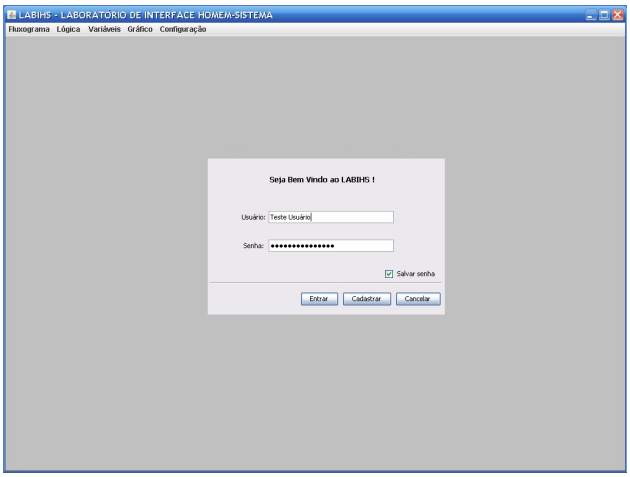


Figura 10.1 – Tela principal com os itens (janela) do menu fechado.

10.1.1 Nome da tela: Inicial

10.1.2 Código da tela: T01

10.1.3 Arquivo: C:\viviane\projetos\Labihs\src\projeto\ProjLabihs.java

10.1.4 Descrição do corpo da tela:

A tela inicial é a tela principal do sistema, onde é exibido o menu e realizado o acesso. Encontra-se inicialmente vazia, pois é aqui que será adicionado o formulário de acesso.

10.1.5 Formulário:

Ainda não foi implementado o formulário de acesso, no entanto, após o término da programação dos requisitos principais do sistema, poderá ser inserido um formulário de acesso localizando-o na parte central do corpo da tela apresentada (figura 10.1).

10.1.6 Regras de validação do formulário:

10.1.6.1 Campo 'usuário': Somente letra (a-z), números (0-9), sublinhado (_), hífen (-) e ponto (.) são permitidos. Tamanho mínimo 6 e máximo de 12 caracteres;

10.1.6.2 Campo 'senha': No mínimo 6 e máximo 12 caracteres, sendo apenas números e letras.

10.1.7 Menu Superior:

- **Fluxograma** – Exibe o fluxograma de visão geral do procedimento;
- **Lógica** – Exibe as *actions* e *checks* na forma de árvore de lógica de sucesso e ainda os botões de navegação no procedimento;
- **Variáveis** – Exibe informações do processo;
- **Gráfico** – Apresenta através de gráficos as informações do processo realizado;
- **Configuração** – Permite ao operador mudar a aparência de cada janela.

10.1.8 Programa da tela:

```
/**
 *
 * @author adm
 */

package projeto;

//importações da aplicação
import projeto.Configuracao.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import projeto.Grafico.Memeater;
```

```
        //nome da classe que instância o JFrame  do pacote  Swing
public class ProjLabihs extends JFrame {
      private JDesktopPane desktop;

        //inicio do método construtor
   public ProjLabihs(){

        //nome do sistema exibido no topo da tela principal
      super("LABIHS - LABORATÓRIO DE INTERFACE HOMEM SISTEMA");

      //cria barra de menus
      JMenuBar barra=new JMenuBar();


 //cria menus
   final JMenu fluxograma = new JMenu("Fluxograma");
   JMenu logica = new JMenu("Lógica");
   final JMenu variaveis = new JMenu("Variáveis");
   JMenu grafico = new JMenu("Gráfico");
   JMenu configuracao = new JMenu("Configuração");

  /*
   //"borda" em torno de cada item do menu
  fluxograma.setBorder(BorderFactory.createLoweredBevelBorder());
  logica.setBorder(BorderFactory.createLoweredBevelBorder());
  variaveis.setBorder(BorderFactory.createLoweredBevelBorder());
  grafico.setBorder(BorderFactory.createLoweredBevelBorder());
  configuracao.setBorder(BorderFactory.createLoweredBevelBorder());
*/

       //cria itens do menu "Fluxograma"
JMenuItem abrirfluxograma=new JMenuItem("Abrir");
JMenuItem sair=new JMenuItem("Sair");

      JMenuItem add = fluxograma.add(abrirfluxograma);
      JMenuItem add1 = fluxograma.add(sair);



  //cria itens do menu "Logica"
JMenuItem abrirlogica=new JMenuItem("Abrir");

      JMenuItem addlogica = logica.add(abrirlogica);

      //cria itens do menu "Variaveis"
JMenuItem abrirvariaveis=new JMenuItem("Abrir");

      JMenuItem addvariaveis = variaveis.add(abrirvariaveis);

            //cria itens do menu "Grafico"
```

```
JMenuItem vergrafico=new JMenuItem("Visualizar");

      JMenuItem addgrafico = grafico.add(vergrafico);

            //cria itens do menu "configuração"
JMenuItem abrirconfiguracao=new JMenuItem("Aparência");

      JMenuItem addconfiguracao = configuracao.add(abrirconfiguracao);


//anexa o menu na barra de menus
barra.add(fluxograma);
barra.add(logica);
barra.add(variaveis);
barra.add(grafico);
barra.add(configuracao);

//anexa a barra de menus a janela do aplicativo
setJMenuBar(barra);

//configura "desktop"
desktop = new JDesktopPane();
desktop.setBackground(Color.lightGray);
desktop.setToolTipText( "Escolha no menu acima o que você desejar visualizar");

   this.getContentPane().add(desktop);

 /*****************************************************************/
      /*INICIO DO CÓDIGO PARA A CLASSE FLUXOGRAMA*/

//configura ouvinte para o item de menu "Abrir"
   abrirfluxograma.addActionListener(


//classe interna para tratar eventos do item de menu "Abrir"
         new ActionListener(){

//exibe janela do fluxograma
   public void actionPerformed(ActionEvent evento){

      //cria a janela
         JInternalFrame frame=new JInternalFrame("Fluxograma",true,true,true,true);

         //obtem painel de conteudo da janela interna
      Container container = frame.getContentPane();
      Fluxograma fluxo = new Fluxograma();

      //anexa ao painel de conteudo da janela interna um objeto da classe "Fluxograma"
      container.add(fluxo,BorderLayout.CENTER);
```

```
//configura o tamanho da janela interna com o tamanho do seu conteudo
   frame.pack();


//anexa a janela interna a "Desktop" e a exibe
desktop.add(frame);

  frame.setLocation(0, 0); //posição x,y
       frame.setSize(new Dimension(420, 715));//tamanho largura, altura
       frame.setVisible(true);

                }
           }
);
 /*FIM DO CÓDIGO PARA A CLASSE FLUXOGRAMA*/

/*********************************************************************/
//BOTAO SAIR
//configura ouvinte para o item de menu "Sair"
sair.addActionListener(

       //classe interna para tratar eventos do item de menu "Sair"
    new ActionListener(){

    //encerra o aplicativo
    public void actionPerformed(ActionEvent evento){
       System.exit(0);
     }
   }
);

/*************************************************************************/
 /*INICIO DO CODIGO PARA A CLASSE LOGICA*/

//configura ouvinte para o item de logica
   abrirlogica.addActionListener(

//classe interna para tratar eventos do item de menu "Abrir"
          new ActionListener(){

       //exibe janela de logica
    public void actionPerformed(ActionEvent evento){

       //cria a janela interna
          JInternalFrame frame=new JInternalFrame("Lógica",true,true,true,true);

          //obtem painel de conteudo da janela interna
       Container container = frame.getContentPane();
       Logica logica = new  Logica();
```

```
      //anexa ao painel de conteudo da janela interna um objeto da classe "Logica"
      container.add(logica,BorderLayout.CENTER);

//configura o tamanho da janela interna com o tamanho do seu conteudo
   frame.pack();

//anexa a janela interna a "Desktop" e a exibe
desktop.add(frame);

 frame.setLocation(420,0); //posiçao x,y
      frame.setSize(new Dimension(600, 240));//tamanho largura, altura
      frame.setVisible(true); //torna visivel

      }
   }
);
 /*FIM DO CODIGO PARA A CLASSE LOGICA*/

/*****************************************************************/

 /*INICIO DO CODIGO PARA A CLASSE VARIAVEIS*/

//configura ouvinte para o item de variaveis
   abrirvariaveis.addActionListener(

//classe interna para tratar eventos do item de menu "Abrir"
         new ActionListener(){


      //exibe janela de variaveis
   public void actionPerformed(ActionEvent evento){

     //cria a janela interna
        JInternalFrame frame=new JInternalFrame("Variáveis",true,true,true,true);

        //obtem painel de conteudo da janela interna
      Container container = frame.getContentPane();
      Variaveis  variavel = new  Variaveis();

      //anexa ao painel de conteudo da janela interna um objeto da classe "Variaveis"
      container.add(variavel,BorderLayout.CENTER);

//configura o tamanho da janela interna com o tamanho do seu conteudo
   frame.pack();

//anexa a janela interna a "Desktop" e a exibe
desktop.add(frame);
           frame.setLocation(420, 240); //posição x,y
           frame.setSize(new Dimension(600, 235));//tamanho largura, altura
          frame.setVisible(true);
```

```
      }
   }
);

 /*FIM DO CODIGO PARA A CLASSE VARIAVEIS*/


/*****************************************************************/

 /*INICIO DO CODIGO PARA A CLASSE GRAFICO*/

//configura ouvinte para o item grafico
   vergrafico.addActionListener(

//classe interna para tratar eventos do item de menu "Abrir"
         new ActionListener(){

      //exibe janela do grafico
   public void actionPerformed(ActionEvent evento){

      //cria a janela interna
         JInternalFrame frame=new JInternalFrame("Gráfico",true,true,true,true);

         //obtem painel de conteudo da janela interna
      Container container = frame.getContentPane();
      Grafico grafico = new  Grafico();

      //anexa ao painel de conteudo da janela interna umo bjeto da classe "Grafico"
      container.add(grafico,BorderLayout.CENTER);

//configura o tamanho da janela interna com o tamanho do seu conteudo
   frame.pack();

//anexa a janela interna a "Desktop" e a exibe
desktop.add(frame);
      frame.setLocation(420, 475); //posição x,y
      frame.setSize(new Dimension(600,240));//tamanho largura, altura
      frame.setVisible(true);

      }
   }
);

 /*FIM DO CODIGO PARA A CLASSE GRAFICO*/

/*****************************************************************/
 /*INICIO DO CÓDIGO PARA A CLASSE CONFIGURACAO*/

//configura ouvinte para o item grafico
   abrirconfiguracao.addActionListener(
```

```
//classe interna para tratar eventos do item de menu "Abrir"
        new ActionListener(){

//exibe janela do configurações
   public void actionPerformed(ActionEvent evento){
    //cria a janela interna
        JInternalFrame frame=new JInternalFrame("Configuração",true,true,true,true);

  //obtem painel de conteudo da janela interna
      Container container = frame.getContentPane();
      Configuracao  configuracao = new  Configuracao();

      //anexa ao painel de conteudo da janela interna um objeto da classe "Configuração"
      container.add(configuracao,BorderLayout.CENTER);

//configura o tamanho da janela interna com o tamanho do seu conteudo
   frame.pack();

  //anexa a janela interna a "Desktop" e a exibe
desktop.add(frame);
          frame.setVisible(true);
      }
   }
);

 /*FIM DO CÓDIGO PARA A CLASSE CONFIGURACAO*/

/************************************************************/

//determina o tamanho da janela do aplicativo
   setSize(1024,768); //largura, altura

//determina que o conteudo anexado a janela seja exibido
   setVisible(true);
   }

      //permite as alterações na dimensão
     public Dimension getMinimumSize() {
         return getPreferredSize();
      }
      public Dimension getMaximumSize() {
         return getPreferredSize();
      }
      public Dimension getPreferredSize() {
         return new Dimension(135,80);
      }
```

```
//executa a aplicação
   public static void main(String args[])
   {

      ProjLabihs pl = new ProjLabihs();
      pl.setDefaultCloseOperation(
      JFrame.EXIT_ON_CLOSE);

    }
  }
```

10.2 Fluxograma

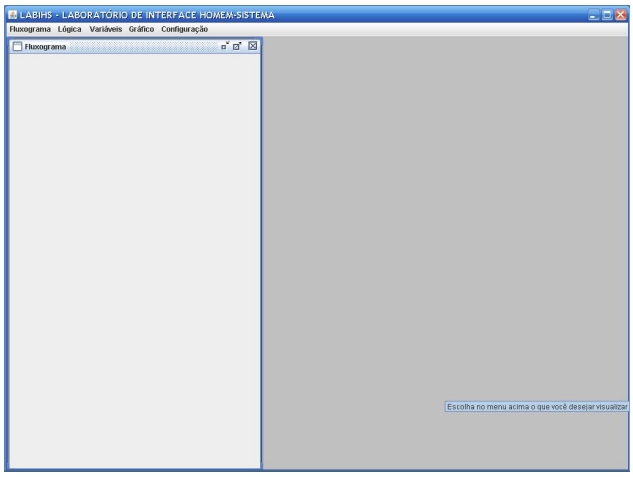


Figura 10.2 – Tela com a janela onde será apresentado o fluxograma dos procedimentos.

10.2.1 Nome da tela: Fluxograma
10.2.2 Código da tela: T02
10.2.3 Arquivo: C:\viviane\projetos\Labihs\src\projeto\Fluxograma.java
10.2.4 Descrição do corpo da tela:

No corpo desta tela é apresentada a janela do fluxograma de visão geral do procedimento, semelhante ao ImPRO. A próxima etapa do desenvolvimento do PBC se baseia na implementação do código para execução das funcionalidades do sistema para o fluxograma, lógica, variáveis e gráfico, onde cada funcionalidade deverá está interligada as outras via código-fonte.

10.2.5 Programa da tela:

```
package projeto;

import java.awt.Dimension;
 import javax.swing.*;


public class Fluxograma extends JPanel{
```

```
  public Dimension getMinimumSize() {
      return getPreferredSize();
    }

    public Dimension getMaximumSize() {
      return getPreferredSize();
    }

    public Dimension getPreferredSize() {
      return new Dimension(135,80);
    }


         public Fluxograma()
{ //aqui entra o código para o fluxograma

       }

    public static void main ( String args[ ] ) {
       Fluxograma f = new Fluxograma();
 f.print("Fluxograma dos Procedimentos " );
    }

private void print(String string) {
    throw new UnsupportedOperationException("Não implementado");
}

    }
```

10.3 Lógica

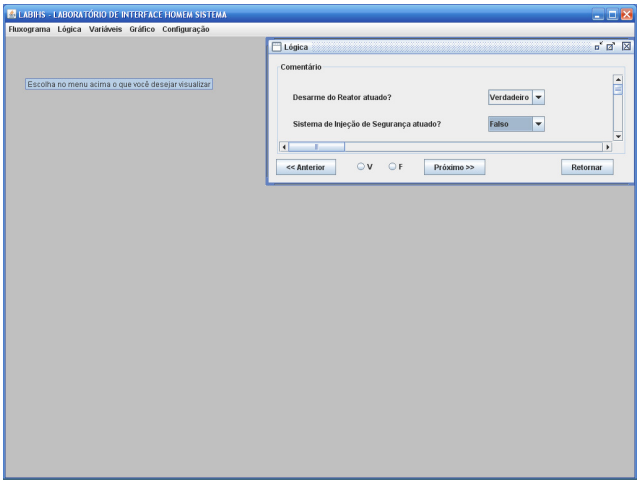


Figura 10.3 – Tela com a janela da lógica de sucesso e botões de navegação no procedimento.

10.3.1 Nome da tela: Lógica

10.3.2 Código da tela: T03

10.3.3 Arquivo: C:\viviane\projetos\Labihs\src\projeto\Logica.java

10.3.4 Descrição do corpo da tela:

Nesta tela é apresentada a janela "Lógica" onde o operador obtém a informação sobre a lógica realizada em cada ação (*Action*) ou verificação (*Check*) do procedimento. O código que será construído trará um modelo semelhante ao ImPro que utiliza em sua lógica booleana estados verdadeiro, falso e desconhecido. (Figura 10.3)

10.3.5 Ações:

Para realizar com sucesso uma ação ou uma verificação o usuário configura a condição de entrada de cada parâmetro utilizado na lógica de sucesso verdadeiro, falso ou desconhecido. O operador também pode optar por ativar o modo automático, onde as escolhas serão realizadas 'roboticamente' pelo sistema.

10.3.6 Botões:

Na parte inferior da janela são exibidos os botões de navegação no procedimento, através dos quais o operador pode navegar e realizar cada ação ou verificação, como é descrito abaixo:

- **<<Anterior** – Botão utilizado para voltar à ação ou verificação precedente;
- **Próximo>>** – Com este botão o operador pode ir para a ação ou verificação seguinte;
- **Retornar** – O operador pode retornar para a ação ou verificação e selecionar o caminho a ser seguido (verdadeiro ou falso).

10.3.7 Programa da tela:

```
/*
 * Logica2.java
 *
 * Created on 19 de Novembro de 2008, 11:25
 */

package projeto;

import java.awt.Dimension;

/**
```

```
 *
 * @author Viviane
 */

public class Logica extends javax.swing.JPanel {

    public Logica() {

        initComponents();

    }

     @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jScrollPane1 = new javax.swing.JScrollPane();

        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();

        jComboBox1 = new javax.swing.JComboBox();
        jComboBox2 = new javax.swing.JComboBox();
        jComboBox3 = new javax.swing.JComboBox();

        jButton1 = new javax.swing.JButton();

        jRadioButton1 = new javax.swing.JRadioButton();
        jRadioButton2 = new javax.swing.JRadioButton();

        jButton2 = new javax.swing.JButton();
        jButton3 = new javax.swing.JButton();

        jScrollPane1.setBorder(javax.swing.BorderFactory.createTitledBorder("Comentário"));

        jLabel1.setText("Desarme do Reator atuado?");

        jLabel2.setText("Sistema de Injeção de Segurança atuado?");

        jLabel3.setText("jLabel3");

        jComboBox1.setModel(new  javax.swing.DefaultComboBoxModel(new  String[] { "Verdadeiro",
"Falso", "Desconhecido"}));

        jComboBox2.setModel(new  javax.swing.DefaultComboBoxModel(new  String[] { "Verdadeiro",
"Falso", " Desconhecido" }));
```

```
    jComboBox3.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Verdadeiro",
"Falso", " Desconhecido" }));

    javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
    jPanel1.setLayout(jPanel1Layout);
    jPanel1Layout.setHorizontalGroup(
       jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
       .addGroup(jPanel1Layout.createSequentialGroup()
          .addGap(23, 23, 23)


.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)

             .addComponent(jLabel3, javax.swing.GroupLayout.Alignment.LEADING)
             .addComponent(jLabel2, javax.swing.GroupLayout.Alignment.LEADING)
             .addComponent(jLabel1, javax.swing.GroupLayout.Alignment.LEADING))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,          76,
Short.MAX_VALUE)


.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)

             .addComponent(jComboBox1,          javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
             .addComponent(jComboBox2,          javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
             .addComponent(jComboBox3,          javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
          .addContainerGap(2563, Short.MAX_VALUE))
    );

    jPanel1Layout.setVerticalGroup(
       jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
       .addGroup(jPanel1Layout.createSequentialGroup()
          .addGap(24, 24, 24)


.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

             .addComponent(jLabel1)
             .addComponent(jComboBox1,          javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
          .addGap(18, 18, 18)


.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

             .addComponent(jLabel2)
             .addComponent(jComboBox2,          javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
            .addGap(18, 18, 18)


.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

                .addComponent(jLabel3)
                .addComponent(jComboBox3,                javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(2641, 2641, 2641))
        );


        jScrollPane1.setViewportView(jPanel1);


        jButton1.setText("<< Anterior");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jRadioButton1.setText("V");

        jRadioButton2.setText("F");

        jButton2.setText("Próximo >>");

        jButton3.setText("Retornar");

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
        this.setLayout(layout);

        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jScrollPane1,      javax.swing.GroupLayout.DEFAULT_SIZE,      484,
Short.MAX_VALUE)
                        .addContainerGap())
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jButton1)
                        .addGap(30, 30, 30)
                        .addComponent(jRadioButton1)
                        .addGap(18, 18, 18)
                        .addComponent(jRadioButton2)
                        .addGap(30, 30, 30)
                        .addComponent(jButton2)
```

```
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,          52,
Short.MAX_VALUE)
                .addComponent(jButton3)
                .addGap(33, 33, 33))))
        );

        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jScrollPane1,          javax.swing.GroupLayout.DEFAULT_SIZE,          500,
Short.MAX_VALUE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,          52,
Short.MAX_VALUE)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jButton2)
                    .addComponent(jRadioButton2)
                    .addComponent(jRadioButton1)
                    .addComponent(jButton3)
                    .addComponent(jButton1))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
    }// </editor-fold>

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

//adicionar o código aqui

}

        // Declaração de variaveis - não modifique
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JButton jButton3;
    private javax.swing.JComboBox jComboBox1;
    private javax.swing.JComboBox jComboBox2;
    private javax.swing.JComboBox jComboBox3;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JRadioButton jRadioButton1;
    private javax.swing.JRadioButton jRadioButton2;
    private javax.swing.JScrollPane jScrollPane1;

    // End of variables declaration
}
```

10.4 Variáveis

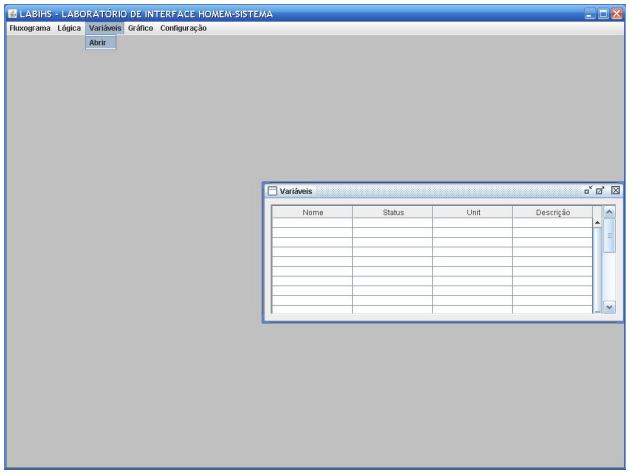


Figura 10.4 – Tela exibindo a janela das variáveis do procedimento.

10.4.1 Nome da tela: Variáveis

10.4.2 Código da tela: T04

10.4.3 Arquivo: C:\viviane\projetos\Labihs\src\projeto\Variaveis.java

10.4.4 Descrição do corpo da tela:

Nesta janela o operador visualizará a tabela de dados do processo onde cada campo pode ser deslocado. Através do código a ser implementado, aqui serão exibidas informações sobre a condição de operação dos componentes e de valores de variáveis da planta.

10.4.5 Programa da tela:

```
/**
 *
 * @author Viviane
 */

package projeto;

import java.awt.Dimension;

public class Variaveis extends javax.swing.JPanel{

    public Dimension getMinimumSize() {
        return getPreferredSize();
     }

     public Dimension getMaximumSize() {
        return getPreferredSize();
```

```
    }

    public Dimension getPreferredSize() {
       return new Dimension(135,80);
    }

    public Variaveis()
  {
    initComponents();
  }

    @SuppressWarnings("unchecked")
  // <editor-fold defaultstate="collapsed" desc="Generated Code">
  private void initComponents() {

    scrollPane1 = new java.awt.ScrollPane();
    jScrollPane1 = new javax.swing.JScrollPane();
    jTable1 = new javax.swing.JTable();

    jTable1.setModel(new javax.swing.table.DefaultTableModel(
       new Object [][] {

//estado inicial dos campos
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
```

```
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null},
            {null, null, null, null}
        },
        new String [] {
            "Nome", "Status", "Unit", "Descrição"
        }
    ) {
        Class[] types = new Class [] {
            java.lang.String.class, java.lang.String.class, java.lang.String.class, java.lang.String.class
        };
        public Class getColumnClass(int columnIndex) {
            return types [columnIndex];
        }
    });
    jTable1.setToolTipText("");
    jTable1.setColumnSelectionAllowed(true);
    jScrollPane1.setViewportView(jTable1);

jTable1.getColumnModel().getSelectionModel().setSelectionMode(javax.swing.ListSelectionModel.SIN
GLE_SELECTION);

    scrollPane1.add(jScrollPane1);

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
    this.setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(scrollPane1,          javax.swing.GroupLayout.DEFAULT_SIZE,          380,
Short.MAX_VALUE)
            .addContainerGap())
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(scrollPane1,          javax.swing.GroupLayout.DEFAULT_SIZE,          280,
Short.MAX_VALUE)
            .addContainerGap())
    );
  }// </editor-fold>

  // Declaração de variaveis - não modifique
  private javax.swing.JScrollPane jScrollPane1;
  private javax.swing.JTable jTable1;
```

```
    private java.awt.ScrollPane scrollPane1;
}
```

10.5 Gráfico

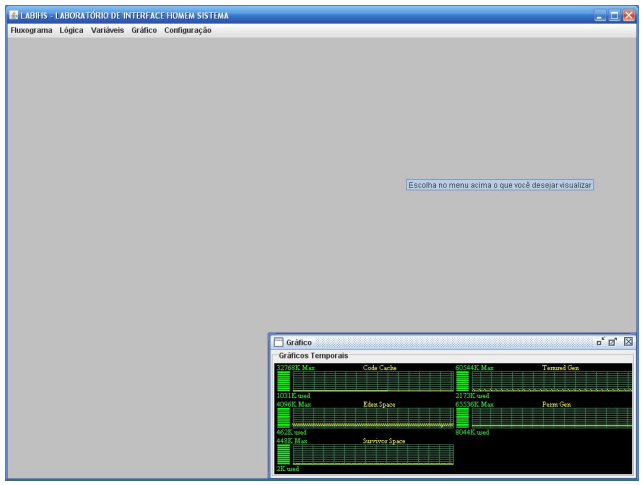


Figura 10.5a – Tela exibindo a janela de gráficos temporais.

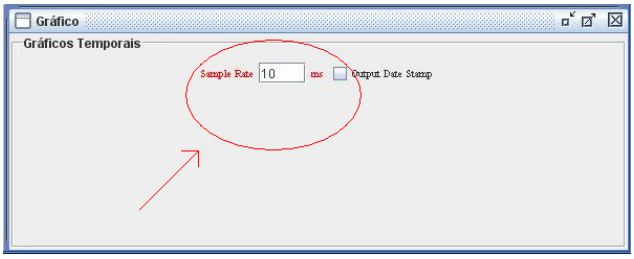


Figura 10.5b – Recurso de alteração da velocidade dos gráficos.

10.5.1 Nome da tela: Gráfico

10.5.2 Código da tela: T05

10.5.3 Arquivo: C:\viviane\projetos\Labihs\src\projeto\Grafico.java

10.5.4 Descrição do corpo da tela:

A figura 10.5a apresenta os gráficos temporais das variáveis envolvidas em cada passo do procedimento, que será construído para auxiliar na decisão do operador sobre conceitos vagos como “aumentar”, “diminuir”, etc. O código do gráfico usado no sistema deverá ser alterado após a programação das funcionalidades do sistema, visto que os valores gráficos deste código não correspondem aos requisitos estabelecidos para o sistema.

10.5.5 Velocidade dos gráficos:

A figura 10.5b, exibe um recurso utilizado para alterar a velocidade dos gráficos, onde para acessá-lo basta dar um clique sobre o nome '*Gráficos Temporais'*, em seguida especificar o valor que deseja e por fim clicar dentro da janela para sair e visualizar os gráficos numa nova velocidade.

10.5.6 Programa da tela:

```
/**
 *
 * @author
 */

package projeto;

import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;
import java.awt.geom.Line2D;
import java.awt.geom.Rectangle2D;
import javax.swing.*;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;
import java.lang.management.*;
import java.util.*;



public class Grafico extends JPanel{

   static JCheckBox dateStampCB = new JCheckBox("Output Date Stamp");
   public Surface surf;
   JPanel controls;
   boolean doControls;
   JTextField tf;

      static java.util.List<MemoryPoolMXBean> mpools =
      ManagementFactory.getMemoryPoolMXBeans();


      static int numPools = mpools.size();


      public Grafico()
   {
               setLayout(new BorderLayout());
      setBorder(new TitledBorder(new EtchedBorder(), "Gráficos Temporais"));
      add(surf = new Surface());
      controls = new JPanel();
```

```
    controls.setPreferredSize(new Dimension(135,80));
    Font font = new Font("serif", Font.PLAIN, 10);
    JLabel label = new JLabel("Sample Rate");
    label.setFont(font);
    label.setForeground(Color.red);
    controls.add(label);
    tf = new JTextField("1000");
    tf.setPreferredSize(new Dimension(45,20));
    controls.add(tf);
    controls.add(label = new JLabel("ms"));
    label.setFont(font);
    label.setForeground(Color.red);
    controls.add(dateStampCB);
    dateStampCB.setFont(font);
    addMouseListener(new MouseAdapter() {
       public void mouseClicked(MouseEvent e) {
          removeAll();
          if ((doControls = !doControls)) {
             surf.stop();


                add(controls);
          } else {
             try {
                surf.sleepAmount = Long.parseLong(tf.getText().trim());
             } catch (Exception ex) {}
             surf.start();
             add(surf);
          }
          validate();
          repaint();
       }
    });
}



public class Surface extends JPanel implements Runnable {

    public Thread thread;
    public long sleepAmount = 1000;
         public int  usageHistCount = 20000;
    private int w, h;
    private BufferedImage bimg;
    private Graphics2D big;
    private Font font = new Font("Times New Roman", Font.PLAIN, 11);
    private int columnInc;
    private float usedMem[][];
    private int ptNum[];
```

```
private int ascent, descent;
private Rectangle graphOutlineRect = new Rectangle();
private Rectangle2D mfRect = new Rectangle2D.Float();
private Rectangle2D muRect = new Rectangle2D.Float();
private Line2D graphLine = new Line2D.Float();
private Color graphColor = new Color(46, 139, 87);
private Color mfColor = new Color(0, 100, 0);
private String usedStr;


public Surface() {
   setBackground(Color.black);
   addMouseListener(new MouseAdapter() {
      public void mouseClicked(MouseEvent e) {
         if (thread == null) start(); else stop();
      }
   });
        int i = 0;
        usedMem = new float[numPools][];
        ptNum = new int[numPools];
}

public Dimension getMinimumSize() {
   return getPreferredSize();
}

public Dimension getMaximumSize() {
   return getPreferredSize();
}

public Dimension getPreferredSize() {
   return new Dimension(135,80);
}


public void paint(Graphics g) {

   if (big == null) {
      return;
   }

   big.setBackground(getBackground());
   big.clearRect(0,0,w,h);


        h = h / ((numPools + numPools%2) / 2);
        w = w / 2;

        int k=0; // index of memory pool.
        for (int i=0; i < 2;i++) {
```

```
                for (int j=0; j < (numPools + numPools%2)/ 2; j++) {
                  plotMemoryUsage(w*i,h*j,w,h,k);
                      if (++k >= numPools) {
                        i = 3;
                        j = (numPools + numPools%2)/ 2;
                        break;
                      }
                  }
                }
            g.drawImage(bimg, 0, 0, this);
        }

          public void plotMemoryUsage(int x1, int y1, int x2, int y2, int npool) {

              MemoryPoolMXBean mp = mpools.get(npool);
              float usedMemory =  mp.getUsage().getUsed();
            float totalMemory =  mp.getUsage().getMax();

            // .. Draw allocated and used strings ..
            big.setColor(Color.green);

                // Print Max memory allocated for this memory pool.
            big.drawString(String.valueOf((int)totalMemory/1024) + "K Max ", x1+4.0f, (float) y1 +
ascent+0.5f);
            big.setColor(Color.yellow);

                // Print the memory pool name.
            big.drawString(mp.getName(),  x1+x2/2, (float) y1 + ascent+0.5f);

                // Print the memory used by this memory pool.
            usedStr = String.valueOf((int)usedMemory/1024)
               + "K used";
            big.setColor(Color.green);
            big.drawString(usedStr, x1+4, y1+y2-descent);

            // Calculate remaining size
            float ssH = ascent + descent;
            float remainingHeight = (float) (y2 - (ssH*2) - 0.5f);
            float blockHeight = remainingHeight/10;
            float blockWidth = 20.0f;
            float remainingWidth = (float) (x2 - blockWidth - 10);

            // .. Memory Free ..
            big.setColor(mfColor);
            int MemUsage = (int) (((totalMemory - usedMemory) / totalMemory) * 10);
            int i = 0;
            for ( ; i < MemUsage ; i++) {
               mfRect.setRect(x1+5,(float) y1+ssH+i*blockHeight,
                        blockWidth,(float) blockHeight-1);
               big.fill(mfRect);
```

```
}

// .. Memory Used ..
big.setColor(Color.green);
for ( ; i < 10; i++)  {
   muRect.setRect(x1+5,(float) y1 + ssH+i*blockHeight,
                blockWidth,(float) blockHeight-1);
   big.fill(muRect);
}

// .. Draw History Graph ..
     if (remainingWidth <= 30) remainingWidth = (float)30;
     if (remainingHeight <= ssH) remainingHeight = (float)ssH;
big.setColor(graphColor);
int graphX = x1+30;
int graphY = y1 + (int) ssH;
int graphW = (int) remainingWidth;
int graphH = (int) remainingHeight;

graphOutlineRect.setRect(graphX, graphY, graphW, graphH);
big.draw(graphOutlineRect);

int graphRow = graphH/10;

// .. Draw row ..
for (int j = graphY; j <= graphH+graphY; j += graphRow) {
   graphLine.setLine(graphX,j,graphX+graphW,j);
   big.draw(graphLine);
}

// .. Draw animated column movement ..
int graphColumn = graphW/15;

if (columnInc == 0) {
   columnInc = graphColumn;
}

for (int j = graphX+columnInc; j < graphW+graphX; j+=graphColumn) {
   graphLine.setLine(j,graphY,j,graphY+graphH);
   big.draw(graphLine);
}

--columnInc;

// Plot memory usage by this memory pool.
if (usedMem[npool] == null) {
            usedMem[npool] = new float[usageHistCount];
   ptNum[npool] = 0;
}
```

```
            // save memory usage history.
            usedMem[npool][ptNum[npool]] = usedMemory;

        big.setColor(Color.yellow);

            int w1; // width of memory usage history.
            if (ptNum[npool] > graphW) {
               w1 = graphW;
            } else {
                  w1 = ptNum[npool];
        }


        for (int j=graphX+graphW-w1, k=ptNum[npool]-w1; k < ptNum[npool];
                                                                k++, j++) {
            if (k != 0) {
               if (usedMem[npool][k] != usedMem[npool][k-1]) {
                        int h1 = (int)(graphY + graphH * ((totalMemory -usedMem[npool][k-
1])/totalMemory));
                        int h2 = (int)(graphY + graphH * ((totalMemory -
usedMem[npool][k])/totalMemory));
                  big.drawLine(j-1, h1, j, h2);
               } else {
                        int h1 = (int)(graphY + graphH * ((totalMemory -
usedMem[npool][k])/totalMemory));
                  big.fillRect(j, h1, 1, 1);
               }
             }
        }
        if (ptNum[npool]+2 == usedMem[npool].length) {
           // throw out oldest point
           for (int j = 1;j < ptNum[npool]; j++) {
               usedMem[npool][j-1] = usedMem[npool][j];
           }
           --ptNum[npool];
        } else {
           ptNum[npool]++;
        }
     }


     public void start() {
        thread = new Thread(this);
        thread.setPriority(Thread.MIN_PRIORITY);
        thread.setName("MemoryMonitor");
        thread.start();
     }


     public synchronized void stop() {
```

```
        thread = null;
        notify();
    }

    public void run() {

        Thread me = Thread.currentThread();

        while (thread == me && !isShowing() || getSize().width == 0) {
            try {
                thread.sleep(500);
            } catch (InterruptedException e) { return; }
        }

        while (thread == me && isShowing()) {
            Dimension d = getSize();
            if (d.width != w || d.height != h) {
                w = d.width;
                h = d.height;
                bimg = (BufferedImage) createImage(w, h);
                big = bimg.createGraphics();
                big.setFont(font);
                FontMetrics fm = big.getFontMetrics(font);
                ascent = (int) fm.getAscent();
                descent = (int) fm.getDescent();
            }
            repaint();
            try {
                thread.sleep(sleepAmount);
            } catch (InterruptedException e) { break; }
            if (Grafico.dateStampCB.isSelected()) {
                System.out.println(new Date().toString() + " " + usedStr);
            }
        }
        thread = null;
    }
}


// Test thread to consume memory
static class Memeater extends ClassLoader implements Runnable {
        Object y[];
        public Memeater() {}
        public void run() {
            y = new Object[10000000];
            int k =0;
            while(true) {
                if (k == 5000000) k=0;
                y[k++] = new Object();
                try {
```

```
                Thread.sleep(20);
          } catch (Exception x){}

                // to consume perm gen storage
           try {
          // the classes are small so we load 10 at a time
          for (int i=0; i<10; i++) {
            loadNext();
          }
      } catch (ClassNotFoundException x) {
                // ignore exception
      }

      }

   }

    Class loadNext() throws ClassNotFoundException {

// public class TestNNNNNN extends java.lang.Object{
// public TestNNNNNN();
//   Code:
//    0:    aload_0
//    1:    invokespecial   #1; //Method java/lang/Object."<init>":()V
//    4:    return
// }

int begin[] = {
   0xca, 0xfe, 0xba, 0xbe, 0x00, 0x00, 0x00, 0x30,
   0x00, 0x0a, 0x0a, 0x00, 0x03, 0x00, 0x07, 0x07,
   0x00, 0x08, 0x07, 0x00, 0x09, 0x01, 0x00, 0x06,
   0x3c, 0x69, 0x6e, 0x69, 0x74, 0x3e, 0x01, 0x00,
   0x03, 0x28, 0x29, 0x56, 0x01, 0x00, 0x04, 0x43,
   0x6f, 0x64, 0x65, 0x0c, 0x00, 0x04, 0x00, 0x05,
   0x01, 0x00, 0x0a, 0x54, 0x65, 0x73, 0x74 };

int end [] = {
   0x01, 0x00, 0x10,
   0x6a, 0x61, 0x76, 0x61, 0x2f, 0x6c, 0x61, 0x6e,
   0x67, 0x2f, 0x4f, 0x62, 0x6a, 0x65, 0x63, 0x74,
   0x00, 0x21, 0x00, 0x02, 0x00, 0x03, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x01, 0x00, 0x01, 0x00, 0x04,
   0x00, 0x05, 0x00, 0x01, 0x00, 0x06, 0x00, 0x00,
   0x00, 0x11, 0x00, 0x01, 0x00, 0x01, 0x00, 0x00,
   0x00, 0x05, 0x2a, 0xb7, 0x00, 0x01, 0xb1, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00 };


// TestNNNNNN
```

```
        String name = "Test" + Integer.toString(count++);

        byte value[];
        try {
            value = name.substring(4).getBytes("UTF-8");
        } catch (java.io.UnsupportedEncodingException x) {
            throw new Error();
        }

        // construct class file

        int len = begin.length + value.length + end.length;
        byte b[] = new byte[len];
        int i, pos=0;
        for (i=0; i<begin.length; i++) {
            b[pos++] = (byte)begin[i];
        }
        for (i=0; i<value.length; i++) {
            b[pos++] = value[i];
        }
        for (i=0; i<end.length; i++) {
            b[pos++] = (byte)end[i];
        }

        return defineClass(name, b, 0, b.length);

         }
         static int count = 100000;

}

public static void main(String s[]) {
    final Grafico demo = new Grafico();
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {System.exit(0);}
        public void windowDeiconified(WindowEvent e) { demo.surf.start(); }
        public void windowIconified(WindowEvent e) { demo.surf.stop(); }
    };
    JFrame f = new JFrame();


    f.addWindowListener(l);
    f.getContentPane().add("Center", demo);
    f.pack();
    //f.setSize(new Dimension(400,500));
    f.setVisible(true);
    demo.surf.start();
        Thread thr = new Thread(new Memeater());
        thr.start();
}
```

}

10.6 Principal

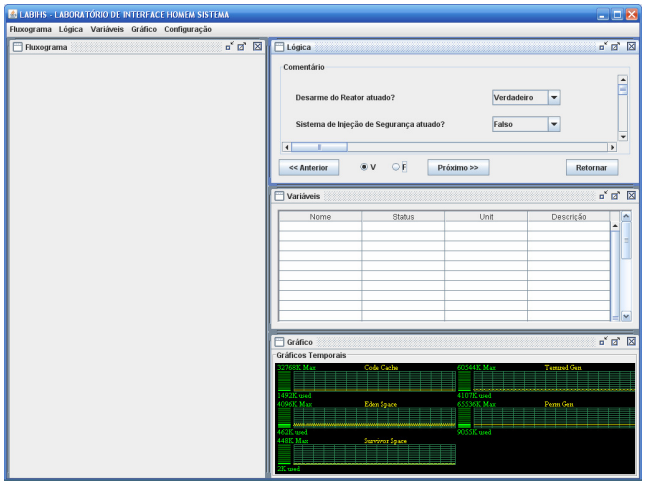


Figura 10.6 – Tela principal com os itens (janela) do menu abertos.

10.6.1 Nome da tela: Principal

10.6.2 Código da tela: T06

10.6.3 Arquivo: C:\viviane\projetos\Labihs\src\projeto\ProjLabihs.java

10.6.4 Descrição do corpo da tela:

A figura 10.6 apresenta alguns detalhes da tela de navegação implementada para os clientes do PBC, onde pode-se ver todos os itens do menu abertos: na janela da esquerda, o local onde serão apresentados os fluxogramas com os passos dos procedimentos; na janela superior direita, a lógica realizada em cada ação ou verificação do procedimento; na janela imediatamente abaixo da direita, os botões de navegação no procedimento onde o usuário pode ir/retornar para a próxima/anterior ação ou verificação e selecionar o caminho a ser seguido (verdadeiro ou falso); na janela logo abaixo a direita, informações, em tempo real, sobre a condição de componentes e de valores de variáveis da planta; e na janela inferior a direita os gráficos ao longo do tempo das variáveis da planta referentes a cada passo do procedimento.

Cada janela interna da tela principal pode ser minimizada, maximizada e fechada e, além disso, seu tamanho e posição podem ser alterados apenas arrastando o mouse.

10.7 Configuração

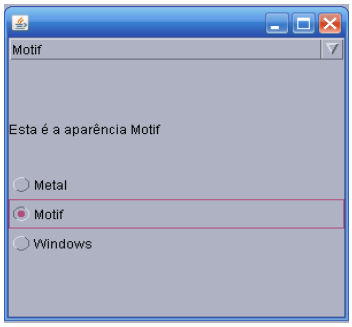


Figura 10.7a – Nesta janela pode ser escolhida a aparência do sistema.

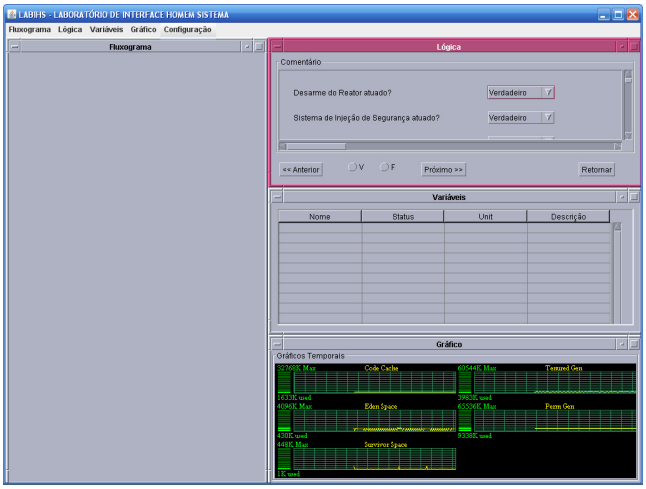


Figura 10.7b – Tela principal com a aparência alterada para o tema Motif.

10.7.1 Nome da tela: Configuração

10.7.2 Código da tela: T07

10.7.3 Arquivo: C:\viviane\projetos\Labihs\src\projeto\Configuracao.java

10.7.4 Descrição do corpo da tela:

A tela de configuração permite que o usuário possa alterar a aparência de todas as janelas. Existem três temas diferentes para a mudança da aparência: Metal, Motif e Windows. Cada tema traz um estilo e cores definidas para o estilo, fundo, ícones dos botões e esqueleto da janela. O interessante é que mesmo havendo somente três temas, estes temas podem ser combinados criando uma variação de estilos para o *layout* do sistema.

10.7.5 Alterando a aparência

Clicando em no item *Configuração* do menu e em seguida aparência, abrir-se-á a janela com os temas oferecidos (figura 10.7a). Após ter realizado a escolha

(radioButton), basta fechar a janela e ao abrir as outras janelas do sistema, elas serão exibidas com a aparência modificada de acordo com o tema escolhido.

10.7.6 Programa da tela:

```
package projeto;


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

public class Configuracao extends JFrame{

   private String strings[]={"Metal","Motif","Windows"};
   private UIManager.LookAndFeelInfo aparencia[];
   private JRadioButton radio[];
   private ButtonGroup grupo;
   private JLabel rotulo;
   private JComboBox comboBox;


      public Configuracao()
   {

   //super("Testando a Aparencia e Comportamento");
   Container container=getContentPane();



   JPanel painelNorte = new JPanel();
   painelNorte.setLayout(new GridLayout(2,2,5,5));



   rotulo= new JLabel("Esta e a aparencia Metal");
   rotulo.setVerticalTextPosition(SwingConstants.CENTER);
   container.add(rotulo);



   comboBox= new JComboBox(strings);
   painelNorte.add(comboBox);



   container.add(painelNorte,BorderLayout.NORTH);
```

```
   radio=new JRadioButton[3];



JPanel painelSul=new JPanel();
painelSul.setLayout(new GridLayout(5,3));



   radio=new JRadioButton[3];
   radio[0]=new JRadioButton("Metal");
   radio[1]=new JRadioButton("Motif");
   radio[2]=new JRadioButton("Windows");


      grupo = new ButtonGroup();


   TratadorDeItens trat = new TratadorDeItens();
      for(int count=0;count<radio.length;count++){
         radio[count].addItemListener(trat);
         grupo.add(radio[count]);

   painelSul.add(radio[count]);
}


   container.add(painelSul,BorderLayout.SOUTH);


   aparencia=UIManager.getInstalledLookAndFeels();
      setSize(300,300);
      setVisible(true);

   radio[0].setSelected(true);
}

//usa UIManager para mudar a aparencia e comportamento da GUI
private void mudeTheLookAndFeel(int valor)
{
//muda aparencia e comportamento
try{
   UIManager.setLookAndFeel(
   aparencia[valor].getClassName());
   SwingUtilities.updateComponentTreeUI(this);
   }

//processa problemas com a mudanca da aparencia e do comportamento
   catch(Exception exception){
```

```
    exception.printStackTrace();
}
}

//executa a aplicação
public static void main(String args[])
{
Configuracao aplicacao = new Configuracao();
    aplicacao.setDefaultCloseOperation(
    JFrame.EXIT_ON_CLOSE);
}


private class TratadorDeItens implements ItemListener{
//processa a seleção da aparência e comportamento feita pelo usuario

public void itemStateChanged(ItemEvent evento)
{
    for(int count=0;count<radio.length;count++)
{
if(radio[count].isSelected()){
    rotulo.setText("Esta é a aparência " + strings[count]);
    comboBox.setSelectedIndex(count);
    mudeTheLookAndFeel(count);
    }
   }
  }
 }
}
```

11. DIAGRAMA DE CASO DE USO

11.1 Cenário

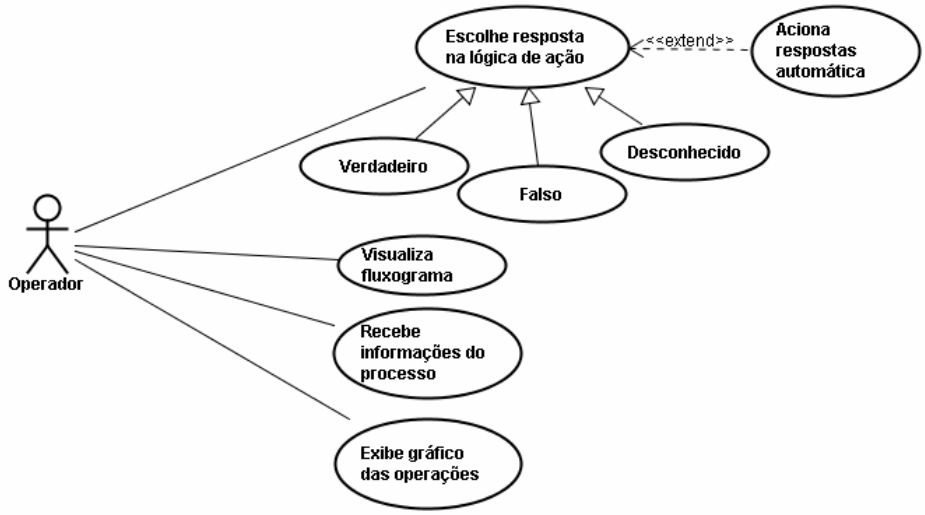


Figura 11.1 – Cenário de uso do Cliente PBC.

11.2 Descrição do caso de uso

REGISTRO DE REQUISITOS	Data de Emissão: 27/03/2009
Projeto:	PBC
1-Analista Responsável: Viviane	
2-Identificação do Caso de Uso: Ator: Operador – Caso de Uso: Escolhe resposta na lógica de ação	
3- Descrição / Justificativa Este caso de uso corresponde à lógica de sucesso, onde o operador obtém a informação sobre a lógica realizada em cada ação (*Action*) ou verificação (*Check*) do procedimento. Para realizar com sucesso uma ação ou uma verificação, o usuário configura a condição de entrada de cada parâmetro utilizado na lógica de sucesso, respondendo aos comentários com verdadeiro, falso ou desconhecido.	
Detalhamento	
1 - Pré-condições:	Acessar o sistema e informar as respostas para cada comentário na lógica de ação.
2 - Pós-condições:	Visualizar o fluxo apresentado no fluxograma do procedimento a partir da resposta fornecida ao comentário da lógica de sucesso do sistema.
3 – Seqüências Lógica de Ações	
Fluxo Básico:	P.A.: Define a condição de entrada de cada ação e verificação com: 1- verdadeiro; 2- falso; 3- desconhecido. P.S.: Captura as respostas da condição de entrada P.S.: Valida cada resposta de entrada P.S.: Exibe o resultado da verificação/ação no fluxograma
Fluxo Alternativo:	P.A.: Clicar no botão de respostas automáticas P.S.: Ativar/Exibir as respostas automáticas P.A.: Utilizar botões de navegação no procedimento P.S.: Conduzir o operador de acordo com cada botão utilizado
4 - Exceções (Regras no banco de Dados)	P.S.: 4.1 Todas as condições de entrada devem ser definidas
5 – Respostas Esperadas	P.S.: Exibe resposta no fluxograma P.S.: Exibe situação do processo
Elaborado por: Viviane F.	**Data: 27/03/2009**

REGISTRO DE REQUISITOS	Data de Emissão: 27/03/2009
Projeto:	PBC
1-Analista Responsável: Viviane	
2-Identificação do Caso de Uso: Ator: Operador – Caso de Uso: Visualiza fluxograma	
3- Descrição / Justificativa Este caso de uso corresponde à visão geral do fluxograma do procedimento, onde todos os passos do procedimento são *renderizados* com o passo de foco no centro da janela. E todas as ações e verificações do procedimento são *renderizadas* como um fluxograma em duas dimensões dentro do retângulo do passo. Estando familiarizado com os passos do procedimento, operador pode atuar expandindo e fechando passos ou navegando diretamente no fluxograma do procedimento.	
Detalhamento	
1 - Pré-condições:	Informar as ações e verificações na árvore lógica de sucesso e/ou seguir com o procedimento através dos botões de navegação
2 - Pós-condições:	Visualizar o processo onde todos os passos do procedimento são *renderizados* com o passo de foco no centro da janela
3 – Seqüências Lógica de Ações	
Fluxo Básico:	P.A.: Define a condição de entrada de cada ação e verificação P.S.: Captura as respostas da condição de entrada P.S.: Valida cada resposta de entrada P.S.: *Renderiza* as ações e verificações do procedimento P.A.: Verifica o resultado apresentado no fluxograma P.A.: Navega sobre o fluxograma do procedimento
Fluxo Alternativo:	P.A.: Expande os passos no fluxograma de procedimento P.S.: Abre o passo expandido P.A.: Fecha os passos no fluxograma de procedimento P.S.: Exibe um fluxograma mais simples
4 - Exceções (Regras no banco de Dados)	P.S.: 4.1 Todas as condições de entrada devem ser definidas
5 – Respostas Esperadas	P.S.: Exibe o fluxo no fluxograma de procedimento P.S.: Abre e/ou fecha passos no fluxograma
Elaborado por: Viviane F.	**Data: 27/03/2009**

REGISTRO DE REQUISITOS	Data de Emissão: 27/03/2009
Projeto:	PBC
1-Analista Responsável: Viviane	
2-Identificação do Caso de Uso: Ator: Operador – Caso de Uso: Recebe informações do processo	
3- Descrição / Justificativa Este caso de uso corresponde às informações do processo, através das quais o operador recebe dados em tempo real sobre a condição de operação, componentes e de valores de variáveis da planta. No sistema todos os componentes da planta deverão ser apresentados como símbolos de dispositivos relacionados a bombas, válvulas, etc.	
Detalhamento	
1 - Pré-condições:	Informar as ações e *checks* na árvore lógica de sucesso, seguir com o procedimento através dos botões de navegação
2 - Pós-condições:	Visualizar as informações do processo em tempo real com os componentes da planta apresentados em forma de símbolos de dispositivos
3 – Seqüências Lógica de Ações	
Fluxo Básico:	P.A.: Define a condição de entrada de cada ação e verificação P.S.: Captura as respostas da condição de entrada P.S.: Valida cada resposta de entrada P.S.: *Renderiza* as ações e verificações do procedimento P.A.: Verifica o resultado apresentado no fluxograma P.A.: Abre/Visualiza a janela *Variáveis* do sistema P.S.: Exibe informações sobre a condição de componentes e de valores de variáveis da planta P.S.: Apresenta todas as informações em tempo real na janela *Variáveis*
Fluxo Alternativo:	P.A.: Reposiciona os campos na tabela das variáveis da planta P.S.: Exibe as alterações realizadas
4 - Exceções (Regras no banco de Dados)	P.S.: 4.1 Todas as condições de entrada devem ser definidas
5 – Respostas Esperadas	P.S.: Exibe condições de componentes e variáveis da planta P.S.: Mantém informações do processo exibidas em tempo real
Elaborado por: Viviane F.	**Data: 27/03/2009**

REGISTRO DE REQUISITOS	Data de Emissão: 27/03/2009
Projeto:	PBC
1-Analista Responsável: Viviane	
2-Identificação do Caso de Uso: Ator: Operador – Caso de Uso: Exibe gráfico das operações	
3- Descrição / Justificativa Este caso de uso corresponde ao diferencial do sistema LABIHS, onde gráficos temporais das variáveis envolvidas em cada passo do procedimento serão incorporados para auxiliar na decisão do operador sobre conceitos vagos como "aumentar", "diminuir" etc., em relação ao caso de ações de avaliação de expressões ambíguas.	
Detalhamento	
1 - Pré-condições:	Depois de realizados os procedimentos de operação, abrir o item gráfico do menu
2 - Pós-condições:	Visualizar as informações do processo através do gráfico de operações
3 – Seqüências Lógica de Ações	
Fluxo Básico:	P.A.: Define a condição de entrada de cada ação e verificação P.S.: Captura as respostas da condição de entrada P.S.: Valida cada resposta de entrada P.S.: *Renderiza* as ações e verificações do procedimento P.A.: Abre o gráfico no menu superior P.S.: Exibe gráficos temporais com informações do processo
Fluxo Alternativo:	P.A.: Altera a velocidade dos gráficos P.S.: Salva alteração no processo atual
4 - Exceções (Regras no banco de Dados)	P.S.: 4.1 Todas as condições de entrada devem ser definidas
5 - Respostas Esperadas	P.S.: Exibe informações sobre andamento do procedimento P.S.: Auxilia na decisão do operador
Elaborado por: Viviane F.	**Data: 27/03/2009**

12. CONCLUSÕES

Este relatório descreve os passos seguidos na programação inicial de um sistema próprio de procedimentos baseados em computador, que deverá ser implantado no simulador do LABIHS.

Até este momento foram construídos o principal diagrama de modelagem, a documentação e a interface que será utilizada no sistema Cliente PBC. Esta documentação, juntamente com o diagrama de caso de uso serve principalmente para auxiliar o próximo programador na implementação das funcionalidades do sistema Cliente PBC do LABIHS.

Com o intuito de garantir a continuidade do desenvolvimento do PBC do LABIHS, este relatório apresentou a análise feita sobre o sistema Cliente, trouxe a descrição das telas que foram desenvolvidas e ainda especificou o que deve ser implementado, dando também o passo inicial na modelagem dos requisitos do sistema.

A partir daqui deverão ser implementados os códigos correspondentes aos requisitos do sistema descritos na sessão 5 deste relatório. Mas adiante, na sessão 11, pode ser encontrada a descrição dos casos de uso que foram elaboradas para garantir o detalhamento dos requisitos do sistema. Estes tópicos, cuja finalidade é auxiliar o programador, fazem parte dos padrões de elaboração dos documentos relacionados à construção de software de computador.

Como trabalho futuro deverão ser implementados os códigos correspondentes aos requisitos do sistema. Para garantir esta implementação foi elaborada uma descrição dos casos de uso do sistema proposto.

Para a implementação do servidor do sistema PBC será utilizada a mesma filosofia de projeto adotada para o detalhamento dos clientes.

13. REFERÊNCIAS

DEITEL, H.M., DEITEL, P.J., 2003, Java Como Programar, Bookman, 4ª ed.

FOWLER, M., 2000, UML Essencial: um breve guia para a linguagem padrão de modelagem de objetos, trad. Vera Pezerico e Chistian Thomas Price. 2ª ed., Bookman, Porto Alegre.

HOSTMANN, C.S., 2004, Big Java. Porto Alegre, Bookman.

HOSTMANN, C.S., CORNELL, G., 2001, Core Java 2, trad. João Eduardo Nobrega Tortello. Makron, São Paulo.

JACOBSON, I., BOOCH, G., RUMBAUGH, J., 1999, The Unified Modeling Language Reference Manual.

JUNG, Y., SEONG, P., KIM, M., 2004, *A Model for Computerized Procedures Based on Flowcharts and Success Logic Trees*, Reliability Engineering and System Safety, ELSEVIER, vol. 26, pg. 351-362.

LARMAN, C., 2004, Utilizando UML e Padrões: uma introdução à análise O.O. e ao processo unificado, trad. Luiz A. Meirelles Salgado e João Tortello. 2ª ed., Bookman, Porto Alegre.

NUREG-0578, 1979, *TMI-2 Lessons learned task force status report and short-term recommendations*, U.S. Nuclear Regulatory Commission Research, Washington.

OLIVEIRA, M.V., 2008, Desenvolvimento e Avaliação de Procedimentos de Operação Computadorizados para Salas de Controle Avançadas de Plantas Industriais, RT-IEN-04/2008.

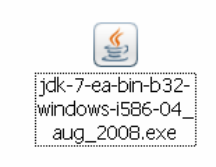
OLIVEIRA, M.V., BRUNO, D.S., CARVALHO, P.V.R., SANTOS, I.J.A.L., GRECCO, C.H., 2009, *Applying computer-based procedures in nuclear power plants,* INAC 2009.

RIGO, W., 2004, Componentes da Interface Gráfica da Linguagem Java, Versão 1.1, Florianópolis. Disponível em: < http://monica.inf.ufsc.br.>

ANEXO I

Instalação das ferramentas para o Desenvolvimento e Execução do Sistema

Antes de prosseguir com o desenvolvimento e dar início à execução do sistema é necessário ter instalado o NetBeans IDE em seu computador mas, antes de tudo, é necessário que o Java Development Kit (JDK) já esteja instalado em sua máquina. As ferramentas que deverão ser obrigatoriamente instaladas para a execução do sistema, são representadas pelos seguintes ícones:



Lembrando que estamos utilizando para a implementação do sistema a IDE NetBeans e que devido às várias vantagens que tornou a atualização do NetBeans bastante significativa, estamos utilizando-o na versão 6.1, no entanto a aplicação também pode ser executada em versões anteriores.

I.1. Instalando o JDK – *versão 7*

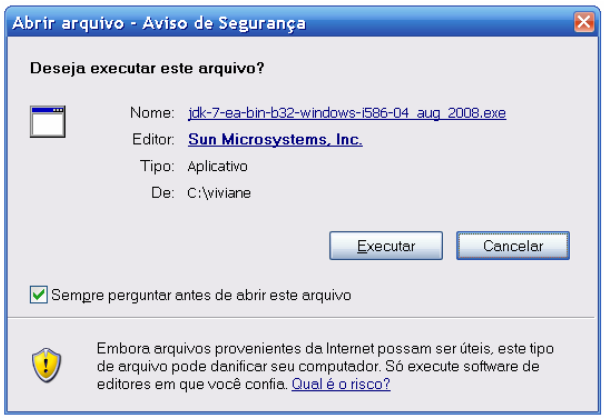
Conhecido como o kit de desenvolvimento, o JDK (Java Development Kit) tem sido oferecido gratuitamente pela SUN. É importante frisar que ele deve ser instalado primeiramente, pois dele depende o sucesso da instalação da ferramenta de desenvolvimento.

Para instalar o JDK a partir do executável, siga corretamente o passo a passo descrito em seguida:

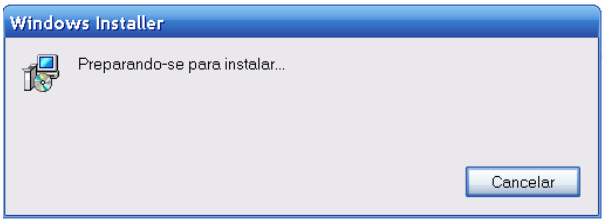
1. Dê um duplo clique no JDK 7, representado com o ícone ao lado.

2. Apenas aguarde quando aparecer a janela abaixo.

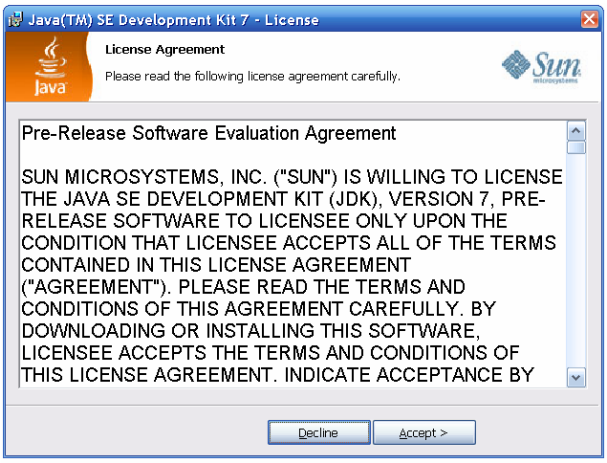
3. Logo em seguida, será exibida uma janela trazendo informações de segurança como o nome e editor do aplicativo. Para começar a instalação basta clicar em *"Executar"*.



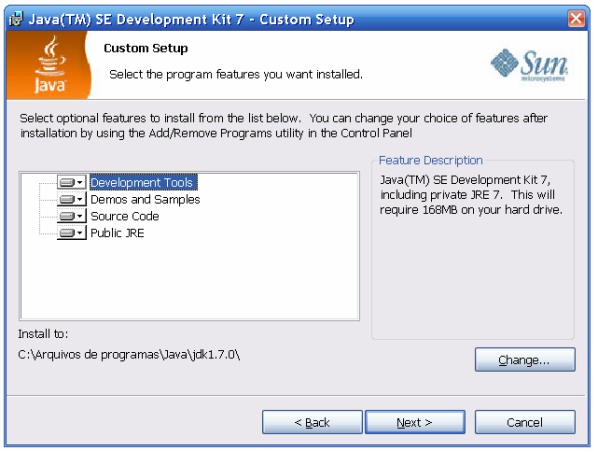
4. Depois aguarde enquanto a aplicação se prepara para instalar.



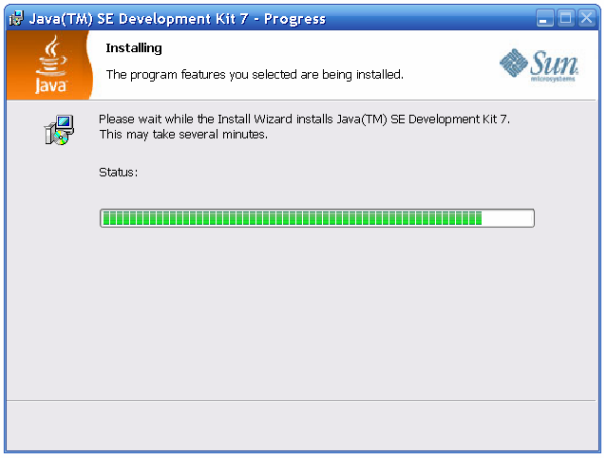
5. A próxima janela apresenta um texto com a licença do programa e dois botões na parte inferior do painel:*"Decline"* e *"Accept >"*. Para prosseguir com a instalação você deverá clicar no botão *"Accept"*, aceitado assim, os termos expressos na licença.



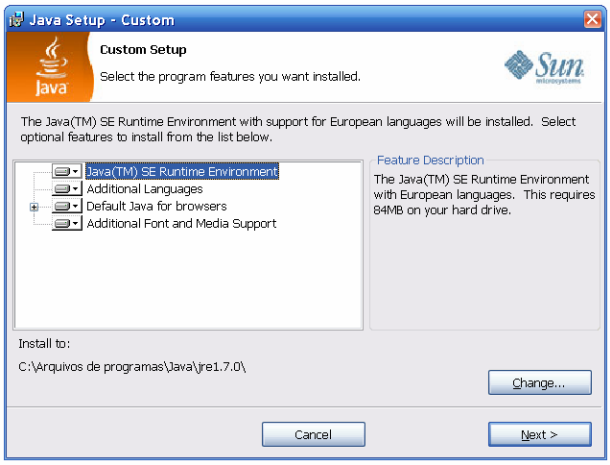
6. Na janela seguinte você poderá determinar o local para instalação clicando em "*Change*..." . Geralmente o local de instalação vem determinado em "C:\Arquivo de programas\Java\...", dessa forma, não é necessário que se faça alteração alguma. Em seguida você deverá apenas clicar no botão "*Next* >" para prosseguir e instalar o JDK.



7. A tela abaixo, apenas apresenta o status de instalação e enquanto a instalação esta sendo executada aguardar alguns instantes.

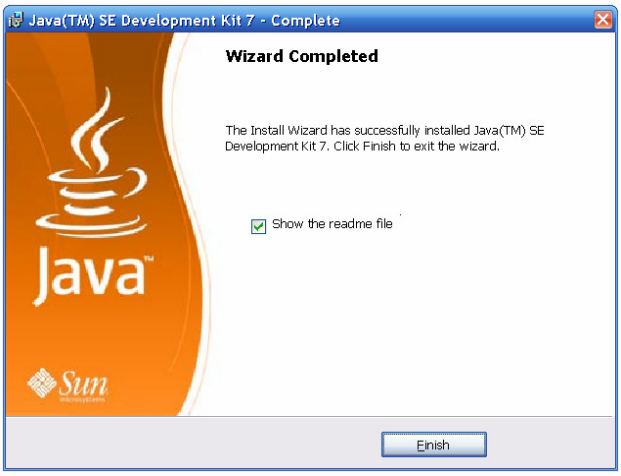


8. Na próxima tela novamente você deverá clicar em *"Next"* para prosseguir com a instalação.



9. A tela abaixo apresenta o progresso da instalação, você deverá aguardar enquanto a instalação esta sendo completada.

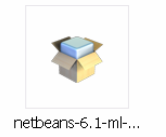
10. Finalmente, no momento em que a instalação é completada você poderá optar se deseja fazer a leitura do arquivo de texto trazido no JDK, ou não. Para que este arquivo não abra é necessário desmarcar a "caixinha" ao lado da mensagem *"Show the readme file"*. Finalmente, basta clicar em *"Finish"* para fechar esta ultima janela de dialogo.



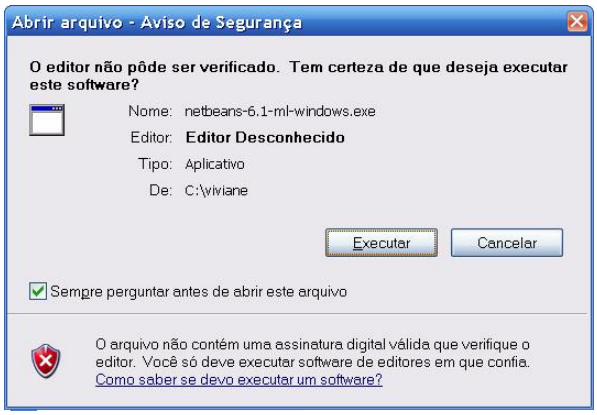
I.2 - Instalando NetBeans – *versão 6.1*

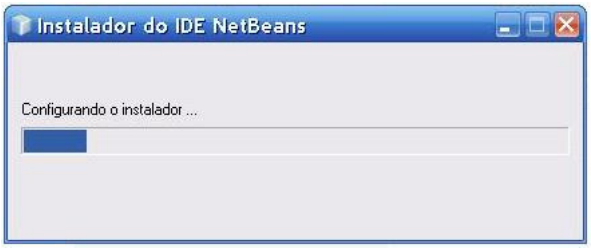
Antes de instalar o IDE, você deverá já ter instalado o kit Java SE Development (JDK) 5 Atualização 14 (versão 1.5.0_14) ou mais recente (inclusive o JDK 6 Atualização 4) em seu sistema. Se não tiver o JDK 5 Atualização 14 ou uma versão mais recente instalada, não será possível prosseguir com a instalação.

1. Primeiramente dê um duplo-clique no ícone do executável NetBeans 6.1, como mostra a figura ao lado.

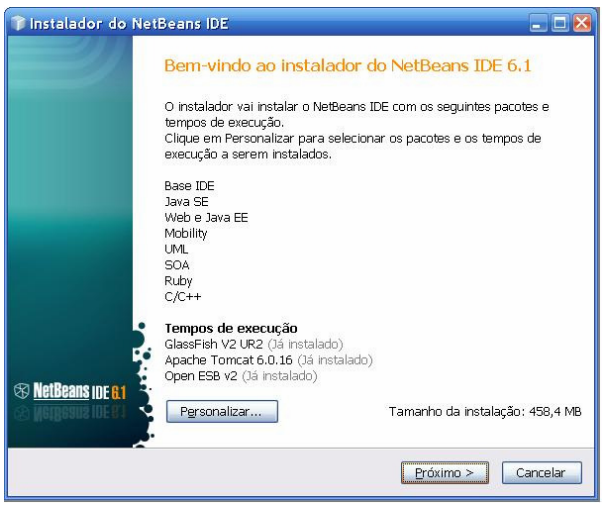


2. Logo em seguida clique em *"Executar"* na seguinte janela de dialogo.

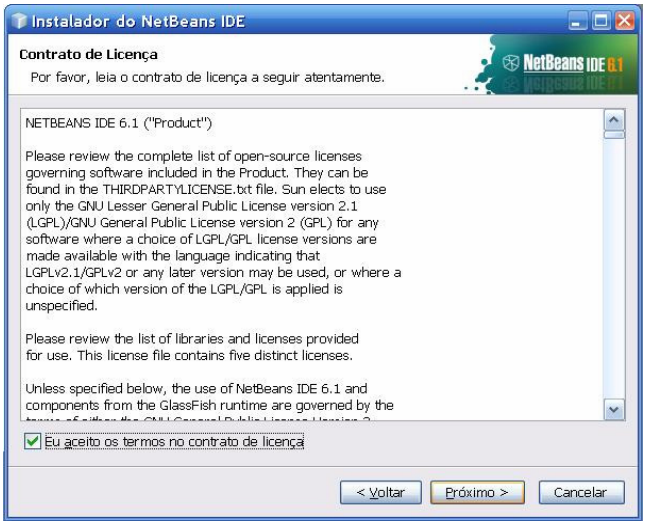




3. Todas as telas de instalação da ferramenta NetBeans são facilmente entendidas pois vem apresentadas com instruções claras e em português. Nesta tela apenas clique no botão “Próximo >”.

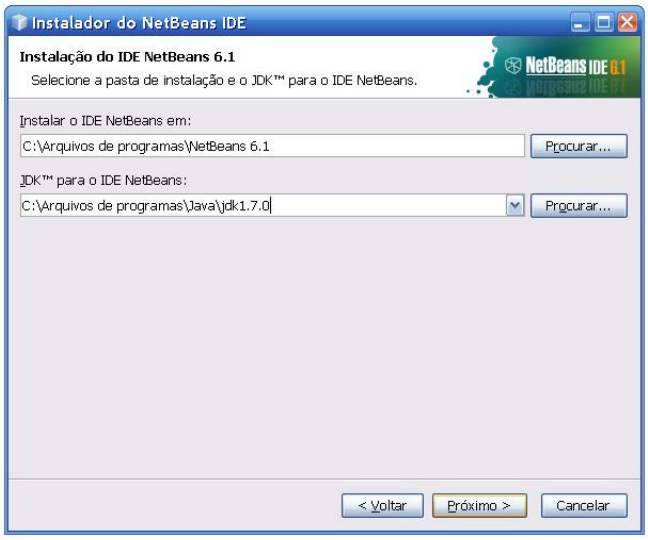


4. Esta tela traz o contrato de licença. Diferentemente das instruções a licença é apresentada em Inglês. No entanto, para continuar com a instalação, basta que você selecione a opção “Eu aceito os termos no contrato de licença” e clique em “Próximo>”.

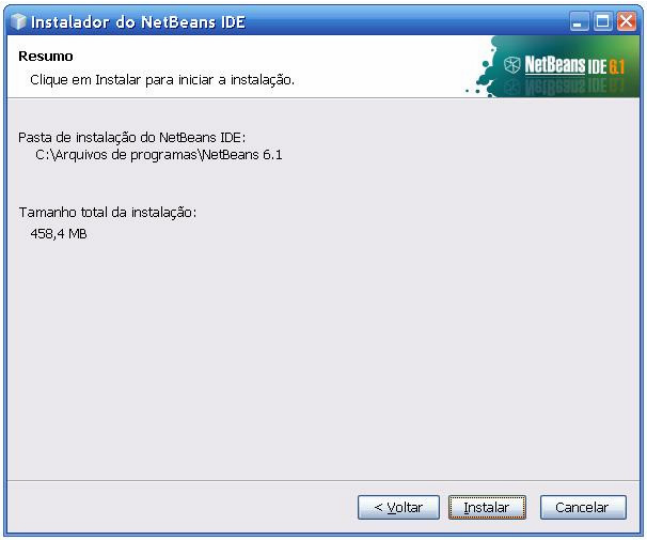


5. Clique em “Procurar” para escolher o local onde será instalado. Recomendamos que você mantenha o diretório que vem apresentado “C:\Arquivos de programas\...” e clique em “Próximo >”.

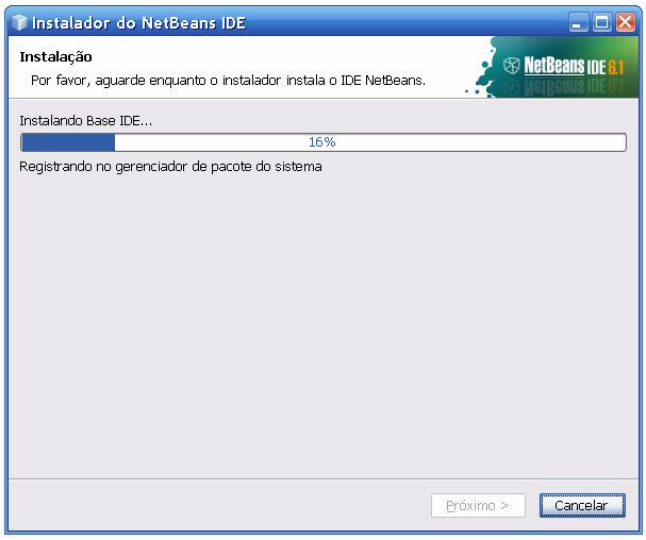
Aqui será, automaticamente, procurado o JDK instalado na máquina, se não encontrar clique em “Procurar”e indique o caminho onde você o instalou. Encontrando-o clique em “Próximo >”.



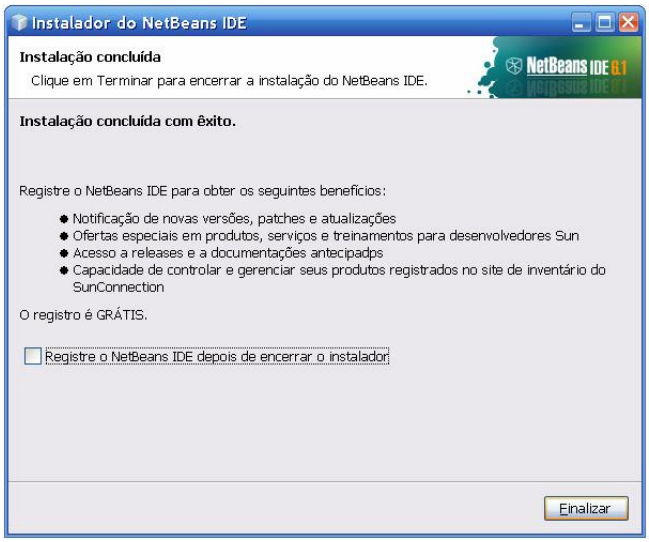
6. Aqui só é preciso verificar o local de instalação da IDE e o tamanho. Se as informações coincidirem com o que deseja clique em “Instalar”, caso contrário clique em “< Voltar” e refaça o passo anterior.



7. O processo de instalação esta sendo executado e pode demorar um pouco. Neste momento não é necessário clicar em nada, apenas aguarde a finalização deste processo.



8. Nesta tela você poderá verificar informações sobre o local de instalação da IDE, como executá-la ou desinstalá-la. Aqui a instalação já foi completada então apenas clique em “Finalizar”.



ANEXO II

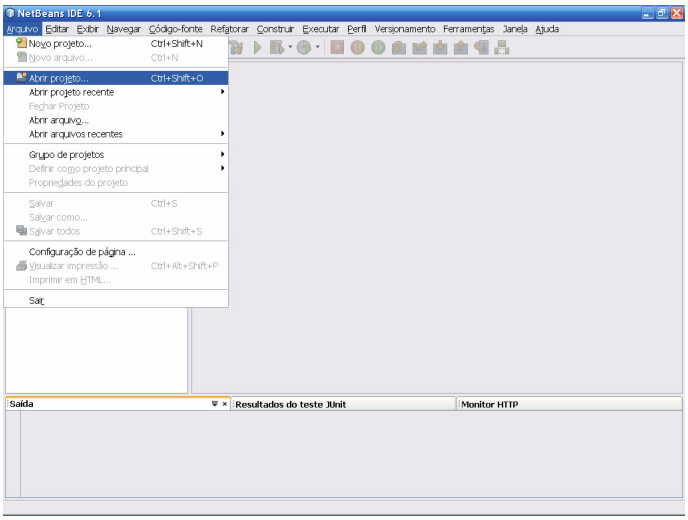
Manual do Usuário

II.1 Abrindo o Projeto PBC

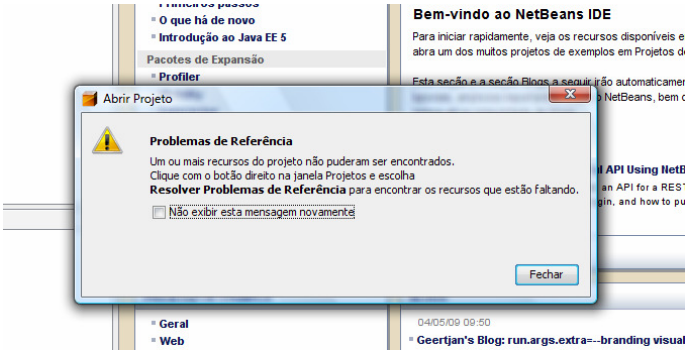
Como descrito no relatório, o desenvolvimento do PBC foi desenvolvido com a IDE NetBeans. O atalho para o NetBeans poderá ser encontrado no Desktop (área de trabalho) logo após sua instalação. Para abrí-lo basta dá um duplo clique sobre seu ícone, como mostra a figura abaixo.

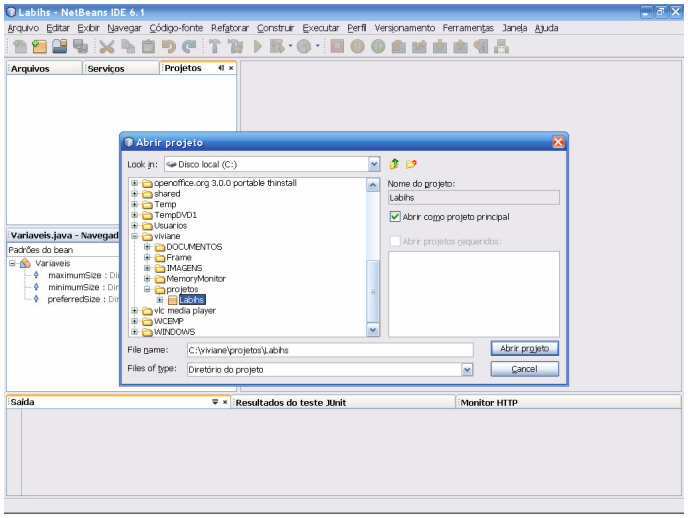
Por se tratar de uma das versões mais completas do NetBeans, a versão 6.1 pode levar para abrir alguns segundos a mais do que versões anteriores. A próxima imagem mostra que a IDE está sendo aberta.

Para abrir o projeto basta clicar em "*Arquivo*" e selecionar a opção "*Abrir projeto*".



Em seguida você deverá apontar para o caminho C:\viviane\projetos\Labihs.



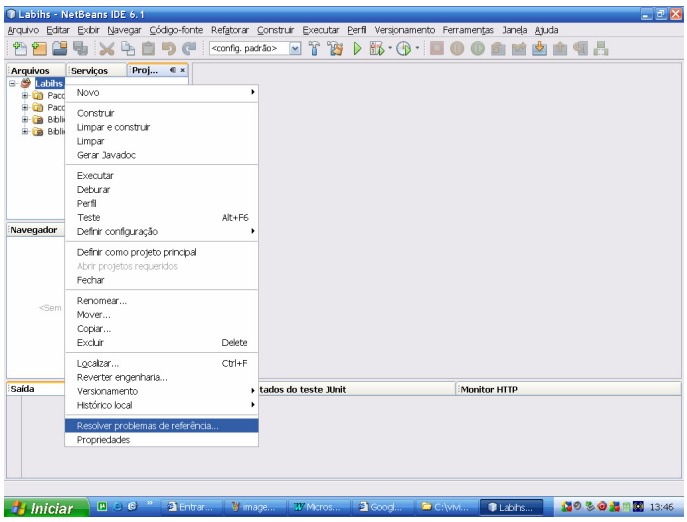


Depois de selecionado o projeto “PBC”, clique em “*Abrir projeto*”. Ao abrí-lo será exibida uma mensagem informando um problema de Referência, como mostra a figura abaixo.

II.2 Resolvendo o problema de Referência

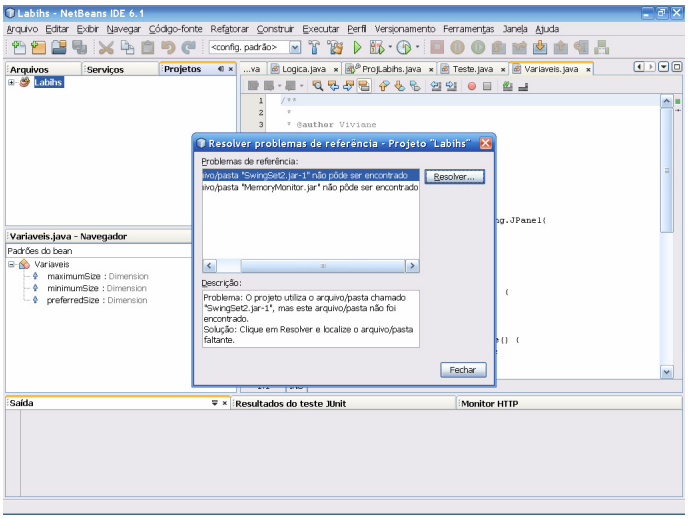
Existem duas maneiras de resolver um problema de Referência, uma é adicionando diretamente as bibliotecas (tópico 3) e outra é feita da seguinte forma:

Depois de verificado a ocorrência de um problema de referência, clique com o botão direto do mouse sobre o nome do projeto e selecione a opção “Resolver problemas de referência”, como mostra a imagem.

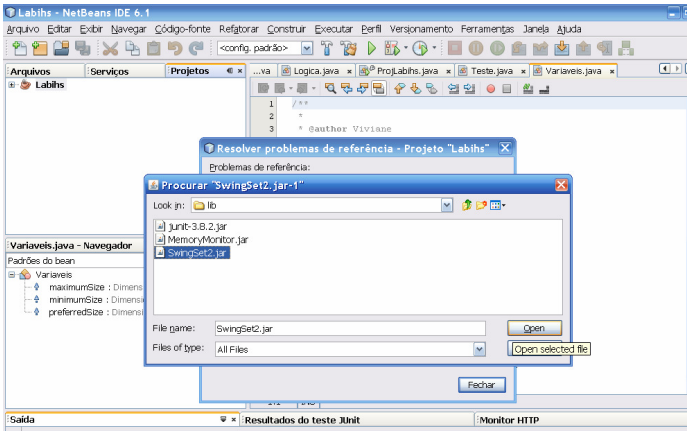


Logo em seguida será aberta uma janela de dialogo onde você terá de verificar quais as referências que foram quebradas, ou seja, as bibliotecas necessárias para o projeto que não estão sendo encontradas. É importante que você saiba em que local estão armazenadas as bibliotecas utilizadas. Neste projeto as bibliotecas foram armazenadas em *C:\viviane\projetos\Labihs\dist\lib*

Na próxima janela, selecione um dos problemas de referência e clique em “*Resolver*”.



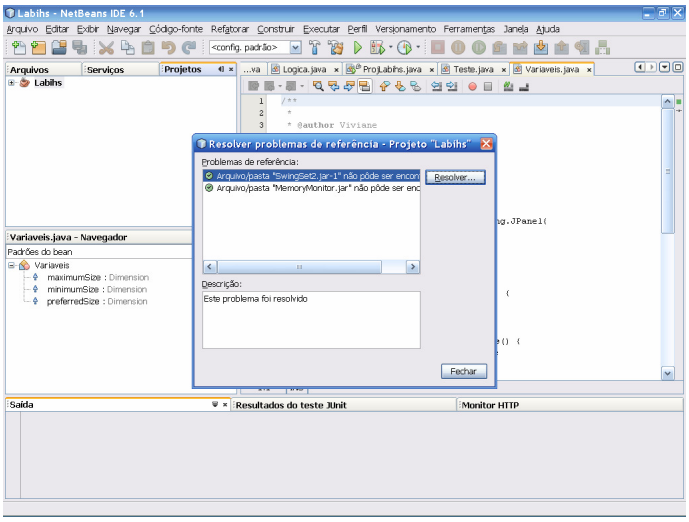
Em seguida aponte para o caminho *C:\viviane\projetos\Labihs\dist\lib*, selecione a biblioteca relacionada ao problema e clique em "*Open*".



Geralmente não é preciso seguir este passo mais de uma vez, pois no momento em que você soluciona um dos problemas de referência, se as bibliotecas estiverem todas no mesmo local, os demais problemas serão automaticamente resolvidos.

No projeto PBC poderá ocorrer de um a três problemas de referência, pois estão sendo utilizadas três bibliotecas, duas adicionais à ferramenta NetBeans (MemoryMonitor.jar e SwingSet2.jar) e outra que já faz parte da IDE (junit-3-8-2.jar). Ao resolver um dos problemas o outro será resolvido, pois as bibliotecas estão armazenadas no mesmo local.

A próxima imagem mostra que os problemas de referência foram resolvidos.

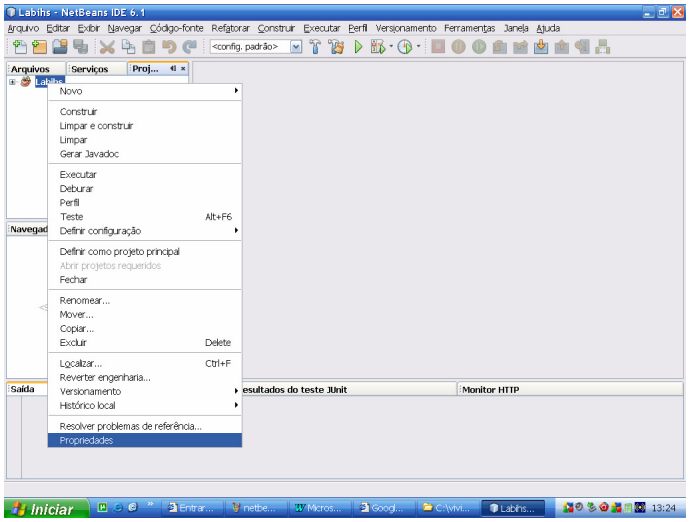


Depois de resolvidos os problemas, o projeto poderá ser compilado e executado, como é mostrado no quarto tópico.

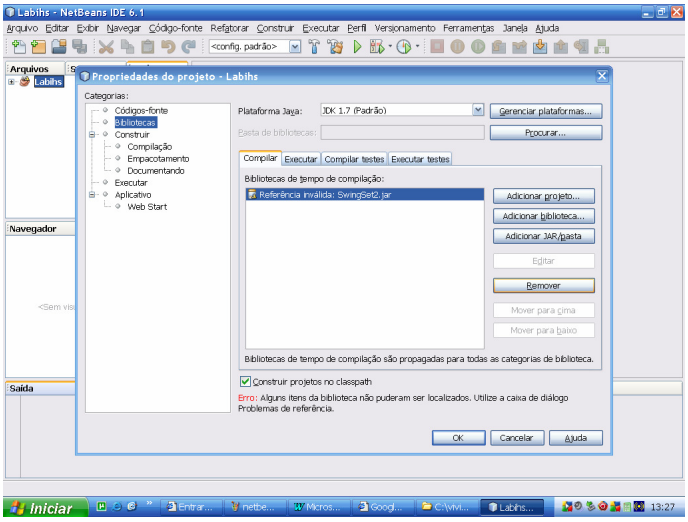
II.3 Adicionando as bibliotecas no NetBeans

Até o momento o projeto PBC está utilizando duas bibliotecas que não fazem parte da lista de bibliotecas do NetBeans, são elas: MemoryMonitor.jar e SwingSet2.jar. Além de utilizar o junit-3-8-2.jar, esta faz parte da lista de bibliotecas do próprio NetBeans. Sempre que for necessário adicionar uma biblioteca, *drive* ou componente no NetBeans, deverá ser feito o seguinte:

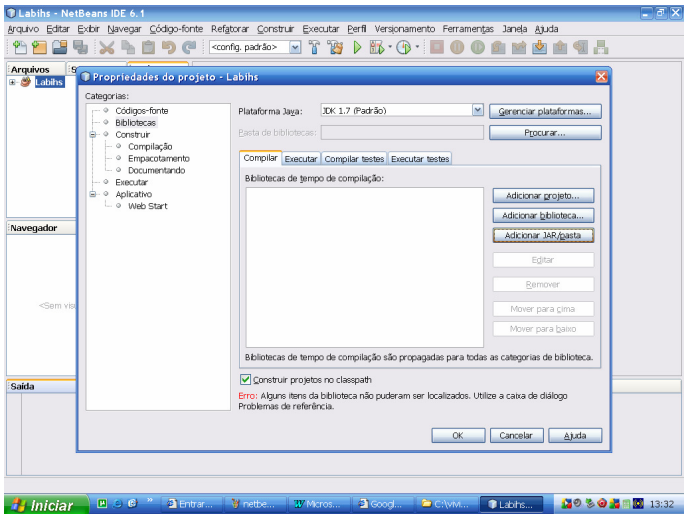
Clique com o botão direito sobre o nome do projeto e em seguida clique em "Propriedades", como mostra a figura.



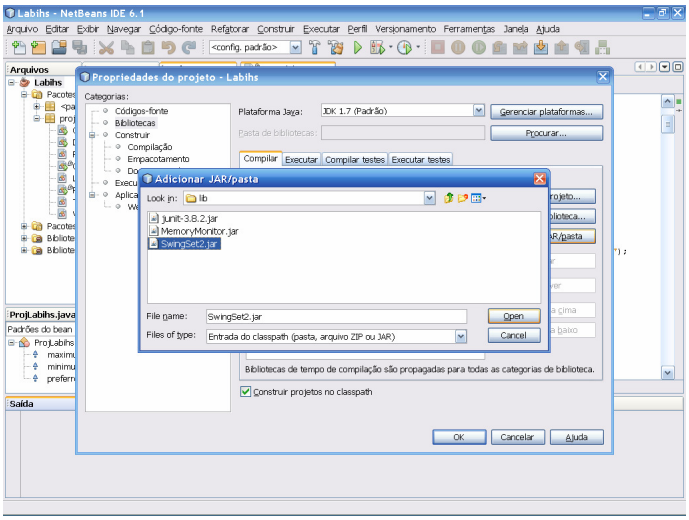
Em seguida será aberta uma janela com várias opções, no campo “categorias” clique em “*Bibliotecas*”, como será mostrado na próxima figura.



A imagem acima mostra no campo “*Compilar*” uma referência inválida, se isso ocorrer, selecione a referência e clique em “*Remover*”.



Em seguida clique em “*Adicionar JAR/pasta*”, para procurar as bibliotecas, driver ou componentes que não fazem parte do NetBeans. Selecione todos com <ctrl> + <a> e clique em “*Open*”. Ao final clique em “*Ok*”.

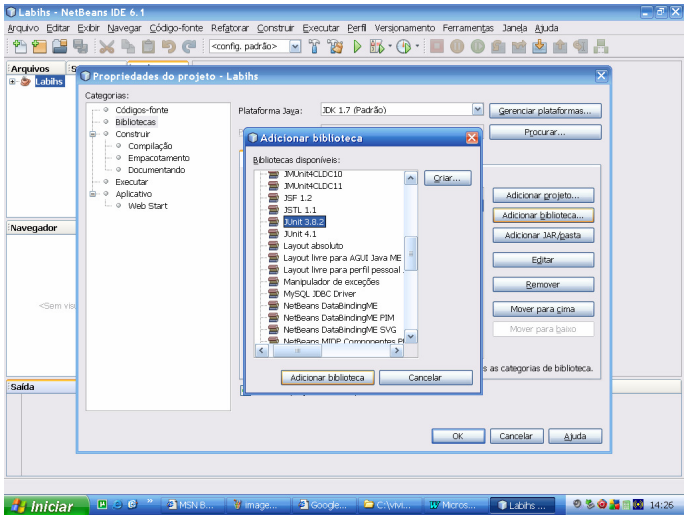


Todas as bibliotecas utilizadas, inclusive a *junit-3-8-2.jar* foram armazenadas em uma única pasta. É importante frisar que ao adicionar novas bibliotecas, estas sejam armazenadas no mesmo local das demais para facilitar em sua utilização.

Abaixo está descrito o caminho para encontrar as bibliotecas que estão sendo utilizadas até então:

- ✓ C:\viviane\projetos\Labihs\dist\lib\MemoryMonitor.jar
- ✓ C:\viviane\projetos\Labihs\dist\lib\SwingSet2.jar
- ✓ C:\viviane\projetos\Labihs\dist\lib\junit-3-8-2.jar

Se desejar encontrar as bibliotecas que fazem parte do NetBeans, clique em "*Adicionar biblioteca*", em seguida será aberta a seguinte janela.

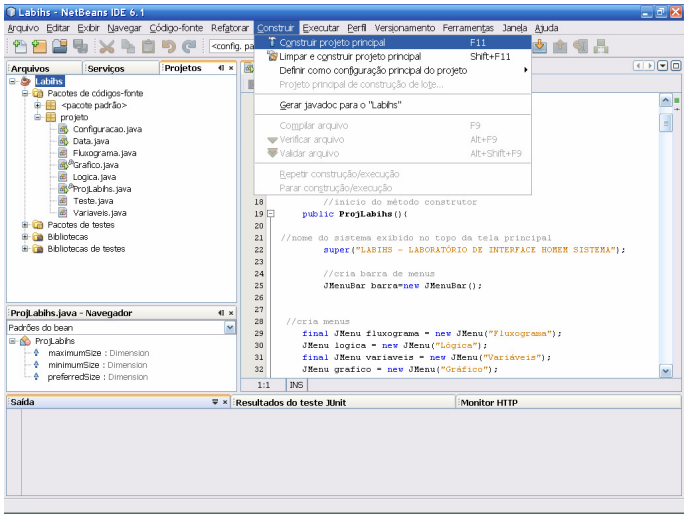


Sempre que for preciso utilizar uma dessas bibliotecas, basta selecioná-la e depois clicar em "*Adicionar biblioteca*".

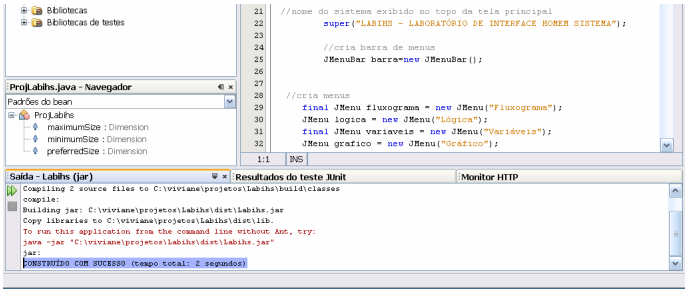
Atualmente para executar o projeto PBC, não será necessário adicionar outras bibliotecas além das citadas, mas futuramente com a programação das funcionalidades do sistema, pode haver tanto a necessidade de adicionar alguma das bibliotecas do NetBeans, quanto bibliotecas ou drivers à parte, como por exemplo, um *driver* de banco de dados, se for acrescentado ao projeto um módulo para gerar histórico.

II.4 Compilando e executando o projeto

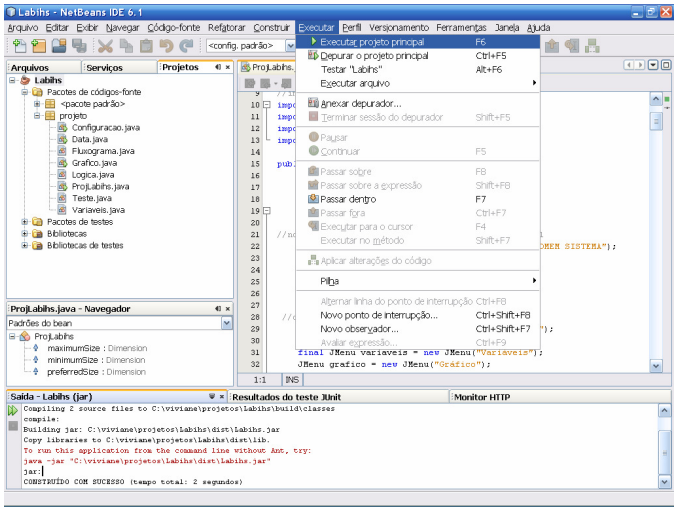
Clique na opção *"Construir"* em seguida clique em *"Construir projeto principal"* (ou *F11*).



Depois de compilado observe na aba "*Saída*", localizada na parte inferior da janela, a ocorrência de erros ou a mensagem "Construído com sucesso".



Para executar o projeto clique em *"Executar"* e depois clique em *"Executar projeto principal"* ou aperte <F6>.



Em seguida apenas aguarde a abertura da janela principal do projeto PBC.

Acessando o site http://www.netbeans.org/ , você poderá obter mais informações sobre o ambiente de desenvolvimento NetBeans, poderá baixar sua nova versão e ainda aprender como utilizar as ferramentas disponíveis na IDE.