UNIVERSIDADE DE SÃO PAULO

ESCOLA POLITÉCNICA

DEPARTAMENTO DE ENGENHARIA MECÂNICA

Sistema SCADA em CORBA / JAVA

PMC 581 – Projeto Mecânico II

Autor: ***Guilherme Felipe Japur de Sá***

NUSP 1029417

Orientador: ***Prof. Marcos R. P. Barreto***

São Pauio

1999

Aos meus pais, Marcos e Marina, pelo incentivo

e apoio todos estes anos de curso.

À Andréa pelo amor, a dedicação e paciência pelas

minhas horas ausentes.

Índice

1. INTRODUÇÃO

A evolução dos sistemas de automação industrial, mais especificamente no que se refere ao controle de dados e malhas de controle de processos industriais, tem oferecido às empresas sistemas cada vez mais variados e complexos, que vão dos sistemas de controle distribuídos (SDCD) aos componentes de chão de fábrica inteligentes. Evolução esta aliada à evolução de outras áreas relacionadas com a tecnologia.

Ainda se tratando de novas tecnologias, vem se observando que o mercado, representado pelos produtores e consumidores de hardware e software, tem procurado padronizar e integrar as diversas ferramentas oferecidas por diferentes fabricantes, visando a interoperabilidade de diferentes linguagens e até de diferentes implementações de uma mesma linguagem, pois muito freqüentemente, observava-se incompatibilidade entre compiladores de uma mesma linguagem, como é o caso de C++, que apesar de ter um conjunto de funções padronizados, muitas de suas funções eram específicas de cada fabricante. Um grande passo rumo à padronização foi dado pela *Sun Microsystems®* ao lançar a linguagem Java, com a especificação totalmente aberta para que qualquer vendedor pudesse implementar a sua própria *Virtual Machine* assim como o seu próprio compilador Java, desde que seguisse as especificações propostas pela *Sun*. Esta condição, que visava justamente criar um padrão em linguagem de programação, valeu à *Microsoft ®* uma ação na justiça por implementar a sua versão de Java proprietário, de tal maneira que a sua *Virtual Machine* fosse incompatível com outras plataformas, utilizando-se para isso sua cômoda posição de mercado. Características da linguagem Java, assim como de sua *Virtual Machine*, serão detalhados nos capítulos que se seguem.

Ainda seguindo a linha das padronizações, no início dos anos 90, um consórcio que reunia, inicialmente, grandes empresas da área de software, começou a formular um padrão para tecnologia de objetos distribuídos, a qual dera o nome de CORBA, sigla para *Common Object Request Broker Architecture*, ou numa tradução livre Arquitetura Comum de

Negociadores de Requisições de Objetos. Este desenvolvimento, em andamento até hoje, visa criar uma estrutura comum de interfaces, que podem ser vistas como um invólucro de objetos que expõe algumas funcionalidades dos objetos, e também um meio de comunicação entre estas interfaces, chamado de ORB. Detalhes da arquitetura CORBA serão vistos nos capítulos adiante.

Neste trabalho serão apresentadas as tecnologias utilizadas no seu desenvolvimento, tais como Java e CORBA, conceitos de objetos distribuídos e um breve estudo sobre sistemas SCADA, comparando a sua arquitetura tradicional (centralizada) com uma arquitetura distribuída.

2. OBJETIVOS

Este trabalho tem por objetivo desenvolver um projeto de sistema SCADA baseada em uma arquitetura distribuída, já tendo definida a arquitetura CORBA / JAVA como plataforma de implementação.

O desenvolvimento do projeto passará por uma série de etapas até a sua conclusão. Primeiramente da definição do escopo do projeto, em que será definido quais as funcionalidades do sistema. Definido o escopo, será feita uma modelagem do sistema, baseada em objetos, o que poderá ser feito utilizando-se alguma ferramenta de apoio como UML (*Unified Modeling Language)* e possivelmente alguma ferramenta que implemente UML.

Feita a modelagem de classes do sistema, parte-se então para a implementação deste, o que deve envolver a escolha de uma implementação de CORBA, assim como de uma implementação de Java.

Após o desenvolvimento, o projeto passa por uma fase de testes, onde serão identificadas possíveis falhas, e a correção destas.

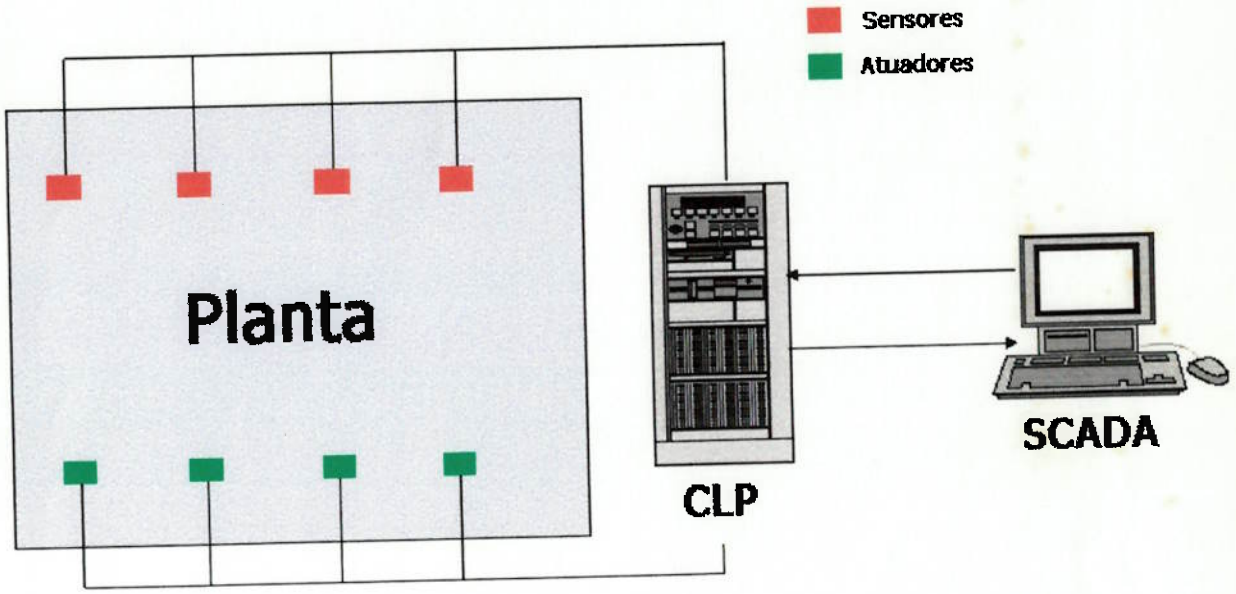
3. SCADA

O termo SCADA (*Supervisory Control And Data Acquisition)* se refere aos softwares utilizados pelas indústrias para se fazer coleta de dados do processo produtivo, e eventualmente intervir neste através de sinais de controle.

Para a coleta de dados, é preciso que hajam alguns componentes no chão de fábrica, que apesar de não serem exatamente parte do sistema SCADA, fazem parte do sistema como um todo, e aqui estaremos chamando de 'SCADA' justamente o sistema como um todo. Os componentes básicos do sistema então são:

- *Sensor*: são responsáveis pela captação de dados de processo. São componentes físicos que quantificam uma grandeza física e geram um sinal elétrico a partir desta medida, que será compreendido pelo computador que estará recebendo o sinal.
- *Atuador*: são responsáveis por gerarem ações a partir de um sinal elétrico. Assim como os sensores, são componentes físicos que têm a função oposta, ou seja, a partir de um sinal elétrico (comando) geram uma ação, alterando o estado da planta.
- *CLP (Controlador Lógico Programável)*: é o componente responsável pela lógica do sistema, ou seja, é este componente que irá receber os sinais gerados pelos sensores e a patir daí decidir que ações devem ser tomadas na planta, gerando comandos para os atuadores.
- *Elementos Gráficos*: são elementos visuais que apresentam um determinado conjunto de dados de processo de maneira gráfica.

Em uma arquitetura tradicional, o sistema possui um ponto (eventualmente mais que um) que centraliza todas as informações dos componentes, e que a cada dado intervalo de tempo (tempo de varredura), o sistema coleta os dados armazenando-os em uma base de dados no ponto central. Uma representação esquemática do sistema em uma arquitetura tradicional é apresentada abaixo:



4. JAVA

Para melhor compreensão do projeto, é necessário uma visão da plataforma Java, e porque a sua escolha neste projeto.

A arquitetura Java foi desenvolvida pela empresa norte-americana *Sun Microsystems ®* com o intuito de ser uma linguagem multi-plataforma, ou seja, que rodasse em qualquer ambiente. A Sun assumiu o lema "write once, run anyware" (escreva uma vez e rode em qualquer lugar) para a nova tecnologia. A sua característica de portabilidade é conseguida pelo fato de o compilador Java criar um arquivo binário para ser executado, não para um sistema operacional ou hardware específico , e sim para um ambiente java, que pode ser visto como uma máquina virtual, ou seja, um ambiente virtual que pode existir em qualquer ambiente. Esse ambiente Java chama-se *Java Virtual Machine (**JVM**)* . Outra característica, então, da linguagem Java que permite a sua portabilidade é o fato de ser interpretado pelo *JVM*, e assim qualquer ambiente que tenha uma *JVM* suporta aplicações em Java

Java é uma linguagem de programação bastante moderna. É totalmente orientada a objeto, e possui um sistema responsável por controlar as instâncias de objetos em memória, chamado "garbage collector". Este sistema é responsável por liberar memória, eliminando objetos assim que estes saem do escopo. Esta característica libera o programador da preocupação de liberar memória no código, tornando o processo de programaçào mais ágil.

Deve-se destacar também a preocupação desprendida pelos desenvolvedores desse ambiente com as aplicações distribuídas em rede, em que o tamanho dos binários devem ser o menor possível, devido ao fato da largura de banda em redes muito grande, como a internet, serem um gargalo no desempenho. Essa presocupação em fazer códigos pequenos, aproveitando o fato de ser um código interpretado e de ser uma linguagem totalmente orientada a objeto, fez com que java se tornasse uma linguagem de grande utilidade para aplicações distribuídas em rede, com uma grande biblioteca de

funcionalidades para rede embitudas nas suas classes base.

Dadas essas características da plataforma java, tem-se feito um esforço no sentido de se portar essa aquitetura java para todso tipo de componente eletrônico, de eltrodomésticos a automóveis, desenvolvendo-se *JVM* para esses processadores, possivelmente em firmware, ou seja, esses componentes teriam 'java nativo', e com a vantagem de serer todos integráveis.

5. CORBA

Ainda seguindo a tendência da plataforma Java de criar uma arquitetura portável e com interoperabilidade, um consórcio formado por mais de 800 empresas, entre produtores e consumidores de software, fundaram um grupo para definir uma especificação de um *framework* para uma arquitetura de objetos distribuídos, que fosse além de independente de plataforma, independente de linguagem de implementação, ainda tendo a característica de interoperabilidade. Dessa idéia foi criada CORBA (Commom Obejct Request Broker Architecture).

Além de prover uma estrutura para que objetos pudessem interagir, independente de plataforma ou de linguagem de implementação, CORBA visa trazer agilidade no processo de desenvolvimento de aplicações que estão distribuídas em uma rede, liberando os desenvolvedores de escrever o código referente à comunicação. A programação é feita de maneira transparente, conforme veremos adiante.

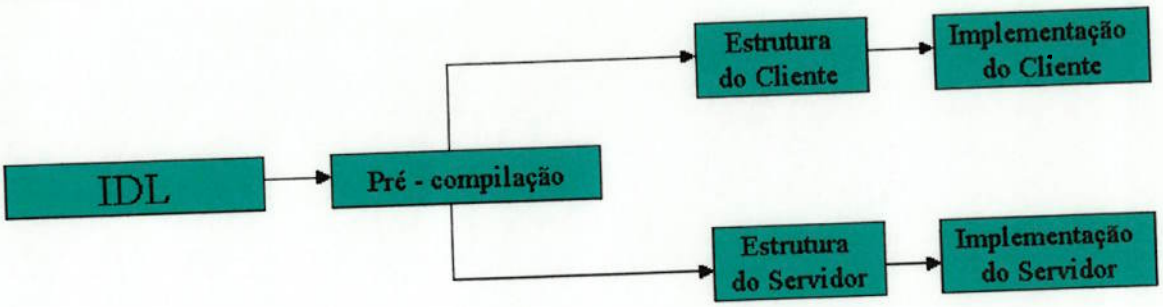
Mas antes de falarmos sobre CORBA e suas características, é preciso falar sobre arquitetura de objetos distribuídos. Podemos definir uma arquitetura baseada objetos distribuídos como um sistema orientado a objeto em que os objetos não necessariamente se encontram na mesma máquina, ou ainda que estejam na mesma máquina, podem estar rodando em processos diferentes. Assim durante a concepção do projeto (modelagem do sistema), o desenvolvedor trata o projeto não como um projeto distribuído pela rede, mas sim como um projeto *Stand-Alone*.

Tendo-se uma noção do que seja uma arquitetura de objetos distribuídos e que CORBA é provem uma estrutura básica para que estas aplicações sejam desenvolvidas, veremos agora os mecanismos da arquitetura CORBA.

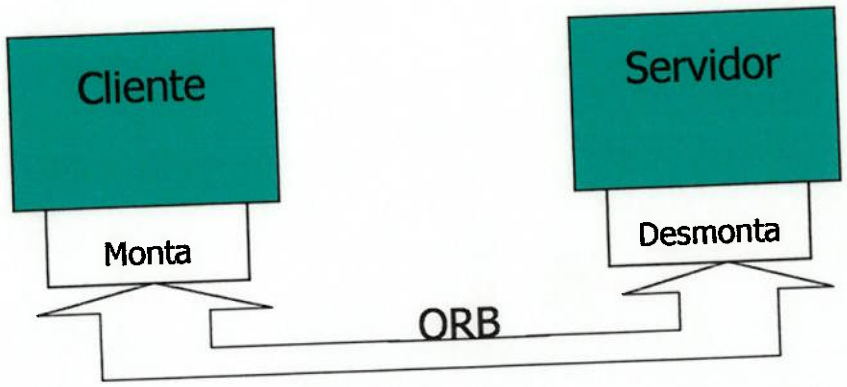
Em primeiro lugar CORBA não é uma linguagem de programação, mas provem uma linguagem para definição de interfaces, chamda de IDL. Na definição da IDL, devem ser definidas todas as funcionalidades das classes de objetos que serão expostas, incluindo-se

nessa definição os métodos, as propriedades, as Exceptions, Vetores, e tudo mais que possa ser usado na troca de dados entre dois objetos.

Definida a IDL, esta é pré-compilada para alguma linguagem de programação. Esta linguagem, não necessariamente mas preferencialmente uma linguagem orientada a objeto, será utilizada para se implementar os objetos em si. O esquema abaixo ilustra esse processo:



CORBA define um bus que é resopnsável por fazer toda a comunicação entre os objetos, tratando de traduzir a referência dos objetos nos diferentes pontos. Esse bus, o ORB, será responsável por montar uma chamada de um cliente e desmontar a chamada no servidor, fazendo o processo inverso para a resposta do servidor. Assim utilizando-se de um adaptador de obejeto (object adapter), é posível conectar os objetos no ORB, permitindo que se comuniquem entre si como se estivessem local. O esquema abaixo iliustra o seu funcionamento:

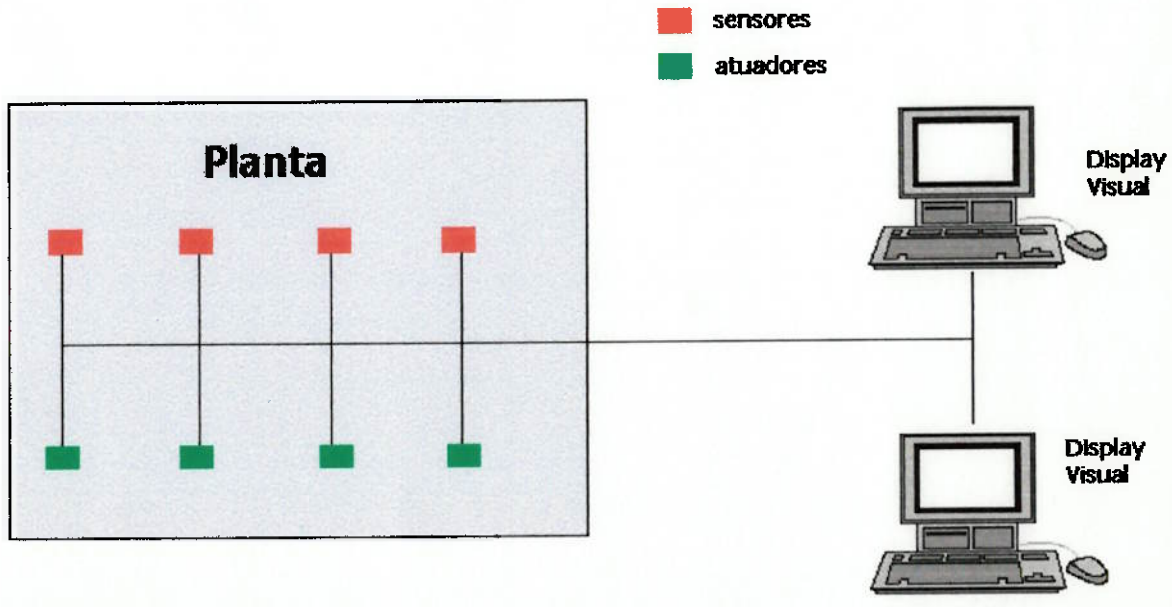


6. DEFINIÇÃO DO ESCOPO DO PROJETO

Como primeira etapa do desenvolviemento do projeto será feita a definição do seu escopo, dando uma visão geral do que o sistema se propõe a fazer.

Como já foi visto, em um sistema SCADA convencional há um ponto que centraliza todas as informações vindas da planta. Assim, sempre que for necessário se acessar algum dado de processo deve-se acessar este ponto centralizador, o que pode gerar um fluxo de dados muito grande pela rede. Além deste alto tráfego na rede, o sistema fica altamente dependente desse ponto centralizador, fazendo com que no caso de uma pane no equipamento ou falha na comunicação, o sistema todo deixe de operar.

Como solução alternativa aos sisetmas SCADA tradicionais, este trabalho se propõe a modelar um sistema SCADA em que seus componentes rodem distribuídos pela rede, mais especificamente, em conjunto com os componentes físicos em campo, e comuniquem entre si independentemente de se ter um ponto centralizador. A estrutura a ser implementada inclui os seus componentes básicos, tais como os apresentado na seção 3 deste trabalho. Um esquema representativo do sistema SCADA distribuído é apresentado abaixo:



No esquema apresentado acima, nota-se que não há um elemento centralizador e que todos os componentes podem comunicar entre si diretamente.

A proposição então vem de encontro ao que foi dito das tecnologias Java e CORBA. A primeira, como já mencionado, permite que seu código seja executado em qualquer ambiente que possua uma *JVM*, o que a faz uma excelente alternativa para se rodar em componentes físicos inteligentes, já que estes podem ter uma *JVM*, possivelmente nativa (em *firmware*) para interpretar o código escrito.

O uso da tecnologia CORBA também se faz bastante atrativo, já que o sistema será totalmente orientado a objeto, na maioria das vezes representando entidades físicas, e será baseado na comunicação entre estes objetos, e esta é uma das facilidades que CORBA proporciona.

7. MODELAGEM DO SISTEMA

Definido o escopo do projeto, foram identificadas as classes do projeto. Estas classes, ainda não abstraídas hierarquicamente estão definidas abaixo com os seus respectivos métodos e propriedades:

SENSOR

Classe de objetos responsável por representar os sensores do sistema. Baseada em um dos componentes físicos do sistema, e assim sendo, os objetos instanciados a partir desta classe serão usados para simular o seu equivalente real.

ATUADOR

Classe de objetos responsável por representar os atuadores do sistema. Assim como os sensores esta classe é baseada em um dos componentes físicos do sistema, e assim sendo, os objetos instanciados a partir desta classe serão usados para simular o seu equivalente real.

SCHEDULE

Os objetos instaciados a partir desta classe terão como função controlar a varredura dos objetos do sistema. Por estarmos trabalhando com uma estrutura distribuída, este elemento pode ser separado dos componentes, e sendo assim uma instância deste pode servir a mais de um componente, trazendo maior flebilidade e eficiência ao sistema.

DISPLAY VISUAL

Esta classe de objetos representa os elementos gráficos do sistema, ou GUIs (General User Interface). Os objetos instanciados a partir desta serão responsáveis por receber dados de um sensor e apresentá-los de maneira gráfica.

COORDENADOR

Este elemento será o responsável por controlar todas as atividades dos objetos do

sistema, mantendo registro dos demais componentes e possibilitando estes de se encontrarem.

Acima temos apenas a identificação das classes finais do sistema, não tendo sido apresentados as propriedades e métodos definidas por cada classe. Ainda no que concerne à modelagem do sistema, esta foi feita utilizando-se UML, que se trata de uma linguagem própria para se fazer modelagem de objetos. Através desta linguagem podem ser identificadas classes genéricas, que servem de base de para outras classes, fazendo valer os conceitos de hierarquia da programação orientada a objeto. Uma breve introdução à linguagem UML, bem como do *software* Rational Rose ®, utilizado para se fazer a modelagem do sistema em UML, estão no Apêndice A.

7.1 Classes do Sistema

O resultado, com todas as as classes identificadas e com todas as suas características estão apresentadas abaixo.

Classe Data

Esta classe representa uma estrutura de Data (dia, mês e ano) que possa ser exposta por um sistema baseado em CORBA, já que este não define tipo de dado para tal fim.

Propriedades

short dia

propriedade que indica o dia do objeto.

short mês

propriedade que indica o mês do objeto.

int ano

propriedade que indica o ano do objeto.

Métodos

String getData()

Este método retorna uma string que representa a data do objeto.

String getDataFormatada(string formato)

Este método retorna uma string que representa a data do objeto da maneira passada pelo parâmetro. Ex: 'dd/mm/aaaa'

Short getDia()

Este método retorna o dia representado no obejto.

Short getMes()

Este método retorna o mês representado no objeto.

Short getAno()

Este método retorna o ano representado no objeto.

String getDiaDaSemana()

Este método retorna o dia da semana representado no objeto.

Short EMaior (Data outra_data)

Este método comprara a data representada pelo obejeto corrente com uma data passada de referência. Retorna 1 caso a data do obejto seja maior, -1 caso a data passada de referência seja maior e 0 (zero) caso as duas datas sejam iguais.

Classe DataHora

Derivada da classe *Data*.

Esta classe incorpora a funcionalidade da classe Data, incorporando horário a esta.

Propriedades

Int hora

Propriedade que indica a hora do objeto.

Int minuto

Propriedade que indica o minuto do objeto.

Int segundo

Propriedade que indica o segundo do objeto.

Métodos

String GetHorario()

Este método retorna o horário representado pelo objeto

String GetHorarioFormatado (string formato)

Este método retorna o hora'rio representado pelo objeto da maneira passada como parâmtero. Ex: 'hh:mm:ss'

Short GetHora()

Retorna a hora representada pelo objeto.

Short GetMinuto()

Retorna o minuto representado pelo objeto.

Short GetSegundo()

Retorna o segundo representado pelo objeto.

Classe Publicação

Esta classe contêm a estrutura básica de todos os tipos de publicação passados entre os componentes do sistema.

Propriedades

Short IDPublicacao

Indica o tipo da publicacao.

DataHora horario

Indica o horário referente à publicação.

Métodos

Short GetID()

Retorna o ID de publicação.

DataHora getHorario()

Retorna o horário da publicação.

Classe pubValor

Derivada da classe Publicação

Esta classe define a estrutura bem como os métodos de acesso a ser passada por uma publicação de valor.

Propriedades

Double valor

Indica o valor que a publicação representa.

Métodos

Double getValor()

Retorna o valor que a publicação representa.

Void setValor(Double novo_valor)

Método que permite alterar o valor que a publicação representa.

Classe Componente

Esta classe descreve as propriedades e métodos básicos de todos os elementos do sistema.

Propriedades

string Nome

Essa variável membro contém o nome que unicamente identifica o objeto no sistema.

Métodos

String getNome ()

Esse método Retorna o nome do componente.

String getID ()

Esse método retorna o ID que unicamente identifica o objeto. O ID é gerado pelo framework de CORBA.

void setNome ()

Este método possibilita a lateração do nome do componente.

Classe Publicador

Essa classe terá definida toda a estrutura dos elementos que são publicadores.

Derivado de Componente

Propriedades

Subscritor listSubscritores

Este atributo é uma lista que contém os subcritores que estão registrados neste publicador. Os métodos Registrar() e desRegistrar() alteram diretamente essa lista.

Publicacao issue

Esse método contém uma instância de publicação, que se refere ao que está sendo publicado pelo objeto. Assim cada classe que se derivar de Publicador, estará publicando um tipo de Publicacao.

Métodos

short Registrar (p_subscritor : Subscritor)

Esse método é exposto pelos publicadores para que os subscritores possam se registrar nestes.

void enviarPublicacao ()

Esse método faz com que o o publicador envie uma publicação para cada subcritor na sua lista. É utilizado para se trabalhar com esquema Push. Neste caso haverá um objeto Schedule associado ao publicador.

short desRegistrar (p_subscritor : Subscritor)

Método utilizado para remover o registro de um subscritor sobre um publicador.

Subscritor[] ListarSubscritores ()

Esta função retorna uma lista com os subcritores registrados no publicador.

Classe Schedule

Classe que define os Schedules, que serão responsáveis pelas varreduras dos componentes.

Derivado de Publicador

Propriedades

long ScanTime

Este parâmetro define o tempo de varredura do Schedule.

int TipoSubscritor

Este parâmetro vai determinar se o Schedule recebe Subscritores ou Históricos.

Métodos

Long getScanTime ()

Este método retorna o tempo de varredura do elemento.

void setScanTime (scan : long)

Este método seta o tempo de varredura de Schedule.

void Executar ()

Este método vai fazer com que os publicadores registrados enviem suas publicações e com que o Histórico mande para a base de dados os seus dados.

void Parar ()

Este método determina a parada da varredura. Assim todos os subscritores registrados no esquema serão parados.

void Rodar ()

Este método faz o Schedule rodar. É chamado ao se instanciar o objeto.

Classe ScheduleSensor

Derivado de Schedule

Propriedades

Sensor[] ListSensores

Variável que contém uma lista dos Sensores registrados n Schedule.

Métodos

short RegistrarSensor (s : Sensor)

Método utilizado pelos Sensores para se subscreverem no Schedule

short desRegistrarSensor ()

Método utilizado pelos Sensores para se desubscreverem no Schedule

Classe Subscritor

Esta é a classe que contém toda a estrutra a ser derivada por todos os subscritores do sistema.

Derivado de Componente

Propriedades

Publicador mPublicador

Esta variável indicará em qual publicador o subscritor está registrado.

Métodos

void Suscrever (IDPublicador : string = default, Tipo : short = default)

Esse método contém todo procedimento que o subscritor deve fazer para se registrar em um publicador.

publicador[] buscarPublicadores (tipo : string)

Esse método implementa o procedimento que deve ser tomado para se fazer busca no sistema de diretórios especificado pela arquitetura CORBA. Recebe como parâmetro uma string que indicará que tipo de publicador está sendo procurado.

void unSubscrever (argname : argtype = default)

Esse método é chamado para se remover o registro do objeto no publicador em que está registrado.

void receberPublicacao ()

Esse método é exposto para que os publicadores possam enviar publicações aos subscritores cmo esquema Push.

Classe Valor

Este tipo, deine alguns métodos e propriedades básicas que todos os publicadores de valor terão.

Derivado de Publicador

Propriedades

Schedule Esquema

Esta propriedade vai conter uma uma referência ao Schedule em que este publicador está registrado.

Métodos

void setEsquema ()

Este método vai setar um novo scheulle para o publicador de valor se registrar.

Schedule getEsquema ()

Este método vai retornar o scheulle em que o publicador de valor está registrado.

Publicacao getPublicacao ()

Retorna a uma publicação de valor.

Classe Sensor

A classe sensor representa uma entidade física. Nesta implemtentação, o sensor terá a oporunidade funcionar como push ou pull, dependendo de como o subscritor se registrar nele. Caso o sensor não tenha se registrado em nenhum Schedule, não será permitido ao subscritor se registrar como push.

Derivado de Valor e de Subscritor

Propriedades

string Variavel

Indica o nome da grandeza física sendo medida.

string Unidade

Indica a unidade da grandeza física sendo medida.

double ValorMax

Indica o máximo valor que a variável pode atingir.

double ValorMin

Indica o mínimo valor que a variável pode atingir.

Atuador[] ListAtuadores

Métodos

Retorna a variável (grandeza física) sendo medida.

string setVariavel (m_variavel : string) : voidgetVariavel ()

Seta a variável sendo medida.

string getUnidade ()

Retorna a unidade da grandeza física sendo medida.

Void setUnidade (m_unid : string)

Seta a unidade da grandez física sendo medida.

double getValorMax ()

Retorna o maior valor a ser medido pelo sensor.

void setValorMax (vMax : double)

Seta o maior valor a ser medido pelo sensor.

double getValorMin ()

Retorna o menor valor a ser medido pelo sensor.

void setValorMin (vMin : double)

Seta o menor valor a ser medido pelo sensor.

void setMaxDeadBand (maxdb : double)

Seta o limite superior da Banda Morta.

double getMaxDeadBand ()

Retorna o limite superior da banda morta setada..

Void setMinDeadBand ()

Seta o limite inferior da Banda Morta.

void getMinDeadBand ()

Retorna o limite inferior da banda morta.

boolean IsDeadBandActive ()

Indica se a Banda morta esta ativa ou não.

void ActivateDeadBand (activar : boolean)

Ativa/Desativa a banda morta.

Short RegistrarAtuador (a : Atuador)

Este método permite o atuador subscrever um sensor.

short desRegistrarAtuador ()

Este método permite ao Atuador se desubscrever do sensor.

short registrarGUI (g : GUI)

Permite ao GUI se registrar no sensor.

short desRegistrarGUI (g : GUI)

Permite ao GUI se desusbscrever do sensor.

short RegistrarAtuador ()

Permite ao Atuador subscrever no sensor.

short desRegistrarAtuador ()

Permite ao atuador de desregistrar do sensor.

Classe Atuador

Classe básica que implementa atuador. Para esta implementação o atuador apenas passa seu estado para um valor proporcional ao valor recebido.

Derivado de Subscritor e de Valor

Propriedades

double Constante

Define uma constante. O atuador no caso vai ser apenas um valor proporcional ao valor de entrada.

Métodos

Void setConstante () :

Vai setar o valor da constante de proporcionalidade.

double getConstante ()

Este método retorna o valor da constante de proporcionalidade.

Classe GUI

Esta é a classe genérica de elementos gráficos.

Derivado de Subscritor

Propriedades

long npontos

Contém o número de pontos a serem plotados.

Publicacao ultimoValor

Mantém registro do último valor publicado

Métodos

Void Refresh ()

Replota os pontos na tela.

Void setNPontos (npontos : long)

Altera o número de pontos a serem plotados.

long getNPontos ()

Lê o número de pontos a serem plotados.

Feita a modelagem, com a identificação das classes do sistema, a próxima etapa será a de implementação do sistema. Uma descrição da implementação, juntamente com o seu código fonte está descrita no próximo capítulo.

8. IMPLEMENTAÇÃO

Conforme mencionado na capítuilo 5 deste trabalho, CORBA é um padrão definido por um consórcio de mais 800 empresas, mas que ainda está desenvolvendo e criando padrões para que todas as implementações sejam portáveis e se comuniquem. Enquanto esse caminho é trilhado, as implementações de ORB existentes no mercado apresentam pequenas diferenças na funcionalidade e no código. No caso desta implementação foi utlilizado o ORBacus ®, da OOC, e uma descrição deste produto pode ser encontrada no apêndice B.

A seguir será apresentado o código de implementaçào do projeto. Primeiramente será apresentado o código das interfaces de CORBA (IDL), em seguida as implementações das classes e finalmente o código dos 'conteiners' ou 'servidores', que são os pedaços do *software* responsáveis por inicializar o ORB, intanciar o objeto e disponibilizá-los para os outros objetos. Por último serão colocados os códigos das applets do sistema. Não serão apresentados os códigos gerados pelo compilador CORBA (IDL para Java), já que estes somariam um grande volume de documentação, e são transparentes para o desenvolvedor.

8.1 IDL

```
// Modulo com todas as interfaces
module scada
{

      // Abaixo estão algumas declarações que serão feitas depois
      // Essa primeira declaração será feita apenas para que se
      // possa se referenciar esses objetos dentro de outros
      // objetos que vêm antes
      interface Subscritor;
      interface Publicador;
      interface Publicacao;
      interface Sensor;
      interface ScheduleSensor;
      interface GUI;
      interface Atuador;
      interface Alarme;
      interface pubValor;

       // Declara uma listas de objetos definidos
      typedef sequence<Subscritor> ListSubscritor;
      typedef sequence<Publicador> ListPublicador;
      typedef sequence<Publicacao> ListPublicacao;
      typedef sequence<pubValor> ListPubValor;
      typedef sequence<ScheduleSensor> ListScheduleSensor;
      typedef sequence<GUI> ListGUI;
      typedef sequence<Sensor> ListSensor;
      typedef sequence<Atuador> ListAtuador;

      // Interface para classe que define um tipo para datas
      interface Data
      {
            void setDia(in short d);
            void setMes(in short m);
            void setAno(in long a);
            string getData();
```

```
        string getDataFormatada(in string formato);
        unsigned short getDia();
        unsigned short getMes();
        unsigned long getAno();
        string getDiaDaSemana();
        short EMaior(in Data datacomp);
        void Adiciona(in string param, in string quant);
    };

    // Interface para classe que define um tipo para data com
Horário
    interface DataHora: Data
    {
        void setHora(in short h);
        void setMinuto(in short m);
        void setSegundo(in short s);
        string getHorario();
        string getHorarioFormatada(in string formato);
        unsigned short getHora();
        unsigned short getMinuto();
        unsigned short getSegundo();
    };

    // Interface Publicacao
    interface Publicacao
    {
        short getIDPublicacao();
        DataHora getHorario();
    };

    // Interface de publicação de valor
    interface pubValor: Publicacao
    {
        double getValor();
        void setValor(in double novo_valor);
    };
```

```
// Interface Basica para Todos os devices
interface Componente
{
      string getNome();
      string getID();
      void setID(in string id);
};

// interface Publicador
interface Publicador: Componente
{
      short Registrar(in Subscritor m_subscritor);
      void enviarPublicacao();
      void desRegistrar(in Subscritor m_subscritor);
      ListSubscritor listarSubscritores();
};

// Interface basica dos Schedules
interface Schedule: Publicador
{
      unsigned long getScanTime();
      void setScanTime(in unsigned long novo_scantime);
      void Executar();
      void Parar();
      void Rodar();
};


// Interface basica dos Schedules
interface ScheduleSensor:Schedule
{
      short Registrar(in Sensor m_sens);
      void desRegistrar(in Sensor m_sens);
};

// interface subcritores
interface Subscritor: Componente
```

```
    {
        ListPublicador buscaPublicadores(in string tipo);
        Publicador buscaPublicador(in string tipo, in string
nome_publicador);
        void Subscrever(in Publicador m_publicador);
        void desSubscrever();
        void receberPublicacao(in Publicacao m_publicacao);
    };

    // interface dos publicadores de valor
    interface Valor: Publicador
    {
        void setEsquema(in Schedule esquema);
        Schedule getEsquema();
        pubValor getPublicacao();
    };

    // interface para os sensores
    interface Sensor: Valor, Subscritor
    {
        string getVariavel();
        void setVariavel(in string nova_variavel);
        string getUnidade();
        void setUnidade(in string nova_unidade);
        double getValMax();
        void setValMax(in double novo_valmax);
        double getValMin();
        void setValMin(in double novo_valmin);
        Alarme getAlarme();
        void setAlarme(in Alarme novo_alarme);
        double getMaxDeadBand();
        void setMaxDeadBand(in double novo_maxdeadband);
        double getMinDeadBand();
        void setMinDeadBand(in double novo_mindeadband);
        boolean IsDeadBandActive();
        void ActivateDeadBand(in boolean novo_deadbandstate);
        short RegistrarGUI(in GUI g);
        short RegistrarAtuador(in Atuador a);
```

```
        short desRegistrarGUI(in GUI g);
        short desRegistrarAtuador(in Atuador a);
    };

    // interface Atuador
    interface Atuador: Valor, Subscritor
    {
        double getConstante();
        void setConstante(in double nova_constante);
        void RecebeValor(in pubValor pv);
    };

    // interface GUI
    interface GUI: Subscritor
    {
        void Refresh();
        void setNPontos(in unsigned long novo_npontos);
        unsigned long getNPontos();
        void setTipo(in short novo_tipo);
        short getTipo();
        ListPubValor getPontos();
        void ReceberValor(in pubValor p);

    };

    // Interface do coordenador
    interface Coordenador
    {
        // Métodos para se registrar
        short RegistrarSensor(in Sensor s);
        short RegistrarAtuador(in Atuador a);
        short RegistrarGUI(in GUI g);
        short RegistrarScheduleSensor(in ScheduleSensor s);
        short RegistrarScheduleHistorico(in ScheduleHistorico
s);
        // Métodos para se desregistrar
        short desRegistrarSensor(in Sensor s);
        short desRegistrarAtuador(in Atuador a);
```

```
            short desRegistrarGUI(in GUI g);
            short desRegistrarScheduleSensor(in ScheduleSensor s);
            // Métodos para de listar objetos
            ListSensor getSensores();
            ListAtuador getAtuadores();
            ListGUI getGUIs();
            ListScheduleSensor getScheduleSensores();
            // Métodos para se pegar objetos específicos
            Sensor getSensor(in string nome);
            Atuador getAtuador(in string nome);
            GUI getGUI(in string nome);
            ScheduleSensor getScheduleSensor(in string nome);
            // busca o número de elementos dado um determinado tipo
            long getQuantidade(in string tipo);
        };


};
```

8.2 Implementações

Classe Data

```
// Implementação da classe Data

package scada;

import java.lang.*;
import java.util.*;

public class Data_impl extends _DataImplBase
{
      private short dia;
      private short mes;
      private int ano;
      private Calendar m_data;

      //O Construtor seta a data atual como default
      public void Data_impl()
      {
            m_data = Calendar.getInstance();
            dia = (short)m_data.get(Calendar.DAY_OF_MONTH);
            mes = (short)m_data.get(Calendar.MONTH);
            ano = (short)m_data.get(Calendar.YEAR);

      }

      // Seta dia
      public void setDia(short d)
      {
            this.dia = d;
      }

      // Seta mes
      public void setMes(short m)
      {
```

```
        this.mes = m;
}

// Seta ano
public void setAno(int a)
{
        this.ano = a;
}

// Retorna a data em forma de uma string default
public String getData()
{
        return getDataFormatada("dd/mm/yy");
}

// retorna uma string em forma no formato especificado
public String getDataFormatada(String formato)
{
        String strDia = new String();
        String strMes = new String();
        String strAno = new String();
        String strData = new String();

        if (formato.compareTo("dd/mm/yy") == 0)
        {       strDia = Integer.toString(dia);
                strMes = Integer.toString(mes);
                strAno = Integer.toString(ano);
                strData = strDia + "/" + strMes + "/" + strAno; }
        else if (formato.compareTo("dd-mm-yy") == 0)
        {       strDia = Integer.toString(dia);
                strMes = Integer.toString(mes);
                strAno = Integer.toString(ano);
                strData = strDia + "-" + strMes + "-" + strAno; }
        else if (formato.compareTo("dd/mmm/yy") == 0)
        {       strDia = Integer.toString(dia);
                strMes = Integer.toString(mes);
                strAno = Integer.toString(ano);
```

```
            strData = strDia + "/" + strMes + "/" + strAno; }
        else
        { System.out.println("Formato Errado");     }

        return strData;
}

// Retorna o dia
public short getDia()
{
        return dia;
}

// Retorna o mes
public short getMes()
{
        return mes;
}

// Retorna o ano
public int getAno()
{
        return ano;
}

// Compara duas datas
public short EMaior (Data datacomp)
{
        if (datacomp.getAno() > this.ano) {return 1;}
        else if (datacomp.getAno() < this.ano) {return -1;}
        else if (datacomp.getMes() > this.mes) {return 1;}
        else if (datacomp.getMes() < this.mes) {return -1;}
        else if (datacomp.getDia() > this.dia) {return 1;}
        else if (datacomp.getDia() < this.dia) {return -1;}
        else return 0;
}
```

```
        // Adiciona uma quantidade à data
        public void Adiciona(String param, String quant)
        {
              Integer valor = new Integer(quant);
              if (param == "d")
              {     this.dia += valor.shortValue(); }
              else if (param == "m")
              {     this.mes += valor.shortValue(); }
              else if (param == "a")
              {     this.ano += valor.intValue(); }
              else
              {     System.out.println("Parmatros incorretos"); }
        }


}       // Fim da classe Data_impl
```

Classe DataHora

```
// Implpementação da classe DataHora

package scada;

import java.lang.*;
import java.util.*;

public class DataHora_impl extends _DataHoraImplBase
{
      private short hora;
      private short minuto;
      private short segundo;
      private short dia;
      private short mes;
      private int ano;
      private Calendar m_date;
      private Data_impl m_data;

      //O Construtor seta a data atual como default
      DataHora_impl()
      {
            super();
            m_date = Calendar.getInstance();
            hora = (short)m_date.get(Calendar.HOUR);
            minuto = (short)m_date.get(Calendar.MINUTE);
            segundo = (short)m_date.get(Calendar.SECOND);
            m_data = new Data_impl();
      }

      // Seta dia
      public void setDia(short d)
      {
            m_data.setDia(d);
      }
```

```
// Seta mes
public void setMes(short m)
{
      m_data.setMes(m);
}

// Seta ano
public void setAno(int a)
{
      m_data.setAno(a);
}

// Set hora
public void setHora(short h)
{
      this.hora = h;
}

// Set minuto
public void setMinuto(short m)
{
      this.minuto = m;
}

// Set segundo
public void setSegundo(short s)
{
      this.segundo = s;
}

// Retorna data em forma de uma string default
public String getHorario()
{
      return getHorarioFormatada("hh:mm:ss");
}

// Retorna uma string em forma no formato especificado
```

```
    public String getHorarioFormatada(String formato)
    {
        String strSeg = new String();
        String strMin = new String();
        String strHora = new String();
        String strDia = new String();
        String strMes = new String();
        String strAno = new String();
        String strData = new String();

        if (formato.compareTo("dd/mm/yy hh:mm") == 0)
        {   strSeg = Integer.toString(segundo);
            strMin = Integer.toString(minuto);
            strHora = Integer.toString(hora);
            strDia = Integer.toString(dia);
            strMes = Integer.toString(mes);
            strAno = Integer.toString(ano);
            strData = strDia + "/" + strMes + "/" + strAno + "
" + strHora + ":" + strMin; }
        else if (formato.compareTo("dd-mm-yy hh:mm") == 0)
        {   strSeg = Integer.toString(segundo);
            strMin = Integer.toString(minuto);
            strHora = Integer.toString(hora);
            strDia = Integer.toString(dia);
            strMes = Integer.toString(mes);
            strAno = Integer.toString(ano);
            strData = strDia + "-" + strMes + "-" + strAno + "
" + strHora + ":" + strMin; }
        else if (formato.compareTo("dd/mmm/yy hh:mm") == 0)
        {   strSeg = Integer.toString(segundo);
            strMin = Integer.toString(minuto);
            strHora = Integer.toString(hora);
            strDia = Integer.toString(dia);
            strMes = Integer.toString(mes);
            strAno = Integer.toString(ano);
            strData = strDia + "/" + strMes + "/" + strAno + "
" + strHora + ":" + strMin; }
        else if (formato.compareTo("hh:mm:ss") == 0)
```

```
        {     strSeg = Integer.toString(segundo);
              strMin = Integer.toString(minuto);
              strHora = Integer.toString(hora);
              strDia = Integer.toString(dia);
              strMes = Integer.toString(mes);
              strAno = Integer.toString(ano);
              strData = strHora + ":" + strMin + ":" + strSeg; }
        else
        { System.out.println("Formato Errado");     }

        return strData;
}

//Metodo getHora
public short getHora()
{
        return this.hora;
}

// Método getMinuto
public short getMinuto()
{
        return this.minuto;
}

// Método getSegundo
public short getSegundo()
{
        return this.segundo;
}

// Método getDataFormatada(String formato)
public String getDataFormatada(String formato)
{
        return m_data.getDataFormatada(formato);
}
```

```
// Método getData()
public String getData()
{
      return m_data.getData();
}

// Método getAno()
public int getAno()
{
      return m_data.getAno();
}

// Método getMes()
public short getMes()
{
      return m_data.getMes();
}

// Método getDia()
public short getDia()
{
      return m_data.getDia();
}

// Compara duas datas
public short EMaior(Data dataref)
{
      DataHora m_ref;
      m_ref = (DataHora) dataref;
      if (m_ref.getAno() > m_data.getAno()) {return 1;}
      else if (m_ref.getAno() < m_data.getAno()) {return -1;}
      else if (m_ref.getMes() > m_data.getMes()) {return 1;}
      else if (m_ref.getMes() < m_data.getMes()) {return -1;}
      else if (m_ref.getDia() > m_data.getDia()) {return 1;}
      else if (m_ref.getDia() < m_data.getDia()) {return -1;}
      else if (m_ref.getHora() > this.hora) {return 1;}
      else if (m_ref.getHora() < this.hora) {return -1;}
```

```
            else if (m_ref.getMinuto() > this.minuto) {return 1;}
            else if (m_ref.getMinuto() < this.minuto) {return -1;}
            else if (m_ref.getSegundo() > this.segundo) {return 1;}
            else if (m_ref.getSegundo() > this.segundo) {return -1;}
            else {return 0;}
      }

      // Método Adiciona()
      public void Adiciona(String param, String quant)
      {
            Integer valor = new Integer(quant);
            if (param == "a")
            { m_data.Adiciona(param, quant); }
            else if (param == "m")
            { m_data.Adiciona(param, quant); }
            else if (param == "d")
            { m_data.Adiciona(param, quant); }
            else if (param == "h")
            { hora += valor.shortValue(); }
            else if (param == "mi")
            { minuto += valor.shortValue(); }
            else if (param == "s")
            { segundo += valor.shortValue(); }
            else
            {     System.out.println("Parmatros incorretos"); }
      }
}
```

Classe Publicação

```
// Implementação da classe Classe Publicacao

package scada;

class Publicacao_impl extends _PublicacaoImplBase
{
	private short IDPublicacao;
	private DataHora horario;
	private DataHora_impl m_hora;

	// Contrutor da classe
	// Cria um horário, logo horário corrente por dafault
	public Publicacao_impl()
	{
		this.m_hora = new DataHora_impl();
		horario = DataHoraHelper.narrow(m_hora);
	}

	// retorna o ID da publicação
	public short getIDPublicacao()
	{
		return this.IDPublicacao;
	}

	// Retorna o horário
	public DataHora getHorario()
	{
		return horario;
	}

}
```

Classe pubValor

```
// Classe pubValor

package scada;

import java.lang.String;
import java.util.Date;

public class pubValor_impl extends _pubValorImplBase
{
      // Declara variáveis contendo as propriedades privadas
      private short IDPub;
      // Java não suporta herança múltipla
      private Publicacao_impl m_pub;
      private double valor;
      private DataHora_impl m_data;
      private Date tmp;

      // Contrutor
      // Cria datas e seta valor atual para estas
      public pubValor_impl(double valor)
      {
            IDPub = 2;
            tmp = new Date();
            this.valor = valor;
            m_data = new DataHora_impl();
            m_data.setHora((short)tmp.getHours());
            m_data.setMinuto((short)tmp.getMinutes());
            m_data.setSegundo((short)tmp.getSeconds());

      }

      // Retorna o valor
      public double getValor()
      {
            return this.valor;
```

```
        }

        // Atualiza o valor
        public void setValor(double valor)
        {
              this.valor = valor;
        }

        // Retorna o ID de publicação
        public short getIDPublicacao()
        {
              return IDPub;
        }

        // Retorna o horário
        public DataHora getHorario()
        {
              return DataHoraHelper.narrow(m_data);
        }
}       // Fim da classe
```

Classe Componente

```
// Classe Componente

package scada;

import java.lang.String;

class Componente_impl extends _ComponenteImplBase
{
	// Declara propriedades privadas
	private String nome;
	private String id;

	// Contrutor
	// recebe nome do componente ao ser criado
	Componente_impl(String nome)
	{
		super();
		this.nome = nome;
	}

	// retorna Nome
	public String getNome()
	{
		return this.nome;
	}

	// retorna ID do componente
	public String getID()
	{
		return this.id;
	}

	// Altera ID do Componente
	public void setID(String id)
	{
```

```
            this.id = id;

        }
}       // Fim da classe
```

Classe Publicador

```
// Classe Publicador

package scada;

import java.util.*;

class Publicador_impl extends _PublicadorImplBase
{
      // Declara propriedades privada
      private Componente m_comp;
      private Hashtable listSub;
      private Publicacao issue;

      // Contrutor
      public Publicador_impl(String nome)
      {
            m_comp = new Componente_impl(nome);
      }

      // Retorna o ID do componentes
      public String getID()
      {
            return m_comp.getID();
      }

      // Retorna o nome
      public String getNome()
      {
            return m_comp.getNome();
      }

      // Altera o ID
      public void setID(String id)
      {
            m_comp.setID(id);
```

```
    }

    // Registra Subscritor
    public short Registrar(Subscritor subs)
    {
        listSub.put(subs.getNome(), subs);
        return 0;
    }

    // Desregistra subscritor
    public void desRegistrar(Subscritor s)
    {
        listSub.remove(s.getNome());
    }

    // Retorna publicação corrente
    public Publicacao getPublicacao()
    {
        return issue;
    }

    // Método abstrato para enviar publicação
    public void enviarPublicacao() {}

    // Lista os subscritores registrados no publicador
    public Subscritor[] listarSubscritores()
    {
        Subscritor[] subs = new Subscritor[listSub.size()];
        int i;
        Enumeration e = listSub.keys();
        i = 1;
        while (e.hasMoreElements())
        {
            subs[i] =
(Subscritor)listSub.get(e.nextElement());
            i++;
        }
        return subs;
```

```
        }

}       // Fim da classe
```

Classe Schedule

```
// Classe Schedule

package scada;

import java.lang.String;
import java.lang.Integer;
import java.util.Hashtable;
import java.util.Enumeration;

public class Schedule_impl extends _ScheduleImplBase implements
Runnable
{
      // Declara propriedades privados
      private int ScanTime;
      private short tipo_sched;
      private Hashtable listSubscritores;
      private SchedRun my_timer;
      private Publicador_impl m_pub;
      private Thread t;
      // Contrutor
      // Cria uma tabela para armazenar os contrutores e instacia
      //um Timer (SchedRun)
      public Schedule_impl(String nome,int scan_time, short tipo)
      {
            this.ScanTime = scan_time;
            this.tipo_sched = tipo;
            listSubscritores = new Hashtable();
            m_pub = new Publicador_impl(nome);
            my_timer = new SchedRun(this, scan_time);
            t = new Thread(this);
            System.out.println("Schedule rodando....");
      }

      // Implementa este método pois a classe implemtenta Runnable
      public void run()
      {
```

```
        my_timer.start();
    }

    // Retorna o ID do COmponente
    public String getID()
    {
        return m_pub.getID();
    }

    // Retorna o nome do componente
    public String getNome()
    {
        return m_pub.getNome();
    }

    // Altera o ID do componente
    public void setID(String id)
    {
        m_pub.setID(id);
    }

    // Altera o Scna Time
    public void setScanTime(int novo_scantime)
    {
        System.out.println("Passou o argumento : " +
Integer.toString(novo_scantime));
        this.ScanTime = novo_scantime;
        this.my_timer.setScanTime(novo_scantime);
    }

    // retorna o Scan Time
    public int getScanTime()
    {
        return this.ScanTime;
    }

    // Método executar
    public void Executar()
```

```
    {
        System.out.println("Executou...");
        //Se for um Schedule para Sensor
        if (tipo_sched == 1)
        {
            System.out.println("Entrou no if...");
            Sensor s;
            // Para cada Subscritor registrado enviar ordem de
            // exectutar
            for (Enumeration e = listSubscritores.elements();
        e.hasMoreElements();)
            {
                s = (Sensor)e.nextElement();
                s.enviaPublicacao();
                System.out.println("Elemento " +
s.getNome());
            }
        }
    }

    // Para o Timer
    public void Parar()
    {
        this.my_timer.Parar();
    }

    // Inicia o Timer
    public void Rodar()
    {
        t.start();
        this.my_timer.start();
    }

    // Registra um subscritor
    public short Registrar(Subscritor s)
    {
        Sensor ss = (Sensor)s;
        System.out.println("Registrou");
```

```
        System.out.println("Registrou " + ss.getNome());
        listSubscritores.put(ss.getNome(), ss);
        System.out.println("Adicionou à lista...");
        return (short)1;
    }

    // Desregistra o subsritor da lista
    public void desRegistrar(Subscritor s)
    {
        m_pub.desRegistrar(s);
    }

    // Lista Subscritores
    public Subscritor[] listarSubscritores()
    {
        return m_pub.listarSubscritores();
    }

    //  Não há publicação para ser enviada
    public void enviarPublicacao()
    {

    }

    // Esta classe implementa o timer, que é o elemento
    responsável por // fazer a varredura
    class SchedRun extends Thread implements Runnable
    {
        // Declara propriedades privadas
        private Schedule_impl meu_dono;
        private int scantime;
        private boolean rodando;

        // Construtor
        public SchedRun(Schedule_impl m_dono, int st)
        {
            this.meu_dono = m_dono;
            this.scantime = st;
```

```
        }

        // Executa
        public void run()
        {
              rodando = true;
              while(rodando)
              {
                    try{
                    meu_dono.Executar();
                    sleep(scantime);
                    } catch (InterruptedException e)
                    { System.out.println(e); }
              }
        }

        // Pára o timer
        public void Parar()
        {
              rodando = false;
        }

        // Altera o Scan Time Corrente
        public void setScanTime(int sc)
        {
              this.scantime = sc;
        }

        // Retorna o Scan Time Corrente
        public int getScanTime()
        {
              return this.scantime;
        }
    }   // Fim da classe SchedRun
}       // Fim da Classe Schedule
```

Classe ScheduleSensor

```
// Classe Schedule

package scada;

import java.lang.String;
import java.lang.Integer;
import java.util.Hashtable;
import java.util.Enumeration;

public class ScheduleSensor_impl extends _ScheduleSensorImplBase
implements Runnable
{
      // Propriedades privadas
      private int ScanTime;
      private short tipo_sched;
      private Hashtable listSubscritores;
      private SchedRun my_timer;
      private Publicador_impl m_pub;
      private String nome;

      // Contrutor
      // Cria uma lista para os sensores e instancia um timer
      public ScheduleSensor_impl(String nome,int scan_time, short
tipo)
      {
            this.ScanTime = scan_time;
            this.tipo_sched = tipo;
            this.nome = nome;
            listSubscritores = new Hashtable();
            m_pub = new Publicador_impl(nome);
            my_timer = new SchedRun(this, scan_time);
            Thread t = new Thread(this);
            t.start();
            System.out.println("Shedulle Sensor rodando...");
      }
```

```
// Inicializa o timer
// Este método é chamado ao se inicializar a Thread
public void run()
{
      my_timer.start();
}

// Retorna o ID do objeto
public String getID()
{
      return m_pub.getID();
}

// Retorna o nome do objeto
public String getNome()
{
      return this.nome;
}

// Seta um novo ID para o objeto
public void setID(String id)
{
      m_pub.setID(id);
}

// Altera o scan time do timer
public void setScanTime(int novo_scantime)
{
      this.ScanTime = novo_scantime;
      this.my_timer.setScanTime(novo_scantime);
}

// Retorna o scan time corrente
public int getScanTime()
{
      return this.ScanTime;
}
```

```
// Manda ordem a todos os sensores registrados para enviarem
// publicações

public void Executar()
{
      DataHora_impl dh = new DataHora_impl();
      System.out.println("Executou as " + dh.getHorario());
      Sensor s;
      // Varre a lista de sensores registrados
      for (Enumeration e = listSubscritores.elements();
e.hasMoreElements();)
      {
            s = (Sensor)e.nextElement();
            s.enviarPublicacao();
      }
}

// Pára o timer
public void Parar()
{
      my_timer.Parar();
}

// Inicia o timer
public void Rodar()
{
      my_timer.run();
}

// Registra novo sensor
public short Registrar(Sensor s)
{
      System.out.println("Registrou " + s.getNome());
      listSubscritores.put(s.getNome(), s);
      System.out.println("Adicionou a lista...");
      return (short)1;
}
```

```
// Desregistra o sensor
public void desRegistrar(Sensor s)
{
	listSubscritores.remove(s.getNome());
}

// Lista todos os sensores registrados
public Subscritor[] listarSubscritores()
{
	return m_pub.listarSubscritores();
}

// Esta classe implementa o timer
class SchedRun extends Thread implements Runnable
{
	private ScheduleSensor_impl meu_dono;
	private int scantime;
	private boolean rodando;

	// Contrutor
	// Só pode ser chamado pelo próprio ScheduleSensor
	SchedRun(ScheduleSensor_impl m_dono, int st)
	{
		this.meu_dono = m_dono;
		this.scantime = st;
	}

	// Executa
	public void run()
	{
		rodando = true;
		while(rodando)
		{
			try{
			meu_dono.Executar();
			sleep(scantime);
```

```
                    } catch (InterruptedException e)
                    { System.out.println(e); }
               }
          }

          // Altera variável para parar
          public void Parar()
          {
               rodando = false;
          }

          // Altera o scan time
          public void setScanTime(int sc)
          {
               this.scantime = sc;
          }

          // retorna o scan Time
          public int getScanTime()
          {
               return this.scantime;
          }
     }    // Fim da classe SchedRun
}         // Fim da classe ScheduleSensor
```

Classe Subscritor

```
// Classe Subscritor

package scada;

import java.util.*;

public class Subscritor_impl extends _SubscritorImplBase
implements Subscritor
{
      // Daclara as propriedades privadas
      private Publicador pub;
      private Publicacao p_cor;
      private Componente_impl m_com;
      private String mNome;

      // Contrutor
      public Subscritor_impl(String nome)
      {
            super();
            mNome = nome;
            m_com = new Componente_impl(nome);
      }

      // Retorna o nome do componente
      public String getNome()
      {
            System.out.println("Chamou getNome()...");
            return mNome;
      }

      // Retorna o ID do componente
      public String getID()
      {
            return m_com.getID();
      }
```

```
        // Altera ID do componente
        public void setID(String id)
        {
              m_com.setID(id);
        }

        // Registra-se no publicador
        public void Subscrever(Publicador p)
        {
              this.pub = p;
              p.Registrar(this);
        }

        // Retorna os publicadores (abstrata)
        public Publicador[] buscaPublicadores(String tipo)
        {
              return null;
        }

        // Retorna publicador específico (abstrata)
        public Publicador buscaPublicador(String tipo, String nome)
        {
              return null;
        }

        // Desregistra o subscritor do publicador
        public void desSubscrever()
        {
              pub.desRegistrar(this);
        }

        // Recebe publicação
        public void receberPublicacao(Publicacao p)
        {
              this.p_cor = p;
        }
}
```

Classe Valor

```
// Classe de Valor

package scada;

import java.lang.*;
import java.util.*;

class Valor_impl extends _ValorImplBase
{
	// Declara propriedades privadas
	private Schedule esquema;
	private double valor;
	private Publicador_impl m_pub;

	// Construtor
	public Valor_impl(String nome)
	{
		m_pub = new Publicador_impl(nome);
	}

	// retorna nome
	public String getNome()
	{
		return m_pub.getNome();
	}

	// Retorna ID
	public String getID()
	{
		return m_pub.getID();
	}

	// Altera ID
	public void setID(String id)
	{
```

```
        m_pub.setID(id);
    }

    // Registra Subscritor
    public short Registrar(Subscritor s)
    {
        return m_pub.Registrar(s);
    }

    // Desregistra subscritor
    public void desRegistrar(Subscritor s)
    {
        m_pub.desRegistrar(s);
    }

    // Envia publicação
    public void enviarPublicacao()
    {
        m_pub.enviarPublicacao();
    }

    // Lista os subsritores registrados
    public Subscritor[] listarSubscritores()
    {
        return m_pub.listarSubscritores();
    }

    // Altera o Schedule
    public void setEsquema(Schedule esq)
    {
        this.esquema = esq;
    }

    // retorna o Schedule
    public Schedule getEsquema()
    {
        return this.esquema;
```

```
        }

        // Retorna a publicação corrente
        public pubValor getPublicacao()
        {
              pubValor_impl p = new pubValor_impl(this.valor);
              pubValor issue = pubValorHelper.narrow(p);
              return issue;
        }
}
```

Classe Sensor

```
// Classe Sensor

package scada;

import java.util.*;
import java.lang.*;

public class Sensor_impl extends _SensorImplBase implements
Runnable
{
      // Declara propriedades privadas
      private Componente_impl m_com;
      private Publicador_impl m_pub;
      private Subscritor_impl m_sub;
      // Listas que vão conter o registraos de sensores,
      // subscritores e atuadoers registrados
      private LinkedList lSub, lGUI, lAtu;
      private Valor_impl m_val;
      private ScheduleSensor esquema;
      private String variavel;
      private String unidade;
      public double valor;
      private double last_valor;
      private double val_max;
      private double val_min;
      private double db_max;
      private double db_min;
      private boolean dbativa;
      private Planta pl;

      // Construtor
      public Sensor_impl(String nome)
      {
            super();
            m_com = new Componente_impl(nome);
            m_pub = new Publicador_impl(nome);
```

```
        m_sub = new Subscritor_impl(nome);
        m_val = new Valor_impl(nome);
        lSub = new LinkedList();
        lGUI = new LinkedList();
        lAtu = new LinkedList();
        valor = 11;
        db_max = 2;
        db_min = 2;
        last_valor = 10;
        val_min = 5;
        val_max = 50;
        dbativa = true;
        // Cria um elemento planta
        pl = new Planta(this, 2000);
        Thread t = new Thread(this);
        t.start();
}


// Métodos de componente
public String getNome()
{
        return m_com.getNome();
}

public String getID()
{
        return m_com.getID();
}

public void setID(String id)
{
        m_com.setID(id);
}

// Métodos de Publicador
public short Registrar(Subscritor s)
```

```
        {
            System.out.println("Subscritor " + s.getNome() + "
recebido");
            lSub.add(s);
            return 0;
        }

        // Registra GUI
        public short RegistrarGUI(GUI g)
        {
            System.out.println("GUI " + g.getNome() + " recebido");
            lGUI.add(g);
            return 0;
        }

        // Desregistra GUI
        public short desRegistrarGUI(GUI g)
        {
            System.out.println("Entrou em desRegistrar GUI");
            GUI tmp;
            for(int i=0; i < lGUI.size(); i++)
            {
                tmp = (GUI)lGUI.get(i);
                if (g.getNome().compareTo(tmp.getNome())==0)
                {
                    lGUI.remove(i);
                }
            }
            return 0;
        }

        // desRegistra Atuador
        public short desRegistrarAtuador(Atuador a)

            System.out.println("Entrou em desRegistrar Atuador");
            Atuador tmp;
            for(int i=0; i < lAtu.size(); i++)
            {
```

```
                tmp = (Atuador)lAtu.get(i);
                if (a.getNome().compareTo(tmp.getNome())==0)
                {
                      lAtu.remove(i);
                }
          }
          return 0;  }

    // Registra atuador para receber publicação
    public short RegistrarAtuador(Atuador a)
    {
          System.out.println("Atuador " + a.getNome() + "
recebido");
          lAtu.add(a);
          return 0;
    }

    // Desregistra o subscritor (abstrata)
    public void desRegistrar(Subscritor s)
    {

    }

    // Envia publicação
    public void enviarPublicacao()
    {
      // Checa condição de banda morta
      if (((valor < (last_valor + db_max))&&(valor > (last_valor
- db_min))) && dbativa)
      {   System.out.println("Valor dentro da Banda Morta..");   }
      // Se condiçõa for da banda morta, enviar publicação
      else
      {
          // Cria publicação e envia para subscritores
          Publicacao p = (Publicacao)pubValorHelper.narrow(new
pubValor_impl(this.valor));
          Subscritor s;
          for(int i=0;i < lSub.size(); i++)
```

```
            {
                  s = (Subscritor)lSub.get(i);
                  s.receberPublicacao(p);
            }


            // Cria publicação e envia para GUIs
            pubValor pv = (pubValor)pubValorHelper.narrow(new
pubValor_impl(this.valor));
            GUI g;
            for(int i=0;i < lGUI.size(); i++)
            {

                  g = (GUI)lGUI.get(i);
                  g.ReceberValor(pv);
            }

            // Envia publicação para atuadores
            Atuador a;
            for(int i=0; i < lAtu.size(); i++)
            {
                  a = (Atuador)lAtu.get(i);
                  a.RecebeValor(pv);
            }

        }
        last_valor = valor;
    }

    // Lista subscritores registrados
    public Subscritor[] listarSubscritores()
    {
          Subscritor[] subs = new Subscritor[lSub.size()];
          for(int i=0; i < lSub.size(); i++)
          {
                    subs[i] = (Subscritor)lSub.get(i);
          }
          return subs;
```

```
}

// Métodos de Subscritor
public void Subscrever(Publicador p)
{

      esquema = (ScheduleSensor)p;
      esquema.Registrar(SensorHelper.narrow(this));
}

public Publicador[] buscaPublicadores(String tipo)
{
      return m_sub.buscaPublicadores(tipo);
}

public Publicador buscaPublicador(String tipo, String nome)
{
      return null;
}

public void desSubscrever()
{
      esquema.desRegistrar(SensorHelper.narrow(this));

}

public void receberPublicacao(Publicacao p)
{
      pubAlarme pala = (pubAlarme)p;
}

// Métodos de Valor
public Schedule getEsquema()
{
      return (Schedule)this.esquema;
}
```

```
	public void setEsquema(Schedule s)
	{
		this.esquema = (ScheduleSensor)s;
	}

	public pubValor getPublicacao()
	{
		pubValor p = pubValorHelper.narrow(new
pubValor_impl(this.valor));
		return p;
	}

	// Métodos do Sensor
	// Retorna variavel
	public String getVariavel()
	{
		return this.variavel;
	}

	// Altera variavel
	public void setVariavel(String v)
	{
		this.variavel = v;
	}

	// Retorna Unidade
	public String getUnidade()
	{
		return this.unidade;
	}

	// Altera Unidade
	public void setUnidade(String unid)
	{
		this.unidade = unid;
	}

	// Retorna valor máximo
```

```
public double getValMax()
{
      return this.val_max;
}

// Altera Valor máximo
public void setValMax(double vm)
{
      this.val_max = vm;
}

// retorna valor mínimo
public double getValMin()
{
      return this.val_min;
}

// Altera Valor mínimo
public void setValMin(double vm)
{
      this.val_min = vm;
}

// Retorna banda morta superior
public double getMaxDeadBand()
{
      return this.db_max;
}

// Altera banda morta superior
public void setMaxDeadBand(double mdb)
{
      this.db_max = mdb;
}

// retorna banda morta inferior
public double getMinDeadBand()
```

```
{
      return this.db_min;
}

// Altera banda morta inferior
public void setMinDeadBand(double mdb)
{
      this.db_min = mdb;
}

// Retorna status da banda morta
public boolean IsDeadBandActive()
{
      return dbativa;
}

// Ativa / Desativa abnada morta
public void ActivateDeadBand(boolean b)
{
      this.dbativa = b;
}

// Incializa a planta
public void run()
{
      pl.start();
}

// Esta classe é responsável por gerar sinais para o sensor,
// simulando uma planta
class Planta extends Thread implements Runnable
{
      // Declara as propriedades privadas
      private Sensor_impl meu_dono;
      private int scantime;
      private boolean rodando;

      // Contrutor
```

```
        Planta(Sensor_impl m_dono, int st)
        {
              this.meu_dono = m_dono;
              this.scantime = st;
        }

        // Inicia a simulação
        public void run()
        {
              rodando = true;
              while(rodando)
              {
                    try{
                    // O valor simulado é um valor aleatório
                    // entre o valo máximo e o valor mínimo
                    meu_dono.valor = (meu_dono.getValMin() +
Math.random()*(meu_dono.getValMax()- meu_dono.getValMin()));
                    System.out.println("Valor corrente: " +
Double.toString(meu_dono.valor));
                    sleep(scantime);
                    } catch (InterruptedException e)
                    { System.out.println(e); }
              }
        }

        // Método inicia a simulação
        public void Rodar()
        {
              rodando = true;
        }

        // Pausa a simulação
        public void Parar()
        {
              rodando = false;
        }

        // Altera tempo de scan
```

```
        public void setScanTime(int sc)
        {
              this.scantime = sc;
        }

        // Retorna tempo de scan
        public int getScanTime()
        {
              return this.scantime;
        }
    }   // Fim da classe Planta
}       // Fim da classe Sensor
```

Classe Atuador

```
// Classe Atuador

package scada;

import java.util.*;
import java.lang.*;

public class atuador_impl extends _AtuadorImplBase
{
      // Declara propriedades privadas
      private Componente_impl m_com;
      private Publicador_impl m_pub;
      private Subscritor_impl m_sub;
      private Hashtable lSub;
      private Valor_impl m_val;
      private Schedule esquema;
      private double my_const;
      private double valor;

      // Construtor
      public atuador_impl(String nome)
      {
            super();
            m_com = new Componente_impl(nome);
            m_pub = new Publicador_impl(nome);
            m_sub = new Subscritor_impl(nome);
            m_val = new Valor_impl(nome);
            lSub = new Hashtable();
            valor = 0.0;
      }


      // Métodos de componente
      public String getNome()
      {
```

```
        return m_com.getNome();
    }

    public String getID()
    {
        return m_com.getID();
    }

    public void setID(String id)
    {
        m_com.setID(id);
    }

    // Métodos de Publicador
    public short Registrar(Subscritor s)
    {
        lSub.put(s.getNome(), s);
        return 0;
    }

    public void desRegistrar(Subscritor s)
    {
        lSub.remove(s.getNome());
    }

    public void enviarPublicacao()
    {
        Publicacao p = (Publicacao)pubValorHelper.narrow(new
pubValor_impl(this.valor));

        Enumeration e = lSub.keys();
        Subscritor s;
        while(e.hasMoreElements())
        {
            s = (Subscritor)lSub.get(e.nextElement());
            s.receberPublicacao(p);
        }
```

```
}

public Subscritor[] listarSubscritores()
{
      Subscritor[] subs = new Subscritor[lSub.size()];
      int i;
      Enumeration e = lSub.keys();
      i = 1;
      while (e.hasMoreElements())
      {
            subs[i] = (Subscritor)lSub.get(e.nextElement());
            i++;
      }
      return subs;
}

// Métodos de Subscritor
public void Subscrever(Publicador p)
{
      m_sub.Subscrever(p);
}

public Publicador[] buscaPublicadores(String tipo)
{
      return m_sub.buscaPublicadores(tipo);
}

public Publicador buscaPublicador(String tipo, String nome)
{
      return m_sub.buscaPublicador(tipo, nome);
}

public void desSubscrever()
{
      m_sub.desSubscrever();

}
```

```
    public void receberPublicacao(Publicacao p)
    {
        pubValor pv = (pubValor)p;
        System.out.println("SAIDA DO ATUADOR: " +
Double.toString(pv.getValor()*my_const));
    }

    //Recebe uma Publicação de valor do publicador registrado
    public void RecebeValor(pubValor p)
    {
        System.out.println("SAIDA DO ATUADOR: " +
Double.toString(p.getValor()*my_const) + " as " +
p.getHorario().getHorario());
    }

    // Métodos de Valor
    public Schedule getEsquema()
    {
        return this.esquema;
    }

    public void setEsquema(Schedule s)
    {
        this.esquema = s;
    }

    public pubValor getPublicacao()
    {
        pubValor p = pubValorHelper.narrow(new
pubValor_impl(this.valor));
        return p;
    }

    // Métodos de Atuador
    public double getConstante()
    {
        return my_const;
```

```
        }

        public void setConstante(double c)
        {
                my_const = c;
        }
} // fim da classe
```

Classe GUI

```
// Classe GUI

package scada;

import java.util.*;
import visual.*;

public class GUI_impl extends _GUIImplBase
{
        // Declara propriedades privadas
        private Publicador pub;
        private pubValor p_cor;
        private Componente_impl m_com;
        private short tipo =1;
        private int npontos;
        private visual.trend meu_dono;
        private java.util.LinkedList lvalores;

        // Construtor
        public GUI_impl(String nome)
        {
                super();
                m_com = new Componente_impl(nome);
                lvalores = new java.util.LinkedList();
                npontos = 10;
        }

        // Associa a um elemento gráfico
```

```
public void setDono(visual.trend m)
{
      this.meu_dono = m;
}

// Retorna nome
public String getNome()
{
      return m_com.getNome();
}

// Retorna ID
public String getID()
{
      return m_com.getID();
}

// Alatera ID
public void setID(String s)
{
      m_com.setID(s);
}

// Subscreve em um publicador
public void Subscrever(Publicador p)
{
      this.pub = p;
      p.Registrar(this);
}

// Retorna publicadores (abstrato)
public Publicador[] buscaPublicadores(String tipo)
{
      return null;
}

// retorna publicador específico (abstrato)
```

```
	public Publicador buscaPublicador(String tipo, String nome)
	{
		return null;
	}

	// Desubscrever (abstrato)
	public void desSubscrever()
	{
	}


	// Recebe publicação
	public void receberPublicacao(Publicacao p)
	{
		if (lvalores.size() >= npontos)
		{	lvalores.removeFirst();	}
		pubValor pv = (pubValor)p;
		System.out.println("Recebendo Publicacao " +
Double.toString(pv.getValor()) + " as " +
pv.getHorario().getHorario());
		lvalores.add(pv);
		this.p_cor = pv;
	}

	// Recebe publicação de  valor do sensor
	public void ReceberValor(pubValor pv)
	{
		System.out.println("Recebendo Publicacao " );
		if (lvalores.size() >= npontos)
		{	lvalores.removeFirst();	}
		System.out.println("Recebendo Publicacao " +
Double.toString(pv.getValor()));
		lvalores.add(pv);
		this.p_cor = pv;
	}

	// Método para atualizar tela (Abstrato)
	public void Refresh()
```

```
    {

    }

    // Método para alterar tipo de GUI (abstrato)
    public void setTipo(short s)
    {

    }

    // método que retorna o tipo de GUI (abstrato)
    public short getTipo()
    {
        return tipo;
    }

    // retorna o número de pontos
    public int getNPontos()
    {
        return npontos;
    }

    // Altera o número de pontos
    public void setNPontos(int np)
    {
        this.npontos = np;
    }

    // Retorna os pontos na forma de um vetor
    public pubValor[] getPontos()
    {
        System.out.println("Entrou em getPontos()");
        pubValor[] p = new pubValor[lvalores.size()];
        System.out.println("Ha " +
Integer.toString(lvalores.size()) + " pontos");
        for (int i=0; i < lvalores.size();i++)
        {
            p[i] = (pubValor)lvalores.get(i);
```

```
            }
            System.out.println("Passou do loop");
            return p;

        }

}
```

Classe Coordenador

```
// Classe coordenador

package scada;

import java.util.*;

public class coordenador_impl extends _CoordenadorImplBase
{
      // Declara listas que vão conter os componentes registrados
      // no sistema
      private LinkedList lSensor;
      private LinkedList lAtuador;
      private LinkedList lGUI;
      private LinkedList lScheduleSensor;

      // Construtor
      // Instancia as listas
      public coordenador_impl()
      {
            super();
            lSensor = new LinkedList();
            lAtuador = new LinkedList();
            lGUI = new LinkedList();
            lScheduleSensor = new LinkedList();
      }

      // Registra Sensor
      public short RegistrarSensor(Sensor s)
      {
            System.out.println("Entrou em registrar sensor...");
            System.out.println("Recebeu o Sensor: " + s.getNome());
            lSensor.add (s);
            return (short)1;
      }
```

```
	// Registra Atuador
	public short RegistrarAtuador(Atuador a)
	{
		System.out.println("Entrou em registrar atuador...");
		lAtuador.add (a);
		System.out.println("Recebeu o Atuador: " + a.getNome());
		return (short)1;
	}

	// Registra GUI
	public short RegistrarGUI(GUI g)
	{
		System.out.println("Entrou em registrar GUI");
		lGUI.add (g);
		System.out.println("Registrou o GUI: " + g.getNome());
		return (short)1;
	}

	// Registra Scheudlle Sensor
	public short RegistrarScheduleSensor(ScheduleSensor s)
	{
		System.out.println("Entrou em Registrar Schedule
Sensor...");
		System.out.println("Pos os meninos para rodar...");
		System.out.println("Recebeu o Sched: " + s.getNome());
		lScheduleSensor.add(s);
		return (short)1;
	}


	// Métodos para desRegistrar objetos

	public short desRegistrarSensor(Sensor s)
	{
		for (int i =0; i < lSensor.size(); i++)
		{
			if (lSensor.get(i).equals(s))
			{	lSensor.remove (i);
```

```
                return (short)1;
            }
        }
        return (short)0;
    }

    public short desRegistrarAtuador(Atuador a)
    {
        for (int i =0; i < lAtuador.size(); i++)
        {
            if (lAtuador.get(i).equals(a))
            {   lAtuador.remove (i);
                return (short)1;
            }
        }
        return (short)0;

    }

    public short desRegistrarGUI(GUI g)
    {
        for (int i =0; i < lGUI.size(); i++)
        {
            if (lGUI.get(i).equals(g))
            {   lSensor.remove (i);
                return (short)1;
            }
        }
        return (short)0;
    }

    public short desRegistrarScheduleSensor(ScheduleSensor s)
    {
        for (int i =0; i < lSensor.size(); i++)
        {
            if (lSensor.get(i).equals(s))
            {   lScheduleSensor.remove (i);
```

```
                return (short)1;
            }
        }
        return (short)0;
    }

    // Métodos para se listar objetos
    public Sensor[] getSensores()
    {
        Sensor[] s = new Sensor[lSensor.size()];
        for (int i=0; i < lSensor.size(); i++)
        {
            s[i] = (Sensor)lSensor.get(i);
        }
        return s;
    }

    public Atuador[] getAtuadores()
    {
        System.out.println("No Atuadores: " + lAtuador.size());
        Atuador[] a = new Atuador[lAtuador.size()];
        for(int i=0; i < lAtuador.size(); i++)
        {
            a[i] = (Atuador)lAtuador.get(i);
        }
        return a;
    }

    public GUI[] getGUIs()
    {
        GUI[] g = new GUI[lGUI.size()];
        for(int i=0; i < lGUI.size(); i++)
        {
            g[i] = (GUI)lGUI.get(i);
        }
        return g;
    }
```

```
	public ScheduleSensor[] getScheduleSensores()
	{
		ScheduleSensor[] s = new
ScheduleSensor[lScheduleSensor.size()];
		for (int i=0; i < lScheduleSensor.size();i++)
		{
			s[i] = (ScheduleSensor)lScheduleSensor.get(i);
			System.out.println("Schedule: " + s[i].getNome());
		}
		return s;
	}


	// Métodos para se buscar objetos específicos
	public Sensor getSensor(String nome)
	{
		Sensor tmp;
		for (int i =0; i < lSensor.size(); i++)
		{
			tmp = (Sensor)lSensor.get(i);
			if (nome.compareToIgnoreCase(tmp.getNome())==0)
			{
				return tmp;
			}
		}
		return null;
	}

	public Atuador getAtuador(String nome)
	{
		Atuador tmp;
		for (int i =0; i < lAtuador.size(); i++)
		{
			tmp = (Atuador)lAtuador.get(i);
			if (nome.compareToIgnoreCase(tmp.getNome())==0)
			{
				return tmp;
```

```
            }
        }
        return null;
    }

    public GUI getGUI(String nome)
    {
        GUI tmp;
        for (int i =0; i <= lGUI.size(); i++)
        {
            tmp = (GUI)lGUI.get(i);
            if (nome.compareToIgnoreCase(tmp.getNome())==0)
            {
                return tmp;
            }
        }
        return null;
    }

    public ScheduleSensor getScheduleSensor(String nome)
    {
        ScheduleSensor tmp;
        for (int i =0; i < lScheduleSensor.size(); i++)
        {
            tmp = (ScheduleSensor)lScheduleSensor.get(i);
            if (nome.compareToIgnoreCase(tmp.getNome())==0)
            {
                return tmp;
            }
        }
        return null;
    }

    public int getQuantidade(String tipo)
    {
        System.out.println("Recebi o param: " + tipo);
        if (tipo.compareTo("sensor")==0)
```

```
            {     return lSensor.size(); }
            else if (tipo.compareTo("atuador")==0)
            {     return lAtuador.size(); }
            else if (tipo.compareTo("GUI") == 0)
            {     return lGUI.size(); }
            else if (tipo.compareTo("Schedulesensor")==0)
            {     return lScheduleSensor.size(); }
            else        {     return 0; }
      }
}
```

8.3 Servidores

Classe CoordenadorServer

Esta classe instancia um objeto Coordenador e aguarda que as outra classe se regiatrem nesta.

```
// Objeto para coordenar os demais objetos

import scada.*;
import java.lang.*;
import java.util.*;
import java.io.*;

public class CoordenadorServer extends Thread implements Runnable
{
      // Função Main
      public static void main(String args[])
      {

            java.util.Properties props = System.getProperties();
            // Indica que será usado o ORBacus
            props.put("org.omg.CORBA.ORBClass",
"com.ooc.CORBA.ORB");
            props.put("org.omg.CORBA.ORBSingletonClass",
"com.ooc.CORBA.ORBSingleton");
            System.setProperties(props);
            System.out.println("Setou as propriedades...");
            // Inicia o ORB
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,
props);
            System.out.println("Inicializou o ORB...");
            // Inicia o BOA
            org.omg.CORBA.BOA boa =
((com.ooc.CORBA.ORB)orb).BOA_init(args, props);
            System.out.println("Inicializou o BOA...");
```

```
            // Seta um modelo de concorrência (Concurrency Model)
            // que suporte Callbacks
            ((com.ooc.CORBA.ORB)orb).conc_model(com.ooc.CORBA.ORB.Co
        ncModel.ConcModelThreaded);

            ((com.ooc.CORBA.BOA)boa).conc_model(com.ooc.CORBA.BOA.
        ConcModel.ConcModelThreadPerRequest);

            // instancia um coordenador
            scada.coordenador_impl c = new scada.coordenador_impl();
            System.out.println("Instanciou o Coordenador ...");
            ((com.ooc.CORBA.ORB)orb).connect(c, "coordenador");

            // Código específico do ORBacus para esperar requests e
            // suportar callbacks
            ((com.ooc.CORBA.BOA)boa).init_servers();
            System.out.println("Entrou no loop...");

        }

        // Classe implementa Runnable
        public void run()
        {

        }
}
```

Classe ScheduleServer

Esta classe instancia um SchdulleSensor e re registra no coordenador.

```
// Classe Schedule Servidor

//package scada;

import java.util.*;
import org.omg.CORBA.*;
import scada.*;


public class ScheduleServer extends Thread implements Runnable
{

      public static void main(String args[])
      {

            java.util.Properties props = System.getProperties();
            // indica que vai usar ORBacus
            props.put("org.omg.CORBA.ORBClass",
"com.ooc.CORBA.ORB");
            props.put("org.omg.CORBA.ORBSingletonClass",
"com.ooc.CORBA.ORBSingleton");
            props.put("ooc.orb.conc_model", "threaded");
            props.put("ooc.boa.conc_model", "thread_per_request");
            System.setProperties(props);
            System.out.println("Setou as propriedades...");

            // Inicializa o ORB
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,
props);
            System.out.println("Inicializou o ORB...");
            // Inicialza o BOA
            org.omg.CORBA.BOA boa =
((com.ooc.CORBA.ORB)orb).BOA_init(args, props);
            System.out.println("Inicializou o BOA...");
```

```
            // Coloca o ORB em loop esperando request
            // Este método é específico do ORBacus para suportar
            // CallBakcs
            ((com.ooc.CORBA.BOA)boa).init_servers();
            System.out.println("Entrou no loop...");

            // Seta modelo de concorrência para suportar callbacks
            ((com.ooc.CORBA.ORB)orb).conc_model(com.ooc.CORBA.ORB.
            ConcModel.ConcModelThreaded);

            ((com.ooc.CORBA.BOA)boa).conc_model(com.ooc.CORBA.BOA.
            ConcModel.ConcModelThreadPerRequest);

            // Instancia um Schedule Sensor
            scada.ScheduleSensor_impl s = new
scada.ScheduleSensor_impl(args[0], 5000, (short)1);
            orb.connect(s);
            ScheduleSensor ss = ScheduleSensorHelper.narrow(s);
            System.out.println("Instanciou o Schedule ..." +
ss.getNome());

            // Localiza o coordenador
            org.omg.CORBA.Object obj =
((com.ooc.CORBA.ORB)orb).get_inet_object("localhost",
(short)5000,"coordenador");
            try
            {
                  scada.Coordenador c =
      scada.CoordenadorHelper.narrow(obj);
                  // registra Scheudlle no Coordenador
                  c.RegistrarScheduleSensor(ss);
            }
           catch (Exception e)
            {
                  System.out.println("Erro: " + e.getMessage());
            }

      }
```

```
        // Implementa Runnable
        public void run()
        {
              System.out.println("Startei...");
        }
}
```

Classe SensorServer

Esta classe instancia um sensor e o registra no coordenador.

```
// Classe Schedule Servidor

import java.util.*;
import java.lang.*;
import org.omg.CORBA.*;
import scada.*;
import java.io.*;
import org.omg.CosNaming.*;

public class SensorServer extends Thread implements Runnable
{
      // Função main
      public static void main(String args[])
      {

            java.util.Properties props = System.getProperties();
            // Indica que usará ORBacus
            props.put("org.omg.CORBA.ORBClass",
"com.ooc.CORBA.ORB");
            props.put("org.omg.CORBA.ORBSingletonClass",
"com.ooc.CORBA.ORBSingleton");
            props.put("ooc.orb.conc_model", "threaded");
            props.put("ooc.boa.conc_model", "thread_per_request");
            System.setProperties(props);
            System.out.println("Setou as propriedades...");

            // Inicializa o ORB
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,
props);
            System.out.println("Inicializou o ORB...");
            // Inicializa o BOA
            org.omg.CORBA.BOA boa =
((com.ooc.CORBA.ORB)orb).BOA_init(args, props);
            System.out.println("Inicializou o BOA...");
```

```
            // Seta modelo de concorrência para suportar Callbacks
            ((com.ooc.CORBA.ORB)orb).conc_model(com.ooc.CORBA.ORB.
            ConcModel.ConcModelThreaded);
            ((com.ooc.CORBA.BOA)boa).conc_model(com.ooc.CORBA.BOA.
            ConcModel.ConcModelThreadPerRequest);

            // Põe o ORB em loop esperando requests
            // Esta funcão é específica do ORBacus e permite
            // callbacks
            ((com.ooc.CORBA.BOA)boa).init_servers();

            // Instancia Sensor
            scada.Sensor_impl p = new scada.Sensor_impl(args[0]);

            // Conecta ao ORB
            orb.connect(p);
            System.out.println("Criou um Sensor " + args[0]);
            Sensor sens = SensorHelper.narrow(p);
            // Seta valores padrão para os sensores
            sens.setVariavel("temperatura");
            sens.setUnidade("oC");
            sens.setValMax(80);
            sens.setValMin(60);
            sens.setMaxDeadBand(1.5);
            sens.setMinDeadBand(1.5);

            // Procura coordenador
            org.omg.CORBA.Object obj =
((com.ooc.CORBA.ORB)orb).get_inet_object("localhost",
(short)5000,"coordenador");
            System.out.println("Encontrou coordenador...");

            try
            {
                scada.Coordenador c =
scada.CoordenadorHelper.narrow(obj);
                // Registra no coordenador
                c.RegistrarSensor(sens);
```

```
                System.out.println("Entrou no loop...");
            }catch (Exception e)
            {
                System.out.println(e.getMessage());
            } // fim do try-catch


        }     // fim do main

        // Implementa Runnable
        public void run()
        {

        }

}       // Fim da classe
```

Classe AtuadorServer

Esta classe instancia um atuador e o registra no coordenador.

```
// Classe Schedule Servidor

import java.util.*;
import org.omg.CORBA.*;
import scada.*;


public class AtuadorServer extends Thread implements Runnable
{
      // Método main
      public static void main(String args[])
      {

            java.util.Properties props = System.getProperties();
            // Indica que o ORBacus será usado
            props.put("org.omg.CORBA.ORBClass",
"com.ooc.CORBA.ORB");
            props.put("org.omg.CORBA.ORBSingletonClass",
"com.ooc.CORBA.ORBSingleton");
            System.setProperties(props);
            System.out.println("Setou as propriedades...");

            // Incializa o ORB
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,
props);
            System.out.println("Inicializou o ORB...");
            // Inicializa o BOA
            org.omg.CORBA.BOA boa =
((com.ooc.CORBA.ORB)orb).BOA_init(args, props);
            System.out.println("Inicializou o BOA...");


            // Seta modelo de concorrência que suporte Callbacks
            ((com.ooc.CORBA.ORB)orb).conc_model(com.ooc.CORBA.ORB.
      ConcModel.ConcModelThreaded);
```

```
            ((com.ooc.CORBA.BOA)boa).conc_model(com.ooc.CORBA.BOA.
        ConcModel.ConcModelThreadPerRequest);

            // Põe o ORB em loop esperando request
            // Esta chamda é específica do ORBacus e permite
            // callbacks
            ((com.ooc.CORBA.BOA)boa).init_servers();
            System.out.println("Entrou no loop...");

            // Instancia um atuador
            scada.atuador_impl s = new scada.atuador_impl(args[0]);
            s.setConstante(3.0);
            System.out.println("Instanciou o Atuador ...");

            // Conecta o atuador ao ORB
            ((com.ooc.CORBA.ORB)orb).connect(s);

            // Procura o Coordenador
            org.omg.CORBA.Object obj =
((com.ooc.CORBA.ORB)orb).get_inet_object("localhost",
(short)5000,"coordenador");
            try
            {
                  Coordenador c = CoordenadorHelper.narrow(obj);
                  // registra o atuador no coordenador
                  c.RegistrarAtuador((Atuador)s);
            }
            catch (Exception e)
            {
                  System.out.println("Erro: " + e.getMessage());
            }     // fim do try-catch
      }           // fim do mátodo main
      // Implementa Runnable
      public void run()
      {

      }
}               // Fim da classe
```

Classe GUIServer

Esta classe cria um GUI, se registra no coordenador e espera uma applet se conectar para poder plotar seus valores.

```
// Classe GUI Servidor

import java.util.*;
import java.lang.*;
import org.omg.CORBA.*;
import scada.*;
import java.io.*;

public class GUIServer extends Thread implements Runnable
{
      // Função main
      public static void main(String args[])
      {

            java.util.Properties props = System.getProperties();
            // indica que o  ORBacus será usado
            props.put("org.omg.CORBA.ORBClass",
"com.ooc.CORBA.ORB");
            props.put("org.omg.CORBA.ORBSingletonClass",
"com.ooc.CORBA.ORBSingleton");
            props.put("ooc.orb.conc_model", "threaded");
            props.put("ooc.boa.conc_model", "thread_per_request");
            System.setProperties(props);
            System.out.println("Setou as propriedades...");

            // incializa o ORB
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,
props);
            System.out.println("Inicializou o ORB...");
            // incializa o BOA
            org.omg.CORBA.BOA boa =
((com.ooc.CORBA.ORB)orb).BOA_init(args, props);
            System.out.println("Inicializou o BOA...");
```

```
            // Seta um modelo de concorrência que suporta callbacks
            ((com.ooc.CORBA.ORB)orb).conc_model(com.ooc.CORBA.ORB.
      ConcModel.ConcModelThreaded);
            ((com.ooc.CORBA.BOA)boa).conc_model(com.ooc.CORBA.BOA.
      ConcModel.ConcModelThreadPerRequest);

            // Põe o ORB em loop esperando request
            // Esta chamda é específica do ORBacus e permite
            // callbacks
             ((com.ooc.CORBA.BOA)boa).init_servers();

            // instancia um GUI
            scada.GUI_impl g= new scada.GUI_impl("guiserver");
            // Conecta ao ORB
            orb.connect(g);
            System.out.println("Criou um Sensor guiserver");
            GUI igui = GUIHelper.narrow(g);

            // Procura o coordenador
            org.omg.CORBA.Object obj =
((com.ooc.CORBA.ORB)orb).get_inet_object("localhost",
(short)5000,"coordenador");
            System.out.println("Encontrou coordenador...");

            try
            {
                  scada.Coordenador c =
scada.CoordenadorHelper.narrow(obj);
                  // Registra no coordenador
                  c.RegistrarGUI(igui);
                  System.out.println("Entrou no loop...");

            }catch (Exception e)
            {
                  System.out.println(e.getMessage());
            }     // fim do try-catch
```

```
        }           // fim do main


        // Implementa Runnable
        public void run()
        {

        }

}
```

8.4 Applets

Há duas applets, cada uma contendo mais de um elemento. A primeira, o coordenador é uma interface gráfica para o objeto coordenador, além de permitir a congiração dos componentes. Já a segunda representa interface gráfica do elemento GUI, e irá se conectar a um GUIServer, descrito anteriormente.

CoordenadorApplet

É constituído de três classes, que estão listadas abaixo.

```
// Classe Coordenador applet

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

public class coordenadorApplet extends Applet
{
      // Lista propriedades privadas
      ListaDeTipo cabeca;
      SensorPanel psens;
      ScheduleSensorPanel pss;
      AtuadorPanel patu;
      Panel p_cor;
      Coordenador c;

      // Instancia os elementos
      public void init()
      {
      cabeca  = new ListaDeTipo(this);
      psens = new SensorPanel(this);
```

```
        pss = new ScheduleSensorPanel(this);
        patu = new AtuadorPanel(this);
        setFont(new Font ("Tahoma", Font.BOLD, 14));

        add("North",cabeca);
        add("Center", psens);
        add("Center",pss);
        add("Center",patu);
        p_cor = (Panel)psens;
        pss.setVisible(false);
        patu.setVisible(false);

        try{
              java.util.Properties prop = new java.util.Properties();
              // inicializa o ORB
              org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(this,
prop);
              System.out.println("ORB iniciado...");

              // Procura o coordenador pelo seu ID
              org.omg.CORBA.Object coord =
orb.string_to_object("IOR:000000000000001a49444c3a73636164612f436f
6f7264656e61646f723a312e3000000000000001000000000000002400010000000
00000a6c6f63616c686f737400138800000000c636f6f7264656e61646f7200");
              if (coord == null)
              {
                    System.out.println("Falhou...");
                    return;
              }
              else
              {     System.out.println("Coordenador encontrado...");
              }
              // Se conecta ao coordenador
              c = CoordenadorHelper.narrow(coord);
              }catch (Exception e)
              {
                    System.out.println("Erro: " + e.getMessage());
              }     // fim do try
```

```
            this.resize(500,500);
        }

        public void setMensagem(String palavra)
        {
            System.out.println(palavra);
        }

        // Muda o painel de controle
        public void mudaPanel(String qual)
        {
            p_cor.setVisible(false);
            System.out.println(qual);
            if (qual.compareTo("sensor") == 0)
            {
                psens.setVisible(true);
                p_cor = (Panel)psens;
            }else if (qual.compareTo("timer") == 0)
            {
                pss.setVisible(true);
                p_cor = (Panel)pss;
            }else if (qual.compareTo("atuador")==0)
            {
                patu.setVisible(true);
                p_cor = (Panel)patu;
            }
            resize(500,500);
        }

        // Envia lista a outros paineis
        public void EnviaLista(String param)
        {
            System.out.println("Parametro recebido: " + param);

            if(param.compareTo("sensor")==0)
            {   System.out.println("Enviando a lista de
sesnores...");
                psens.RecebeLista (c.getSensores());
```

```
            }else if (param.compareTo("atuador")==0)
            {     patu.menswrite("Consegui chamar...");
                  patu.RecebeLista (c.getAtuadores());
            }else if (param.compareTo("Schedulesensor")==0)
            {     pss.RecebeLista(c.getScheduleSensores());
            } else
            {
            }


      }

      // Envia determinada lista de parâmetros dependendo de quem
      // pediu
      public void EnviaLista2(String param)
      {
            System.out.println("Parametro recebido: " + param);

            if(param.compareTo("sensor")==0)
            {     psens.RecebeLista2 (c.getScheduleSensores());
            }else if (param.compareTo("atuador")==0)
            {     patu.menswrite("Consegui chamar...");
                  patu.RecebeLista2 (c.getSensores());
            }else if (param.compareTo("Schedulesensor")==0)
            {     pss.RecebeLista(c.getScheduleSensores());
            } else
            {
            }


      }

}     // fim da applet
```

```
// Lista de Tipo

import java.awt.*;
import java.awt.event.*;
import java.lang.*;

public class ListaDeTipo extends Panel implements ItemListener
{
      coordenadorApplet meu_dono;
      Checkbox oSensor, oAtuador, oTimer, oGUI;
      CheckboxGroup grupo;
      String selecionado;

      // Cria elementos de CheckBox
      public ListaDeTipo(coordenadorApplet c)
      {
            grupo  = new CheckboxGroup();
            this.selecionado = "sensor";
            meu_dono = c;
            setLayout(new GridLayout(1,3));
            oSensor = new Checkbox("Sensor", grupo, true);
            oSensor.setFont(new Font ("Tahoma", Font.BOLD, 12));
            oSensor.addItemListener(this);
            add(oSensor);
            oAtuador = new Checkbox("Atuador", grupo, false);
            oAtuador.setFont(new Font ("Tahoma", Font.BOLD, 12));
            oAtuador.addItemListener(this);
            add(oAtuador);
            oTimer = new Checkbox("Timer", grupo, false);
            oTimer.setFont(new Font ("Tahoma", Font.BOLD, 12));
            oTimer.addItemListener(this);
            add(oTimer);
            add(new Label("           "));
            add(new Label("           "));
      }
```

```
    // Diz qual o item selecionado no momento
    public String getSelecionado()
    {
          return this.selecionado;
    }

    // Responde aos cliques
    public void itemStateChanged(ItemEvent e)
    {
          if (e.getItemSelectable() == oSensor)
          {     selecionado = "sensor";
                meu_dono.setMensagem("Clicou em Sensor");   }
          else if (e.getItemSelectable() == oAtuador)
          {     selecionado = "atuador";
                meu_dono.setMensagem("Clicou em Atuador");  }
          else if (e.getItemSelectable() == oTimer)
          {     selecionado = "timer";
                meu_dono.setMensagem("Clicou em Timer");    }
          else
          {     meu_dono.setMensagem("Clicou em Nenhum...");      }
          meu_dono.mudaPanel(selecionado);
    }
}         // fim da applet
```

```
// Classe Sensor Panel

import java.lang.*;
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import scada.*;

public class SensorPanel extends Panel implements ItemListener,
ActionListener
{
      coordenadorApplet meu_dono;
      LinkedList sensores, timers;

      Label mens;
      Checkbox bmAtiva;
      TextField bmmin, bmmax, vmax, vmin, variavel, unid;
      Label lbmmin, lbmmax, lvmax, lvmin, lvar, lunid;
      Panel ptemp;
      Button atualiza, refresh, registra;
      java.awt.List lst, listTimer;
      // Variáveis de dados
      Double m_bmmin, m_bmmax, m_vmax, m_vmin;
      String m_var, m_unid;
      boolean bbmativa;

      // Construtor
      // Cria os elementos gráficos
      public SensorPanel(coordenadorApplet m)
      {
            this.meu_dono = m;
            sensores = new LinkedList();
            timers = new LinkedList();

            setLayout(new GridLayout(12,2));
            add(mens = new Label("SENSOR"));
            add(new Panel());
```

```
add(lst = new java.awt.List(1,true));
lst.setMultipleMode(false);
add(new Panel());
lst.addItemListener(this);
add(lvar = new Label("Variável: "));
add(ptemp = new Panel());
ptemp.setLayout(new GridLayout(1,3));
ptemp.add(variavel = new TextField());
ptemp.add(new Panel());
add(lunid = new Label("Unidade: "));
add(ptemp = new Panel());
ptemp.setLayout(new GridLayout(1,3));
ptemp.add(unid = new TextField());
ptemp.add(new Panel());
add(lvmin = new Label("Valor Minimo: "));
add(ptemp = new Panel());
ptemp.setLayout(new GridLayout(1,3));
ptemp.add(vmin = new TextField());
ptemp.add(new Panel());
add(lvmax = new Label("Valor Máximo: "));
add(ptemp = new Panel());
ptemp.setLayout(new GridLayout(1,3));
ptemp.add(vmax = new TextField());
ptemp.add(new Panel());
add(bmAtiva = new Checkbox("Banda Morta Ativa"));
bmAtiva.addItemListener(this);
add(new Panel());
add(lbmmin = new Label("Banda Morta Minima: "));
add(ptemp = new Panel());
ptemp.setLayout(new GridLayout(1,3));
ptemp.add(bmmin = new TextField());
ptemp.add(new Panel());
add(lbmmax = new Label("Banda Morta Maxima: "));
add(ptemp = new Panel());
ptemp.setLayout(new GridLayout(1,3));
ptemp.add(bmmax = new TextField());
ptemp.add(new Panel());
```

```
            add(new Label("Timers"));
            add(listTimer = new java.awt.List(1,true));
            add(ptemp = new Panel());
            ptemp.add(refresh = new Button("Refresh"));
            ptemp.add(atualiza = new Button("Atualiza"));
            add(ptemp = new Panel());
            ptemp.add(registra = new Button("Registra"));
            ptemp.add(new Panel());
            atualiza.addActionListener(this);
            refresh.addActionListener(this);
            registra.addActionListener(this);
      }

      public void menswrite(String m)
      {

      }

      // Recebe lista de sensores
      public void RecebeLista(Sensor[] s)
      {
            System.out.println("Entrou na funcao..." +
Integer.toString(s.length));
            for (int j=sensores.size()-1; j >= 0; j--)
            {
                  sensores.remove(j);
            }

            for (int i=0; i < s.length; i++)
            {
                  System.out.println("Nome: " + s[i].getNome());
                  sensores.add(s[i]);
            }
      }

      // Recebe a lista de ScheduleSensor
      public void RecebeLista2(ScheduleSensor[] s)
      {
```

```
            System.out.println("Entrou na funcao..." +
Integer.toString(s.length));
            for (int j=timers.size()-1; j >= 0; j--)
            {
                  timers.remove(j);
            }

            for (int i=0; i < s.length; i++)
            {
                  System.out.println("Nome: " + s[i].getNome());
                  timers.add(s[i]);
            }
      }

      // Replota
      public void Refresh()
      {
            Sensor m_item;
            lst.removeAll();
            for(int i = 0; i < sensores.size(); i++)
            {
                  m_item = (Sensor)sensores.get(i);
                  lst.add(m_item.getNome(), i);
            }

            ScheduleSensor m_item2;
            listTimer.removeAll();
            for(int i=0; i < timers.size(); i++)
            {
                  m_item2 = (ScheduleSensor)timers.get(i);
                  listTimer.add(m_item2.getNome(), i);
            }

      }

      // Responde aos cliques
      public void actionPerformed(ActionEvent ev) {
            if (ev.getActionCommand().compareTo("Atualiza") == 0)
```

```
            {
                  atualizaValores();
                  System.out.println("Clicou em atualiza!");
            }else if (ev.getActionCommand().compareTo("Refresh") ==
0)
            {
                  System.out.println("Clidou em refresh");
                  meu_dono.EnviaLista("sensor");
                  meu_dono.EnviaLista2("sensor");
                  Refresh();
            }else if
(ev.getActionCommand().compareTo("Registra")==0)
            {
                  RegistrarSchedule();
            }

      }

      // Responde ao clique na checkbox
      public void itemStateChanged(ItemEvent e){
            if (e.getItemSelectable() == bmAtiva)
            {
                  if (e.getStateChange() == ItemEvent.SELECTED)
                  {     bbmativa = true;
                        System.out.println("Selecionado");
                  }
                  else
                  {     bbmativa = false;
                        System.out.println("Deselecionado");
                  }
                  Sensor m_item =
(Sensor)sensores.get(lst.getSelectedIndex());
                  m_item.ActivateDeadBand(bbmativa);

            }else
            {
                  System.out.println("Afetado " +
lst.getSelectedItem());
```

```
                MostraDados();
            }
        }

        // registra Sensor em no ScheduleSensor
        public void RegistrarSchedule()
        {
            System.out.println("Entrou em registrar Schedule...");
            if (lst.getSelectedIndex() < 0 ||
listTimer.getSelectedIndex() < 0)
            {
                System.out.println("Nenhum item selecionado...");
                return;
            }
            Sensor m_item;
            m_item = (Sensor)sensores.get(lst.getSelectedIndex());

            System.out.println("Sensor " + m_item.getNome() + "
selecionado");
            if (m_item.getEsquema() != null)
            {
                System.out.println("Desubscrevendo...");
                m_item.desSubscrever();
            }
            ScheduleSensor ss =
(ScheduleSensor)timers.get(listTimer.getSelectedIndex());
            System.out.println("Schedule " + ss.getNome() + "
selecionado");
            ss.Registrar(m_item);

        }

        // Mostra os dados referentes ao Sensor na tela
        public void MostraDados()
        {
            Sensor m_item;
            m_item = (Sensor)sensores.get(lst.getSelectedIndex());

            System.out.println("Item Selecionado: " +
```

```
m_item.getNome());
            variavel.setText(m_item.getVariavel());
            unid.setText(m_item.getUnidade());
            bmmin.setText(Double.toString(m_item.getMinDeadBand()));
            bmmax.setText(Double.toString(m_item.getMaxDeadBand()));
            vmin.setText(Double.toString(m_item.getValMin()));
            vmax.setText(Double.toString(m_item.getValMax()));
            bmAtiva.setState(m_item.IsDeadBandActive());
      }

      // Atualiza valores referentes aos sensores
      public void atualizaValores()
      {
            Sensor m_item;
            m_item = (Sensor)sensores.get(lst.getSelectedIndex());
            System.out.println("Item Selecionado: " +
m_item.getNome());
            if (variavel.getText().compareTo("")!=0)
            {
                  m_item.setVariavel(variavel.getText());
            }
            if (unid.getText().compareTo("") != 0 )
            {
                  m_item.setUnidade(unid.getText());
            }
            Double val = Double.valueOf(bmmin.getText());
            if (!val.isNaN())
            {
                  m_item.setMinDeadBand(val.doubleValue());
            }
            val = Double.valueOf(bmmax.getText());
            if (!val.isNaN())
            {
                  m_item.setMaxDeadBand(val.doubleValue());
            }
            val = Double.valueOf(vmin.getText());
            if (!val.isNaN())
            {
```

```
                m_item.setValMin(val.doubleValue());
            }
            val = Double.valueOf(vmax.getText());
            if (!val.isNaN())
            {
                m_item.setValMax(val.doubleValue());
            }
        }

}
```

```
// Classe ScheduleSensor Panel

import java.lang.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import java.util.*;
import scada.*;

public class ScheduleSensorPanel extends Panel implements
ItemListener, ActionListener
{
      LinkedList lSched;
      coordenadorApplet meu_dono;

      Label mens;
      TextField scan;
      Label lScan;
      Panel ptemp;
      Button atualiza, refresh, rodar, parar;
      java.awt.List lst;
      // Variáveis de dados
      Double m_scan;


      public ScheduleSensorPanel(coordenadorApplet m)
      {
            this.meu_dono = m;
            lSched = new LinkedList();
            setLayout(new GridLayout(10,2));
            add(mens = new Label("TIMER"));
            add(new Panel());
            add(lst = new java.awt.List(1,true));
            lst.setMultipleMode(false);
            add(new Panel());
            lst.addItemListener(this);
            add(ptemp = new Panel());
            ptemp.setLayout(new GridLayout(1,2));
```

```
        ptemp.add(lScan = new Label("Scan Time"));
        ptemp.add(scan = new TextField());
        add(new Panel());
        add(ptemp = new Panel());
        ptemp.add(refresh = new Button("Refresh"));
        ptemp.add(atualiza = new Button("Atualiza"));
        atualiza.addActionListener(this);
        refresh.addActionListener(this);
        add(ptemp = new Panel());
        ptemp.add(rodar = new Button("Rodar"));
        ptemp.add(parar = new Button("Parar"));
        rodar.addActionListener(this);
        parar.addActionListener(this);


}

// Recebe lista de Schedulles disponíveis
public void RecebeLista(ScheduleSensor[] s)
{
        for(int j=0;j < lSched.size(); j++)
        {
                lSched.remove(j);
        }


        for(int i= 0; i < s.length; i++)
        {
                lSched.add(s[i]);
        }

}

// Busca Schudeles
public void Refresh()
{
        ScheduleSensor m_item;
```

```
            lst.removeAll();
            for(int i = 0; i < lSched.size(); i++)
            {
                  m_item = (ScheduleSensor)lSched.get(i);
                  lst.add(m_item.getNome());
            }

      }

      public void menswrite(String m)
      {

      }

      // Responde a cliques do mouse
      public void actionPerformed(ActionEvent ev) {
            if (ev.getActionCommand().compareTo("Atualiza") == 0)
                  {     System.out.println("Clicou em atualiza!");
                        atualizaValores();
                  }
            else if (ev.getActionCommand().compareTo("Refresh") ==
0)
                  {     System.out.println("Clicou em Refresh!");
                        meu_dono.EnviaLista("Schedulesensor");
                        Refresh();
                  }
            else if (ev.getActionCommand().compareTo("Rodar") == 0)
                  {     System.out.println("Clicou em Rodar!");
      }
            else if (ev.getActionCommand().compareTo("Parar") == 0)
                  {     System.out.println("Clicou em parar!");

                        ScheduleSensor m_item = (ScheduleSensor)
lSched.get(lst.getSelectedIndex());
                        m_item.Rodar();
                  }
            else
                  {     System.out.println("Clicou em porra
nenhuma!");
```

```
                        ScheduleSensor m_item = (ScheduleSensor)
lSched.get(lst.getSelectedIndex());
                        m_item.Parar();
                    }
        }

        public void itemStateChanged(ItemEvent e){
                    System.out.println("Afetado " +
lst.getSelectedItem());
                    MostraDados();
        }

        // Mostra dados referentes ao SchedulleSensor selecionado
        public void MostraDados()
        {
                ScheduleSensor m_item = (ScheduleSensor)
lSched.get(lst.getSelectedIndex());
                System.out.println("Item Selecionado: " +
m_item.getNome());
                scan.setText(Integer.toString(m_item.getScanTime()));
        }

        // Atualiza valores referentes ao SchedulleSensor
        public void atualizaValores()
        {
                ScheduleSensor m_item = (ScheduleSensor)
lSched.get(lst.getSelectedIndex());
                Integer val = Integer.valueOf(scan.getText());
                m_item.setScanTime(val.intValue());

        }

}
```

```
// Classe atuador Panel

import java.lang.*;
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import scada.*;

public class AtuadorPanel extends Panel implements ItemListener,
ActionListener
{
      // Declara propriedades
      coordenadorApplet meu_dono;
      LinkedList atuadores, sensores;

      Label mens;
      TextField txtConst;
      Panel ptemp;
      Button atualiza, refresh, registra;
      java.awt.List lst, listSensores;
      // Variáveis de dados
      Double m_const;
      boolean bbmativa;
      Sensor s_atual;

      // Construtor
      public AtuadorPanel(coordenadorApplet m)
      {
            this.meu_dono = m;
            atuadores = new LinkedList();
            sensores = new LinkedList();

            setLayout(new GridLayout(6,2));
            add(mens = new Label("ATUADOR"));
            add(new Panel());
            add(lst = new java.awt.List(2,true));
            lst.setMultipleMode(false);
```

```
            add(new Panel());
            lst.addItemListener(this);
            add(new Label("Constante: "));
            add(ptemp = new Panel());
            ptemp.setLayout(new GridLayout(1,3));
            ptemp.add(txtConst = new TextField());
            add(new Label("Sensores"));
            add(listSensores = new java.awt.List(2,true));
            listSensores.setMultipleMode(false);
            add(ptemp = new Panel());
            ptemp.add(refresh = new Button("Refresh"));
            ptemp.add(atualiza = new Button("Atualiza"));
            atualiza.addActionListener(this);
            refresh.addActionListener(this);
            add(ptemp = new Panel());
            ptemp.add(new Label("     "));
            ptemp.add(registra = new Button("Registra"));
            registra.addActionListener(this);


      }

      public void menswrite(String m)
      {
            System.out.println(m);
      }

      // recebe atuadores registrados
      public void RecebeLista(Atuador[] s)
      {
            System.out.println("Entrou na funcao..." +
Integer.toString(s.length));
            for (int j=atuadores.size()-1; j >= 0; j--)
            {
                  atuadores.remove(j);
            }

            for (int i=0; i < s.length; i++)
```

```
        {
            System.out.println("Nome: " + s[i].getNome());
            atuadores.add(s[i]);
        }
    }

    // Recebe Sensores registrados
    public void RecebeLista2(Sensor[] s)
    {
        for (int j=sensores.size()-1; j >= 0; j--)
        {
            sensores.remove(j);
        }

        for (int i=0; i < s.length; i++)
        {
            System.out.println("Nome: " + s[i].getNome());
            sensores.add(s[i]);
        }
    }

    // Busca atuadores e sensores
    public void Refresh()
    {
        Atuador m_item;
        lst.removeAll();
        for(int i = 0; i < atuadores.size(); i++)
        {
            m_item = (Atuador)atuadores.get(i);
            lst.add(m_item.getNome(), i);
        }

        Sensor m_item2;
        listSensores.removeAll();
        for (int i=0; i < sensores.size(); i++)
        {
            m_item2 = (Sensor)sensores.get(i);
```

```
                listSensores.add(m_item2.getNome(), i);
            }

        }

        // Atualiza valores do atuador corrente
        public void AtualizaValor()
        {
            Atuador m_item;
            m_item = (Atuador)atuadores.get(lst.getSelectedIndex());

            System.out.println("Selecionou : " + m_item.getNome());
            Double val = Double.valueOf(txtConst.getText());
            if (!val.isNaN())
            {
                m_item.setConstante(val.doubleValue());
            }
        }

        // Mostra dados do atuador na tela
        public void MostraDados()
        {
            Atuador m_item;
            m_item = (Atuador)atuadores.get(lst.getSelectedIndex());

            System.out.println("Item Selecionado: " +
m_item.getNome());

        txtConst.setText(Double.toString(m_item.getConstante()));
        }

        // responde a cliques do mouse
        public void actionPerformed(ActionEvent ev) {
            if (ev.getActionCommand().compareTo("Atualiza") == 0)
            {           AtualizaValor();
                        System.out.println("Clicou em atualiza!");
            }else if (ev.getActionCommand().compareTo("Refresh") ==
0)
            {
```

```
                System.out.println("Clidou em refresh");
                meu_dono.EnviaLista("atuador");
                meu_dono.EnviaLista2("atuador");
                Refresh();
          }else if
(ev.getActionCommand().compareTo("Registra")==0)
          {
                System.out.println("Clidou em registra");
                if (lst.getSelectedIndex() < 0 ||
listSensores.getSelectedIndex() < 0)
                {     System.out.println("Nenhum item
selecionado");
                      return;
                }

                Atuador m_item;
                m_item =
(Atuador)atuadores.get(lst.getSelectedIndex());
                Sensor m_item2;
                m_item2 =
(Sensor)sensores.get(listSensores.getSelectedIndex());
                if (s_atual == null)
                {
                      m_item2.RegistrarAtuador(m_item);
                }else
                {
                      s_atual.desRegistrarAtuador(m_item);
                      m_item2.RegistrarAtuador(m_item);
                }
                s_atual = m_item2;
          }

     }
     // responde a cliques no checkbox
     public void itemStateChanged(ItemEvent e){
          MostraDados();
     }

}          // fim da classe
```

Classe GUIApplet

```
// Classe Trend
package visual;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;
import scada.*;

public class trend extends Applet {
    TrendControls controls;
    TrendCanvas canvas;
    public LinkedList l;
    Coordenador coordenador;
    GUI meu_gui;
    timer t;

    // inicializa a applet
    public void init() {
      setLayout(new BorderLayout());
      canvas = new TrendCanvas();
      add("Center", canvas);
      add("South", controls = new TrendControls(canvas, this));
      try{
            System.out.println("Iniciando o ORB");
            java.util.Properties prop = new java.util.Properties();
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(this,
prop);
            // Inicializa o ORB
            System.out.println("ORB iniciado...");

            // Procura o Coordenador pela sua referência
            org.omg.CORBA.Object coord =
orb.string_to_object("IOR:000000000000001a49444c3a73636164612f436f
6f7264656e61646f723a312e3000000000000001000000000000000240001000000
00000a6c6f63616c686f737400138800000000c636f6f7264656e61646f7200");
```

```
            // Conecta-se ao coordenador
            coordenador = CoordenadorHelper.narrow(coord);
            if (coord == null)
            {
                  System.out.println("Falhou...");
                  return;
            }
            else
            {     System.out.println("Coordenador encontrado...");
            }
            // Conecta-se ao GUI
            meu_gui = coordenador.getGUI("guiserver");
            System.out.println("GUI encontrado:" +
meu_gui.getNome());
            // Instancia uma lista e um timer
            l = new LinkedList();
            t = new timer(this, 2000);
            t.start();

       }catch(Exception e)
       {
             System.out.println("Erro: " + e.getMessage());
       }
     }

     public void destroy() {
         remove(controls);
         remove(canvas);
     }

     public void start() {
       controls.setEnabled(true);
     }

     public void stop() {
       controls.setEnabled(false);
     }
```

```
    public void processEvent(AWTEvent e) {
        if (e.getID() == Event.WINDOW_DESTROY) {
            System.exit(0);
        }
    }

    // Atualiza valores na tela
    public void Refresh()
     {

           pubValor[] p = meu_gui.getPontos();
           System.out.println("Recebeu " +
Integer.toString(p.length) + "pontos");
           if (p.length < 2) return;
           for(int i=(l.size()-1); i >=0 ; i--)
           {
                l.remove(i);
           }
           System.out.println("Removeu todos os pontos");
           for(int i=0; i < p.length; i++)
           {
                l.add(i, p[i]);
           }
           System.out.println("Montou vetor de " +
Integer.toString(l.size()) + " pontos");
           canvas.redraw(l);
     }

    // Busca os sensores
    public Sensor[] BuscarSensores()
    {
     return coordenador.getSensores();
    }

    // Função Main
    public static void main(String args[]) {
     Frame f = new Frame("ArcTest");
     trend arcTest = new trend();
```

```
        arcTest.init();
        arcTest.start();

        f.add("Center", arcTest);
        f.setSize(300, 400);
        f.show();
    }

    public String getAppletInfo() {
        return "Trend do sistema Scada";
    }

    // Subscreve-se em um sensor
    public void SubscreverSensor(Sensor s) {
        s.RegistrarGUI(meu_gui);
    }

    // Caso esteja subscrito, dessubscreve primeiro
    public void SubscreverSensor(Sensor s, Sensor last) {
        last.desRegistrarGUI(meu_gui);
        s.RegistrarGUI(meu_gui);

    }

}
```

```
// Classe TrendCanvas

package visual;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.lang.*;
import java.util.*;
import scada.*;


class TrendCanvas extends Canvas {
    int          startAngle = 0;
    int          endAngle = 45;
    boolean      filled = false;
    Font   font;
    LinkedList   ll;

     // Plota o gráfico
    public void paint(Graphics g) {
     Rectangle r = getBounds();
     int hlines = 10;
     int vlines = 10;
     int valory, ult_valor;
     int altura;
     int nvalores;


     altura = r.height - 50;


     for (int i = hlines; i > 0; i--) {
         g.setColor(Color.blue);
         g.drawLine(r.width/vlines, (altura/hlines)*i, r.width,
(altura/hlines)*i);
         g.setColor(Color.black);
         g.drawString(Integer.toString(altura -
```

```
(altura/hlines)*i),r.width/vlines-25, (altura/hlines)*i);
      }
      g.setColor(Color.blue);
      for (int i = 1; i <= vlines; i++) {
          g.drawLine((r.width/vlines)*i, 0, (r.width/vlines)*i,
altura);
      }

      g.setColor(Color.red);
      nvalores = ll.size();
      if (nvalores < 2) return;

      pubValor x1,x2;
      DataHora dd;
      for(int i=1; i<nvalores; i++)
      {
            x1 = (pubValor)ll.get(i-1);
            x2 = (pubValor)ll.get(i);
            dd = x1.getHorario();
            g.drawLine((i)*(r.width/nvalores), r.height -
(int)x1.getValor() - 50, (i+1)*(r.width/nvalores),r.height -
(int)x2.getValor() - 50);
            g.setColor(Color.black);
            g.drawString(dd.getHorario(), (i)*(r.width/nvalores),
altura+20);
            g.setColor(Color.red);
      }

      if (filled) {
          //g.fillArc(0, 0, r.width - 1, r.height - 1, startAngle,
endAngle);
      } else {
          //g.drawArc(0, 0, r.width - 1, r.height - 1, startAngle,
endAngle);
      }

      g.setColor(Color.black);
      g.drawLine(r.width/vlines, altura, r.width, altura);
      g.drawLine(r.width/vlines, altura, r.width/vlines, 0);
```

```
        g.setFont(font);
        int sx = 10;
        int sy = r.height - 28;

    }

        // Recebe valores e plota gráfico
    public void redraw(LinkedList valores) {
        this.ll = valores;
        repaint();
    }
}           // Fim da classe
```

```
// Classe TrendControls

package visual;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;
import scada.*;

class TrendControls extends Panel implements ItemListener,
ActionListener {
      // Declara as propriedades
    java.awt.List lista;
    TrendCanvas canvas;
    trend meu_dono;
    java.util.LinkedList h;
    Button buscar, refresh;
    Sensor sens_atual;
    TextField vari, unid;

      // Construtor
    public TrendControls(TrendCanvas canvas, trend m) {
      this.canvas = canvas;
      meu_dono = m;
      buscar = new Button("Buscar");
      buscar.addActionListener(this);
      add(buscar);
      add(new Label("Ponto "));
      add(lista = new java.awt.List(3, false));
      lista.setMultipleMode(false);
      lista.addItemListener(this);
      Panel p = new Panel();
      p.setLayout(new GridLayout(2,2));
      p.add(new Label("Variavel"));
      p.add(vari = new TextField());
      p.add(new Label("Unidade"));
      p.add(unid = new TextField());
```

```
  add(p);
  h = new LinkedList();
}

  // Recebe valores
public void RecebeValores(LinkedList hreceb)
  {
        if (h.size() >= 10)
        {
              h.removeFirst();
        }
        h.add(new Double(Math.random()*100));
        canvas.redraw(h);

  }

  // Busca os sensores registrados no coordenador
private void BuscarSensores()
{
  Sensor[] ss = meu_dono.BuscarSensores();
  for(int i=(lista.getItemCount()-1); i >=0 ; i--)
  {
        lista.remove(i);
  }
  for (int i=0; i < ss.length; i++)
  {
        System.out.println("Nome: " + ss[i].getNome());
        h.add(i, ss[i]);
        lista.add(ss[i].getNome(),i);
  }
}

  // responde a cliques do mouse
public void actionPerformed(ActionEvent ev) {
  String label = ev.getActionCommand();
  if (label.compareTo("Buscar")==0)
  {
```

```
            BuscarSensores();
        }else if(label.compareTo("Refresh")==0)
        {

        }
    }

        // Subscreve o GUI a um sensor
    public void SubscreverSensor() {
        Sensor s = (Sensor) h.get(lista.getSelectedIndex());
        System.out.println("Sensor " + s.getNome() + " selecionado");
        if (sens_atual == null)
        {     System.out.println("Vai registrar o primeiro sensor");
            meu_dono.SubscreverSensor(s);     }
        else
        {     System.out.println("Vai registrar outro sensor");
            meu_dono.SubscreverSensor(s, sens_atual);    }

        sens_atual = s;
        vari.setText(s.getVariavel());
        unid.setText(s.getUnidade());

    }

        // Responde a uma mudança de estado
    public void itemStateChanged(ItemEvent e){
        SubscreverSensor();
    }

}
```

9. CONSIDERAÇÕES FINAIS

A estrutura apresentada acima define um *framework* para a implementação de sistema SCADA distribuído, ou seja, foram apresentados as classes da estrutura de um sistema SCADA. Com isso espera-se ter sido este o primeiro passo no desenvolvimento de um ssitema que com toda a complexidade que as aplicações comerciais exigem.

Tendo o seu código aberto e sua estrutura documentada, espera-se que novos trabalhos sejam feitos nesta área, sem que se tenha que refazer o que está implementado e documentado nestas páginas.

A aplicação prática deste trabalho, tornar-se-á mais evidente quando o mercado dispuser de equipamentos de hardware que suportem aplicações java sem a necessidade de componentes adicionais, os chamados equipamentos com 'java nativo'. Equipamentos estes, que acreditamos, sejam a tendência dos fabricantes, e que não tardam para se tornatem a chegar ao mercado. Com esta visão, e considerando a facilidade que o desenvolvimento de componentes nos proporciona em termos de reutilização de código, acreditamos ser este um projeto de grande valor no que concerne à sua utilidade prática.

10. BIBLIOGRAFIA

- BARRETO, M.R.P. **UML: Unified Modeling Language.** COMDEX Brazil 1999. São Paulo, 1999.
- CAMPIONE, M; WALRATH, K. **Java Tutorial, The**. 2 ed. Palo Alto, Addison Wesley, 1998.
- ORBACUS for C++ and Java. Massachusetts, Object Oriented Concepts, 1999.
- ORFALI, R ; HARKEY, D. **Java and CORBA, Client / Server Programming with**. 2 ed. Nova Iorque, Wiley, 1998.
- ROSENBERG, J. **CORBA in 14 days, Teach Yourself.** 1 ed. Indiana, SAMS, 1998.

APÊNDICE A - UML

UML é uma linguagem utilizada na modelagem de projetos orientados a objeto. Esta linguagem, define 9 tipos de diagramas, que são:

- ✓ Diagrama de classe
- ✓ Diagrama de Objeto
- ✓ Diagrama de Caso de Uso
- ✓ Diagrama de Sequência
- ✓ Diagrama de Colaboração
- ✓ Diagrama de Estado
- ✓ Diagrama de Atividade
- ✓ Diagrama de Componente
- ✓ Diagrama de Instalação

Destes usaremos apenas três, que foram aplicáveis no caso. São eles:

Diagrama de Uso de Caso (Use Case):

Diagrama usado para modelar o comportameto do sistema, identificando *atores* e de que pretendem usar o sistema.

Diagrama de Classes:

Diagrama que mostra o conjunto de classes do sistema com os seus relacionamentos.

Diagrama de Sequência

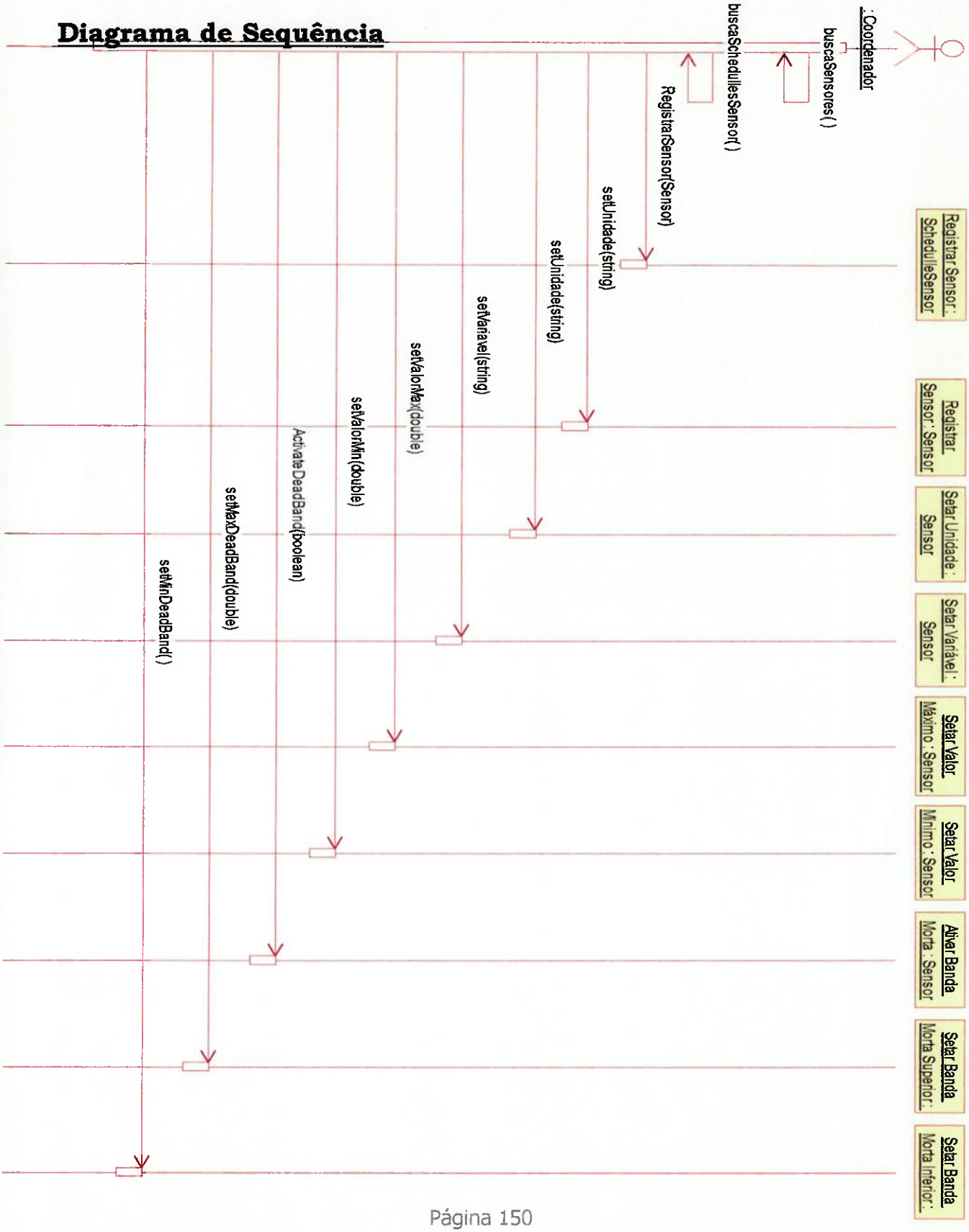
Diagrama que mostra a sequência temporal de iteração entre os elementos.

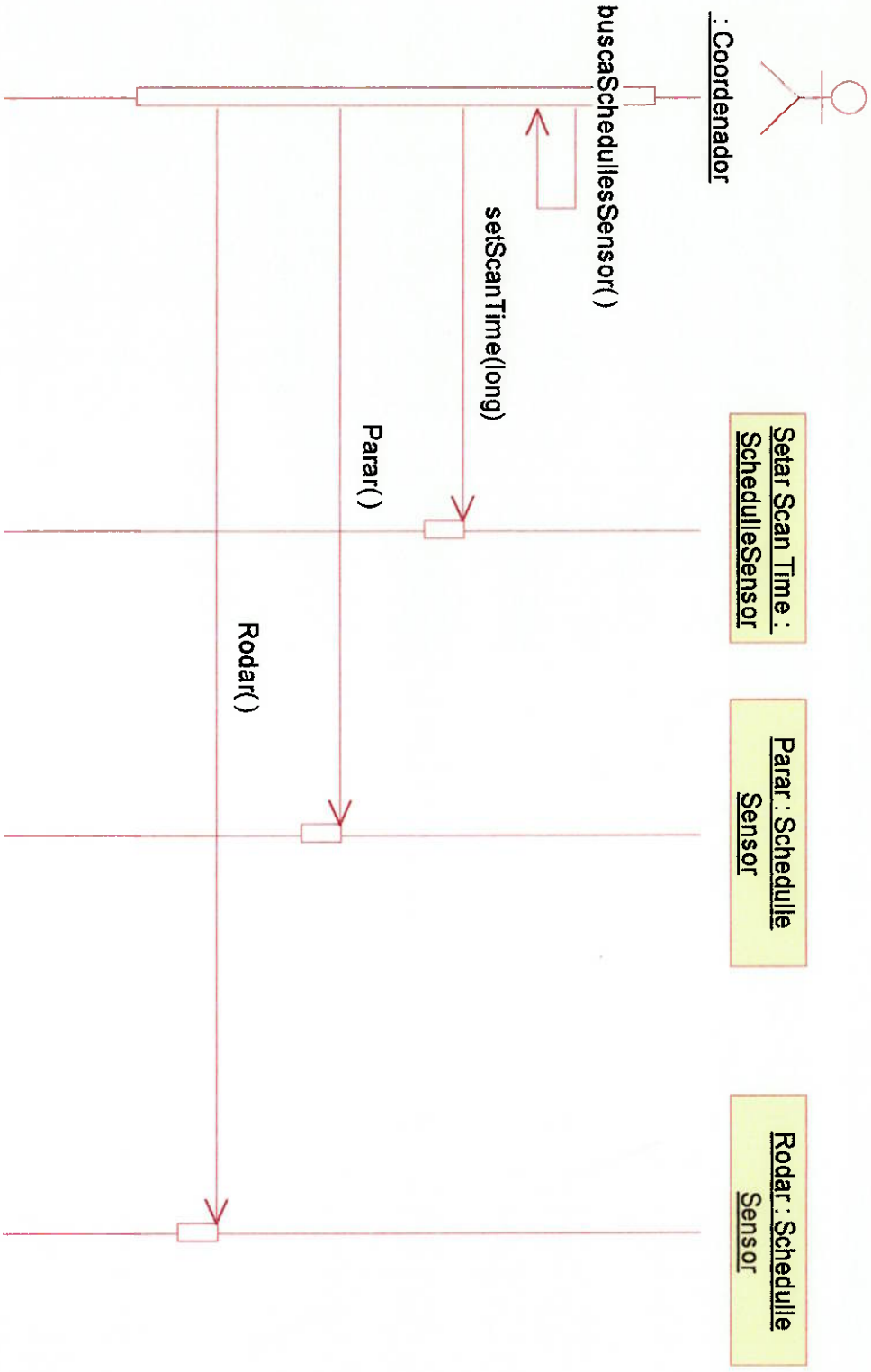
Para esta modelagem, foi ulitizada uma ferramenta case, que possui uma série de funcionalidades para ajudar, não só a desenhar os diagrama, como também analisar o que está sendo feito, atulizar modificações automaticamente e gerar documentação do sistema. No caso específico deste projeto foi utilizado o Rational Rose ®, que é uma ferramenta contendo todas as funcionalidades descritas acima.

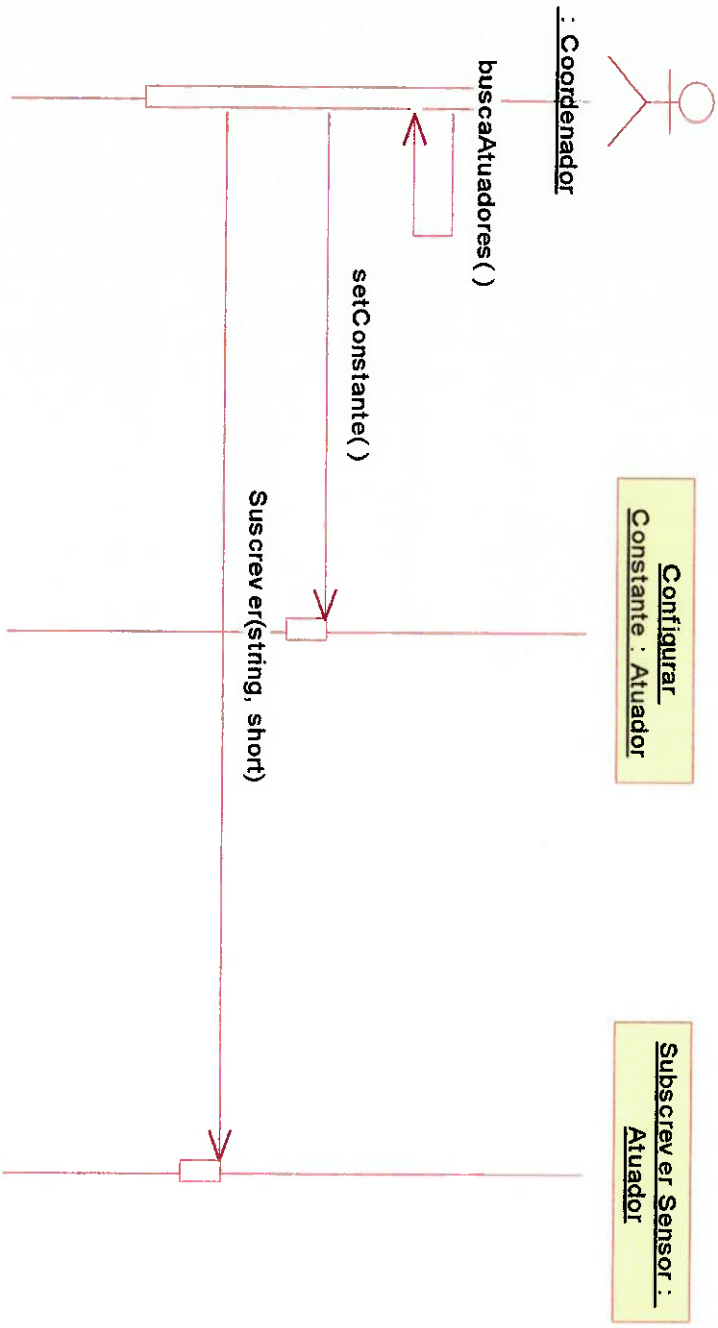
Os diagramas gerados, bem como uma cópia da documentação estão anexos às páginas que se seguem.

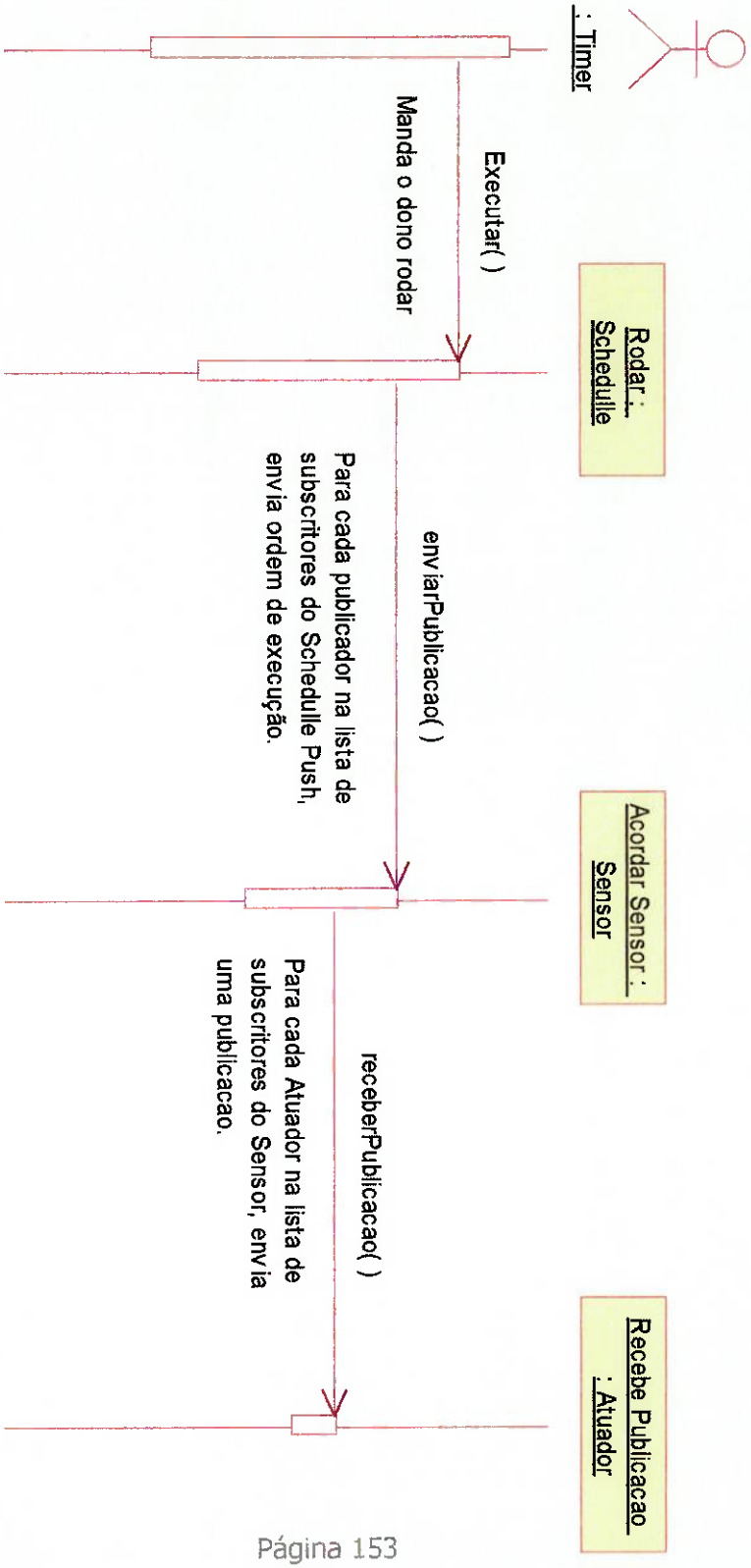
Diagrama de Classes

Diagrama de Sequência




: Coordenador
buscaSchedullesSensor()
setScanTime(long)
Parar()
Rodar()
Setar Scan Time : SchedulleSensor
Parar : Schedulle Sensor
Rodar : Schedulle Sensor


: Coordenador
buscaAtuadores( )
setConstante( )
Suscrever(string, short)
Configurar
Constante : Atuador
Subscrever Sensor :
Atuador


: Timer
Executar( )
Manda o dono rodar
Rodar : Schedulle
Para cada publicador na lista de subscritores do Schedulle Push, envia ordem de execução.
enviarPublicacao( )
Acordar Sensor : Sensor
Para cada Atuador na lista de subscritores do Sensor, envia uma publicacao.
receberPublicacao( )
Recebe Publicacao : Atuador

Documentação gerada pelo Rational Rose ®

Table of Contents

LOGICAL VIEW REPORT

Logical View

PUBLICADOR

Essa classe terá definida toda a estrutura dos elementos que são publicadores.

Derived from Componente

Private Attributes:

listSubscritores : Subscritor

Este atributo é uma lista que contém os subcritores que estão registrados neste publicador. Os métodos Registrar() e desRegistrar() alteram diretamente essa lista.

issue : Publicacao

Esse método contém uma instância de publicação, que se refere ao que está sendo publicado pelo objeto. Assim cada classe que se derivar de Publicador, estará publicando um tipo de Publicacao.

Public Operations:

Registrar (p_subscritor : Subscritor) : short

Esse método é exposto pelos publicadores para que os subscritores possam se registrar nestes.

enviarPublicacao () : void

Esse método faz com que o o publicador envie uma publicação para cada subcritor na sua lista. É utilizado para se trabalhar com esquema Push. Neste caso haverá um objeto Schedulle associado ao publicador.

desRegistrar (p_subscritor : Subscritor) : short

Método utilizado para remover o registro de um subscritor sobre um publicador.

ListarSubscritores () : Subscritor[]

Esta função retorna uma lista com os subcritores registrados no publicador.

VALOR

Este tipo, deine alguns métodos e propriedades básicas que todos os publicadores de valor terão.
Derived from Publicador

Private Attributes:

Esquema : Schedulle

Esta propriedade vai conter uma uma referência ao Schedulle em que este publicador está registrado.

Public Operations:

setEsquema () : void

Este método vai setar um novo scheulle para o publicador de valor se registrar.

getEsquema () : Schedulle

Este método vai retornar o scheulle em que o publicador de valor está registrado.

getPublicacao () : Publicacao

Retorna a uma publicação de valor.

SCHEDULLE

Classe que define os Schedulles, que serão responsáveis pelas varreduras dos componentes.
Derived from Publicador

Private Attributes:
Este parâmetro define o tempo de varredura do Schedulle.

TipoSubscritor : intScanTime : long

Este parâmetro vai determinar se o Schedulle recebe Subscritores ou Históricos.

Public Operations:

getScanTime () : Long

Este método retorna o tempo de varredura do elemento.

setScanTime (scan : long) : void

Este método seta o tempo de varredura de Schedulle.

Executar () : void

Este método vai fazer com que os publicadores registrados enviem suas publicações e com que o Histórico mande para a base de dados os seus dados.

Parar () : void

Este método determina a parada da varredura. Assim todos os subscritores registrados no esquema serão parados.

Rodar () : void

Este método faz o Schedulle rodar. É chamado ao se instanciar o objeto.

COMPONENTE

Esta classe descreve as propriedades e métodos básicos de todos os elementos do sistema.

Private Attributes:

Nome : string

Essa variável membro contém o nome que unicamente identifica o objeto no sistema.

Public Operations:

getNome () : string

Esse método Retorna o nome do componente.

getID () : string

Esse método retorna o ID que unicamente identifica o objeto. O ID é gerado pelo framework de CORBA.

setNome () : void

Este método possibilita a lateração do nome do componente.

SENSOR

A classe sensor representa uma entidade física. Nesta implemtentação, o sensor terá a oporunidade funcionar como push ou pull, dependendo de como o subscritor se registrar nele. Caso o sensor não tenha se registrado em nenhum schedulle, não será permitido ao subscritor se registrar como push.

Derived from Valor, Subscritor

Private Attributes:

Variavel : string

Indica o nome da grandeza física sendo medida.

Unidade : string

Indica a unidade da grandeza física sendo medida.

ValorMax : double

Indica o máximo valor que a variável pode atingir.

ValorMin : double

Indica o mínimo valor que a variável pode atingir.

ListAtuadores : Atuador[]

Public Operations:

Retorna a variável (grandeza física) sendo medida.

setVariavel (m_variavel : string) : voidgetVariavel () : string

Seta a variável sendo medida.

getUnidade () : string

Retorna a unidade da grandeza física sendo medida.

setUnidade (m_unid : string) : void

Seta a unidade da grandez física sendo medida.

getValorMax () : double

Reorna o maior valor a ser medido pelo sensor.

setValorMax (vMax : double) : void

Seta o maior valor a ser medido pelo sensor.

getValorMin () : double

Retorna o menor valor a ser medido pelo sensor.

setValorMin (vMin : double) : void

Seta o menor valor a ser medido pelo sensor.

setMaxDeadBand (maxdb : double) : void

Seta o limite superior da Banda Morta.

getMaxDeadBand () : double

Retorna o limite superior da banda morta setada..

setMinDeadBand () : void

Seta o limite inferior da Banda Morta.

getMinDeadBand () : void

Retorna o limite inferior da banda morta.

IsDeadBandActive () : boolean

Indica se a Banda morta esta ativa ou não.

ActivateDeadBand (activar : boolean) : void

Ativa/Desativa a banda morta.

RegistrarAtuador (a : Atuador) : short

Este método permite o atuador subscrever um sensor.

desRegistrarAtuador () : short

Este método permite ao Atuador se desubscrever do sensor.

registrarGUI (g : GUI) : short

Permite ao GUI se registrar no sensor.

desRegistrarGUI (g : GUI) : short

Permite ao GUI se desusbscrever do sensor.

RegistrarAtuador () : short

Permite ao Atuador subscrever no sensor.

desRegistrarAtuador () : short

Permite ao atuador de desregistrar do sensor.

SUBSCRITOR

Esta é a classe que contém toda a estrutra a ser derivada por todos os subscritores do sistema.

Derived from Componente

Private Attributes:

Publicador : Publicador

Esta variável indicará em qual publicador o subscritor está registrado.

Public Operations:

Suscrever (IDPublicador : string = default, Tipo : short = default) : void

Esse método contém todo procedimento que o subscritor deve fazer para se registrar em um publicador.

buscarPublicadores (tipo : string) : publicador[]

Esse método implementa o procedimento que deve ser tomado para se fazer busca no sistema de diretórios especificado pela arquitetura CORBA. Recebe como parâmetro uma string que indicará que tipo de publicador está sendo procurado.

unSubscrever (argname : argtype = default) : void

Esse método é chamado para se remover o registro do objeto no publicador em que está registrado.

receberPublicacao () : void

Esse método é exposto para que os publicadores possam enviar publicações aos subscritores cmo esquema Push.

ATUADOR

Classe básica que implementa atuador. Para esta implementação o atuador apenas passa seu estado para um valor proporcional ao valor recebido.

Derived from Subscritor, Valor

Private Attributes:

Constante : double

Define uma constante. O atuador no caso vai ser apenas um valor proporcional ao valor de entrada.

Public Operations:

setConstante () :

Vai setar o valor da constante de proporcionalidade.

getConstante () : double

Este método retorna o valor da constante de proporcionalidade.

GUI

Esta é a classe genérica de elementos gráficos.

Derived from Subscritor

Private Attributes:

npontos : long

tipo : int

ultimoValor : Publicacao

Public Operations:

Refresh () :

setNPontos (npontos : long) :

getNPontos () : long

getTipo () : int

setTipo (tipo : int) :

PUBLICACAO

Classe básica que define os tipos de publicação.

Private Attributes:

IDPublicacao : int

Propriedade que identifica o tipo da publicação.
Propriedade indicando o horário da pubklicação.

horario : DataHora

Public Operations:

getID () : long

Retorna o ID de Publicação, que vai indicar que tipo de publicação.

getHorario () : datetime

Retorna o horáio da publicação.

PUBVALOR

Classe que contém os parâmtros a serem passados por uma publicação de valor.

Derived from Publicacao

Private Attributes:

valor : double

Parâmetro contendo o valor da publicação.

Public Operations:

getValor () : double

Retorna o valor da publicação.

setValor (valor : double) :

Este método seta o valor da publicação.

DATA

Classe para definir Data.

Private Attributes:

dia : int

Indica o dia.

mes : int

Indica o mês.

ano : int

Indica o ano.

diasemana : string

Indica o dia da semana.

Public Operations:

getData (formato : string) : string

método para retornar um a string que represnete a data.

getDia () : int

Este método retona o dia.

getMes () : int

Este método retorna o mês.

getAno () : int

Este método retorna o ano.

getDiaDaSemana () : string

Este método retorna o dia da semana.

EMaior (outradata : Data) : int

Este métod compara data do objeto com outra data. Retorna 1 caso esta data seja maior, -1 caso a outra data seja maior e 0 caso sejam iguais.

adiciona (parametro : string, valor : int) : void

Adiciona um período à data. A variável parâmetro indica qual período será somado: 'd' indica dia, 'm' indica mês e 'a' indica ano. A variável valor indica quanto será somado.
Retoma uma string formatada representando a data.

getDataFormatada (formato : string) : string

DATAHORA

Classe que define data e horário.

Derived from Data

Private Attributes:

hora : int

Indica a hora.

minuto : int

Indica o minuto.

segundo : int

Indica o segundo.

Public Operations:

getHorario () : string

Retorna uma String representando o horário.

getHora () : int

Retorna a hora.

getMinuto () : int

Retorna o minuto.

getSegundo () : int

Retorna o segundo.

getHorarioFormatado (formato : string) : string

Este método retorna um string representando o horário formatado.

SCHEDULLESENSOR

Derived from Schedulle

Private Attributes:

ListSensores : Sensor[]

Variável que contém uma lista dos Sensores registrados n schedulle.

Public Operations:

RegistrarSensor (s : Sensor) : short

Método utilizado pelos Sensores para se subscreverem no Schedulle

desRegistrarSensor () : short

Método utilizado pelos Sensores para se desubscreverem no Schedulle