UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Geração de Sistemas Supervisórios a partir de Modelos Orientados a Objetos

por

LEANDRO ROSNIAK TIBOLA

Dissertação submetida à avaliação, como requisito
parcial para a obtenção do grau de Mestre em
Ciência da Computação

Prof. Dr. Carlos Eduardo Pereira
Orientador

Porto Alegre, novembro de 2000.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Tibola , Leandro Rosniak
Geração de Sistemas Supervisórios a partir de Modelos Orientados a Objetos / por Leandro Rosniak Tibola. - Porto Alegre: PPGC da UFRGS, 2000.
63f.:il.

Dissertação (Mestrado) - Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2000. Orientador: Pereira, Carlos E.

1. Sistemas Supervisórios . 2. Orientação a Objetos. 3. Modelagem. 4. Integração de Ambientes de Modelagem e Sistemas Supervisórios. 5. SCADA I. Pereira, Carlos E. II. Título.

Dedico este trabalho aos meus pais Dervis e Otilia Tibola, minha irmã Luciana, meu irmão Laércio (*in memoriam*) e minha namorada Angela Cristina Pessotto.

Agradecimentos

Aprender, pesquisar, descobrir e conhecer. Estas foram as principais motivações que me levaram a empreender jornadas semanais de praticamente 500 Km entre Frederico Westphalen e Porto Alegre para realizar o curso de mestrado. Duas noites dormindo no banco de um ônibus, viagens de carona, tanto no verão quanto inverno. Isto só possível com a paciência, a compreensão e o apoio de muitas pessoas, verdadeiros amigos. Para todas elas, meus sinceros agradecimentos.

Agradeço inicialmente aos meus pais, Dervis e Otilia, que na ausência de recursos financeiros, me ensinaram o valor do trabalho, da abnegação, do compromisso, da responsabilidade e da humildade como meios para o sucesso. Pai, teu exemplo e obras falam mais que palavras. Mãe, obrigado pelas lagrimas derramadas e pelas orações fervorosas, sei que as tuas têm muito poder. Minha irmã Luciana, que de longe acompanhou minha trajetória, obrigado pelas palavras de encorajamento. O caminho está aberto, tu também poderás trilhá-lo. Meu irmão Laércio, que partiu dias antes que eu começasse o mestrado, mas deixou muito boas lembranças. Ainda quero alcançar uma disciplina igual a tua.

Durante este curso, tive a imensa felicidade de contar com a companhia, o apoio e a compreensão da minha namorada Angela, que permitiu que eu dividisse a maior parte do meu tempo entre o trabalho na URI e o mestrado, ficando para ela o pouquinho que sobrava. Você merece um "*upgrade*". Te amo, obrigado.

Gostaria de agradecer a "turma" do IEE, aos professores João Netto e Renato Ventura, aos colegas Wilson Pardi Jr., Carlos Mitidieri, Isabel Fernandes, Rafael Wild, Marcelo Göetz, João Pacheco e Sérgio Süess, Cristiano Brudna. Um agradecimento especial ao colega e "guia" durante o mestrado, Leandro Buss Becker, que desde o primeiro em que nos conhecemos foi mais que um amigo, foi um irmão. Que teu caminho seja repleto de sucesso.

Dedico um agradecimento especial ao meu orientador, professor Carlos Eduardo Pereira, que viu em um interiorano o potencial para ser mestre. Desculpe-me, Prof. Carlos, pelas minhas limitações, gostaria de ter feito mais e melhor. Admiro tua capacidade de fazer "milhares" de coisas, envolvido com tudo e com todos. Deus te abençoe.

Desejo agradecer de forma especial ao Prof. Nelson Zang e ao Prof. Evandro Preuss, pelo estímulo, companheirismo e compreensão demostrado no período de estudos. Acima da posição hierárquica sempre sobressaiu-se vossa amizade. Agradeço à Universidade Regional Integrada, pelo apoio para que a realização deste curso fosse possível.

Não poderia deixar de agradecer aos colegas de "república" João "Tutti" Colle, Giovani De Cezaro e Sandro "Seco" Negrello. Amigos, não dividimos somente um apartamento, dividimos nossos problemas, nossos sonhos, nossa comida, nosso dinheiro e principalmente nossa amizade. Valerem as gargalhadas, as "bóias", as "Skol". "Somamos" um apartamento com mais três vencedores.

Finalmente, gostaria de dizer que o mestrado foi um imenso aprendizado para mim, que nunca tinha saído de casa. Melhorei como profissional, mas aprimorei muito o homem, o filho, o irmão, o namorado e o amigo. Percebi que o crescimento deve ser do todo e a técnica de nada vale se o humano for esquecido.

Sumário

Lista de Abreviaturas 7

Lista de Figuras 8

Resumo 9

Abstract 10

1 Introdução 11

1.2 Sistemas Supervisórios 13
1.3 Modelagem Orientada a Objeto 15
1.4 Motivação 17
1.5 Objetivos do Trabalho 19
1.6 Contribuição do Autor 19
1.7 Organização do Volume 19

2 Estado da Arte 20

2.1 Ferramenta para Sistema Supervisório 20
2.1.1 Elipse Windows 23
2.1.2 Análise do Sistema Supervisório 28
2.2 Ferramentas de Modelagem Orientadas a Objeto 30
2.2.1 SIMOO 32

3 Projeto Conceitual da Ferramenta *Supervisory Designer* 36

3.1 O Modelo Conceitual 36
3.2 Exemplo de Visões: Automóvel 37
3.2.1 A "Visão" do Motorista 38
3.2.2 A "Visão" do Mecânico 40
3.3 Proposta de Integração com Sistemas Supervisórios 43
3.4 Conclusão 46

4 Estudo de Caso 48

4.1 O Modelo Orientado a Objetos do Sistema de Engarrafamento de Bebidas 48
4.2 Usando o Modelo no *Supervisory Designer* 50
4.2.1 A criação de Elementos de Visualização (EV) 52
4.2.2 A apresentação dos Elementos de Visualização (EV) 58

5 Conclusões e Trabalhos Futuros 60

5.1 Trabalhos Futuros 61

Bibliografia 62

Lista de Abreviaturas

CLP	Controlador Lógico Programável
CASE	*Computer Aided Software Engineering*
DCS	*Distributed Control Sistems*
DDE	*Dynamic Data Exchange*
DELET	Departamento de Engenharia
IEE	Instrumentação Eletro-Eletrônica
MET	*Model Editor Tool*
PCS	*Process Control Systems*
PLC	*Programable Logical Controller*
SCADA	*Supervisory Control & Data Aquisition Systems*
UML	*Unified Modified Language*
OPC	Object Linking and Embedding for Process Control
AS	Structured Analysis

Lista de Figuras

FIGURA 1.1 - Exemplo de um Sistema de Controle em Rede. 12
FIGURA 1.2 - Representações da "*Tag* Nível" de um Tanque. 14
FIGURA 2.1 - Exemplo da Tela do Organizer. 24
FIGURA 2.2- *Tag* do Tipo Demo. 25
FIGURA 2.3 - *Tag* do Tipo RAM. 25
FIGURA 2.4 - A Página de Alarmes do *Tag* Demo. 26
FIGURA 2.5 - Janela de Script do *Tag* Demo. 28
FIGURA 2.6 - Criação de Diferentes Tags e Definição seus Valores 29
FIGURA 2.7 - Diferença dos Valores do Elemento Gráfico e da *Tag*. 30
FIGURA 2.8 - Alteração do Alarme LoLo para as *Tags* Pressao 30
FIGURA 2.9 - Diagrama de Classes do SIMOO. 34
FIGURA 2.10 - Diagrama de Instâncias no SIMOO. 35
FIGURA 3.1 - O Diagrama de Classes do Modelo Conceitual do *Supersory Designer*. 36
FIGURA 3.2 - Diagrama de Classes e Instâncias do Automóvel. 37
FIGURA 3.3 - A Visão de Automóvel do Motorista 39
FIGURA 3.4 - A "Visão" de Automóvel do Mecânico. 42
FIGURA 3.5 - Características Básicas Definidas para a *Tag* no Elipse Windows. 44
FIGURA 3.6 - Definição de Atributos de Métodos no SIMOO-RT. 45
FIGURA 3.7 - O Esquema Ideal de Integração das Ferramentas SIMOO e Elipse. 45
FIGURA 3.8 - Esquema Detalhado de Integração das Ferramentas SIMOO e Elipse… 46
FIGURA 4.1 - Exemplo de Supervisório Presente no Elipse Software. 48
FIGURA 4.2 - O Modelo de Classes da Engarrafadora de Bebidas. 49
FIGURA 4.3 - Janelas de Atributos e Alarmes da Classe Tanque. 49
FIGURA 4.4 - A Estrutura do Arquivo de Visualização (.vis). 50
FIGURA 4.5 - O Modelo Expandido no *Supervisory Designer* 51
FIGURA 4.6 - A janela *General* da Configuração do *Gauge*. 53
FIGURA 4.7 - A Janela *Position* da Configuração do *Gauge*. 53
FIGURA 4.8 - A Janela *Advanced* da Configuração do *Gauge*. 54
FIGURA 4.9 - A Janela *General* da Configuração do *Display*. 55
FIGURA 4.10 - A Janela *General* da Configuração do *Bitmap Sequence*. 56
FIGURA 4.11 - A Janela *Images* da Configuração do *Bitmap Sequence*. 57
FIGURA 4.12 - A Janela *General* da Configuração do *Bar Graph*. 58
FIGURA 4.13 - Elementos de Visualização da Classe Tanque. 59

Resumo

Este trabalho aborda o tema da geração de sistemas supervisórios a partir de modelos orientados a objetos. A motivação para realização do trabalho surgiu com o estudo de sistemas supervisórios e de ferramentas de suporte à modelagem de sistemas usando orientação a objetos. Notou-se que nos primeiros, apesar de possuírem como principal objetivo a visualização de estados e grandezas físicas relacionadas a componentes de plantas industriais (nível de um tanque, temperatura de um gás, por exemplo), os modelos computacionais utilizados baseiam-se em estruturas de dados não hierárquicas, nas quais variáveis de contexto global e não encapsuladas, as chamadas "*tags*", são associadas às grandezas físicas a serem visualizadas. Modelos orientados a objeto, por outro lado, constituem uma excelente proposta para a criação de modelos computacionais nos quais a estrutura e semântica dos elementos de modelagem é bastante próxima a de sistemas físicos reais, facilitando a construção e compreensão dos modelos.

Assim sendo, a proposta desenvolvida neste trabalho busca agregar as vantagens do uso de orientação a objetos, com conceitos existentes em sistemas supervisórios, a fim de obter-se ferramentas que melhor auxiliem o desenvolvimento de aplicações complexas. Classes e suas instâncias são usadas para modelagem de componentes da planta industrial a ser analisada. Seus atributos e estados são associados às grandezas físicas a serem visualizadas. Diferentes formas de visualização são associadas às classes, aumentando assim o reuso e facilitando o desenvolvimento de sistemas supervisórios de aplicações complexas.

A proposta conceitual desenvolvida foi implementada experimentalmente como uma extensão à ferramenta SIMOO-RT, tendo sido denominada de "*Supervisory Designer*". A ferramenta desenvolvida estende o modelo de objetos e classes de SIMOO-RT, permitindo a adição de informações específicas para supervisão – tais como as definições de limites para os atributos. A ferramenta foi validada através do desenvolvimento de estudos de casos de aplicações industriais reais, tendo demonstrado diversas vantagens quando comparada com o uso de ferramentas para construção de sistemas supervisórios disponíveis comercialmente).

Palavras chave: Sistemas Supervisórios, projeto orientado a objeto, modelagem orientada a objeto, SIMOO-RT, *SCADA*.

TITLE: " SUPERVISORY SYSTEMS DESIGN BASEAD ON OBJECT-ORIENTED MODELS ".

Abstract

This work approaches the generation theme of supervisory systems from oriented models to objects. The motivation to carry out this work came up from the study of supervisory system and of support tools to the modelling of systems using orientation of objects. We realised that in former ones, in spite of having as main objective the visualisation of states and physics greatness related to industrial maps components (e.g. tank level, gas temperature) the computational models used, based themselves on structures of no hierarchical datum, in which variables from global context and not encapsulated, the called "tags" are associated to physics greatness to be visualised. Oriented models to objects, on one hand, constitute an excellent proposal to the creation of computational models in which the structure and semantic of modelling elements is quite close to the physics real systems, making the construction and the comphreension of models easy.

In such case, the proposal developed in this work searches to join the advantages of the use of oriented objects, with existed concepts in supervisory systems, in order to get tools that better help the development of complex applications. Classes and their instance are used to modelling of industrial maps components to be analysed. Their attributes and states are associated to physics greatness to be visualised. Different forms of visualisation are associated to classes, increasing in this way the reuse and making the development of supervisory system of complex application easy.

The conceptual proposal develop was implemented experimentally as an extension to SIMOO-RT tool, had been denominated "Supervisory Designer", allowing the addiction of specific information to supervision - such as the limit definitions to attributes. The toll was validity through the development of studies from cases of real industrial applications, had showed several advantages when comparing with the use of tools to construction of supervisory systems available commercially.

Keywords: Supervisory systems, oriented project to object, oriented modelling to object, SIMOO-RT, SCADA.

1 Introdução

Desde o advento da Revolução Industrial até nossos dias, as técnicas de produção têm sofrido muitas modificações. Dentre os muitos sistemas usados nas empresas para controlar e gerenciar a produção, os que mais têm se difundido são os PCS (Sistemas de Controle de Processos ou *Process Control Systems*), os SCADA (Sistemas de Controle Supervisório e Aquisição de Dados ou *Supervisory Control & Data Aquisition Systems*) e os DCS (Sistemas de Controle Distribuído ou *Distributed Control Systems*).

Os PCS (Sistemas de Controle de Processos ou *Process Control Systems*) são sistemas computacionais que associam o controle estatístico de processos com equipamentos de controle automático. Seu principal objetivo é reduzir o desvio da produção efetiva em relação à produção programada. Os PCS têm como foco a tentativa de combinar o controle estatístico de processos com as potencialidades dos equipamentos de controle de processo, a fim de minimizar as variações da produção, através da detecção e remoção de causas detectáveis, utilizando os equipamentos de controle automático para o ajuste e a contabilização destas causas. Típicos exemplos de aplicações de PCS são as plantas de produção, usinas de energia elétrica e sistemas de transporte.

Os DCS (Sistemas de Controle Distribuído ou *Distributed Control Systems*) são sistemas de computadores criados para controlar processos industriais. Eles recebem estes nomes devido à arquitetura do seu projeto. O processamento é dividido entre muitos processadores ao invés de usar apenas um único computador com elevada capacidade de processamento. Geralmente os processadores são construídos em módulos separados, os quais são otimizados para uma função particular. Um módulo, por exemplo, pode fornecer uma interface com o operador, outro pode registrar o histórico de dados, etc. Freqüentemente os módulos são fisicamente distribuídos pela planta da fábrica. Isto é ainda mais freqüente quando os módulos estão conectados a instrumentos de controle existentes na planta, uma vez que isto reduz drasticamente o custo de cabeamento para comunicação. Esta distribuição física dos módulos leva à necessidade de um processamento distribuído, impondo requisitos de coordenação e sincronização dos diversos processos. Módulos com missão crítica possuem redundância, pois dois ou mais módulos idênticos realizam exatamente a mesma

atividade. Somente um deles efetua funções de computação no processo real. Ele é conhecido como equipamento primário ou líder. O outro módulo, conhecido como equipamento reserva, aguarda por uma falha do primário para entrar em ação. Se isto ocorrer, ele toma o controle do módulo primário e passa, ele próprio, a ser o módulo primário. Assim, o equipamento com problema pode ser consertado sem perda de controle do processo.

Por sua vez, os SCADA (Sistemas de Controle Supervisório e Aquisição de Dados ou *Supervisory Control & Data Aquisition Systems*) podem ser definidos como um sistema computacional, geralmente localizado fora do chamado chão-de-fábrica, o qual possui um programa que realiza as operações de leitura e gravação de valores, os oriundos de CLP's, os quais encontram-se localizados no chão-de-fábrica (a figura 1.1 mostra os equipamentos de chão de fábrica, os CPL´s, em um nível intermediário e por fim o ambiente de rede e o acesso a Internet). Seu principal objetivo é a supervisão dos processos de produção e a aquisição de dados destes mesmos processos, através de equipamentos específicos para este fim (CLP's ou controladores lógicos programáveis). Um dos grandes diferencias dos SCADA dos sistemas anteriores é a possibilidade de alteração, durante seu funcionamento, dos parâmetros que os CLP's estão monitorando. Outra característica dos SCADA é a utilização de gráficos para a visualização das variações dos valores observados.

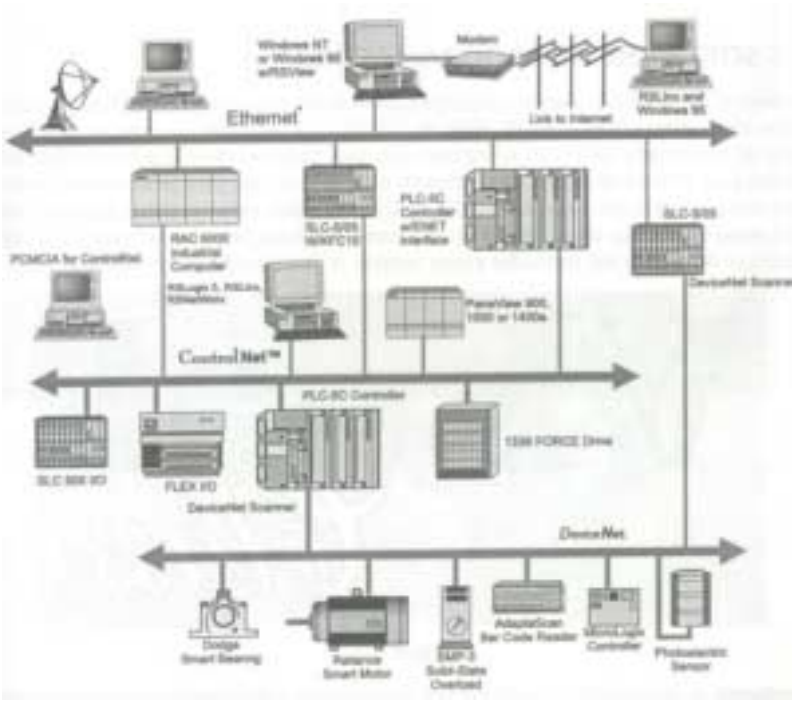

FIGURA 1.1 - Exemplo de um Sistema de Controle em Rede.

No presente estudo, que trata da geração de sistemas supervisórios a partir de modelos orientados a objeto, nos concentraremos nos sistemas do tipo SCADA ou simplesmente sistemas supervisórios, os quais são descritos na próxima seção.

1.2 Sistemas Supervisórios

São considerados sistemas supervisórios aqueles sistemas computacionais que apresentam uma estrutura de comunicação entre computadores e equipamentos de aquisição de dados, seguida da manipulação e análise destes dados e conseqüente apresentação ao usuário, em um formato previamente definido.

A principal função deste tipo de sistema é realizar a coleta de dados no chão de fábrica, permitindo a visualização destes dados através das diferentes camadas hierárquicas da organização. Outra função importante é a possibilidade de o sistema supervisório executar ações baseadas em parâmetros antecipadamente informados, fazendo com que o usuário participe apenas quando as ações exijam a intervenção humana. Em um sistema supervisório ideal, estas ações podem ser "disparadas" de qualquer um dos níveis hierárquicos, onde se encontra o usuário, bastando autorização para isto.

Geralmente, os componentes principais de um sistema supervisório são as *Tags*. As *Tags* são todas as variáveis numéricas ou alfanuméricas envolvidas na aplicação. Podem representar variáveis matemáticas, lógicas, vetores ou *strings* ou ainda representar pontos de entrada e saída de dados do processo que está sendo controlado. Neste caso, correspondem as variáveis do processo real (como temperatura, vazão, nível, etc.), sendo a ligação entre controladores industriais e o sistema. Seus tipos variam dependendo do controlador e do sistema supervisório que estão sendo utilizados. As *Tags* mais freqüentes são aquelas usadas para comunicação entre controlador e sistema e as que armazenam valores aritméticos, matrizes e textos. É com base nos valores das *Tags* que os dados coletados são apresentados para o usuário.

Um sistema supervisório deve apresentar os dados coletados pelo CLP para o usuário de uma forma simples e significativa. Isto é possível através da utilização de gráficos, botões e ícones coloridos. A apresentação destes gráficos é alterada de acordo com a variação dos valores das *Tags*, como pode ser isto na figura 1.2. Como algumas *Tags* sofrem mudanças freqüentes, é possível criar animações de figuras com a

alternância dos valores. Em contrapartida elementos gráficos associados a uma *Tag* podem ser usados como pontos de entrada de dados a serem enviados para o CLP, alterando sua configuração.

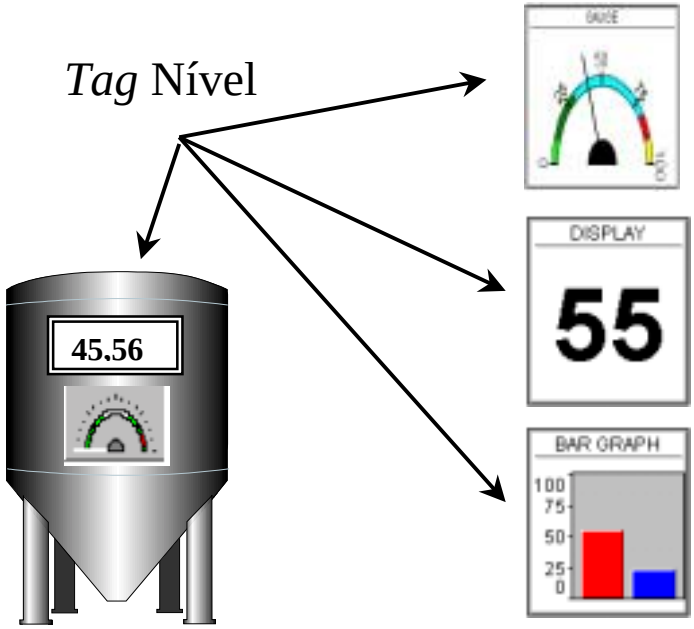

FIGURA 1.2 - Representações da "*Tag* Nível" de um Tanque.

Além da apresentação dos valores coletados, outra importante função do sistema supervisório é de monitorar uma *Tag* e informar ao usuário quando seu valor atingir certos limites (geralmente altíssimo, alto, baixo e baixíssimo ou segundo a terminologia de sistemas supervisórios comerciais *High-High, High, Low* e *Low-Low*). Enquanto o valor permanecer dentro de um intervalo que não seja crítico, as mensagens são apenas informativas, mas quando estes valores atingem um dos limites definidos, mensagens de alarmes são acionadas (na forma de texto, som ou animações). Algumas ações automáticas podem estar previstas quando da ocorrência destes alarmes, como, por exemplo, se o limite *High-High* de temperatura para uma caldeira é atingido, o sistema supervisório pode ligar um equipamento resfriador ou desligar a caldeira.

A variação dos valores das *Tags*, inicialmente visualizados, pode também ser arquivada em disco, formando um histórico das atividades do sistema. Possuir um histórico do sistema é útil ao permitir uma análise temporal dos dados na procura de eventuais problemas ou possibilidades de melhoria. A manipulação do arquivo de histórico possibilita a geração de relatórios pré-definidos, consultas de períodos específicos em arquivo e a utilização comparativa destes valores com valores atuais em gráficos de tendências por exemplo. Muito próxima da criação de históricos do sistemas

está a conexão com bancos de dados. Conectar o sistema supervisório a um banco de dados permite a recuperação de informações dos sistemas de forma mais rápida e segura, além de permitir a utilização de informações corporativas no processo que está sendo monitorado.

1.3 Modelagem Orientada a Objeto

Quando analisamos um sistema, criamos modelos da área de aplicação que nos interessa. Podemos criar um modelo do sistema como um todo ou de apenas uma parte deste.

O modelo representa um aspecto da realidade e é construído de tal forma que nos ajuda a entendê-la. O modelo é mais simples do que a realidade, uma vez que geralmente simplificações são assumidas.

Usando o paradigma de orientação a objetos, a maneira como modelamos a realidade difere da modelagem convencional, como, por exemplo, a adotada pelo método de análise estruturada (SA) [You 89]. Modelamos o mundo em termos de tipos de objetos e daquilo que acontece a esses tipos de objetos.

Segundo [Mar 95] a modelagem orientada a objetos possui muitos benefícios, dos quais destacamos os mais adequados ao nosso contexto, apresentados a seguir:

- **Reusabilidade**: classes presentes em um modelo são projetadas para que possam ser reusadas em muitos modelos de sistemas . Um repositório deve ser preenchido por uma coleção sempre crescente de classes reusáveis.
- **Estabilidade**: as classes identificadas normalmente correspondem a conceitos do domínio da aplicação, os quais tendem a ser mais estáveis que o desenvolvimento tecnológico. Isto faz com que os modelos tendam a ser válidos mesmo com atualizações tecnológicas.
- **O desenvolvedor pensa em termos de comportamento dos objetos, não em detalhes de baixo nível**: o encapsulamento oculta os detalhes e torna as classes complexas fáceis de serem usadas.
- **Classes de complexidade sempre crescente são construídas:** classes são construídas de classes, as quais são por sua vez construídas de outras classes.

- **Modelagem mais rápida:** modelos são criados mais rapidamente a partir de componentes ou modelos preexistentes.
- **Refinamento durante a construção**: a evolução constante do modelo durante sua construção é possível, levando a resultados finais muito melhores. Os trabalhos são refinados repetidas vezes, o modelo é aprimorado à medida que é implementado.
- **Modelagem mais realística**: a análise OO modela a área de aplicação de uma maneira mais próxima da realidade do que a análise convencional. Nas técnicas convencionais, o paradigma muda à medida que passamos da análise para o projeto e do projeto para a implementação. Com as técnicas OO, a análise, os projeto e a implementação usam o mesmo paradigma e o refinam sucessivamente.
- **Melhor comunicação entre profissionais de informática e pessoas da área:** As metodologias OO encorajam um melhor entendimento quando os usuários finais e os desenvolvedores compartilham um modelo comum.
- **Independência do projeto:** as classes são projetadas para serem independentes de plataformas de *hardware* e ambientes de *software*. O desenvolvedor não precisa se preocupar com o ambiente ou esperar até que ele seja especificado.
- **Interoperabilidade**: programas de muitos fornecedores diferentes podem funcionar juntos. Existe uma maneira padronizada de encontrar classes e de interagir com elas.
- **Migração:** aplicativos existentes ou que não sejam orientados a objetos podem ser preservados ao encaixá-los em um envoltório OO, de forma que a comunicação com eles seja por meio de mensagens OO padrões.
- **Melhores ferramentas CASE**: as ferramentas CASE usarão técnicas gráficas para projetar classes e suas interações e para usar objetos existentes adaptados a novos modelos.
- **Bibliotecas de classes:** as bibliotecas independentes de aplicações também são importantes e são criadas com as facilidades das ferramentas CASE. As corporações devem criar suas próprias bibliotecas de classes que reflitam seus padrões internos e necessidades de modelos.

1.4 Motivação

A criação de sistemas supervisórios complexos é sem dúvida facilitada com o uso das ferramentas comerciais disponíveis atualmente. Estas ferramentas enfatizam o ambiente de janelas e proporcionam ao projetista a possibilidade de selecionar, dentro de um conjunto de gráficos previamente definido, o modo de visualização de um determinado valor. Como descrito acima, os sistemas supervisórios são baseados em elementos chamados *Tags*, que armazenam valores em memória originários de equipamentos controladores. Cada *Tag* possui suas próprias definições dentro do sistema e cada alteração deve ser realizada para a *Tag* específica. Uma *Tag*, porém, não pode ser usada em outro sistema supervisório, pois está restrita ao sistema atual, forçando uma redefinição em um novo sistema. A criação de novos sistemas, por sua vez, deve ser realizada desde o princípio, não sendo possível a adoção de um modelo previamente definido, desprezando qualquer trabalho de modelagem do ambiente que tenha sido efetuado anteriormente.

As ferramentas de modelagem baseadas no paradigma de orientação a objeto, têm-se apresentado como uma alternativa eficiente na confecção de modelos que representem de forma significativa a complexidade do mundo real, especialmente em aplicações industriais. Estas ferramentas permitem a criação de classes e a definição de atributos para os componentes do processo de produção (como um tanque ou uma válvula) e seu instanciamento imediato, conforme a necessidade do projetista. A definição de classes com atributos comuns a uma gama de componentes de processo, pode ser facilmente definida através do uso do conceito de generalização. É possível aproveitar estas classes em sua forma original ou redefini-las criando novas classes com maior grau de especificação (tanque de água, tanque de combustível submerso, por exemplo), usando conceitos como herança e agregação. A reutilização de um modelo ou de parte deste na criação de um novo modelo permite uma economia razoável de tempo, sempre bem-vinda quando os prazos de projetos são exíguos. As ferramentas de modelagem baseadas no paradigma de orientação a objeto seguem, pela própria definição, uma metodologia, o que permite que diferentes projetistas utilizem os mesmos símbolos para representar suas idéias, tornando, assim, seu diálogo mais claro e produtivo.

Diante deste contexto, em [Tib 99] foi realizado um estudo comparativo envolvendo tanto ferramentas de modelagem baseadas no paradigma de orientação a objeto como sistemas supervisórios comerciais. Os resultados obtidos neste estudo comprovaram a necessidade de, em um mesmo ambiente, construirmos o modelo da planta industrial e a partir dele definirmos os principais parâmetros para um sistema supervisório. Por outro lado, foi identificado que é aceitável que possamos reutilizar o modelo como um todo ou apenas suas partes para a modelagem de outra(s) planta(s) ou diferentes modos de visão de uma mesmo modelo de planta industrial, dependendo do nível de proximidade ou distância de um objeto específico. Partindo de um modelo orientado a objeto, poderíamos definir no próprio modelo:

- os valores limites para cada atributo da classe;
- formas de representação gráfica para cada atributo da classe;
- formas de representação gráfica para a classe;
- formas de representação gráfica de um nível hierárquico de classes no modelo.

A definição de valores limites na classe permitiria a herança dos mesmos para todas as sub-classes desta, economizando considerável tempo de definição. A redefinição dos valores limites na instância seria possível.

Um atributo pode ter significado diferente para diferentes pessoas em diferentes níveis organizacionais. O operador do equipamento pode estar muito interessado na variação da temperatura de um forno, enquanto o supervisor tem interesse somente quando a temperatura ultrapassar os limites inferior e superior. Teremos, então, para um mesmo atributo, várias visões deste atributo, persistindo seu valor e alterando a forma de apresentação.

As representações de classe e nível de classes são interessantes a partir do momento que permitiriam uma navegação não seqüencial no modelo, a criação de várias visões para um mesmo elemento e, ao reaproveitar a classe ou o nível em outro modelo, todas as definições existentes seriam utilizadas, evitando sua redefinição.

A realização do trabalho citado motivou o desenvolvimento desta dissertação, cujo principal objetivo é a geração de sistemas supervisórios a partir de modelos orientados a objeto.

1.5 Objetivos do Trabalho

O principal objetivo deste trabalho é a geração de sistemas supervisórios a partir de modelos orientados a objetos. Dentre as vantagens que se espera alcançar, destacam-se: o aumento da reusabilidade destes modelos e maior facilidade de extensão. Outro objetivo é a possibilidade da geração, a partir do modelo, de uma estrutura que possa ser utilizada em sistemas supervisórios de terceiros, externos a ferramenta de modelagem SIMOO-RT, o qual será detalhado no item 2.2.1.

1.6 Contribuição do Autor

Como contribuições do autor podemos citar uma proposta de como integrar o desenvolvimento de sistemas supervisórios com ferramentas de modelagem orientadas a objeto. Isto foi feito identificado-se os conceitos presentes nas ferramentas e definindo-se relacionamentos possíveis entre estes conceitos.

1.7 Organização do Volume

Primeiramente, no Capítulo 2, descrevem-se as características principais de sistemas supervisórios e de ferramentas de modelagem orientadas a objeto. Para facilitar a compreensão dos conceitos descritos as características dos sistemas são baseadas nas ferramentas Elipse [Eli 98] e SIMOO-RT [Cop 96,Bec 99], as quais serão usadas freqüentemente ao longo do trabalho. Cabe destacar que estas ferramentas representam bem os modelos atualmente utilizados comercialmente, ou seja, sua estrutura e comportamento são semelhantes a outras ferramentas comerciais.

O Capítulo 3 descreve o projeto conceitual do modelo e conceitua seus componentes básicos: Visão, Elementos de Visualização e Referência. Apresenta, ainda, um exemplo aonde são aplicados os conceitos acima.

O Capítulo 4 mostra um estudo de caso, exibindo as funcionalidades da ferramenta.

O Capítulo 5 expõe as conclusões alcançadas com o desenvolvimento deste trabalho e sugestões para continuidade do mesmo.

2 Estado da Arte

Neste capítulo são detalhadas algumas características sobre os sistemas supervisórios e são apresentadas algumas ferramentas para seu desenvolvimento, em especial o programa Elipse Windows. Também são descritas algumas ferramentas de modelagem orientada a objeto, especificamente o SIMOO-RT.

2.1 Ferramenta para Sistema Supervisório

Na escolha de uma ferramenta para o desenvolvimento de um sistema supervisório devemos levar em consideração a estrutura física com a qual estará envolvido.

Geralmente, a estrutura física que cerca um sistema supervisório apresenta três camadas, como visto na Figura 1.1. Na primeira situam-se os equipamentos industriais (motores, sensores, leitores de código de barras, etc.). Esta camada é caracterizada pela conexão dos equipamentos aos CLP's (Controladores Lógicos Programáveis) através de meios e protocolos específicos. Esta comunicação é feita através de protocolos de comunicação de baixo nível e exige o cumprimento de certos requisitos para ser efetivada com sucesso. Temos como exemplo desta camada a comunicação usando PPI ou MPI, Modbus, Fieldbus, Profibus, entre outros.

Na segunda camada ocorre a comunicação entre o(s) CLP(s) e o microcomputador, no qual o sistema supervisório está sendo executado. Neste ponto, os dados são agrupados por equipamento pelo CLP, havendo identificação inequívoca do CLP e de seus dados. O CLP possui memória para armazenamento intermediário de um certo número de operações realizadas pelo equipamento, o que permite que a comunicação com o microcomputador possa ser retomada após uma parada momentânea. A comunicação entre ambos normalmente é feita de forma serial ou, mais recentemente, através de placas *Ethernet*. Existe nesta camada uma relação direta do tipo de CLP com o sistema supervisório. A comunicação perfeita entre ambos vai depender da utilização do driver específico para o CLP pelo sistema supervisório. Serão necessárias configurações adequadas em ambos para que os dados recebidos pelo CLP e transmitidos para o supervisório possam ser corretamente interpretados. Endereçamento de posição de memória, tamanho dos bytes, modos de leitura e escrita do CLP,

comunicação com outras aplicações entre outras configurações são necessárias para que CLP e microcomputador funcionem corretamente.

Na terceira camada se situam o microcomputador com o sistema supervisório, os demais sistemas da indústria (administrativos, de fornecedores, de parceiros, etc.) e o acesso externo à organização. A interconexão do sistema supervisório com outros sistemas, nesta camada, é possível, por exemplo, através do uso de redes *Ethernet* e do protocolo TCP/IP. O supervisório deve estar preparado para a comunicação com outros sistemas supervisórios, quer sejam locais ou remotos, permitindo assim a visualização de dados ou atuação de usuários sobre a linha de produção, mesmo que este não esteja física e localmente presente. Em nosso estudo vamos nos ater ao sistema supervisório, ao programa enfim, abstraindo a estrutura básica e a comunicação entre seus componentes.

A partir da estrutura física é possível definir os pontos de conexão entre os dispositivos e o sistema supervisório. Como descrito anteriormente, estes pontos são denominados *Tags* na terminologia de sistemas supervisórios. Elas são as variáveis definidas na aplicação supervisória e possuem ligação com os pontos de entrada e saída do CLP que está monitorando o processo controlado. Podem ser de vários tipos: numéricos, alfanuméricos, *strings*, posições de memória, aritméticas, etc..

A simples possibilidade de associação de *Tags* com portas de I/O do CLP não é suficiente para suportar a complexidade das aplicações dos sistemas supervisórios. A necessidade de programar uma ou um conjunto de ações para uma ou mais situações ocorridas no processo é de muita utilidade dentro do monitoramento e controle do mesmo. Muitas destas ações podem ser realizadas pelo sistema supervisório, através da execução de uma linguagem de programação, que é embutida no próprio programa supervisório e conhecida como *Scripts*. Esta linguagem geralmente é proprietária, mas segue um conjunto de comandos de uma linguagem de programação conhecida, como a *Basic* ou a C. Os *Scripts* permitem uma flexibilidade muito grande aos supervisórios, pois possibilitam maior proximidade com operações de baixo nível e acesso a dispositivos e sistema operacional.

Existe comercialmente, um grande número de programas para a criação de sistemas supervisórios, dentre eles podemos citar:

- **UniSoft**: é uma ferramenta desenvolvida pela empresa brasileira Indusoft Ltda [Ind 00]. É utilizada para o desenvolvimento de interfaces homem-máquina, como estação de supervisão local de processos industriais e como estação concentradora de dados em processos distribuídos. É baseado em um microcomputador interligado a um processo ou máquina através de um controlador lógico programável, estação remota ou outro equipamento de aquisição de dados.
- **LookOut**: é um programa de automação industrial para a criação de interfaces homem-máquina e SCADA, produzida pela empresa americana National Instruments - NI [NAT 99A]. A empresa apresenta o LookOut como uma arquitetura baseada em objetos, no qual é possível criar objetos e conectá-los entre si sem a necessidade de programação, compilação ou *scripts*. Sua utilização é recomendada para sistemas de telemetria, sistemas de controle de processos contínuos e discretos e em processos batch.
- **BridgeVIEW**: é um ambiente gráfico para desenvolvimento de aplicações de automação. Também produzido pela National Instruments, possui a linguagem de programação gráfica G para a criação de supervisórios padrão [NAT 99B]. O BridgeVIEW é dirigido para criação de aplicações relacionadas com equipamentos de controle de entrada e saída de processos, tais como câmeras de vídeo, equipamentos de aquisição de dados e controladores diretos de processo.
- **Genesis32**: é uma família de aplicações cliente/servidor, fabricada pela empresa ICONICS [ICO 98]. O Genesis32 utiliza o padrão OPC (Object Linking and Embedding for Process Control) para o desenvolvimento de aplicações para MS-Windows 9X e NT [ICO 98]. Os clientes possuem características diferenciadas: o GraphWorkX32 é um conjunto de ferramentas de desenho e animações gráficas para criação de HMI; o TrendWorkX32 cria vários tipos de gráficos de tendências a partir de um histórico de dados; o AlarmWorkX32 permite a configuração de alarmes para pontos críticos do processo usando voz, e-mail e telefone, além de gerar relatórios, tabelas e histogramas; o ControlWorkX32 consiste em um *kernel* independente para programação nas linguagens compatíveis com a norma IEC 1131-3 [ICO 98].

2.1.1 Elipse Windows

O Elipse Windows é um programa de supervisão de processos comercializado pela empresa brasileira e gaúcha Elipse Software Ltda [Eli 98]. Ele permite monitorar variáveis de processo em tempo real, através de gráficos e objetos animados como barras, tendências, mostradores, medidores, etc. É possível fazer acionamentos e enviar ou receber informações para os equipamentos de aquisição de dados (CLP) através de pontos de acionamento, réguas e botões de acionamento [Eli 98]. A ligação de *Tags* de entrada e saída ao CLP podem ser feitas através de dois tipos: *Tag* PLC (onde é possível associar uma variável a um CLP) e *Tag* Bloco PLC (onde é possível associar um conjunto de variáveis a um equipamento externo).

2.1.1.1 Componentes do Elipse Windows

A versão do Elipse Windows aqui utilizada foi a 2.0 SP1, a qual apresenta um conjunto de componentes que facilitam a criação de aplicativos de supervisão, como veremos a seguir.

- *Organizer*

O *Organizer* permite uma visão global de todo o aplicativo, em forma de árvore hierárquica de opções, facilitando as edições e configurações de todos os objetos envolvidos no sistema, como mostrado na Figura 2.1.

A idéia do Organizer pode ser comparada à organização de arquivos no formato da árvore de diretórios. Neste caso, o aplicativo propriamente dito seria o diretório raiz do sistema. A partir do aplicativo o usuário pode visualizar os demais componentes: *Tags* (Etiquetas), *Screens* (Telas), *Alarms* (Alarmes), *Scripts* (Manuscritos), *Recipes* (Receitas), *Historics* (Históricos) e *Reports* (Relatórios), que são os componentes fundamentais de um aplicativo de supervisão e serão descritos no decorrer deste capítulo. Clicando sobre as opções dessa árvore o usuário vai navegando no aplicativo de maneira que todas as opções ficam disponíveis, desde a criação de Tags até a alteração do tamanho de um objeto numa tela específica.

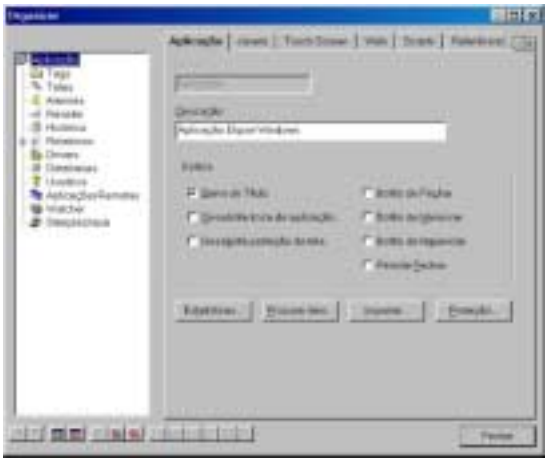

FIGURA 2.1 - Exemplo da Tela do Organizer.

- *Tipos* de Tags

Como descrito no item 1.2, as *Tags* são todas as informação que o sistema irá receber e enviar junto a equipamentos de aquisição de dados (CLP). Devido a sua importância na composição do sistema, veremos nesta seção os tipo de *Tags* disponíveis no Elipse Windows.

As *Tags* são representadas por uma pasta no *Organizer*, dentro da qual podemos criar várias outras *Tags* ou grupos de *Tags.* Os tipos de *Tags* presentes no Elipse Windows são:

- **Crono**: cria um novo cronômetro (permitindo realizar ações em tempos específicos);
- **Bloco PLC**: cria um bloco de variáveis conectadas a um equipamento externo (PLC);
- **DDE**: cria uma variável que permite trocar dados com outros programas utilizando DDE;
- **Demo**: cria uma variável que simula variações de valor dentro da aplicação;
- **Expressão**: cria uma variável com valor calculado;
- **Matriz**: cria uma matriz para armazenar tabelas de dados auxiliares.
- **RAM**: cria uma variável para armazenar valores auxiliares.

Para melhor entendermos a criação de uma *Tag*, exemplificaremos com a *Tags* do tipo Demo e RAM, as quais serão descritas a seguir.

- *Tag Demo*

As *Tags* Demo permitem gerar curvas definidas ou valores randômicos conforme o tipo de curva escolhida. Em *Tags* Demo podemos ajustar as opções Limite Inferior (valor mínimo para o *Tag*), Limite Máximo (valor máximo), Incremento (incremento quando uma curva dente de serra estiver selecionada), Espera (número de períodos entre cada geração de valor para a *Tag* Demo - se for 1, um valor é gerado a cada período, se for 2, gera um valor a cada dois períodos), Período (define o número de milisegundos entre a geração de cada novo valor para a *Tag* Demo, usado junto com o atributo Espera para controlar o intervalo de tempo para a variação dos dados).

FIGURA 2.2- *Tag* do Tipo Demo.

- *Tag RAM*

As Tags do tipo RAM são de utilização interna, para guardar valores em memória. Estes valores são voláteis, ou seja, são guardados enquanto o aplicativo Elipse estiver executando.

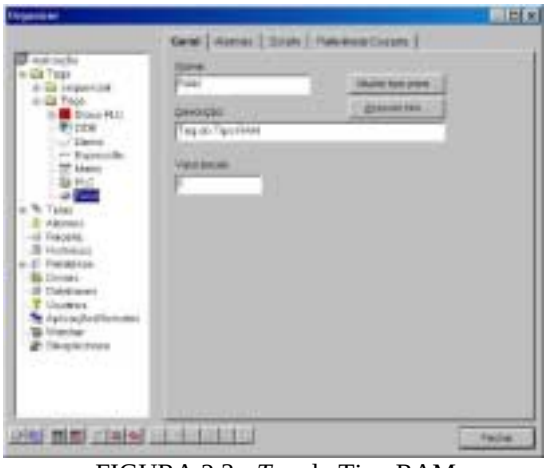

FIGURA 2.3 - *Tag* do Tipo RAM.

- *Página de alarmes*

Cada *Tag* definido possui uma página de Alarmes (discutidos na seção 1.2), onde podem ser configurados quatro intervalos e prioridades para alarmes, como pode ser visto na figura 2.4. Alarmes são usados para sinalizar algum problema e então tomar as ações apropriadas usando *Scripts*.

Para visualizar os Alarmes configurados para um *Tag*, é necessário criar um objeto de tela e atribuir os alarmes do *Tag* a ele. Este objeto pode mostrar também alarmes já ocorridos que estejam registrados em um arquivo de Históricos (arquivos aonde são armazenados dados de processos para análises futuras).

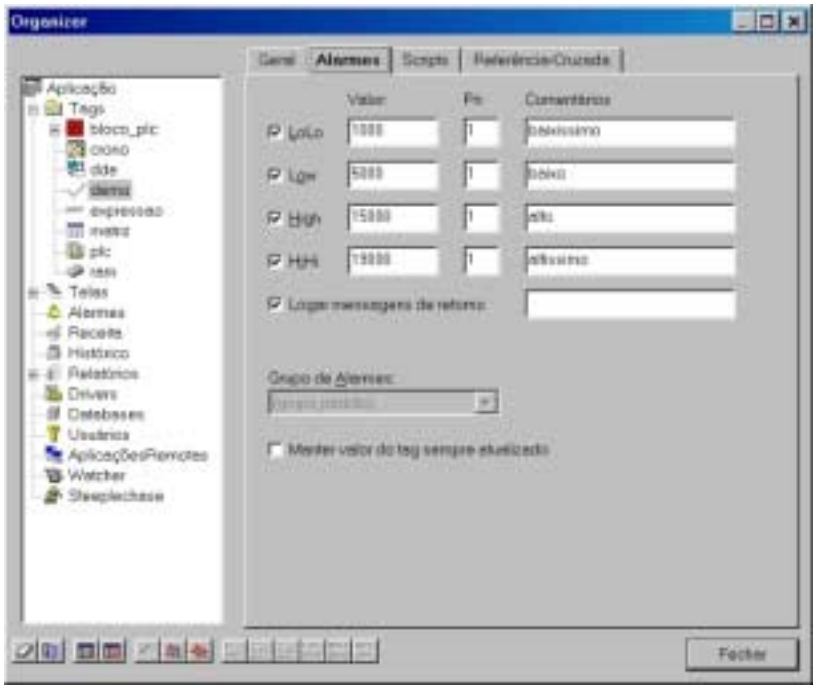

FIGURA 2.4 - A Página de Alarmes do *Tag* Demo.

No Elipse Windows estão disponíveis os seguintes alarmes:

- **LoLo** – Alarme baixo crítico. É usado quando o valor do Tag está abaixo de um mínimo, ou seja, extremamente baixo.
- **Low** – Alarme baixo. É usado quando o valor do Tag está abaixo do normal.
- **High** – Alarme alto. É usado quando o valor do Tag está mais alto do que o normal.
- **HiHi** – Alarme alto crítico. É usado quando o valor do Tag está acima de um máximo, ou seja, extremamente alto.
- **Valor** – define os limites para cada situação possível nos alarmes acima.

- **Prioridade** – define a prioridade para cada situação de alarme. Números pequenos indicam alta prioridade.

- *Página de Scripts*

Cada *Tag* definido no Elipse também possui uma página de *Scripts* (descritos na seção 1.2), aonde podem ser editados programas para o *Tag* usando a linguagem Elipse Basic. *Scripts* são usados para criar ações lógicas e executá-las quando um evento específico ocorrer, como mostrado na figura 2.5.

Scripts de *Tags* normalmente estão associados ao valor do *Tag*, significando que eles podem ser executados quando o valor do *Tag* mudar ou estiver em situação de alarme. *Scripts* podem ser executados nas seguintes situações:

- **OnAck** – executa o *Script* quando o alarme for sinalizado como reconhecido pelo usuário.
- **OnAlarmHigh** – executa o *Script* quando o valor definido para Alarme alto (*High*) for alcançado.
- **OnAlarmHiHi** – executa o *Script* quando o valor definido para Alarme para Alarme alto crítico (*HiHi*) for alcançado.
- **OnAlarmLow** – executa o *Script* quando o valor definido para Alarme baixo (*Low*) for alcançado.
- **OnAlarmLoLo** – executa o *Script* quando o valor definido para Alarme baixo crítico (*LoLo*) for alcançado.
- **OnAlarmReturn** – executa o *Script* assim que a situação de alarme tiver terminado.
- **OnChangeValue** – executa o *Script* quando o valor do *Tag* mudar.

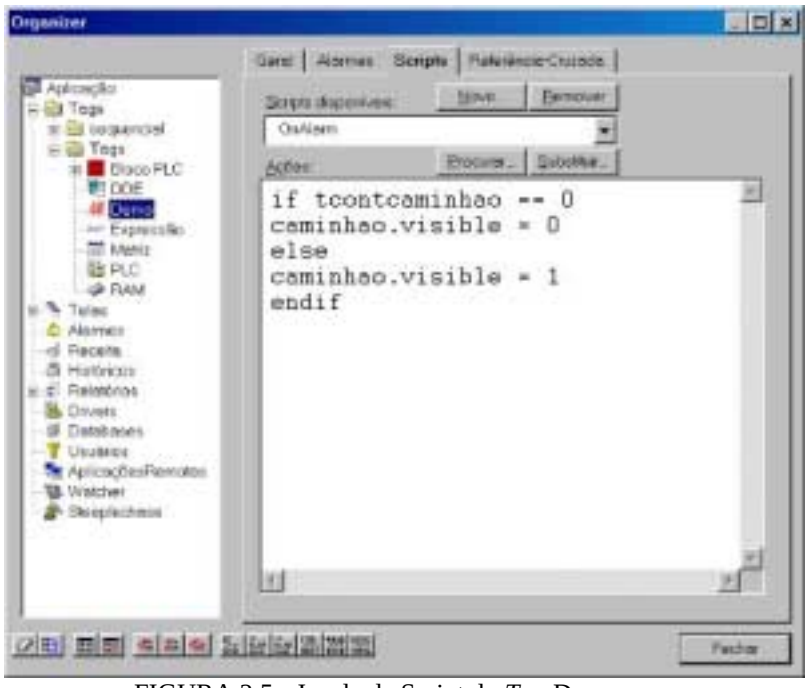

FIGURA 2.5 - Janela de Script do *Tag* Demo.

2.1.2 Análise do Sistema Supervisório

Como descrito anteriormente, as *Tags* são o conceito fundamental nas aplicações de sistemas supervisórios, o Elipse Windows em nosso estudo, e realizam a comunicação com equipamentos de aquisição de dados, o que permite que conectemos nossa aplicação com o "chão de fábrica". Deste modo o Elipse mostrou pontos fortes: uma interface gráfica flexível, facilidade para criação de elementos gráficos, capacidade de comunicação com a grande maioria do CLP's disponíveis comercialmente, possibilidade de conexão com outros aplicativos Windows através do DDE, inclusive utilizando os recursos de rede e mais recentemente da Internet, comunicação com o "chão de fábrica" por meio dos *Tags* e a capacidade de resposta a eventos por meio da Linguagem Elipse *Basic* nos *Scripts* – disponíveis em todos os objetos da aplicação.

Apesar disto, o Elipse Windows apresenta, inicialmente, a desvantagem de não ser portável. É impossível portar suas aplicações para sistemas não-Windows, devido à não geração de estrutura de classes em uma linguagem portável (como Java, por exemplo) o que impede sua utilização em uma gama de computadores que rodam sistemas operacionais mais robustos ou mais específicos para plantas industriais.

Outra desvantagem do Elipse e também da quase totalidade dos sistemas supervisórios comerciais disponíveis é seu enfoque centrado na *Tag.* Para supervisionar uma planta industrial, é necessário que variáveis lógicas estejam diretamente ligadas a

pontos de entrada e saída dos equipamentos da mesma. Além disso as *Tags*, ao serem criadas, se tornam independentes. Não há o conceito de classe de *Tags*, o que impede o instanciamento de uma delas, permitindo que uma definição prévia possa ser herdada pelas diversas instâncias. Não é possível a agregação de duas ou mais *Tags* objetivando formar uma nova *Tag*.

Não existe ainda vínculo com os valores definidos na Tag com os valores apresentados em um elemento gráfico, o que causa um desconforto e trabalho dobrado. Podemos tomar como exemplo um tanque no qual queremos monitorar pressão, nível e temperatura. No Elipse, é necessário criar três *Tags*, que são componentes diferentes e não possuem relação hierárquica ou de herança. O problema surge quando, criado este primeiro tanque, surge a necessidade de criarmos mais dois tanques. Teremos, então, criado 9 *Tags* diferentes para três tanques, que também são elementos diferentes, como podemos observar na figura 2.6.

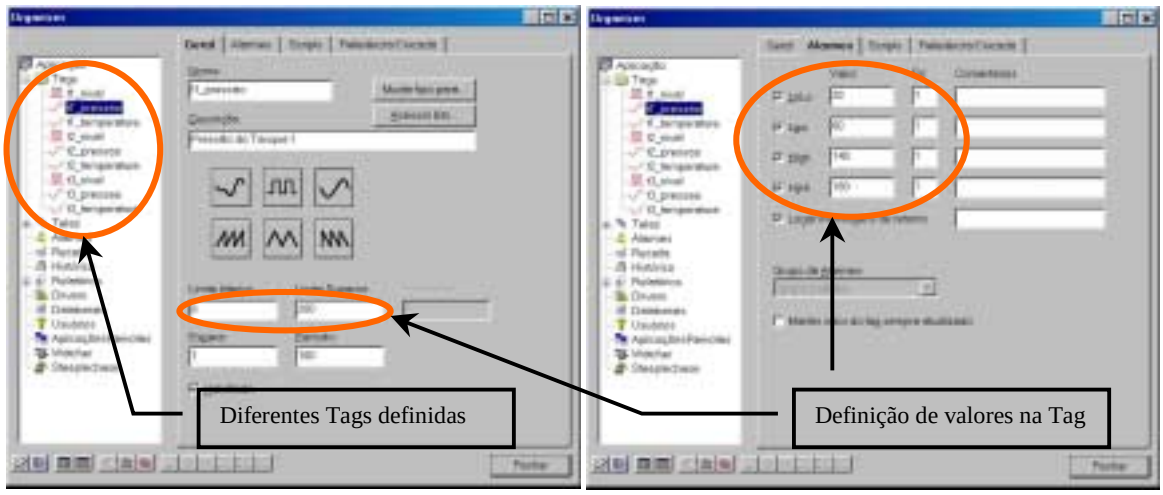

FIGURA 2.6 - Criação de Diferentes Tags e Definição seus Valores

Ao definirmos uma representação gráfica para os valores destas *Tags* constatamos que os valores definidos para a Tag não são automaticamente incorporados no elemento gráfico, como podemos perceber na figura 2.7.

Pode-se perceber, ainda, uma desvantagem muito forte: no momento da alteração do valor de um dos alarmes, (LoLo por exemplo), é notória a necessidade da alteração ser efetivada em todas as *Tags* individualmente. Vemos na figura 2.8 os nove elementos gráficos que foram ligados com as nove *Tags*. Estes devem ter seus valores para o alarme LoLo alterados individualmente, para a respectiva *Tag*.

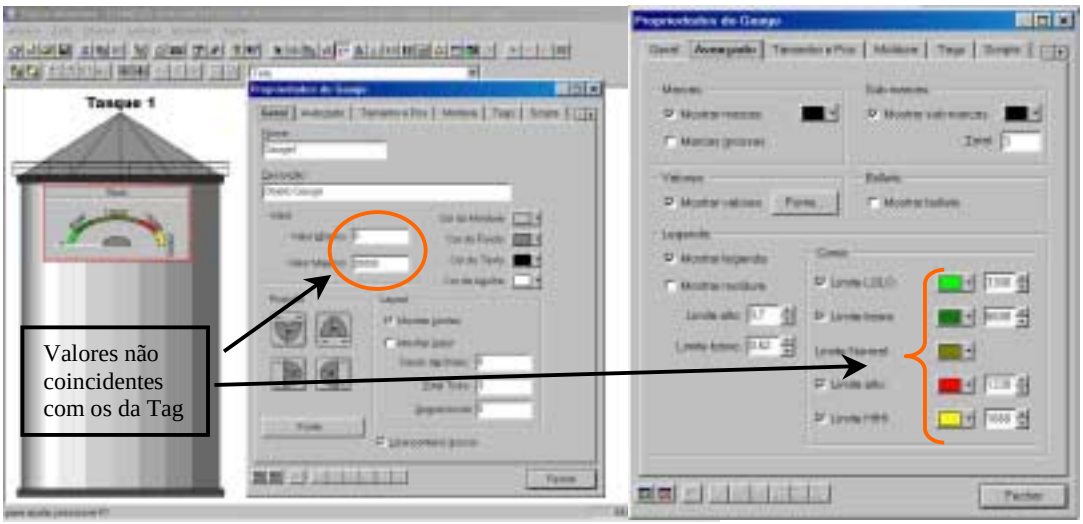

FIGURA 2.7 - Diferença dos Valores do Elemento Gráfico e da *Tag*.

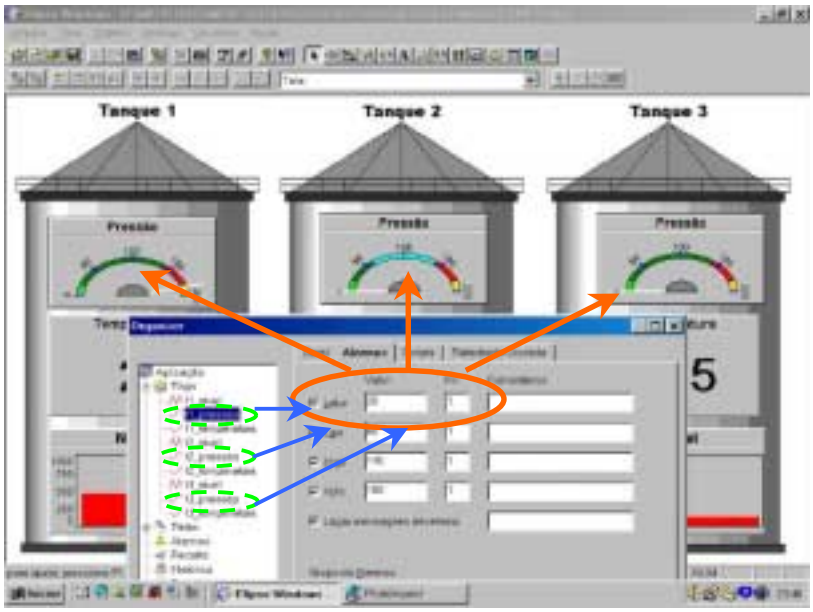

FIGURA 2.8 - Alteração do Alarme LoLo para as *Tags* Pressao.

2.2 Ferramentas de Modelagem Orientadas a Objeto

Nesta seção são descritas sucintamente algumas ferramentas de modelagem baseadas no paradigma de orientação a objeto.

Estas ferramentas, sendo algumas do tipo CASE (*Computer Aided Software Engineering*), permitem ao usuário criar e refinar modelos menores dentro de um modelo global, o qual representa o domínio do problema. O modelo global contém classes, categorias de classes, objetos, operações, subsistemas, módulos, dispositivos e os relacionamentos entre os mesmos. Cada um destes componentes possui propriedades

para identificá-los e caracterizá-los. A notação da ferramenta oferece ícones para representar graficamente cada tipo de componente e de relacionamento.

Geralmente, é possível utilizar um gerador de código, com o qual pode-se produzir código fonte a partir das informações contidas em um determinado modelo, em uma linguagem de programação suportada pela ferramenta.

O código gerado para cada componente do modelo selecionado é função da especificação do mesmo e das suas propriedades de geração de código, bem como das propriedades do modelo. Estas propriedades fornecem as informações específicas da linguagem necessárias para mapear o modelo para código alvo.

Neste trabalho nos concentramos particularmente na ferramenta SIMOO-RT, mas dentre as diversas ferramentas disponíveis comercial ou academicamente, podemos destacar as seguintes:

- **Rational Rose**: é uma ferramenta CASE projetada e distribuída pela empresa americana Rational Software Inc [Rat 99], segue a metodologia UML (*Unified Modeling Language*) [Dou 98, Dou 99]. A UML possibilita a definição de diferentes diagramas para representar as características estáticas e dinâmicas de um modelo lógico e um modelo físico para representar os componentes definidos na análise e no projeto orientados a objeto.

  A interface gráfica do Rational Rose permite visualizar, criar, modificar e manipular os componentes do modelo. Esta interface é composta por três tipos de janelas: janela da aplicação, janelas dos diagramas, janelas de especificações. A ferramenta permite ao usuário controlar quais ícones de componentes, relacionamentos e propriedades aparecem em cada diagrama, utilizando os recursos oferecidos pela janela de aplicação.

  A barra de ferramentas de diagramas mostrada ao longo do lado esquerdo da janela exibe as ferramentas representadas por ícones, que podem ser adicionados ao diagrama corrente. Para cada tipo de diagrama um conjunto de ferramentas apropriado é exibido, de acordo com a sua natureza. Os diagramas existentes são os seguintes : Diagrama de Classes (representa as descrições genéricas do sistema), Diagrama de Objetos (mostra os objetos do sistema, suas ligações e as suas mensagens), Diagrama de Componentes (apresenta os relacionamentos entre os subsistemas e os módulos que constituem o modelo), Diagrama de

Estados (relaciona os possíveis estados para uma classe dada, os eventos que causam a transição de um estado para outro e as ações que resultam de uma mudança de estado) e Diagrama de Distribuição (contém os processadores, os dispositivos e as conexões que formam o modelo).

Utilizando-se o gerador de código da Rational Rose, é possível produzir código fonte nas linguagens C++, Java e MS-Visual Basic a partir das informações contidas em um determinado modelo.

- **ObjecTime**: é uma ferramenta CASE produzida pela empresa canadense ObjecTime Ltd [Obj 99]. Basicamente, a ferramenta suporta a metodologia denominada ROOM (*Real-Time Object-Oriented Modeling*) [Sel 94].

  A ferramenta ObjecTime pode ser usada para criar um modelo orientado a objeto e executável de um sistema de tempo-real e suporta estratégias de construção de modelos comumente usadas. O componente primário de um modelo ObjecTime é uma classe ator, que representa um conjunto de objetos, usando-se a definição abrangente de um objeto como uma máquina lógica de escopo arbitrário. Uma interface de ator é definida em termos de portas e protocolos, os quais representam conjuntos de mensagens relacionadas.

  Atores possuem um comportamento, o qual é descrito por uma máquina de estado chamada diagrama *ROOM-Charts* e por variáveis de estado estendidas definidas em termos de classes de dados. Um ator ObjecTime pode possuir uma estrutura interna complexa arbitrária, definida por referências para outras classes de atores cujos atores são seus componentes. O comportamento de um ator pode também ser descrito hierarquicamente definindo-se subestados a partir de estados. A ferramenta incorpora uma linguagem de programação convencional. Citações na sintaxe desta linguagem são usadas para representar detalhes de baixo nível tais como passagem de mensagens e manipulação de valores de dados. ObjecTime oferece implementação de herança. Em particular, todas as propriedades de uma classe ator complexa podem ser herdadas por suas subclasses.

2.2.1 SIMOO

O SIMOO foi desenvolvido no âmbito da uma tese de Doutorado no Instituto de Informática da Universidade Federal do Rio Grande do Sul [Cop 96]. A partir de sua

definição várias extensões foram sendo adicionadas à ferramenta, especialmente uma extensão para tempo-real - o SIMOO-RT, resultado de uma Dissertação de Mestrado no mesmo Instituto [Bec 99].

A ferramenta SIMOO tem como objetivo a criação de modelos orientados a objetos, que possuem recursos para permitir uma interação plena entre o usuário e o modelo.

O SIMOO adota uma configuração de hierarquia que permite o modelo ser detalhado por níveis conforme a necessidade. O uso de elementos de interface e dos monitores de visualização permitem ao usuário manter a separação entre o domínio do modelo e os recursos de visualização e resultados obtidos.

Os modelos que são construídos na ferramenta editora de modelos (chamada MET – *Model Editor Tool*) são compostos por vários tipos de diagramas: o diagrama de classes (usado para descrever os diversos tipos de hierarquias do modelo) e o diagrama de instâncias (usado para descrever "exemplos" das classes e especificar as relações entre elas), o diagrama de cenários, o diagrama de fluxo de dados e o *Deployment Diagram* , os três últimos, específicos do SIMOO-RT.

O editor suporta a descrição de aspectos estáticos e dinâmicos do modelo. Os aspectos estruturais são descritos graficamente, enquanto que o comportamento dinâmico de cada entidade é descrito na linguagem C++ usando-se recursos de biblioteca. A partir da descrição do modelo, o editor gera automaticamente o código executável necessário. Também possibilita a construção de uma biblioteca de elementos autônomos os quais evoluem à medida que novos modelos são desenvolvidos.

No nível mais alto de abstração há uma única entidade representando todo o sistema sob modelagem. Esta entidade é posteriormente refinada em níveis de abstração mais inferiores, construindo, portanto, uma hierarquia de agregação. Tal estratégia de modelagem tem como objetivo uma reutilização mais fácil de entidades, assim também como um refinamento progressivo, quando necessário.

O SIMOO também utiliza o conceito de meta-classes e meta-objetos, especialmente para visualização. Meta-classes descrevem a estrutura de uma classe enquanto que meta-objetos concentram-se nos aspectos individuais de cada objeto.

2.2.1.1 O Diagrama de Classes

O Diagrama de Classes é construído de maneira hierárquica. Diferentes tipos de hierarquia podem ser identificados: agregação, herança e refinamento. Neste diagrama, classes podem ser criadas e descritas assim como também outros elementos específicos. A figura 2.9 mostra um típico diagrama de classes.

A ferramenta editora de modelos também permite ao usuário definir como as classes irão se comunicar entre si. A comunicação entre classes é feita através do uso de pontos de conexão. Há dois tipos de pontos de conexão, de acordo com a comunicação que é empregada. O primeiro tipo é através de mensagens que usam os pontos de conexão de mensagens. O segundo tipo é através de portas de entrada e de saída.

Neste diagrama uma relação entre duas classes significa que uma classe é capaz de comunicar-se com outra classe, mas isso não significa necessariamente que todas as instâncias destas classes devam possuir uma relação de comunicação.

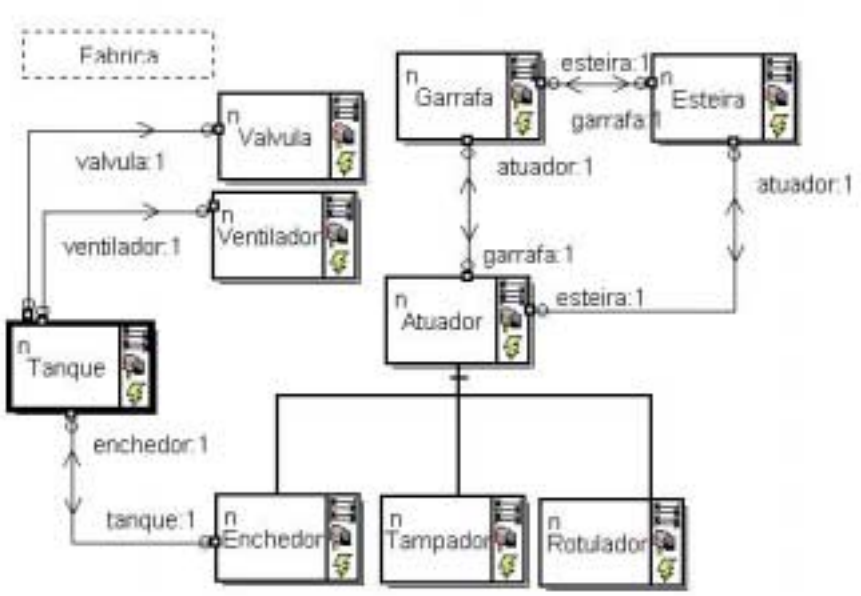

FIGURA 2.9 - Diagrama de Classes do SIMOO.

2.2.1.2 O Diagrama de Instâncias

O Diagrama de Instâncias é usado para instanciar os elementos genéricos descritos no diagrama de classes e também para definir a comunicação entre eles. Uma característica especial do SIMOO é que o usuário deve necessariamente criar um diagrama de instâncias a partir do qual o modelo de simulação executável e o código da aplicação serão gerados. A necessidade de um diagrama de instâncias é baseado no fato

de que entidades de simulação concretas correspondem a objetos, e não a classes. A figura 2.10 mostra um exemplo de diagrama de instâncias.

Cada retângulo mostra um elemento específico que foi descrito genericamente no diagrama de classes. Pode haver tantas instâncias da mesma classe quanto necessárias para construir o modelo, também como conjuntos de elementos (arranjos) do mesmo tipo de classe. As relações feitas neste diagrama indicam uma comunicação entre dois ou mais elementos, diferentemente do diagrama de classes onde uma relação entre classes significa que uma classe conhece outra classe, ou seja, que é possível haver comunicação entre elas.

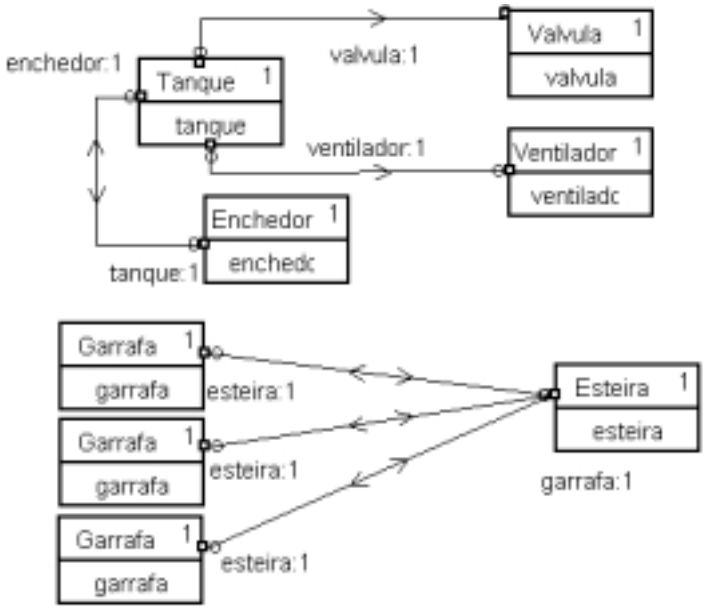

FIGURA 2.10 - Diagrama de Instâncias no SIMOO.

2.2.1.3 SIMOO-RT e SIMOO

O SIMOO-RT é uma extensão do SIMOO que possibilita criar cenários adicionais aos do SIMOO (como diagrama de cenários, diagrama de fluxo de dados e *Deployment Diagram)*. Esta extensão adiciona propriedades ao SIMOO que permitem a sua utilização como plataforma para implementação de sistemas de tempo real distribuídos (STRD), especialmente aqueles voltados para automação industrial. Dentre estas propriedades, destacam-se as facilidades para especificação de restrições temporais, o suporte para descrição de comportamento do modelo através de máquinas de estados e diagrama de eventos e capacidade de fazer geração automática de código.

3 Projeto Conceitual da Ferramenta *Supervisory Designer*

O presente capítulo descreverá o modelo conceitual da proposta de integração entre modelos orientados a objeto e sistemas supervisórios, o qual servirá de base para construção da ferramenta *Supervisory Designer*.

A fim de facilitar a explanação dos diferentes conceitos utilizar-se-á o exemplo de um automóvel e suas diferentes visões por parte do motorista e do mecânico.

3.1 O Modelo Conceitual

A Figura 3.1 apresenta o diagrama de classes do modelo conceitual da ferramenta *Supervisory Designer*, a qual consiste de uma proposta de unificação entre sistemas de modelagem orientados a objeto e sistemas supervisórios. Conforme pode ser visto no lado direito da figura 3.1, utiliza-se de conceitos usuais de orientação a objeto, tais como classes, atributos, estados e métodos. A classe "Classe" representa uma classe da estrutura do SIMOO que está sendo manipulada. Uma "Classe" é uma agregação (de zero ou mais) atributos, estados ou métodos, os quais são representados pelas classes "Atributo", "Estado" e "Método", respectivamente. Cada "Classe" possui uma ou mais visões associadas (as quais são representadas pela classe "Visão", sendo que cada "Visão" referencia uma ou mais classes). Uma "Visão" é composta por "Elemento de Visualização" e "Referência". "Elementos de Visualização" são representações gráficas de "Estado" ou "Atributos" de uma classe. "Referência" são "associações" entre visões ou seja, uma visão pode ser construída de forma hierárquica, assim visões podem referenciar outras visões.

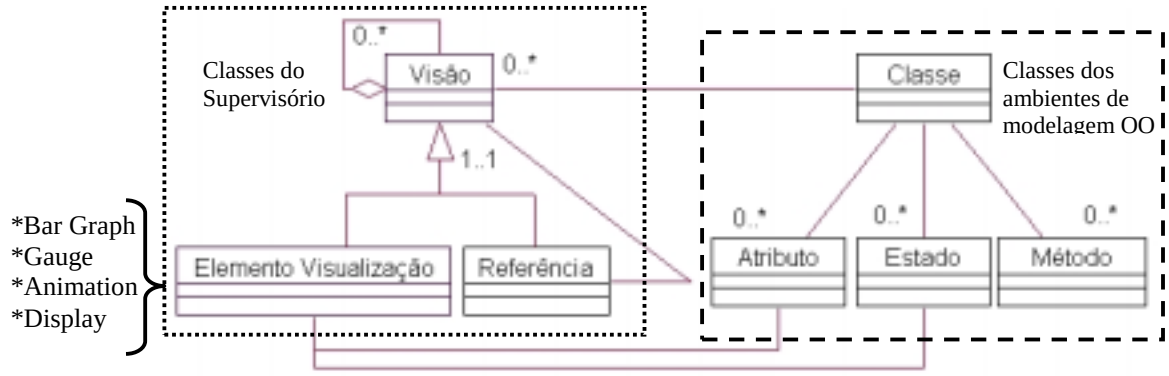

FIGURA 3.1 - O Diagrama de Classes do Modelo Conceitual do *Supersory Designer*.

3.2 Exemplo de Visões: Automóvel

Para concretizar e facilitar a compreensão dos conceitos apresentados, consideramos aqui uma aplicação simples de um automóvel, cujo modelo orientado a objeto é apresentado na figura 3.2.

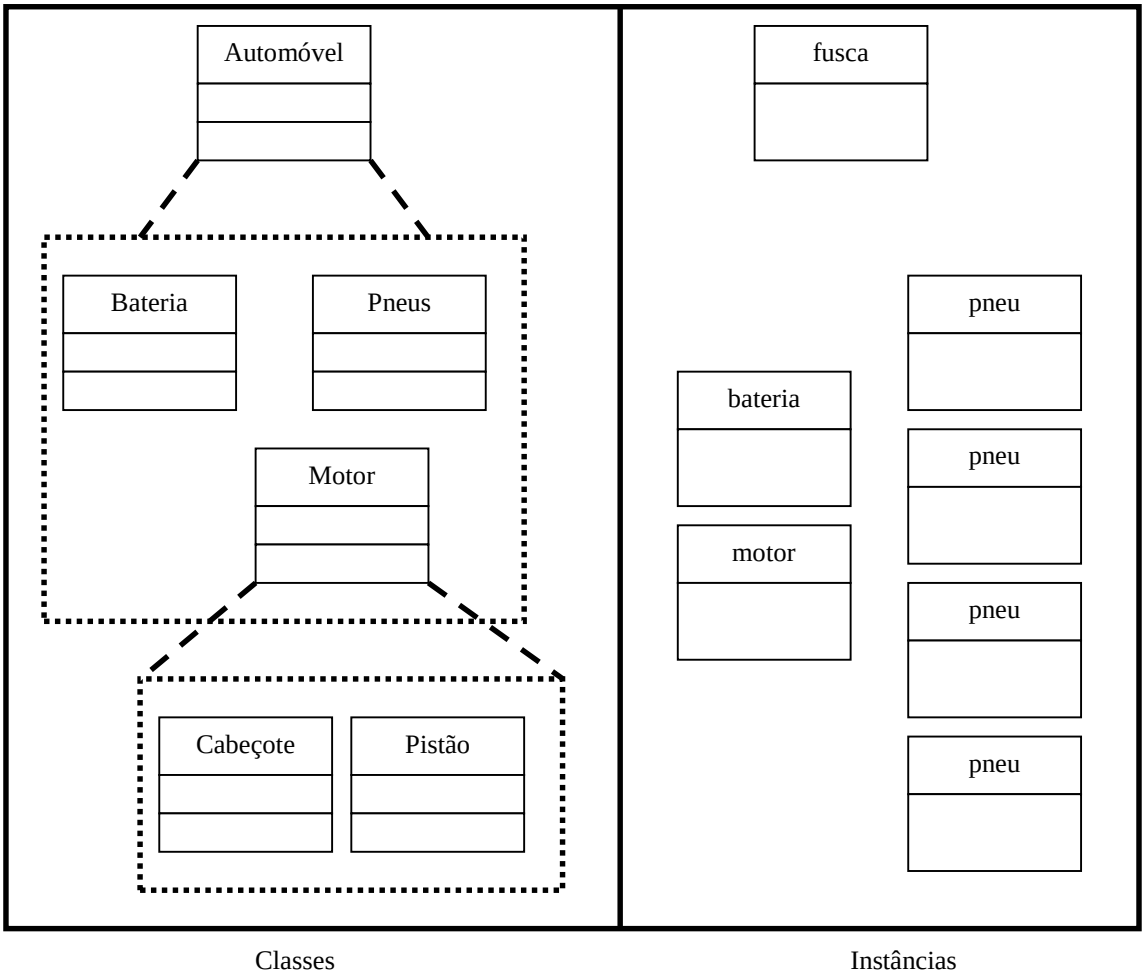

FIGURA 3.2 - Diagrama de Classes e Instâncias do Automóvel.

O exemplo permite comprovar que modelos orientados a objeto tendem a ser muito próximos do modelo real, pois permitem criar classes diretamente relacionadas com elementos presentes no mundo real. Estas classes podem, por sua vez, serem instanciadas inúmeras vezes, permitindo a reutilização das mesmas, além de garantir a fidelidade dos objetos em relação a classes.

É importante notar a possibilidade de utilizarmos outros conceitos de orientação a objeto, tais como herança, onde uma classe (veículo terrestre, por exemplo) pode ser redefinida, originando uma nova classe (automóvel, por exemplo), contendo os atributos

e métodos da classe original (veículo terrestre), somados aos atributos e métodos específicos da nova classes (automóvel); agregação, pelo qual podemos relacionar duas ou mais classes com uma terceira, formando, assim, uma classe que é a agregação de outras classes.

3.2.1 A "Visão" do Motorista

O motorista, no mundo real, preocupa-se basicamente em dirigir seu automóvel. Algumas informações básicas estão disponíveis para o motorista quando este dá a partida no automóvel (carga de bateria, nível de combustível, nível do óleo, temperatura do motor, etc.), outras tornam-se disponíveis quando este está em movimento (quilometragem percorrida, por exemplo). Um terceiro caso, a princípio, pode se manifestar nos dois momentos anteriores, como um pneu furado, por exemplo.

A Figura 3.3 apresenta quatro níveis distintos a serem considerados separadamente. No primeiro nível encontra-se o Mundo Real, assim identificado na figura, o qual representa os objetos físicos que estão sendo modelados. Podemos identificar as representações de um carro, um motor, uma bateria e um pneu. No segundo nível, encontra-se o Modelo OO, também identificado na figura. Percebe-se que uma seta tracejada direciona um objeto do Mundo Real (objeto pneu, por exemplo) para uma classe do Modelo OO (classe Pneus, por exemplo). Este direcionamento indica que o objeto físico (pneu) possui características desejáveis e significativas, as quais estarão presentes na classe que a representa no Modelo OO. Estas características do objeto físico (os atributos da classe) podem ser vistos na figura 3.3, no segundo nível, logo abaixo da representação da classe, dentro de uma chave. Os atributos são compostos por um tipo e um nome. O terceiro nível apresenta cinco Elementos de Visualização (uma de várias formas de representação gráfica de um atributo). Uma seta contínua liga o atributo de uma classe com a sua forma de apresentação (o atributo "tensão", da classe "Bateria", aponta para um semáforo e o atributo "temperatura" da classe "Motor" aponta para um Elemento de Visualização composto por um *gauge* e um *led*). Este nível apresenta uma visão geral do automóvel para o motorista, tendo como diferencial o atributo "pressão" da classe "Pneus", o qual identifica uma referência (elemento de visualização que permite acessar outra visão, geralmente mais detalhada). Finalmente, o quarto nível demostra o detalhamento da referência do atributo "pressão"

da classe "Pneus". Este detalhamento mostra a relação dos atributos "pressão", de quatro instâncias da classe "Pneu", com um elemento de visualização.

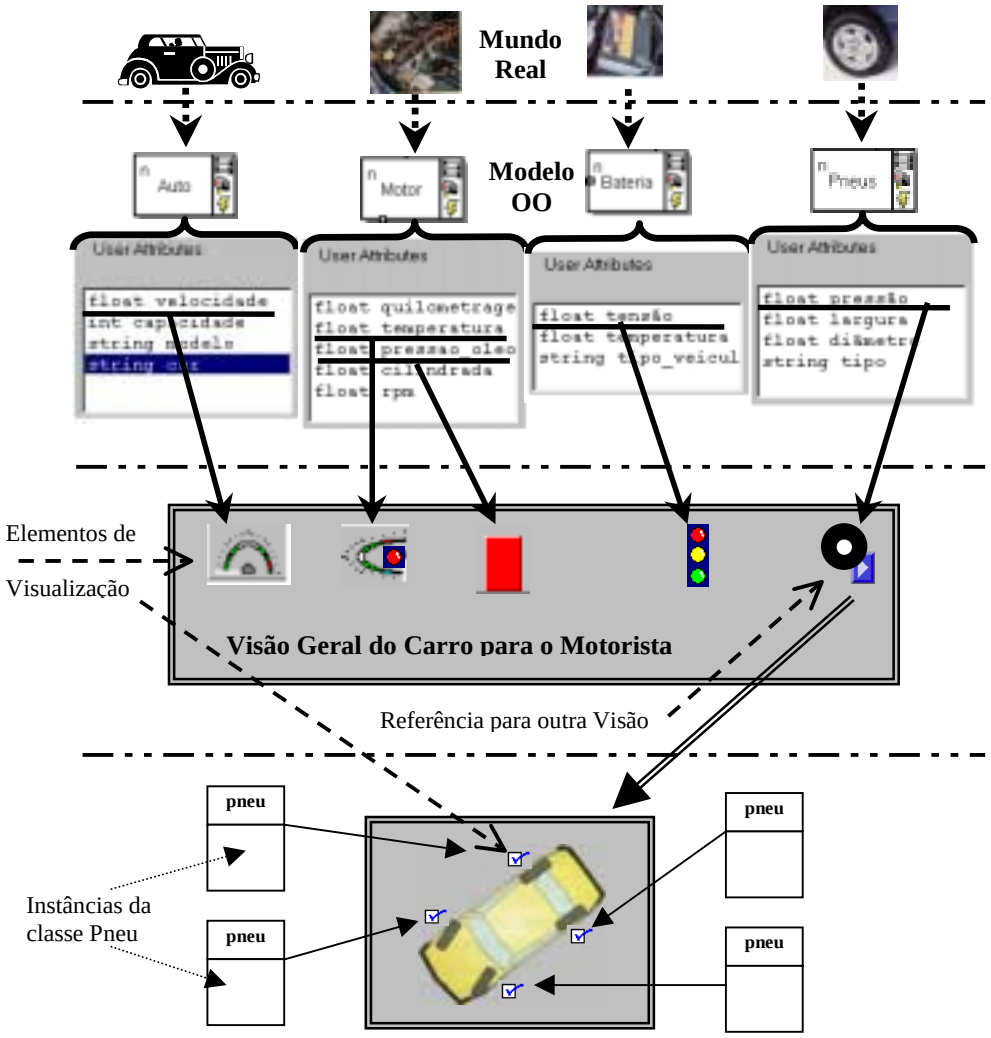

FIGURA 3.3 - A Visão de Automóvel do Motorista

A partir da figura 3.3, nota-se que o motorista possui duas "Visões". Na primeira a situação do automóvel pode ser vista de um modo mais genérico: os Elementos de Visualização apresentam o valor de um atributo de uma classe de forma gráfica, indicando uma situação que pode ou não ser significativa para o motorista (pressão do óleo, temperatura do motor, por exemplo). Percebe-se um diferencial em relação ao elemento de visualização do atributo "pressão" da classe Pneus, pois este é uma referência que aponta para outra Visão. Assim, se no mundo físico um dos pneus estivesse furado, a instância correspondente no modelo OO teria um valor muito baixo para o atributo "pressão". O Elemento de Visualização representaria esta situação com a mudança da cor do pneu, o que chamaria a atenção do motorista, porém não indicaria

qual dos pneus foi afetado. Para identificar qual a posição do pneu defeituoso o motorista deveria consultar a visão dos pneus, a qual apresenta a situação de cada um deles.

3.2.2 A "Visão" do Mecânico

O mecânico, pela natureza de seu trabalho, interessa-se por informações com maior grau de precisão do que as informações percebidas nas "Visões" do motorista. Em alguns casos, a "Visão" do motorista não possui relevância para o mecânico, como a visão dos pneus, por exemplo. Em outras situações, a "Visão" do mecânico é completamente desconhecida pelo motorista, interessando somente ao primeiro. Percebe-se, assim, que pessoas distintas (em níveis hierárquicos distintos) possuem diferentes visões dos mesmos objetos físicos. Esta diferenciação nas "Visões" torna necessária uma representação gráfica adequada a cada tipo de usuário (motorista e mecânico, por exemplo).

Na "Visão" do mecânico, apresentada na figura 3.4, podem ser vistos quatro níveis. Como no exemplo do motorista, o primeiro nível apresenta o Mundo Real, representado pelas figuras de um motor e de uma bateria. No segundo nível também são apresenta as classes (Motor, Cabeçote, Pistão e Bateria) e seus respectivos atributos. No terceiro nível, a "Visão" geral do carro para o mecânico possui os mesmos Elementos de Visualização presentes na "Visão" do motorista (atributos "temperatura" e "pressão" da classe "Motor"). Neste nível nota-se, também, que os Elementos de Visualização dos atributos das classes "Auto" e "Pneus" foram substituídos pelos Elementos de Visualização relacionados com as classes "Cabeçote" e "Pistão". Observa-se, ainda, que um mesmo atributo ("tensão" da classe "Bateria") é representado por Elementos de Visualização diferentes (motorista - um conjunto de *leds*; mecânico - conjunto de *leds* e *display* numérico). Para o motorista basta saber se a carga é suficiente ou não. Para o mecânico, a tensão exata da carga da bateria pode representar um diagnóstico preciso de alguma anomalia no automóvel, assim este EV é uma referência para uma "Visão" mais detalhada da classe "Bateria". O quarto nível apresenta o detalhamento da classe "Bateria", onde informações importantes para o mecânico podem ser visualizadas (tais como a temperatura suportada pela bateria, sua tensão e o tipo de veículos nos quais pode ser utilizada). Este nível é totalmente diferente, tanto para motorista, quanto para mecânico, mas ambos são partes do mesmo automóvel. Sua representação necessita ser

a mais próxima do nível em que se encontra a pessoa que faz uso das informações representadas por ela.

Observa-se, a partir dos exemplos anteriores e da figura 3.1 (diagrama de classes do modelo conceitual do *Supervisory Designer*) que uma mesma classe (auto, motor, bateria, pneu, por exemplo) pode ser representada por um ou mais Elementos de Visualização - EV. Um grupo de EV's, da mesma classe ou de classes diferentes, pode ser agrupado em uma mesma "Visão". Uma "Visão" pode apresentar EV's e "Referências" para outras "Visões".

Uma "Visão" é a representação de uma ou mais informações significativas para seu observador, através de EV's e/ou "Referências". Um atributo pode ser visto em diferentes "Visões" (como o atributo "tensão" da classes "Bateria", visível tanto para o motorista quanto para o mecânico).

Segundo estas afirmações, constata-se que o modelo do *Supervisory Designer* possibilita a rápida definição de "Visões", "Referências" e "Elementos de Visualização" para um sistema supervisório, possibilita, ainda, a utilização de um modelo de classes já existente.

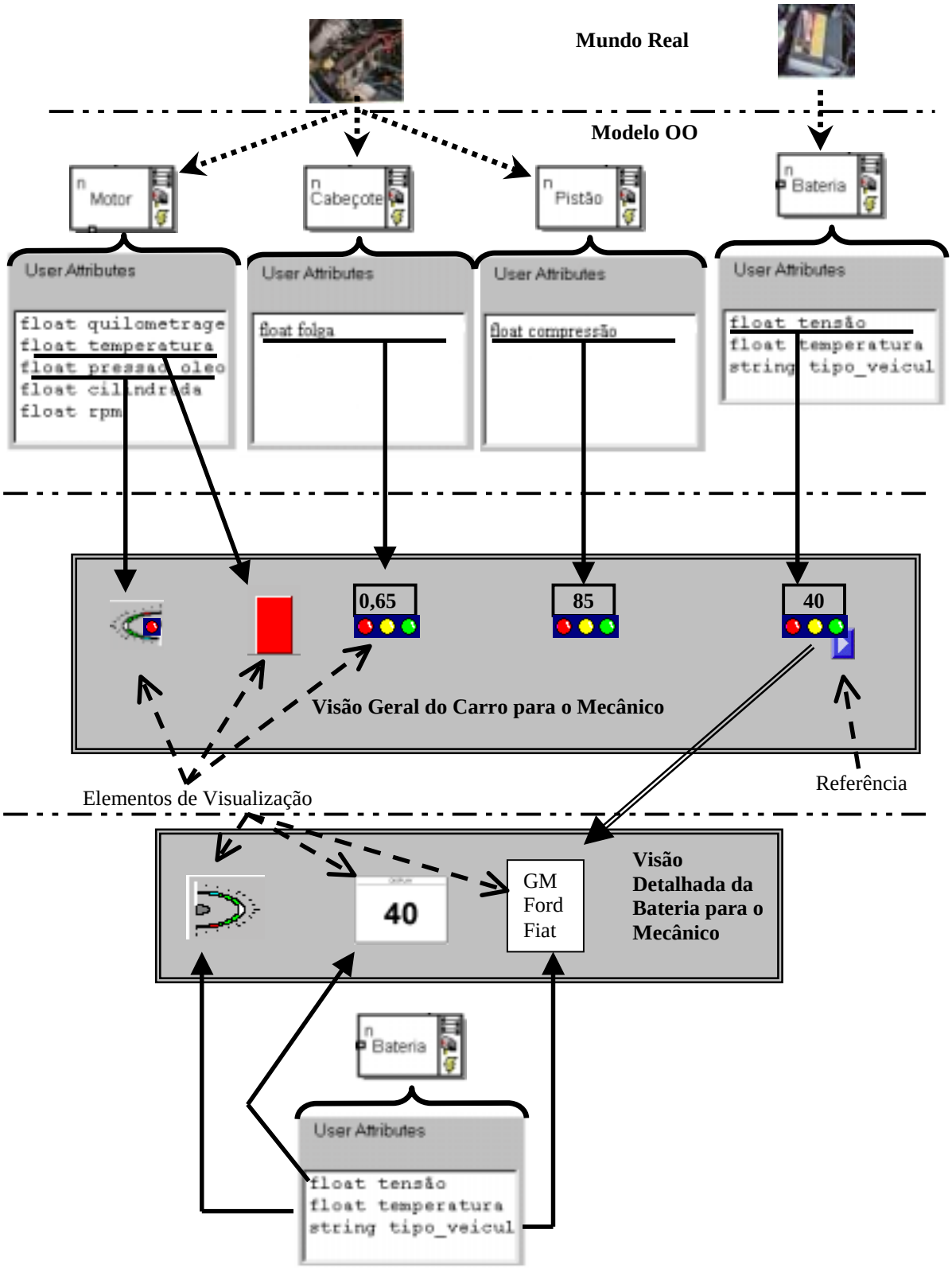

FIGURA 3.4 - A "Visão" de Automóvel do Mecânico.

3.3 Proposta de Integração com Sistemas Supervisórios

A integração de uma ferramenta de modelagem OO com uma ferramenta de desenvolvimento de sistemas supervisórios tornaria o trabalho de desenho, projeto, desenvolvimento e implantação destes sistemas muito mais rápido e produtivo, permitindo o reaproveitamento de muitos componentes e até projetos inteiros, se uma ferramenta pudesse fazer uso das qualidades da outra, interagindo entre si.

Uma das vantagens imediatas da integração destas ferramentas permitiria que, durante a modelagem, a especificação de características de supervisão fosse definida para as classes, de modo que durante a confecção do modelo as instâncias herdassem estas características e seus respectivos valores padrão. Esta condição evita a redefinição individual destes valores para as *Tags*, como acontece no Elipse. A figura 3.5 apresenta as características básicas definidas para cada *Tag* no Elipse Windows e que necessariamente deveriam estar presentes na definição da classe no SIMOO-RT.

Quando uma classe é editada na ferramenta SIMOO-RT, pode-se perceber a inexistência de uma janela para a definição dos valores e das características apresentadas na figura 3.5. Estão presentes, como era de se esperar, o conjunto de atributos presentes na classe, bem como os métodos implementados pela mesma. A figura 3.6 apresenta duas janelas, uma contendo os atributos da classe (sem qualquer característica associada à cada atributo, exceto o tipo) e outra contendo na parte superior os métodos e sua mensagem de ativação e na parte inferior uma porção do código deste método.

No presente trabalho propõe-se a seguinte estratégia para a integração das ferramentas de modelagem orientada a objeto e de geração de sistemas supervisórios (vide figuras 3.5 e 3.6). A presente proposta permitiria a definição das características e valores referentes a *Tag*, para os atributos ou métodos na ferramenta SIMOO-RT, em tempo de projeto. Outra melhoria permitiria a definição de diferentes modos de visualização para o atributo, para a classe ou para o método. Esta visualização poderia ficar armazenada na forma de um catálogo básico, permitindo ao usuário, na confecção do supervisório, escolher a representação mais conveniente, editar uma representação existente ou simplesmente criar uma representação nova e adiciona-la ao catálogo. A figura 3.7 apresenta o esquema ideal de integração entre as duas ferramentas de forma mais ampla, onde podemos ver o diagrama de classes na parte mais superior. Deste

diagrama é selecionada uma classe. Para esta classe são definidos atributos (como float temperatura, float pressão, int nível , etc...) os quais, por sua vez, podem ser utilizados como entrada para um diagrama de estado ou como atributos para um EV (elemento de visualização). Utilizando-os para um EV, os atributos são associados a uma forma de visualização (um gráfico ou figura qualquer) e posteriormente são exportados para uma ferramenta de criação de sistemas supervisórios, a qual lê as definições e as apresenta em tela.

A figura 3.8, por sua vez, apresenta as alterações necessárias para o incremento de algumas características de supervisão no SIMOO-RT. Após a definição de um atributo, é definido um conjunto de valores de controle para este atributo. Assim para efeito de visualização, o atributo “int temperatura” teria os seguinte indicativos: unidade, valor mínimo, valor máximo, valor mínimo-mímino, valor máximo-máximo, prioridade e texto. Estes indicativos podem ser usados em tempo de projeto na ferramenta SIMOO-RT para definição de “valores de supervisão”, para que no momento da importação no sistema supervisório estes valores não precisem novamente ser definidos, antecipando uma etapa de construção do sistema.

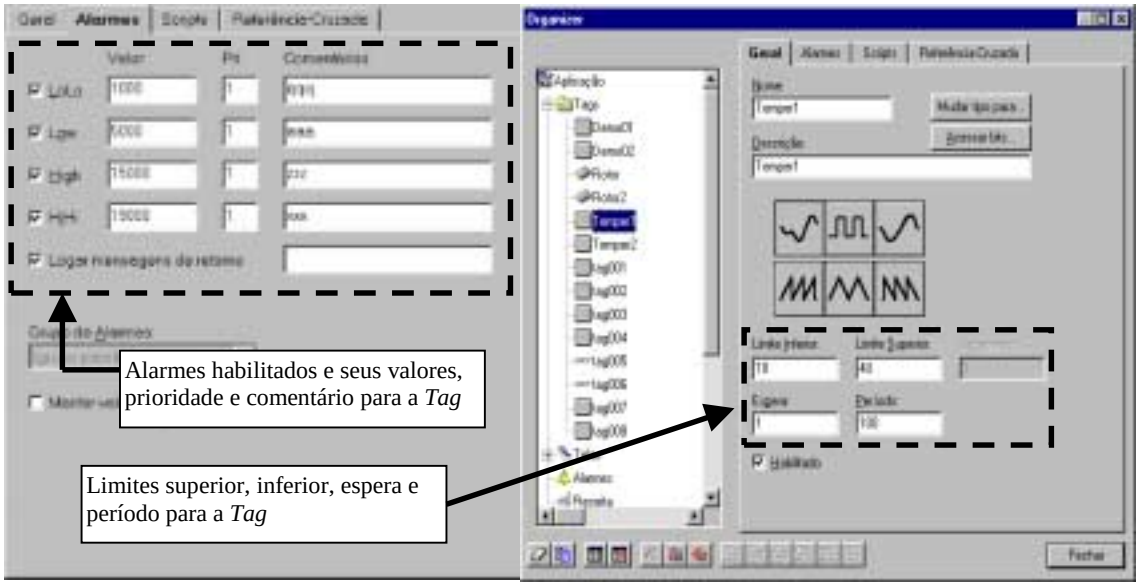

FIGURA 3.5 - Características Básicas Definidas para a *Tag* no Elipse Windows.

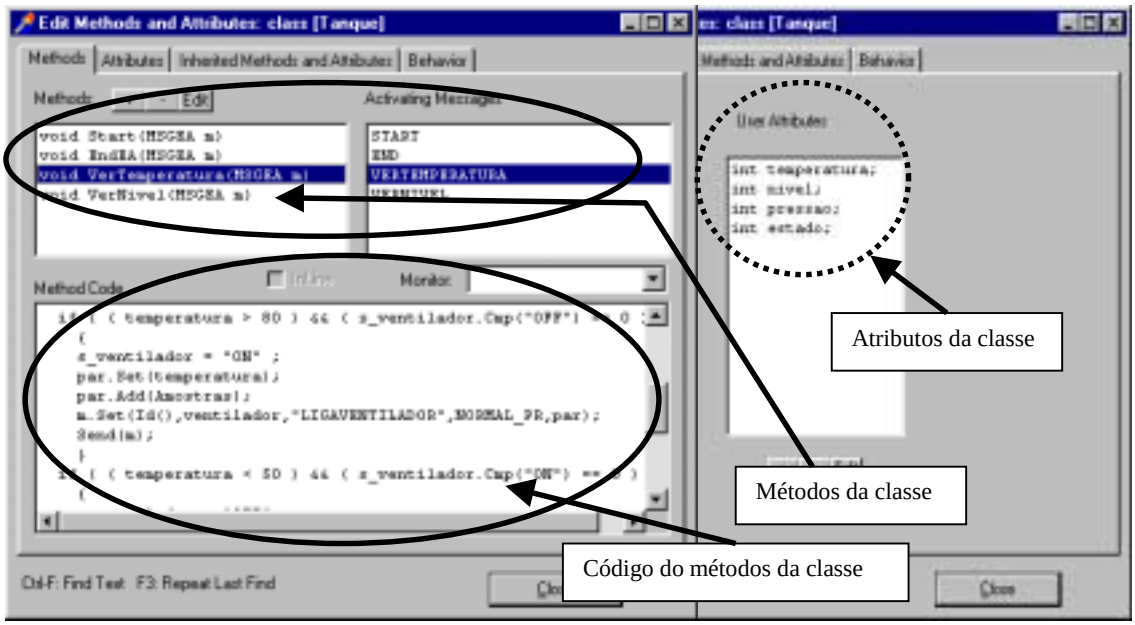

FIGURA 3.6 - Definição de Atributos de Métodos no SIMOO-RT

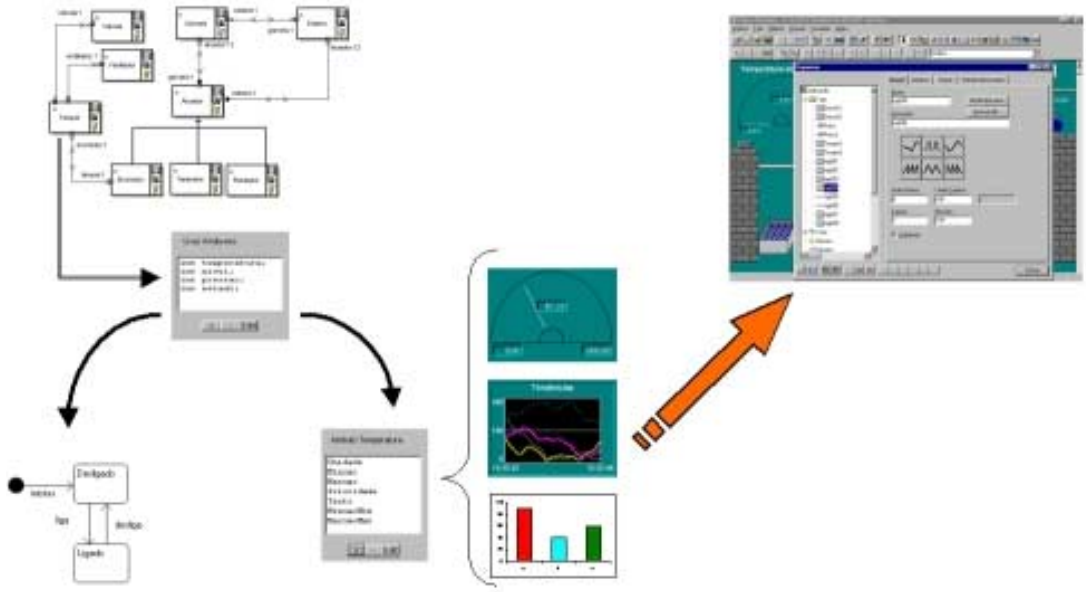

FIGURA 3.7 - O Esquema Ideal de Integração das Ferramentas SIMOO e Elipse.

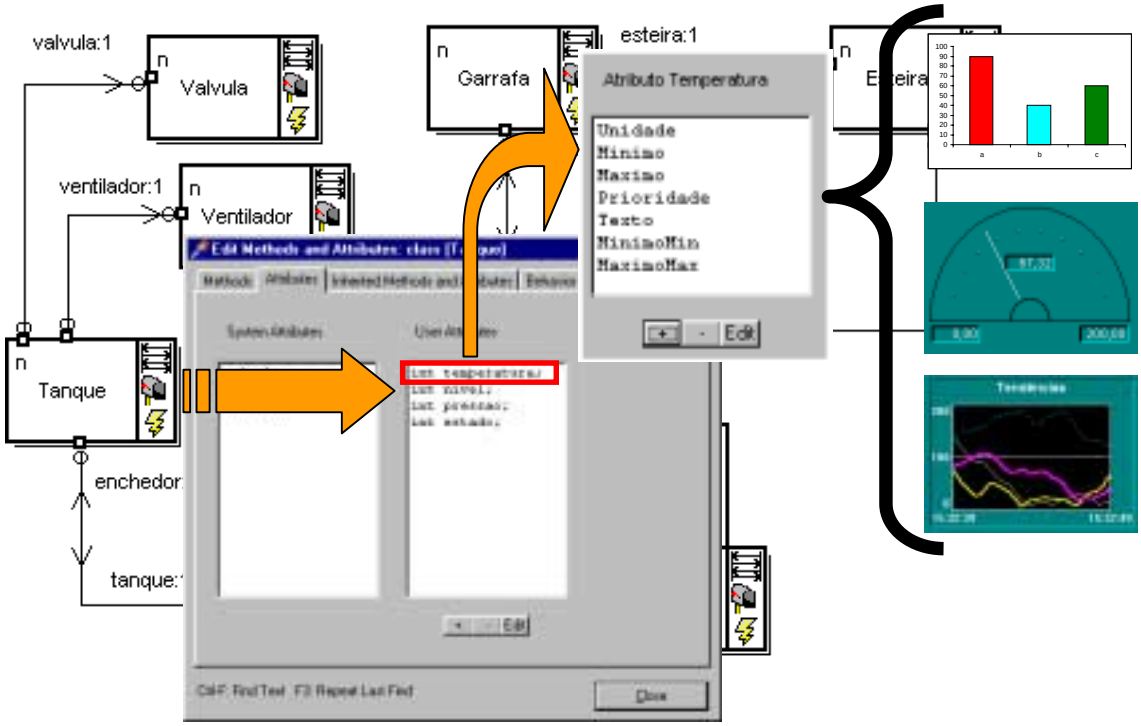

FIGURA 3.8 - Esquema Detalhado de Integração das Ferramentas SIMOO e Elipse.

3.4 Conclusão

A criação de modelos baseados no mundo real é facilitada pelo uso de ferramentas CASE, especialmente aquelas que fazem uso do paradigma de orientação a objeto, como é o caso do SIMOO. O SIMOO, como exposto no Capítulo 2, permite a definição do diagrama de classes e do diagrama de instâncias, criação de classes e definição de atributos e métodos para as mesmas, o que a torna uma ferramenta adequada para a modelagem de sistemas computacionais, especialmente os industriais.

O programa de supervisão de processos Elipse Windows, também apresentado no Capítulo 2, mostrou, por sua vez, fácil utilização na criação dos supervisórios de processos, através de uma interface intuitiva (*Organizer*), de componentes "pré-definidos" (as *Tags* de diversos tipos), da linguagem de programação (*Scripts*) e dos *drives* para equipamentos CLP's presentes na mesma.

A integração entre ambas as ferramentas passa, primeiramente, pela redefinição de parte da implementação do SIMOO-RT, visto que esta ferramenta não possui componentes voltados para a supervisão de processos. É preciso inserir os componentes que não estão presentes no SIMOO e, conseqüentemente, interligá-los com os

diagramas de classes e instâncias. Para a realização desta alteração é necessária a definição de valores de alarmes para os atributos da classe. Após a definição dos valores dos alarmes, a criação de elementos de visualização relacionados com os atributos e métodos também se faz necessária. O armazenamento destas definições em arquivos possibilita, na seqüência, a realização de troca dos dados com o sistema supervisório Elipse.

As implementações descritas no parágrafo anterior são factíveis no SIMOO-RT, pois este programa, sendo de uso acadêmico, possibilita sua alteração e extensão. O programa Elipse Windows, assim como os demais programas analisados, é um programa comercial e até o momento da realização deste estudo não permitia a modificação do seu código.

4 Estudo de Caso

Este capítulo apresenta um estudo de caso mostrando a geração de sistemas supervisórios a partir de modelos orientados a objeto, utilizando para isto a ferramenta *Supervisory Designer*, como forma de mostrar suas potencialidades.

O presente estudo de caso baseou-se em um sistema supervisório exemplo contido no programa Elipe Software (que simula o funcionamento de uma engarrafadora de bebidas), buscando, assim, criar um modelo orientado a objetos e, a partir deste, um sistema supervisório semelhante a outro já existente e em funcionamento, como mostra a figura abaixo.

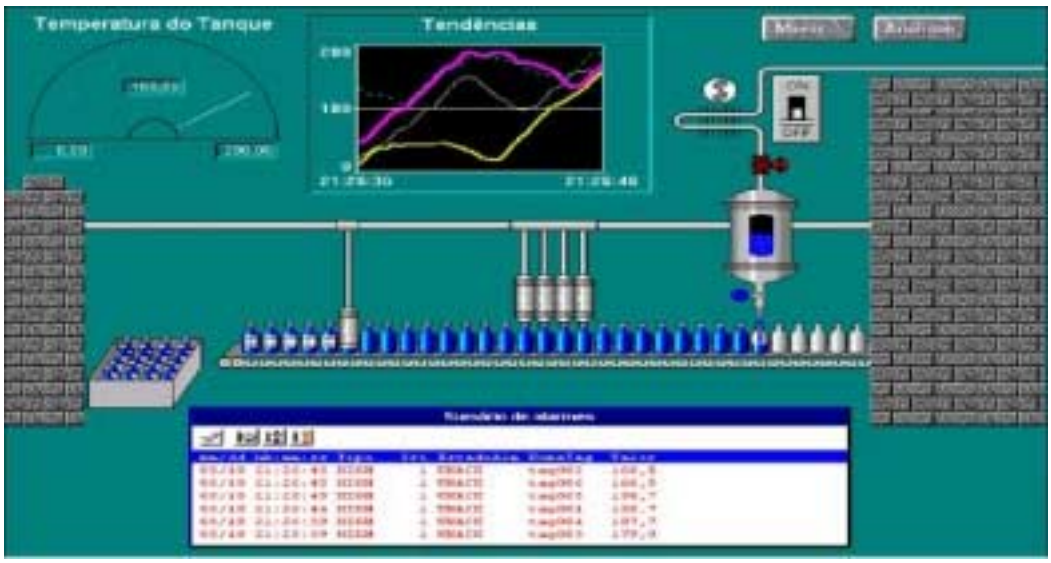

FIGURA 4.1 - Exemplo de Supervisório Presente no Elipse Software.

4.1 O Modelo Orientado a Objetos do Sistema de Engarrafamento de Bebidas

A modelagem do sistema apresenta as seguintes classes: Garrafa, Esteira, Valvula, Ventilador, Atuador (e suas herdeiras Enchedor, Tampador e Rotulador) e Tanque, na qual nos concentraremos. A figura 4.2 apresenta o modelo de classes.

Para fornecer uma estrutura hierárquica que pudesse ser facilmente representada em forma de diretórios ou árvore, foram acrescentadas outras três classes (Tanque1, Tanque2, e Tanque3) no nível logo abaixo da classe Tanque. Estas classes por sua vez também estão indicando outros níveis abaixo de si, no total de dois. Este níveis possuem uma ou mais classes, permitindo, assim, visualizar perfeitamente a hierarquia destas classes no modelo.

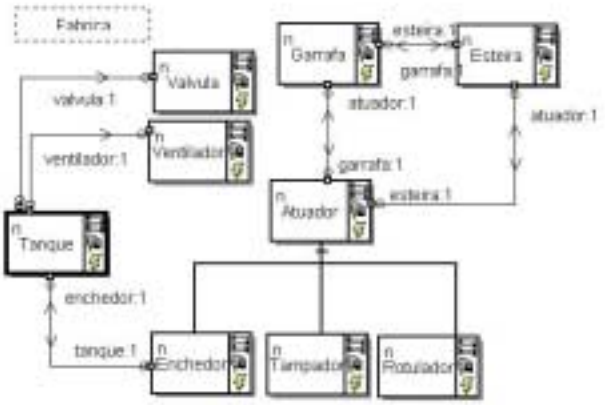

FIGURA 4.2 - O Modelo de Classes da Engarrafadora de Bebidas.

A primeira etapa de modelagem caracteriza-se por dois momentos distintos. No primeiro faz-se a definição dos atributos para a classe Tanque, através da página *Attributes* da janela *Edit Methods and Attributes*, definida no SIMOO para este fim. Isto pode ser visto na janela da esquerda na figura 4.3. A simples adição de um atributo nesta janela habilita a definição de valores de visualização para o mesmo. Em um segundo momento definem-se os valores para um ou mais atributos da classe, permanecendo na mesma janela utilizando a página *Visualization*, visível na figura 4.3. A classe Tanque possui os seguintes atributos: int temperatura, float nivel, float pressao e int estado. Para estes atributos são informados valores referentes a sua visualização no sistema supervisório (alarmes, como visto na seção 2.1).

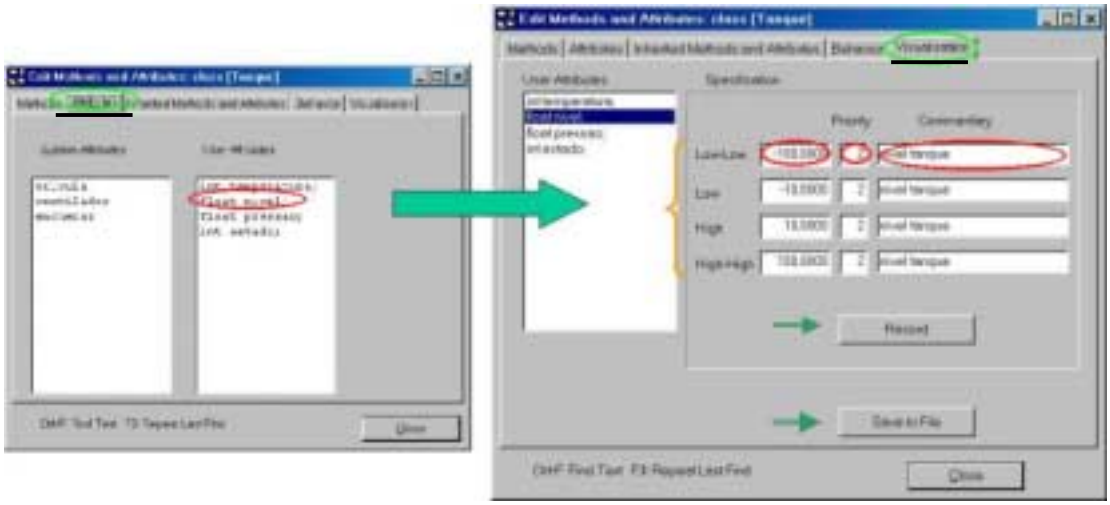

FIGURA 4.3 - Janelas de Atributos e Alarmes da Classe Tanque.

Para permitir o *Supervisory Designer* conhecer os valores definidos até aqui, a página *Visualization* possibilita salvar os atributos e seus valores associados em um arquivo tipado (preferencialmente com o nome da classe com e a extensão ".vis" - Tanque.vis). A estrutura do arquivo Tanque.vis pode ser vista na figura 4.4.

```
TEstVis = record
    Classe          : String[MaxLength];
    Tam             : integer ;
    Atributo        :Array [0..MaxArray] of String[MaxLength];
    Hi              :Array [0..MaxArray] of String[MaxLength];
    HiHi            :Array [0..MaxArray] of String[MaxLength];
    Low             :Array [0..MaxArray] of String[MaxLength];
    LowLow          :Array [0..MaxArray] of String[MaxLength];
    Precisao        :Array [0..MaxArray] of String[MaxLength];
    Unidade         :Array [0..MaxArray] of String[MaxLength];
    PriHi           :Array [0..MaxArray] of String[MaxLength];
    PriHihi         :Array [0..MaxArray] of String[MaxLength];
    PriLow          :Array [0..MaxArray] of String[MaxLength];
    PriLolo         :Array [0..MaxArray] of String[MaxLength];
    ComHi           :Array [0..MaxArray] of String[MaxLength];
    ComHihi         :Array [0..MaxArray] of String[MaxLength];
    ComLow          :Array [0..MaxArray] of String[MaxLength];
    ComLolo         :Array [0..MaxArray] of String[MaxLength];
    // Atributos Instrumented -> true or false
    AtribInstr      :Array [0..MaxArray] of String[MaxLength];
    end;
```

FIGURA 4.4 - A Estrutura do Arquivo de Visualização (.vis).

4.2 Usando o Modelo no *Supervisory Designer*

Tendo como base as definições anteriores, podemos acionar o *Supervisory Designer*. Ao ser carregada a ferramenta apresenta a classe de nível mais alto no modelo. O sinal "+" à esquerda de uma classe indica a existência de níveis inferiores abaixo deste, contendo outras classes. A partir deste ponto a navegação pelas classes é facilitada pela possibilidade de clicar e expandir (ou encolher) um ou mais níveis. A figura 4.5 apresenta o modelo Fabrica com seus sub-níveis expandidos.

Na janela do *Supervisory Designer* estão presentes os seguintes componentes:

- ***Classes***: situada do lado esquerdo do janela, são apresentadas todas as classes do diagrama atual, na forma de estrutura em árvore. As classes que possuírem um sub-nível são apresentadas com o sinal “+” antes de seu nome. Cada *nodo* da estrutura em árvore contém o nome das classes e um ponteiro para a respectiva

classe. Isto permite, em um mesmo componente, realizar a representação hierárquica do modelo e manipular os dados da classe selecionada.

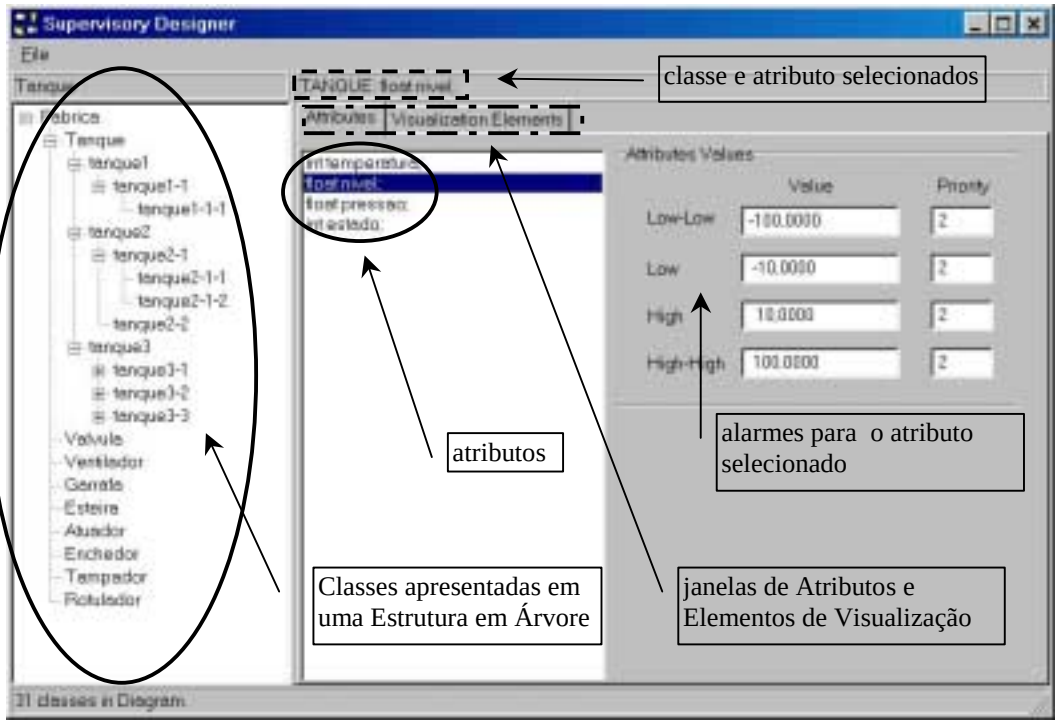

FIGURA 4.5 - O Modelo Expandido no *Supervisory Designer* .

- ***Items***: a segunda parte da janela está subdividida em duas janelas, com funções diferentes:
  - ***Attributes***: esta janela apresenta os atributos que a classe possui e os seus respectivos valores para os alarmes, conforme configurados no Form EditarInstancia. Os valores dos atributos da classe são mostrados após a seleção da respectiva classe.
  - ***Visualization Elements***: esta janela demonstra as possíveis representações gráficas de uma atributo. Cada *bitmap* está relacionado com um EV diferente. Na atual implementação, somente a adição ou deleção são possíveis, em uma futura seqüência deste trabalho, a alteração será possível.

Como os valores dos alarmes foram pré-definidos para os atributos (vide figura 4.3) e estão relacionados diretamente com a classe Tanque, quando um dos atributos é selecionado, seus valores são apresentados na caixa "*Attributes Values*". É permitida a

alteração ou manutenção dos valores dos alarmes, os quais serão copiados para a instância do Elemento de Visualização definido na janela "*Visualization Elements*".

A partir da visualização da estrutura hierárquica das classes e seus respectivos atributos e alarmes no *Supervisory Designer*, podemos criar Elementos de Visualização (instanciando uma das classes TGauge, TbarGraph, TDisplay ou TimgAnim). Neste estudo de caso, foram criados quatro elementos de visualização: *bitmap sequence* (identificado como *animation*), *bar graph*, *display* e *gauge*. A criação de EV's é melhor explicitada nas seções a seguir.

4.2.1 A criação de Elementos de Visualização (EV)

Uma vez que acessamos a estrutura de classes do SIMOO, pode-se acessar os atributos destas classes e ligar estes a um elemento de visualização. A escolha pode ser feita entre um dos tipos de visualização disponíveis: Gauge, Display, Bar Graph ou Bitmap Animation. Será criado, então, um objeto do tipo TEV, o qual terá como informações: a classe atual, o nome da classe, os atributos da classe e os dados dos alarmes para os atributos da classe.

Nas ações subseqüentes são configurados os valores para cada tipo específico de EV (*Gauge*, *Display*, *Bar Graph* ou *Bitmap Animation*) e sua visualização é apresentada conforme um *bitmap* associado.

4.2.1.1 Gauge

O elemento de visualização *Gauge* possui três janelas, cada uma tendo um conjunto de itens alteráveis:

- **Janela *General***: esta janela permite configurações relacionadas com as cores e a orientação do Gauge, como pode ser visto na figura 4.6. Seus itens são:
  - *Description*: breve descrição do EV;
  - *Values:* valores de configuração máximo e mínimo, são carregados a partir do arquivo de visualização (classe.vis, por exemplo), caso exista.
  - *Colors*: permite configurar as cores de quadro, de fundo, do texto e da agulha ou ponteiro.
  - *Rotation*: permite definir a rotação a que será submetido o Gauge.

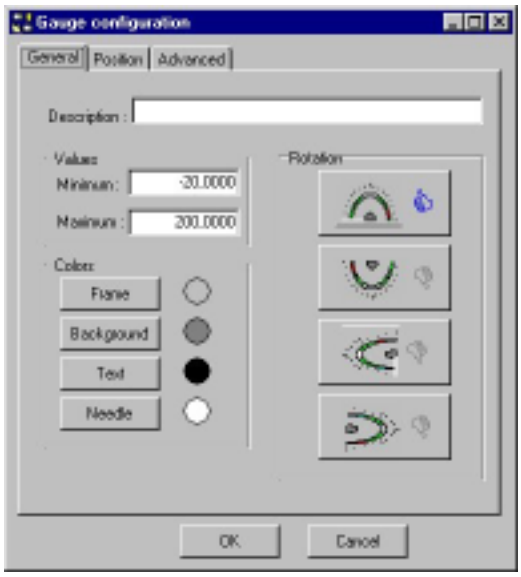

FIGURA 4.6 - A janela *General* da Configuração do *Gauge*.

- **Janela *Position***: esta janela está relacionada diretamente com o tamanho do EV e seu posicionamento na tela do Sistema Supervisório de destino. Como a janela *Position* é idêntica para todos EV's, portanto será apresentada somente no EV Gauge. A figura 4.7 apresenta a janela *Position*.

FIGURA 4.7 - A Janela *Position* da Configuração do *Gauge*.

Os itens presentes na aba *Position* são:

- X e Y: coordenadas X e Y para posicionamento do EV.

- Height e Breadth: altura e largura que o EV terá. Não há definição de uma medida, ficando para o Sistema Supervisório de destino sua identificação.
- Frame: pode configurar este item como visível ou não visível, se haverá borda, qual sua cor e espessura; se haverá título, qual o texto, sua cor de fundo, sua fonte e separador.

- **Janela *Advanced***: *Advanced* tem itens relacionados com os alarmes dos valores dos atributos e sua apresentação, vistos na figura 4.8. Eles são:
  - *Marks*: mostra e configura a cor das marcas referentes aos valores no Gauge.
  - *Sub-Marks*: mostra e configura a cor e a quantidade das sub-marcas referentes aos valores no Gauge.
  - *Values*: mostra e configura o tipo do caracter para os valores no Gauge.
  - *Legend*: pode mostrar todas ou exibir/ocultar individualmente todos os alarmes. Definem-se as cores e os valores que LoLo, Lo, Hi, HiHi e Normal (somente cor) terão. Os valores apresentados inicialmente para os alarmes são carregados do arquivo de visualização.

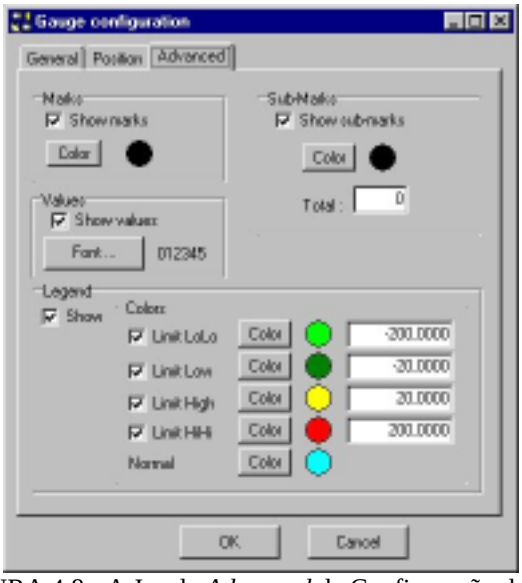

FIGURA 4.8 - A Janela *Advanced* da Configuração do *Gauge*.

4.2.1.2 Display

O elemento de visualização Display possui as janelas *Position* e *General*, mostrada a seguir :

- **Janela *General***: esta janela contém os seguintes itens (figura 4.9):
  - *Description*: permite uma descrição resumida do Display.
  - *Alignment*: possibilita indicar o alinhamento do Display nos sentidos horizontal (esquerda, centro e direita) e vertical (acima, no centro e abaixo).
  - *Font*: permite alterar o tipo de fonte do Display.
  - *Color*: permite alterar a cor da fonte do Display.
  - *Format*: indica qual a forma de apresentação do Display: texto, numérico ou data e hora. No caso do Display numérico podem ser definidos tamanho e precisão. Sufixo e prefixo podem ser adicionados.

FIGURA 4.9 - A Janela *General* da Configuração do *Display*.

4.2.1.3 Bitmap Sequence

O elemento de visualização *Bitmap Sequence* apresenta três janelas: *General*, *Position*, apresentada anteriormente, e *Images*. As janelas *General* e *Images* são descritas a seguir.

- **Janela *General***: esta janela apresenta os seguintes itens (figura 4.10):
  - *Description*: permite uma pequena descrição do elemento.
  - *Transparent*: caso este item esteja assinalado o EV terá seu fundo transparente no supervisório destino.
  - *Border*: apresenta ou não a borda envolta do EV.

- *Background color*: permite alterar a cor de fundo do elemento.
- *Refresh*: tempo no qual o elemento de visualização é atualizado.

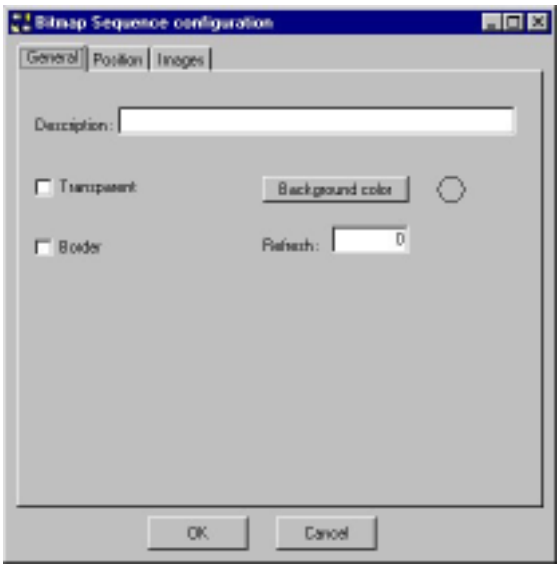

FIGURA 4.10 - A Janela *General* da Configuração do *Bitmap Sequence*.

- **Janela *Images*:** esta janela permite selecionar e ordenar figuras estáticas que, apresentadas em seqüência dão a impressão de movimento. O movimento é realizado quando o EV atinge um determinado valor ou está dentro de um determinado intervalo, conforme for o caso. Os seguintes itens estão presentes nesta janela (na figura 4.11):
  - *Images chart:* este quadro apresenta o nome da figura (*Bitmap*), a sua localização no disco rígido (*Localization*) e o valor máximo (ou máximo e mínimo) que esta pode atingir para permanecer visível.
  - *Number of Images:* item que contém o número de imagens presentes no elemento de visualização.
  - *Add:* este botão permite adicionar uma figura ao quadro de imagens.
  - *Del:* este botão retira uma figura do quadro de imagens.
  - *Manual*: este item permite a configuração de um intervalo diferente de valores para cada imagem.
  - *Automatic*: este item, por sua vez, recebe dois valores, máximo e mínimo, e distribui o intervalo entre o número de figuras presentes. A figura 4.11 apresenta os modos *Automatic* e *Manual*.

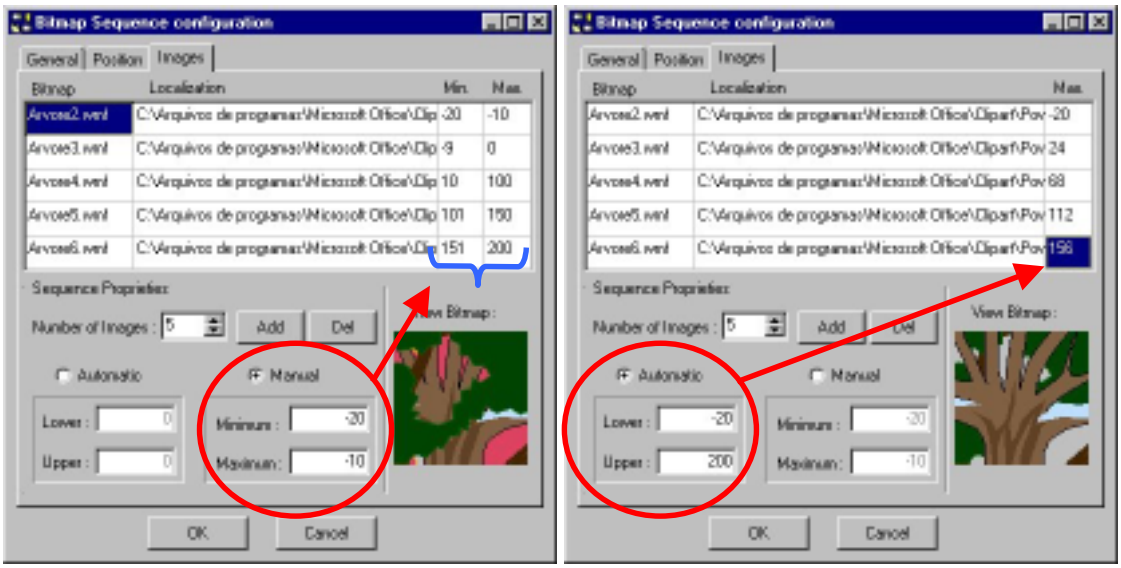

FIGURA 4.11 - A Janela *Images* da Configuração do *Bitmap Sequence*.

4.2.1.3 Bar Graph

O EV *Bar Graph* possui duas janelas: *General* e *Position*. A seguir são apresentados os componentes da janela *General*.

- **Janela *General:*** esta janela permite a configuração de máximo e mínimo para um gráfico, a orientação e as cores para o gráfico e sua régua, além de ajustes no texto. Apresentada na figura 4.12.
  - *Description*: permite uma pequena descrição do elemento.
  - *Limits:* carrega os valores definidos para o atributo no Form EditarInstancia para os campos *Inferior* e *Superior*, mas permite sua alteração a qualquer momento.
  - *Orientation*: indica qual será a orientação do gráfico: para cima, para baixo, para a esquerda ou para a direita.
  - *Enable Ruler*: habilita ou não a régua do gráfico.
  - *Colors of ruler:* permite escolher as cores de fundo, do texto e da grade da régua, bem como a posição de sua apresentação: superior, inferior, à direita ou à esquerda do gráfico.
  - *Text of Unit:* permite identificar a unidade de medida que está sendo apresentada pelo gráfico.

- *Font of Ruler:* o tipo de letra para a unidade.
- *Background color:* cor de fundo do elemento de visualização.
- *Bar color:* cor da barra do gráfico.

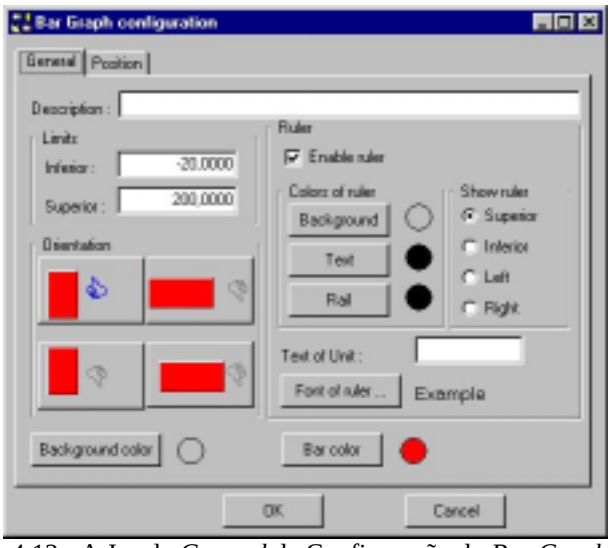

FIGURA 4.12 - A Janela *General* da Configuração do *Bar Graph*.

4.2.2 A apresentação dos Elementos de Visualização (EV)

A partir das ações descritas no texto acima, os EV's criados no *Supervisory Designer* são apresentados na janela "*Visualization Elements*", como pode ser visto na figura 4.13. Cada EV inserido é apresentado como um ícone representando seu tipo (um quadrado contendo uma figura e o nome do EV).

Na figura 4.13 percebe-se, ainda, que os quatro elementos de visualização, inseridos para o atributo "nível" da classe Tanque não ocupam todo o espaço da página *Visualization Elements*, a qual pode conter outros elementos para outros atributos da mesma ou de outras classes. O quadro tracejado determina o espaço onde serão apresentados estes Elementos de Visualização. Um duplo clique sobre o EV permite a edição dos seus valores, apresentando as janelas descritas nas seções anteriores.

A configuração dos valores de alarmes para os atributos e a simples definição de valores para os elementos de visualização não garante que os mesmos possam ser transportados para um programa supervisório. Para que isto seja possível o *Supervisory Designer* permite gravar todas as definições efetuadas em arquivo texto, o qual pode ser lido pelo programa supervisório destino. Este arquivo possui preferencialmente o nome

da classe seguido da extensão .sup, assim teríamos Tanque.sup. Estes nomes podem ser alterados pelo usuário no momento do salvamento da estrutura do supervisório.

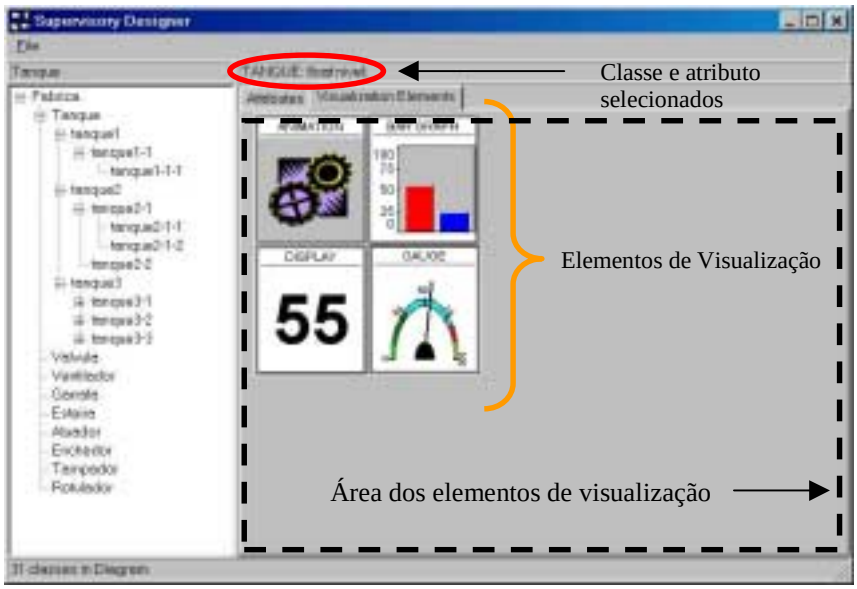

FIGURA 4.13 - Elementos de Visualização da Classe Tanque.

A troca de informações entre ferramenta de modelagem e sistema supervisório, descrita acima, foi criada no âmbito do projeto "*Tool Support to the Generation of Supervisory Control Software from Object-Oriented Specifications of Real-Time Industrial Automation Systems*" [Par 00], realizado na UFRGS.

Até o momento da finalização deste trabalho não tinha sido possível realizar a integração, na forma de exportação e importação, entre os ambientes SIMOO-RT e Elipse Windows. Sendo o Elipse Windows uma ferramenta comercial, não tivemos acesso ao seu código fonte. Apesar disso, mantivemos reuniões com a equipe de desenvolvimento da empresa Elipse, apresentamos a ferramenta SIMOO-RT, suas características, potencialidades e possibilidades de integração entre ambas as ferramentas. As pessoas participantes da equipe de desenvolvimento do Elipse Windows se mostraram receptivas as idéias apresentadas, para intensificar as intenções de integração, recebemos uma versão beta (em fase de finalização) do Elipse Windows e, em contrapartida, entregamos a eles uma versão do SIMOO-RT para estudo.

5 Conclusões e Trabalhos Futuros

Os principais pontos abordados por esta dissertação envolvem o uso de modelos criados a partir de ferramentas de modelagem orientadas a objeto por ferramentas geradoras de sistemas supervisórios. Este trabalho apresentou uma possibilidade de integração das ferramentas de modelagem com as ferramentas de supervisórios, através da incorporação de características de supervisão em ferramentas de modelagem.

Durante a análise do estado-da-arte, realizou-se o estudo das ferramentas geradoras de sistemas supervisórios, e dentre outras, optou-se pelo Elipse Windows, descrito no Capitulo 2. Esta ferramenta serviu como parâmetro para o levantamento das características presentes em sistemas supervisórios, possibilitou o entendimento de seus componentes principais (as *Tags*), permitiu um comparativo de sua interface gráfica com as demais ferramentas de supervisórios, e apresentava grande número de usuários dentro do grupo de pesquisa do IEE. Um aspecto importante em relação ao Elipse Windows foi o diálogo realizado entre os pesquisadores do IEE e os técnicos e gerentes de seu fabricante, objetivando a realização de trabalhos conjuntos, visando à integração das ferramentas Elipse e SIMOO-RT.

Dentre todas as ferramentas de modelagem orientadas a objeto pesquisadas, a escolhida neste estudo foi a SIMOO-RT, igualmente descrita no Capítulo 2. Esta ferramenta de modelagem possui vários pontos positivos em sua adoção: sua larga utilização por pesquisadores da UFRGS, possibilidade de extensão da mesma devido à disponibilidade do código fonte, possibilidade de descrição de restrições temporais, dentre outras. Outro grande motivo para sua escolha foi a possibilidade de definição de modelos orientados a objeto de forma simples e consistente, permitindo a utilização da quase totalidade dos conceitos de orientação a objeto. Os modelos criados possuem estrutura hierárquica, permitindo uma representação mais próxima da realidade da estrutura organizacional a qual representam.

A principal contribuição desta dissertação é a proposta de um modelo de integração entre modelagem OO e sistemas supervisórios através da incorporação de características de supervisão na ferramenta de modelagem SIMOO-RT. Para realizar este intento foram adicionadas as seguintes funcionalidades à ferramenta SIMOO-RT:

- definição de valores de alarmes (*Low*, *Low-Low*, *High*, *High-High*), prioridade e comentários para os atributos das classes, permitindo seu armazenamento em disco;
- implementação do *Supervisory Designer*, componente do SIMOO-RT que possibilita:
  - em tempo de modelagem, a definição de quatro diferentes Elementos de Visualização (EV's) para os atributos de uma classe ou para a classe como um todo;
  - a utilização dos valores de alarmes definidos na classe na definição do Elemento de Visualização;
  - a criação de múltiplas "Visões" para uma mesma classe, correspondendo a diferentes enfoques para diferentes pessoas e níveis organizacionais;
  - a criação de "Referências", que possibilitam o acesso a outras "Visões";
  - o armazenamento dos dados do SIMOO-RT, referentes a supervisão, em arquivo, permitindo sua leitura por uma ferramenta de geração de sistemas de supervisão.

Contabiliza-se, também, como contribuição, a aproximação dos pesquisadores do IEE com a empresa Elipse Software Ltda. Durante as trocas de *e-mails* e visitas, tanto na empresa quanto na Universidade, percebeu-se o interesse recíproco e a complementariedade dos trabalhos realizados por ambas.

5.1 Trabalhos Futuros

Para trabalhos futuros, pode-se realizar a expansão do *Supervisory Designer*, permitindo a definição de um repositório de Elementos de Visualização pré-definidos; o estudo da possibilidade de integração desta ferramenta com outras ferramentas de supervisão, em acréscimo a Elipse Windows e principalmente a intensificação do contato com a empresa Elipse Software, visando ao consolidação de convênios para a integração da ferramentas Elipse Windows e SIMOO-RT ou o desenvolvimento de uma ferramenta que agregue as funcionalidades de ambas.

Bibliografia

[ARO 93] ARONS, Henk de Swaan. Object Orientation in Simulation. In: EUROPEAN SIMULATION SYMPOSIUM, ESS,1993, Delft, NL. **Proceedings**... Netherlands: University of Tecnology, 1993.

[BEC 99a] BECKER, Leandro Buss. **Ambiente de Modelagem e Implementação de Sistemas Tempo Real usando o Paradigma de Orientação a Objetos.** Porto Alegre: PPGC da UFRGS, 1999.

[BEC 99b] BECKER, Leandro et al. Proposal of an Integrated Object-Oriented Environment for the Design of Supervisory Software for Real-Time Industrial Automation Systems. In: WORKSHOP ON OBJECT-ORIENTED REAL-TIME DEPENDABLE SYSTEMS, 1999, Santa Barbara, USA. **Proceedings**... [S.l.:s.n.], 1999.

[BEC 99c] BECKER, Leandro et al. An Integrated Environment for the Complete Development Cycle of an Object-Oriented Distributed Real-Time Systems. In: INTERNATIONAL SIMPOSIUM ON OBJECT-ORIENTED REAL-TIME COMPUTING, 1999, Saint-Malo, France. **Proceedings**... [S.l.:s.n.], 1999.

[BOO 91] BOOCH, G. **Object-Oriented Development**. Redwood City, CA: Benjamin/Cummings, 1991.

[COP 96] COPSTEIN, Bernardo. **Um Ambiente de Simulação Interativa Visual Orientado a Objetos.** Porto Alegre: CPGCC da UFRGS, 1996.

[DIS 2000] DISTRIBUTED Control Systems, Disponível em: < http://homepages.nildram.co.uk/~mevans/dcs0001.html> , Acesso em: jul. 2000.

[DOU 98] DOUGLASS, Bruce Powel. **Real-Time UML:** Developing Efficient Objects for Embedded Systems. Reading, Massachusetts: Addison-Wesley, 1998.

[DUO 99] DOUGLASS, Bruce Powel. **Doing Hard Time:** Developing Real-Time Sytems with UML, Objects, Frameworks and Patterns. Reading, Massachusetts: Addison-Wesley, 1999.

[ELI 98] ELIPSE. **Elipse Windows:** Sistema de Supervisão e Controle. Porto Alegre, 1998.

[HAL 91] HALANG, W.; STOYENKO, A. **Constructing Predictable Real-Time Systems**. Kluwer: Academic Publishers, 1991.

[ICO 98] ICONICS. **Genesis32 Manual**. Foxborough, MA, 1998.

[IND 2000] INDUSOFT. **Unisoft Manual**. Disponível em: < www.indusoft.com >. Acesso em: jul. 2000.

[KEL 98] KELLER, G.; TEUFEL, T. **SAP R/3 Process Oriented Implementation**. Reading, Massachusetts:Addison-Wesley, 1998.

[KIM 94a] KIM, Kane et al. A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials. In: COMPSAC, 1994, Taipei. **Proceedings**... [S.l.:s.n.], 1994. p. 392-402.

[KIM94b] KIM, Kane et al. Distinguishing Features and Potential Roles of the RTO.k Object Model. In: WORKSHOP ON OBJECT-ORIENTED REAL-TIME DEPENDABLE SYSTEMS, 1994, Dana Point. **Proceedings**... [S.l.:s.n.], 1994. p. 36-45.

[KIM 96] KIM, Kane et al. The DREAM Library Support for PCD and RTO.k programming in C++. In: WORKSHOP ON OBJECT-ORIENTED REAL-TIME DEPENDABLE SYSTEMS, 1996, Laguna Beach. **Proceedings**... [S.l.:s.n.], 1996. p. 59-68.

[KIM 95] KIM, Kane et al. A Timeliness-Guaranteed Kernel Model - DREAM Kernel - and Implementation Techiniques. In: RTCSA, 1995, Tokyo. **Proceedings**... [S.l.:s.n.], 1995. p. 80-87.

[MAR 95] MARTINS, James, ODELL, James J. **Análise e Projeto Orientados a Objeto.** São Paulo: Makron Books, 1995.

[MEY 88] MEYER, Bertrand. **Object Oriented Software Construction.** New York: Prentice-Hall, 1988.

[NAT 99a] NATIONAL INSTRUMENTS. **LookOut Manual**. Disponível em: < www.natinst.com > . Acesso em: set. 1999.

[NAT 99b] NATIONAL INSTRUMENTS. **BridgeVIEW Manual**. Disponível em: < www.natinst.com > . Acesso em: set. 1999.

[OBJ 99] OBJECTIME. **Object Time Manual**. Disponível em: < www.objectime.com >. Acesso em: ago. 1999.

[PAR 2000] PARDI JUNIOR, W.; PEREIRA, C. **Tool Support to the Generation of Supervisory Control Software from Object-Oriented Specifications of Real-Time Industrial Automation Systems.** Porto Alegre: CPGCC da UFRGS, 2000.

[PAR 98] PARDI JUNIOR, Wilson. **Simulação e Visualização de Plantas Industriais usando Técnicas de Orientação a Objetos**. Porto Alegre: IEE/UFRGS, 1998. Relatório parcial de bolsa DTI.

[PER 94a] PEREIRA, Carlos; DARSCHT, Pablo. Using Object-Orientation in Real-Time Applications: An Experience Report. In: TOOLS EUROPE, Versailles, França. **Proceedings...** [S.l.:s.n.], 1994.

[PER 94b] PEREIRA, Carlos. Real Time Active Objects in C++/Real-Time UNIX. In: ACM SIGPLAN WORKSHOP ON LANGUAGES, COMPILER, AND TOOL SUPPORT FOR REAL-TIME SYSTEMS, Orlando, EUA. **Proceedings...** [S.l.:s.n.], 1994.

[PER 94c] PEREIRA, Carlos. Enhancing Real-Time Object-Oriented Analysis Methods. In: EUROMICRO WORKSHOP ON REAL-TIME PROGRAMMING, 6., Vaesteraas, Sweden. **Proceedings...** [S.l.:s.n.], 1994. p. 104-109.

[PER 95] PEREIRA, Carlos. Temporal Reasoning on Object-Oriented Real-Time Specifications by using Constraint Propagation Techniques. In: IFAC/IFIP WORKSHOP ON REAL-TIME PROGRAMMING, 20., FL-USA. **Proceedings...** [S.l.:s.n.], 1995.

[PER 96a] PEREIRA, Carlos; FRIGERI, A.; DARSCHT, P.; HALANG, W. Object-Oriented Development of Real-Time Industrial Automation Systems. In: IFAC Triennial World Congress, San Francisco, USA. **Proceedings...** [S.l.:s.n.], 1996.

[PER 96b] PEREIRA, Carlos. Métodos de Análise de Sistemas Tempo-Real Usando Técnicas de Orientação a Objetos. SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, São Carlos, SP. **Proceedings...** [S.l.:s.n.], 1996.

[RAT 99] RATIONAL ROSE CORP. **UML Notation Guide Versão 1.1**. Disponível em: < www.rational.com >. Acesso em: ago. 1999.

[RUM 94] RUMBAUGH, James et al. **Modelagem e Projetos Baseados em Objetos**. Rio de Janeiro: Campus, 1994.

[SAP 98a] SAP AG. **PI-PCS Interface**. Disponível em: < www.sap.com >. Acesso em: set. 1998.

[SAP 98b] SAP AG. **SAP APO: Advanced Planner & Optimizer**. Disponível em: < www.sap.com >. Acesso em: set. 1998.

[SEL 94] SELIC, Bran et al. **Real Time Object Oriented Modelling**. [S.l.]: John Wiley and Sons, 1994.

[SHL92] SHLAER, S.; MELLOR, S. **Object Life Cycles**: Modeling the World in States. Englewood Cliffs, NJ: Yourdon Press, 1992.

[STR 90] STROUSTROUP, Bjarne; ELLIS, Margaret A. **Anottated C++:** Reference Manual. Reading, Massachusetts: Addison Wesley, 1990.

[TIB 99] TIBOLA, Leandro Rosniak. **Avaliação de Possibilidade de Geração de Sistemas Supervisórios a Partir de Modelos Orientados a Objetos de Aplicações Industriais**. Porto Alegre: CPGCC da UFRGS, 1990. (TI - 808).

[WIL 94] WILHELM, Bob. Structural Implications of Concurrent Execution Models for OO Real Time Systems. In: OOPSLA, 1994, Portland, USA. **Proceedings...** [S.l.:s.n.], 1994.

[WIL 98] WILD, Rafael e PEREIRA, Carlos. Contribuição à Avaliaçã de Características Tempo-Real em Barramentos de Campo. In: SIMPÓSIO DE SISTEMAS DE SIMULAÇÃO E CONTROLE, 1., 1998, Rio de Janeiro. **Anais...** [S.l.:s.n.], 1998. p. 13-18.

[YOU 89] YOURDON, Edward. **Análise Estruturada Moderna.** Rio de Janeiro: Campus, 1990.