JAIR JONKO ARAUJO

INTERFACE DE COMUNICAÇÃO ENTRE AMBIENTE DE MODELAGEM E SISTEMA SUPERVISÓRIO

Monografia apresentada ao Curso de Bacharelado em Informática do Instituto de Física e Matemática da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Bacharel em Informática.

Ênfase: Sistemas de Computação

Orientador: Leandro Buss Becker, MSc.
UFRGS
Co-orientador: Prof. Gil Carlos R. Medeiros, MSc.
UFPEL

Pelotas - RS
2000

A663 Araujo, Jair Jonko

Interface de comunicação entre ambiente de modelagem e sistema supervisório / Jair Jonko Araujo. – 2000. – 62p.

Monografia (bacharelado) – Universidade Federal de Pelotas. Instituto de Física e Matemática, 2000.

1. Informática 2. Sistemas de computação 3. Interface de comunicação I. Título

CDD: 001.6

Catalogação CRB 10/926

A meus pais pela oportunidade da vida, pelo amor e pela dedicação.

Agradecimentos

Gostaria fazer um agradecimento especial ao meu orientador, Leandro Buss Becker, pela atenção e dedicação na orientação deste trabalho, atendendo prontamente as todas as minhas dúvidas e indicando variado material de estudo e fornecendo suporte em cada etapa do trabalho. Gostaria de agradecer também ao prof. Carlos Eduardo Pereira por atendido prontamente ao meu pedido de oportunidade de trabalho junto ao Grupo de Automação da UFRGS, permitindo-me a escolha do tema do trabalho e fornecendo-me as orientações iniciais. Também gostaria de agradecer ao meu co-orientador prof. Gil Carlos Medeiros, não só pela orientação na estrutura da monografia, mas por todo o apoio que recebi durante o curso e na fase de seleção do trabalho de conclusão.

Finalmente, deixo um agradecimento a tantos, não citados nominalmente, mas sem cuja colaboração, este trabalho não teria sido realizado, como marco de conclusão do curso de Bacharelado em Informática, especialmente a meus colegas de trabalho no CEFET/RS que sempre criaram meios para que houvesse tempo para realizar as atividades relacionadas ao curso e ao trabalho.

Sumário

1 Introdução 11
2 Sistemas Supervisórios 14
2.1 Sistemas SCADA 14
3 A Modelagem Orientada a Objeto e o SIMOO-RT 19
3.1 Modelagem Orientada a Objeto 19
3.2 SIMOO-RT 21
3.2.1 O Diagrama de Classes 22
3.2.2 O Diagrama de Instâncias 23
3.2.3 Editor de Diagrama de Transição de Estados 23
3.3 Supervisory Designer 24
4 Arquitetura COM 27
5 Projeto e Implementação da Interface 33
5.1 Modificações no SIMOO-RT 33
5.2 Definição e implementação da interface e do objeto servidor 37
5.2.1 Definição 37
5.2.2 Implementação 38
6 Validação da Interface 44
6.1 Implementação do cliente COM 44
6.2 Estudo de Caso 48
7 Conclusões 53
Anexo1 54
Glossário 58
Bibliografia 60

Lista de Abreviaturas ou Siglas

AO/C++	*Active Object/C++*;
API	*Application Program Interface*;
CASE	*Computer Aided Software Engineering*;
CLP	Controlador Lógico Programável;
CLSIDs	*Class Identifier*;
COM	*Component Object Model*;
CORBA	*Object Management Group's Common Object Request Broker Architecture*;
DCE	*OSF's Distributed Computing Environment* (atualmente *The Open Group's*;
DCOM	*Distributed Component Object Model*;
DCS	*Distributed Control Sistems*;
DLL	*Dinamic Link Libraries*;
EV	Elemento de Visualização;
EVBarGraph	Elemento de Visualização tipo *Bar Graph*;
EVBitAnim	Elemento de Visualização tipo *Bitmap Sequence*;
EVDisplay	Elemento de Visualização tipo *Display*;
EVGauge	Elemento de Visualização tipo *Gauge*;
EXE	*Executable Program*;
GUIDs	*Globally Unique Identifiers*;
HTTP	*Hypertext Transfer Protocol*;
I/O	*Input/Output*;
IDL	*Interface Definition Language*;
IID	*Interface Identifier*;
IPX/SPX	*Internetwork Packet Exchange*;
LAN	*Local Area Network*;
MET	*Model Editor Tool*;
MIDL	*Microsoft's Interface Definition Language*;
NDR	*Network Data Representation*;

NT	*Microsoft's New Technology*;
OLE	*Object Link Embedding*;
ORPC	*Object Remote Procedure Call*;
OSF	*Open Software Foundation* (atualmente *The Open Group)*;
PCS	*Process Control Systems*;
PLC	*Programmable Logic Controller*;
RPC	*Remote Procedure Call*;
RT	*Real-Time*;
SCADA	*Supervisory Control and Data Aquisition Systems*;
SCM	*Service Control Manager*;
SIMOO	Não é uma sigla e sim o nome da ferramenta (lembra "simulação orientada a objetos") ;
STDE	*State and Transitions Diagram Editor*;
TCP/IP	*Transmission Control Protocol / Internet Protocol*;
UDP	*User Datagram Protocol*;
WAN	*Wide Area Network*;

Lista de Figuras

1.1 - Relação entre elementos de um sistema de industrial..... 12
2.1 - Tela típica de um sistema supervisório..... 15
2.2 - Estrutura física de um sistema de automação..... 16
3.1 - Diagrama de classes do SIMOO RT..... 22
3.2 - Diagrama de instâncias do SIMOO RT..... 23
3.3 - Visão do diagrama de transição de estados SIMOO-RT..... 24
3.4 - O Supervisory Designer do SIMOO-RT..... 26
4.1 - Visão geral do empacotamento dos objetos..... 30
4.2 - Arquitetura DCOM..... 31
5.1 - Janela anterior a modificação..... 34
5.2 - Janela atual..... 35
5.3 - Janela CreateEV..... 35
5.4 - Janelas de edição dos EVs (EVGauge, EVDisplay, EVBitAnim, EVBarGraph) 36
5.5 - Objetos comuns aos Evs..... 38
5.6 - Objetos específicos dos Evs..... 39
5.7 - Definição da Interface no editor de Type Library..... 40
5.8 - Implementação da propriedade do objeto servidor..... 41
5.9 - Seleção da Interface com o usuário..... 42
5.10 - Janela de Exportação dos EVs..... 43
6.1 - Janela principal do software supervisório de teste..... 44
6.2 - Janela de edição de telas..... 45
6.3 - Janela de criação de EVs..... 46
6.4 - Código da classe de criação de objetos COM..... 47
6.5 - Código parcial dos botões *COM*, *Load* e *ToMod*..... 48
6.6 - Classes atributos e EVs do estudo de caso..... 49
6.7 - Atributos e EVs (vista parcial das classes Sensor e SenNivel) 50
6.8 - Atributos e EVs (vista parcial das classes Bomba e Tanque) 50
6.9 - Transferência de dados entre o SIMOO-RT e o supervisório de teste..... 51
A1.1 - Representação gráfica de uma classe..... 54

A1.2 - Abordagens para a composição de um paradigma de simulação..... 54
A1.3 - Subdiagrama..... 55
A1.4 - Relacionamentos de conhecimento..... 55
A1.5 - Tipos de pontos de conexão..... 56
A1.6 - Relacionamento de criação (GERADOR/CLIENTE) 57
A1.7 - Elementos gráficos do diagrama de instâncias..... 57

Resumo

O presente trabalho tem como objetivo definir e implementar uma interface de comunicação entre o ambiente SIMOO-RT e um sistema supervisório. O SIMOO-RT é um ambiente integrado para modelagem, simulação e implementação de sistemas de tempo real distribuídos, que faz uso do paradigma de orientação a objetos. Este trabalho integra uma dissertação de mestrado, em desenvolvimento no Laboratório de Automação da UFRGS, que prevê a descrição de propriedades voltadas para a geração de sistemas supervisórios. O objetivo do mesmo é permitir integração do SIMOO-RT com sistemas supervisórios, que são softwares que permitem monitorar e controlar, em tempo de execução, um determinado sistema de controle ou automação. Esta interface será implementada utilizando-se a tecnologia COM da Microsoft. Com a sua utilização, será possível a construção de sistemas supervisórios a partir dos elementos que foram definidos no ambiente de modelagem, garantindo a consistência entre o sistema implantado e o sistema que foi modelado.

Palavras chaves: Supervisório, SCADA, COM, DCOM, Modelagem Orientada a objetos, SIMOO-RT.

1 Introdução

A necessidade do aumento de produção, da redução de custos e do aumento da qualidade de bens de consumo, tem motivado a constante evolução dos sistemas de produção. Isto tem sido caracterizado, principalmente, pela evolução tecnológica dos equipamentos, a qual é possível pela contínua melhoria dos componentes eletro-eletrônicos, principalmente os ligados à micro-eletrônica.

Equipamentos autônomos, interdependentes, ligados entre si formando redes e fluxo contínuo de produtos diferentes pelas linhas de produção tornam os equipamentos e as plantas de produção cada vez mais difíceis de serem projetados, especificados e administrados.

Os sistemas computacionais que permitem o controle desses dispositivos também crescem em complexidade. Uma das principais características dos sistemas de automação industrial é o alto grau de acoplamento desses sistemas com os componentes situados na planta. O atendimento das restrições de tempo e a dinâmica do comportamento dos componentes situados na planta são fundamentais na operação correta do sistema.

A fim de monitorar e controlar, em tempo de execução, um sistema já implementado são usados sistemas supervisórios, os quais permitem, por exemplo, a visualização e até mesmo a mudança de parâmetros de controle por usuários do sistema. A visualização dos parâmetros do processo monitorado pode ser mapeada para diferentes níveis hierárquicos dentro de uma fábrica, variando desde a monitoração de instrumentos no chão de fábrica até a coleta de informações estatísticas a serem usadas a nível gerencial.

Estudos de casos [BEC99b e TIB99] demonstram que um dos maiores problemas nas ferramentas de modelagem existentes é a fraca interação entre modelagem, visualização e supervisão. O projetista geralmente modela e simula o sistema em uma determinada ferramenta, identificando e corrigindo os erros de projeto. Entretanto, quando este for implantado, o supervisório será construído sem nenhum

vínculo com as definições do sistema modelado, o que poderá acarretar incoerência entre a modelagem e a supervisão. Este fato motivou o desenvolvimento de um ambiente para modelagem e geração de sistemas supervisórios, descritos a partir do SIMOO-RT [BEC99a], através de uma Dissertação de Mestrado que se encontra em andamento no Instituto de Informática da Universidade Federal do Rio Grande do Sul.

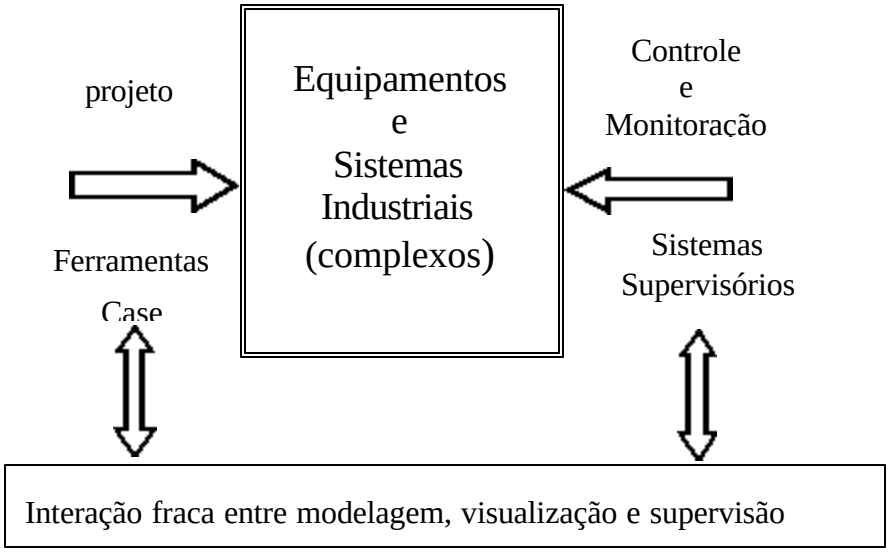

Figura 1.1- Relação entre elementos de um sistema de industrial.

O objetivo principal deste trabalho é definir e implementar uma interface capaz de permitir a interação entre o SIMOO-RT e um sistema supervisório. Assim, a partir da especificação do modelo, os elementos de visualização especificados na fase de modelagem também poderão ser utilizados no projeto do sistema supervisório. De maneira reversa, as alterações dos parâmetros dos elementos de visualização que forem efetuadas no supervisório, devem gerar atualizações dos mesmos no modelo.

Esta interface de comunicação deve ser implementada utilizando um mecanismo de interação entre objetos independente de linguagem de programação e de sistema operacional. Por isso a utilização da arquitetura COM da Microsoft é fundamental, por ser um mecanismo bastante aceito pelo mercado de software na atualidade.

A fim de validar a interface proposta, também foi construído um sistema supervisório contendo as funcionalidades mínimas necessárias para agregar e editar elementos de visualização. Este sistema é capaz de utilizar as funcionalidades da interface para atualizar, no ambiente de modelagem, as modificações feitas pelo projetista nos elementos de visualização a partir do supervisório.

Este trabalho está organizado da seguinte maneira:

- O Capítulo 2 define os Sistemas Supervisórios, apresentando suas principais características e funcionalidades;
- No Capítulo 3 é feita uma breve descrição do paradigma de orientação a objetos, juntamente com a apresentação das principais características da ferramenta SIMOO-RT, na qual se baseia o presente trabalho;
- O Capítulo 4 apresenta a tecnologia COM, definindo sua arquitetura e funcionamento;
- O Capítulo 5 trata da definição e do desenvolvimento da interface entre o SIMOO-RT e o supervisório, apresentando as etapas que foram executadas para realização do objetivo proposto;
- O Capítulo 6 trata da validação dos dados gerados através dessa interface, com a sua correta agregação nos dois sistemas que estão sendo comunicados, e
- Finalmente, o Capítulo 7 apresenta uma análise dos resultados obtidos e as conclusões deste trabalho.

2 Sistemas Supervisórios

A evolução dos equipamentos industriais, com a introdução crescente de sistemas de automação industrial, torna complexa a tarefa de monitorar, controlar e gerenciar esses sistemas produtivos. Diferentes arquiteturas de sistemas computacionais tem sido desenvolvidas e propostas para esse gerenciamento.

Dentre os muitos sistemas usados para controlar e gerenciar a produção, os que mais tem 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 Sistems*).

Na próxima seção são apresentadas as principais características dos sistemas SCADA, uma vez que estes se encontram inseridos no contexto deste trabalho.

2.1 Sistemas SCADA

Os sistemas de supervisão, ou simplesmente SCADA, permitem que sejam monitoradas e rastreadas informações do processo produtivo. Tais informações são primeiro coletadas através de equipamentos de aquisição de dados, seguido da manipulação e análise destes dados e posteriormente são apresentadas ao usuário. Estas informações podem ser visualizadas por intermédio de quadros sinóticos animados, com indicações instantâneas das variáveis de processo (temperatura, pressão, contagem, etc), e armazenadas em bases de dados relacionadas ao processo do cliente. As análises dos dados podem ser feitas dentro do supervisório através de tabelas e gráficos de tendência ou fora dele pelos softwares comerciais comuns como Access, Excel, etc.

Outra função importante é a possibilidade do sistema supervisório executar ações baseadas em parâmetros antecipadamente informados, fazendo que usuário participe do processo de controle apenas quando as ações exijam a intervenção humana. A figura 2.1 mostra uma tela típica de um sistema supervisório.

Até aproximadamente 1998, os principais fornecedores de softwares de supervisão eram empresas de software, como é o caso da Wonderware, Iconics, Intellution, etc. Atualmente alguns fabricantes de hardware entraram no mercado, como é o caso da G.E. Fanuc, Allen-Bradley, Siemens e etc.

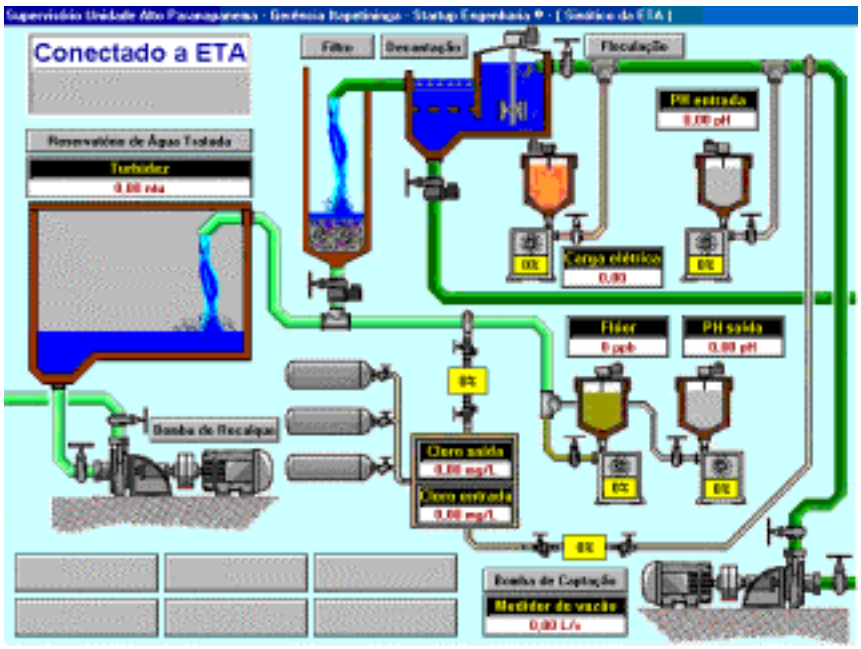

Figura 2.1 - Tela típica de um sistema supervisório

Geralmente, a estrutura física que cerca um sistema supervisório apresenta três camadas, conforme pode ser observado na figura 2.2.

Na primeira situam-se os equipamentos industriais (motores, sensores, leitores de código de barras, etc.), os quais geralmente estão conectados a um Controlador Lógico Programável (CLP), que é um equipamento microprocessado capaz de gerenciar o acionamento destes componentes, que são ligados em placas de entradas e saída, executando um software de controle armazenado em memória. Esta comunicação é feita através de protocolos comunicação de baixo nível e exige o cumprimento de certos requisitos para ser efetivada com sucesso. Temos como exemplo nesta camada, a comunicação usando protocolos *Modebus*, *Fieldbus Fundation*, *Profibus*, entre outros.

Na segunda camada ocorre a comunicação entre o(s) CLP(s) (ou somente os sensores e atuadores, como no caso do *Fieldbus*, por exemplo) e o microcomputador, no qual o sistema supervisório está sendo executado. Neste ponto, os dados são agrupados

por equipamento pelo CLP, havendo identificação inequívoca destes 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 da placas *Ethernet*. Existe nesta camada, uma relação direta do tipo de CLP com o sistema supervisório e 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, comunicação com outras aplicações entre outras configurações, são necessárias para que CLP e microcomputador funcionem corretamente.

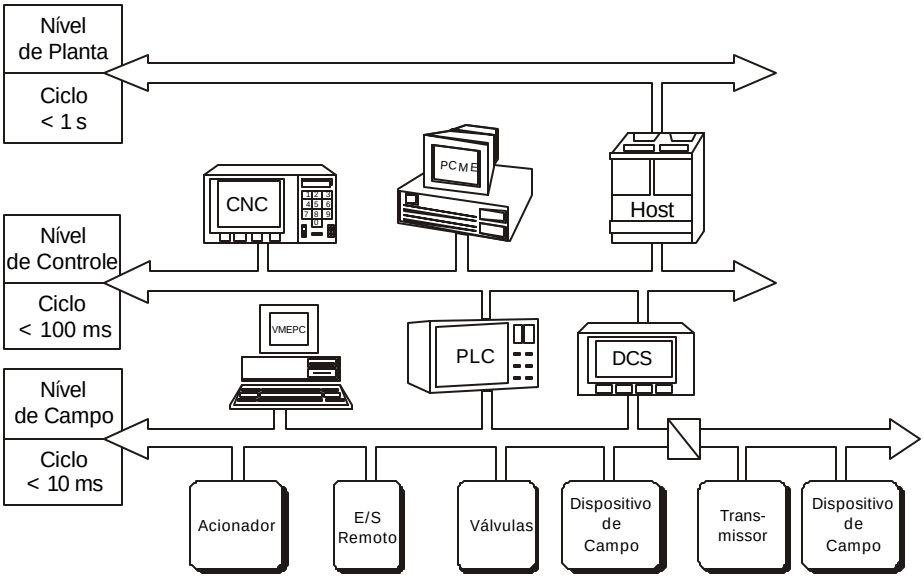

Figura 2.2 - Estrutura física de um sistema de automação

Na terceira camada situam-se o microcomputador com o sistema supervisório, os demais sistemas da industria (administrativos, de fornecedores, de parceiros, etc.) e o acesso externo à organização. A interconexão do sistema supervisório com outros sistemas, nesta camada, é possível 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.

A partir da estrutura física é possível definir os pontos de conexão entre os dispositivos e o sistema supervisório, denominados *Tags*, que 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.

As *Tags* são todas as variáveis numéricas ou alfanuméricas envolvidas na aplicação, podendo executar funções computacionais (operações matemáticas, lógicas, com vetores ou *strings*, etc.) ou representar pontos de entrada e saída de dados do processo que está sendo controlado. Neste caso, correspondem às variáveis do processo real (como temperatura, vazão, nível, etc.), se comportando como a ligação entre o controlador e o sistema. Seus tipos variam dependendo do CLP e do sistema supervisório que estão sendo utilizados. As *Tags* mais freqüentes são aquelas usadas para comunicação entre CLP e o sistema e as que permitem realizar operações aritméticas, manipular 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. Apresentação destes gráficos é alterada de acordo com a variação dos valores das *Tags*. Como algumas 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.

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. Há necessidade de programar ações para as situações ocorridas no processo que podem ser realizadas pelo sistema supervisório, através da execução de uma linguagem de programação, embutida no próprio programa, conhecida como *Scripts.* Esta linguagem geralmente é proprietária, mas segue um conjunto de comandos de uma linguagem de programação conhecida, como o *Basic* ou o C.

Os *Scripts* permitem uma flexibilidade muito grande aos supervisórios, pois possibilitam maior proximidade com operações de baixo nível e acesso à dispositivos e sistema operacional. Geralmente estão associados às variações das *Tags*, podendo ser executados quando o seu valor mudar ou estiver em situação de alarme.

Uma situação de alarme é identificada quando o valor da *Tag* ultrapassa uma faixa pré-estabelecida. Esta faixa possui dois valores para o limite superior e dois valores para o limites inferior, sendo denominados altíssimo, alto, baixo e baixíssimo (*High-High, High, Low* e *Low-Low*). 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.

As variações dos valores das *Tags*, inicialmente visualizadas, podem também ser arquivados em disco, formando um histórico das atividades do sistema. Possuir um histórico do sistema é útil por 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 próximo da criação de históricos do sistema está a conexão com bancos de dados. Conectar o sistema supervisório a um banco de dados permite a recuperação de informações do sistema de forma mais rápida e segura, além de permitir a utilização de informações corporativas no processo que está sendo monitorado.

3 A Modelagem Orientada a Objetos e o SIMOO-RT

Este capítulo tem por finalidade elucidar os principais conceitos da modelagem orientada a objetos e apresentar o ambiente SIMOO-RT, que será usado no contexto deste trabalho. Posteriormente será apresentado o módulo *Supervisory Designer* que se encontra em desenvolvimento e do qual o presente trabalho é parte integrante.

3.1 Modelagem Orientada a Objetos

Segundo Rumbaugh [RUM97], "um modelo é uma abstração de alguma coisa, cujo propósito é permitir que se conheça essa coisa antes de construí-la". Constrói-se modelos para representar aspectos da realidade, a fim de executar projetos de maneira mais rápida e eficiente. Em todas as áreas pode-se utilizar modelos, citando-se como exemplo os modelos arquiteturais, as miniaturas de aviões e automóveis para testes em túnel de vento, esboços a lápis para pintura entre outros.

O modelo é construído com ênfase no aspecto da realidade que se está interessado em entender melhor. Assim ele é mais simples do que a realidade, omitindo os detalhes que não interessam ao propósito em estudo.

A abordagem baseada em objetos preocupa-se primeiro em identificar os objetos, que são entidades discretas e distintas, contidos no domínio da aplicação e depois estabelecer procedimentos relativos a eles. Convém salientar que algumas propriedades de um sistema podem ser mal representadas pelos modelos, o que é considerado normal, pois nenhuma abstração pode incluir tudo e não se deve sobrecarregar o modelo com construções que o tornem pesado, pois o que se busca na modelagem é exatamente simplificar a realidade.

Na modelagem de sistemas orientados a objetos, o autor citado caracteriza diversos elementos que compõem um modelo orientado a objetos, dos quais destaca-se dois. O primeiro é o diagrama de classes cujos nós são as classes e os arcos são os relacionamentos entre as mesmas, representando o comportamento estático do modelo,

e o segundo é o diagrama de estados cujos nós são estados e os arcos são transições entre os mesmos, causadas pelos eventos, representando o comportamento dinâmico do modelo.

O paradigma de orientação a objetos é uma técnica bastante popular no desenvolvimento de sistemas de propósito geral. Conforme estudos citados por Becker [BEC 98] este paradigma também se mostra bastante adequado para descrever sistemas de tempo-real distribuídos, como por exemplo os sistemas de automação industrial. O autor destaca como vantagens da utilização desse paradigma o fato de que os "objetos permitem estruturar a informação de uma forma lógica, aumentando a tratabilidade de sistemas complexos". Cita ainda outros benefícios:

- **Fácil manutenção e compreensão:** os modelos orientados a objetos permitem um mapeamento direto da semântica do domínio do problema para o modelo a ser desenvolvido o que é altamente benéfico, uma vez que os objetos do modelo podem representar e comportarem-se como os elementos complexos do mundo real.
- **Modularidade:** as classes/objetos identificáveis permitem um particionamento lógico do domínio do problema. Cada uma das classes de objetos representa um pedaço próprio no domínio do problema que pode ser examinado separadamente ou dentro do contexto do sistema.
- **Concorrência:** objetos são unidades naturais para execução concorrente, permitindo que aplicações do mundo real com concorrência intrínseca sejam modeladas naturalmente.
- **Melhor gerenciamento da complexidade:** técnicas de desenvolvimento orientadas a objetos fornecem conceitos de abstração poderosos para tratar com a complexidade. Os diferentes tipos de hierarquia (generalização/especialização, agregação e serviços hierárquicos) facilitam a representação e o gerenciamento em vários níveis de detalhes.

Existem diversas ferramentas de modelagem orientadas a objetos em âmbito acadêmico e comercial. Estas ferramentas, conhecidas como ferramentas 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, 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, as ferramentas CASE agregam 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.

3.2 SIMOO-RT

Um exemplo de ferramenta CASE, a qual o presente trabalho tem por base, é o SIMOO-RT, resultado de uma Dissertação de Mestrado do Instituto de Informática da Universidade Federal do Rio Grande do Sul [BEC99a]. Este trabalho incorporou uma extensão para tempo-real, na ferramenta SIMOO que foi desenvolvido no âmbito de uma tese de Doutorado [COP97]. Esta ferramenta tem como objetivo principal a criação de modelos orientados a objetos, com recursos para permitir a simulação do modelo.

Os modelos são construídos no editor de modelos (denominado MET – *Model Editor Tool*) e são compostos por diversos diagramas tais como o diagrama de classes, o diagrama de instâncias e o diagrama de transição de estados. Este último formaliza a especificação dos possíveis estados de execução das classes/objetos e dos eventos que provocam as transições entre estes estados.

Desta forma, o editor suporta a descrição de aspectos estáticos e dinâmicos do modelo. Os aspectos estáticos são descritos graficamente, através dos diagramas de classe e de instâncias, enquanto que o comportamento dinâmico de cada entidade é descrito, nos diagramas de transição de estados e em *templates* na linguagem C++. A partir da descrição do modelo, o editor pode gerar automaticamente o código executável necessário para a simulação e também para a aplicação final.

A partir do modelo criado é possível gerar código para uma aplicação final, na linguagem AO/C++. Esta linguagem combina as propriedades de orientação a

objetos existente em C++ com os benefícios dos sistemas operacionais UNIX de tempo real (UNIX-TR) orientado a processos, como por exemplo a plataforma QNX.

A seguir, serão descritos o diagrama de classes, o diagrama de instâncias, o diagrama de transição de estados e o *supervisory designer* que são elementos importantes para execução do trabalho proposto.

3.2.1 O Diagrama de Classes

As classes e os seus relacionamentos são criadas e descritas no diagrama de classes. Os modelos são construídos de maneira hierárquica, sendo que diversos tipos de associações entre as classes podem ser identificados, tais como agregação, herança, etc. Classes definidas em um determinado nível são consideradas "componentes agregados" da classe de mais alto nível que está sendo detalhada, assim um modelo é representado por uma única classe que inclui todas as demais. Tal estratégia de modelagem tem como objetivo uma reutilização mais fácil de entidades, assim também como um refinamento progressivo quando necessário. A figura 3.1 mostra um típico diagrama de classes.

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.

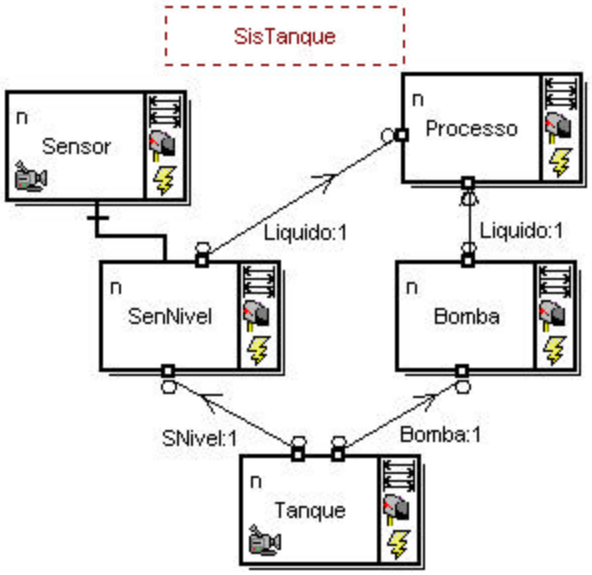

Figura 3.1- Diagrama de classes do SIMOO-RT.

3.2.2 O Diagrama de Instâncias

Este editor permite a especificação da estrutura estática das entidades do modelo, em tempo de execução. Através deste diagrama podem ser eliminadas as ambigüidades presentes em diagramas que incluem somente classes, pois ele especifica as instâncias, ou objetos, dos elementos descritos no diagrama de classes, bem como define as relações de comunicação entre eles.

Uma característica especial do SIMOO-RT é que o usuário sempre deve criar um diagrama de instâncias, a partir do qual o modelo de simulação ou de execução será gerado. A necessidade de um diagrama de instâncias é baseada no fato de que entidades de execução concretas correspondem a objetos, e não a classes. A figura 3.2 mostra um exemplo de diagrama de instâncias.

Cada retângulo especifica um elemento que foi descrito genericamente no diagrama de classes. Pode haver tantas instâncias da mesma classe quanto necessárias para construir o modelo. 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.

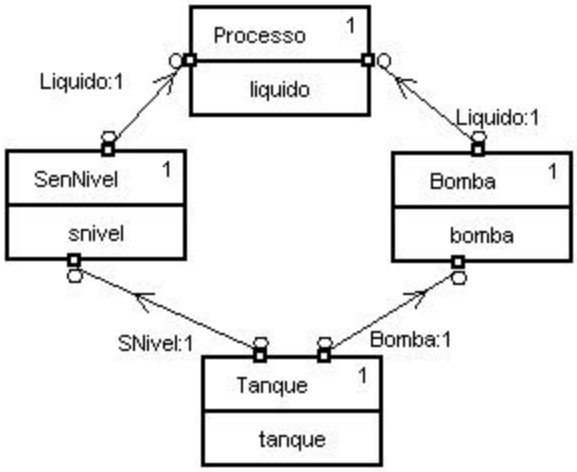

Figura 3.2 - Diagrama de instâncias do SIMOO-RT.

3.2.3 Editor de Diagrama de Transição de Estados

A ferramenta STDE permite a descrição dos possíveis estados de execução das classes/objetos e os eventos que provocam as transições entre estes estados. São

possíveis três tipos de estados: estado inicial, estado final ou estado normal. Aos estados do tipo normal é permitido associar uma seqüência de atividades, que são executadas assim que ocorrer a transição para o mesmo. Não é possível associar atividades aos estados inicial e final, pois eles são associados ao método de criação e de destruição da instância. A figura 3.3 mostra um exemplo de um diagrama de transição de estados.

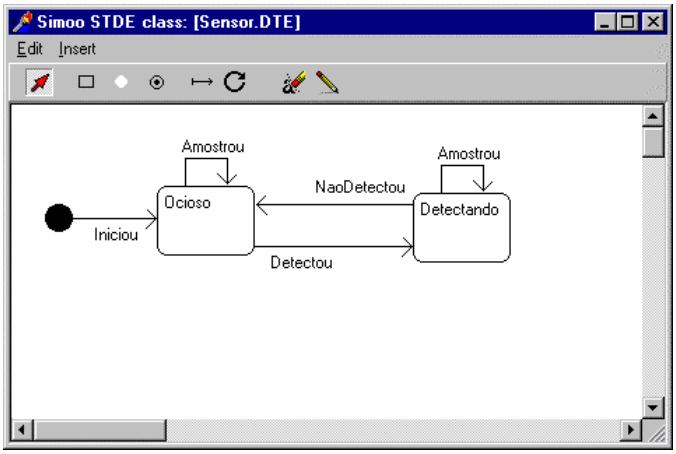

Figura 3.3 - Visão do diagrama de transição de estados SIMOO-RT.

3.3 *Supervisory Designer*

Uma expansão importante ao ambiente SIMOO-RT que atualmente se encontra em construção é o *Supervisory Designer* [TIB00]. O objetivo principal deste trabalho é permitir a utilização de modelos orientados a objetos descritos na ferramenta SIMOO-RT na criação de sistemas supervisórios.

Para tanto é necessário definir, em tempo de modelagem, propriedades úteis para a construção de um sistema supervisório. Como exemplo destas propriedades, cita-se os limites para os atributos das classes e a possibilidade de escolher a forma de sua representação no sistema supervisório.

O elemento básico de sua estrutura é o Elemento de Visualização (EV). Um EV representa a possibilidade de apresentação gráfica de um atributo, de um estado, de um método ou mesmo uma classe.Um ou mais EVs formam uma janela no supervisório.

Como citado acima, pode-se especificar limites para um atributo, os quais serão usados como campos indicadores de alarmes, sendo usados para posterior

comparação com um dispositivo de aquisição de dados. Foram definidos os seguintes campos para indicação de alarmes:

- *Low-Low*: que contém o valor usado como Mínimo do Baixo e representa uma situação extrema descendente;
- *Low*: contém o valor usado como Baixo e representa uma situação intermediária descendente;
- *High*: contém o valor usado como Alto e representa uma situação intermediária ascendente;
- *High-High*: contém o valor usado como Máximo do Alto e representa uma situação extrema ascendente.

Para todos os alarmes podem ser indicados um comentário e um grau de prioridade (a qual o sistema deve responder com maior ou menor rapidez). Estes dois últimos elementos não são verificados no modelo, ficando a cargo do sistema supervisório de destino a validação dos mesmos.

Os EVs que podem ser definidos na modelagem são *Gauge*, *Display*, *Bar Graph* ou *Bitmap Animation*.

Um EV do tipo *Bitmap Animation* têm por finalidade permitir a criação de animação no supervisório. Um conjunto de imagens *bitmaps* podem ser apresentados em uma seqüência previamente definida no EV. A alternância das imagens é condicionada a mudança de um intervalo de valores e os atributos desse EV permitem armazenar informações relativas a apresentação do EV no programa supervisório destino, tais como descrição, transparência, cores, borda e *refresh*.

Um EV do tipo *Bar Graph* tem por finalidade possibilitar a criação de elementos que possam ser representados como um gráfico de barras. Ela permite a definição da orientação da barra, cores, título, grade, etc.

Um EV do tipo *Display* possui a função de apresentar valores lidos diretamente de um coletor de dados ou simulados pelo sistema, apresentando-os de três formas: texto simples, o próprio valor numérico ou como data/hora. Este EV conta com atributos para alinhamentos horizontal e vertical, formato de apresentação (texto, numérico ou data), tamanho, precisão, prefixo e sufixo.

Um EV do tipo *Gauge* possui atributos para valores máximo e mínimo, configuração cores, quantidade e visibilidade de vários elementos gráficos (agulha, marcas, fundo, fonte, legenda, Alarmes LoLo, Lo, Hi, HiHi e Normal), rotação do Gauge e suas cores. Este elemento gráfico, além de apresentar valores máximo e mínimo, presentes nos anteriores, e possui ainda, referência aos alarmes *Low*, *LowLow*, *High* e *HighHigh*.

Além dos atributos citados em cada EV, existem diversos atributos que são comuns a todos os EVs tais como relacionados diretamente com a posição, tamanho, visibilidade e características do título na apresentação do elemento de visualização.

A ferramenta permite acessar os atributos definidos para as classes, especificar valores para alarmes associados a estes atributos e ligá-los a um elemento de visualização. Quando um EV for definido para um atributo será criado um objeto que conterá o nome da classe, os atributos da classe, os dados dos alarmes para os atributos da classe e os valores para cada tipo específico de EV, o qual poderá ser usado posteriormente na implementação do supervisório para o sistema de automação que está sendo modelado.

Na figura 3.4 podemos ter uma visão geral do *supervisory designer*.

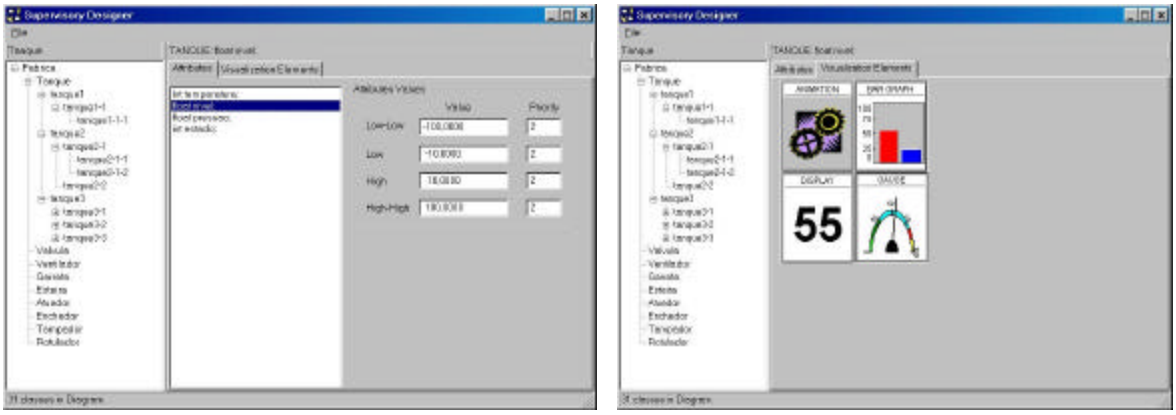

Figura 3.4 - O *Supervisory Designer* do SIMOO-RT.

4 Arquitetura COM

O presente capítulo trata da arquitetura COM e de sua extensão para ambientes distribuídos, denominada DCOM. Esta tecnologia foi utilizada na implementação da interface de comunicação entre o SIMOO-RT e sistemas supervisórios.

À medida que a complexidade dos sistemas computacionais aumenta, os softwares passam a ter suas funções distribuídas em diferentes componentes, aumentando a flexibilidade e a reusabilidade, reduzindo os custos de desenvolvimento e manutenção.

Surge, assim, a necessidade de criar mecanismos que permitam a esses diferentes componentes, instalados na mesma máquina ou em computadores diferentes, comunicarem-se entre si. Diferentes arquiteturas têm sido propostas ao longo da década de 90, destacando-se atualmente o *Object Management Group's Common Object Request Broker Architecture* (CORBA) e o *Component Object Model* (COM).

Nos sistemas operacionais, os processos são protegidos uns dos outros. Quando um processo necessita se comunicar com outro processo ele não pode chamá-lo diretamente: tem que usar alguma forma de comunicação entre os processos provida pelo sistema operacional. O *Component Object Model* (COM) padroniza esta comunicação de uma maneira transparente, definindo a interação entre módulos servidores e módulos clientes através de uma interface específica. Mais do que uma única tecnologia, existe uma imensa infra-estrutura no sistema operacional usada para prover comunicação transparente entre dois módulos de um software, ou seja, um arquivo executável e uma biblioteca (DLL) ou dois arquivos executáveis.

O DCOM (*Distributed Component Object Model*) foi colocado no mercado em meados de 1996, pouco depois do COM, e estende COM para suportar comunicação entre objetos em computadores diferentes de forma transparente, seja em uma rede local (LAN), uma rede de grandes distâncias (WAN) ou Internet, podendo usar qualquer protocolo de transporte, incluindo TCP/IP, UDP, IPX/SPX e NetBios. As plataformas

que dispõem de implementações DCOM são Windows 95/98/NT 4.0, Apple Macintosh, Sun Solaris, AIX, MVS e Linux.

No modelo COM, as classes de objetos e as interfaces são nomeadas usando *Globally Unique Identifiers* (GUIDs). Quando GUIDs são usados para referenciar classes particulares de objetos, eles são chamados *Class Identifier* (CLSIDs) e quando são usadas para referenciar interfaces são denominadas *Interface Identifier* (IID). Estes GUIDs são números extensos (16 bytes - 38 dígitos) que garantem, estatisticamente, a unicidade e a nomeação descentralizada para objetos e classes.

Uma interface define o comportamento ou as capacidades de um componente de software com um conjunto de métodos e propriedades, garantindo a consistência semântica do objeto que a suporta. Cada objeto pode suportar diversas interfaces, identificadas através de um IID que juntamente com CLSID, são registrados no *registry* quando executamos o processo de registro do servidor COM. O *registry* é um banco de dados que contém as configurações do sistema operacional da família Windows e informações sobre pacotes, componentes e aplicativos instalados.

As interfaces são definidas com a *Microsoft's Interface Definition Language* (MIDL) e correspondem a assinatura dos métodos dos componentes, ou seja, quais tipos de dados entram no componente, sob forma de parâmetro, ou saem dele, sob forma de valor de retorno. Estas informações são descritas em um arquivo IDL (*Interface Definition Language*), em uma linguagem de definição semelhante ao "C", contendo os tipos básicos de dados e mais algumas palavras chave. Os clientes interagem com os servidores invocando métodos descritos na interface, sendo que a implementação propriamente dita do objeto é escondida do cliente.

As interfaces são pouco intuitivas, o que é compensado nas ferramentas de desenvolvimento pela introdução de ferramentas para construção de interfaces, que a partir do preenchimento de campos em caixas de diálogo, fazem a construção do código das mesmas.

Uma classe COM é o código fonte que implementa uma ou mais interfaces em qualquer uma das linguagens suportadas. A princípio, qualquer linguagem pode ser usada para criar componentes COM, desde que sua especificação seja em nível binário. Linguagens como C++, Java, Object Pascal (Delphi), Visual *Basic* e até mesmo Cobol podem ser utilizadas para codificar objetos COM. Qualquer classe suporta, no mínimo,

a interface padrão `IUnknown`, que inclui o método `QueryInterface`, `AddRef` e `Release`.

O método `QueryInterface` recebe um IID e devolve um ponteiro para a interface correspondente. Para interagir com um componente, uma aplicação cliente precisa conhecer ou ter capacidade de encontrar, no mínimo, uma CLSID e uma IID para uma interface suportada por esta classe. Com estas informações, o cliente pode solicitar ao COM para criar um objeto e retornar o ponteiro para a interface apropriada.

Os métodos `AddRef` e `Release` implementam uma contagem de referências mantida por cada objeto, assim quando a contagem chega a zero o objeto assume que não existem mais clientes e pode autodestruir-se. Qualquer método que retorna um ponteiro para interface precisa chamar `AddRef` e os clientes precisam chamar `Release` após encerrar a utilização da interface.

Quando um cliente solicita um CLSID, o COM utiliza o *Service Control Manager* (SCM) para localizar e ativar o servidor. Cada classe COM está associada com outra classe denominada *Class Factory* que é responsável por criar instâncias da classe. Quando um cliente requisita a criação de um novo objeto, o servidor é localizado, carregado (DLL) ou disparado (EXE) e a *Class Factory* é requisitada para criar o objeto. O cliente recebe um ponteiro para interface e as chamadas subseqüentes são invocadas diretamente para a instância criada. Para criar as instâncias são usadas funções do sistema operacional tais como `CoCreateInstanceEx` ou `CoGetClassObjet`.

Conforme já citado, uma ou mais classes podem ser empacotadas num servidor no formato de DLL ou em um arquivo executável. Quando a implementação é em DLL, o servidor é denominado no processo (*in-process server*) e geralmente é mais simples, pois quando ela é carregada pelo processo cliente, ficam ambos no mesmo espaço de endereçamento de memória. Se a classe for empacotada num executável separado, o servidor é denominado fora do processo (*out-process server*), podendo executar na mesma máquina que o cliente ou numa máquina remota (acessada através do DCOM). Se as DLLs forem executadas em outra máquina ou em um ambiente hospedeiro, elas são consideradas fora de processo. Esse esquema pode ser visualizado na figura 4.1.

Quando o objeto servidor é trocado de lugar, seja de máquina ou até mesmo de rede, não há a necessidade de uma recompilação do mesmo. A independência de localização é uma das características que facilitam a utilização e manutenção de objetos COM.

Para permitir a comunicação entre aplicativos que residem em espaços diferentes de memória, o COM utiliza a metodologia de *marshling*, que consiste no processo ler uma informação da memória do processo cliente e passar esta informação para um *buffer* de memória *flat*, onde será lida pelo outro processo.

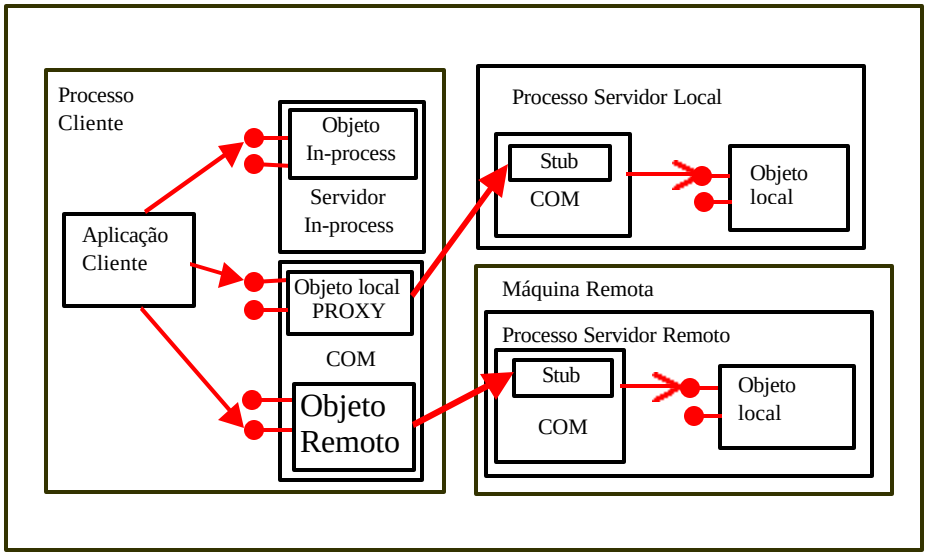

Figura 4.1 – Visão geral do empacotamento dos objetos.

A comunicação pode ser executada de forma transparente entre o cliente e o servidor porque o compilador MIDL gera dois códigos denominados *proxy* e *stub*. O *proxy*, que corresponde ao lado do cliente da API para os objetos que suportam a interface, é responsável pela linearização dos parâmetros (*marshaling*), enquanto o *stub* é responsável por decodificar as requisições recebidas dos clientes e disparar o objeto apropriado no servidor. Estes dois códigos garantem a compatibilidade binária entre as diferentes linguagens utilizadas ou, no caso do DCOM, diferentes sistemas operacionais, máquinas e protocolos de rede. Assim, um cliente COM interage com um objeto COM obtendo um ponteiro para uma das interfaces do objeto e invocando métodos através deste ponteiro, como se o objeto estivesse no espaço de endereçamento do cliente.

O DCOM estende a transparência de empacotamento do COM ao permitir que os servidores executem em outra máquina. A comunicação entre objetos numa rede

é baseada no protocolo *Object Remote Procedure Call* (ORPC), uma extensão orientada a objetos do OSF's DCE RPC, projetado para utilizar múltiplos protocolos de transporte, incluindo protocolos Internet como HTTP. Os *proxies* e *stubs* são responsáveis pela linearização das diferentes representações de dados utilizados em cada máquina para um formato neutro, denominado *OSF's Network Data Representation* (NDR) e utilizado através da rede.

Para que seja possível criar um objeto remotamente, as bibliotecas COM precisam saber o nome da máquina onde está o objeto servidor. Uma vez que o nome do servidor e o *Class Identifier* (CLSID) são conhecidos, uma parte das bibliotecas COM chamadas *Service Control Manager* (SCM) será responsável pela conexão entre as máquinas.

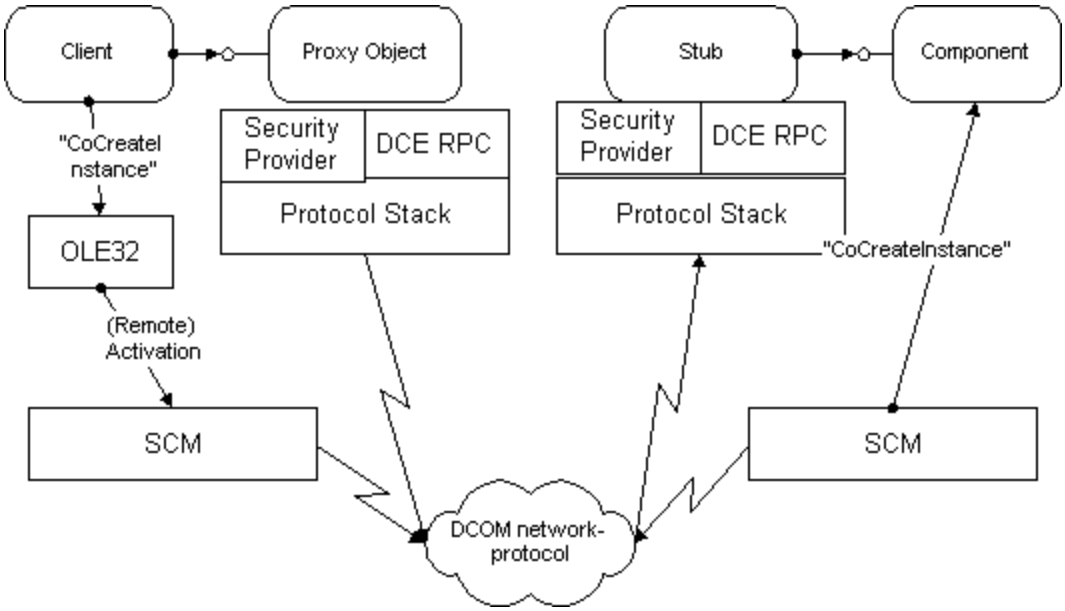

Figura 4.2 - Arquitetura DCOM

Quando um cliente requisita uma *Class Factory* para um objeto remoto como apresentado na figura 4.2, o SCM local contata o SCM da máquina remota que localiza e dispara o servidor retornando uma conexão RPC. Uma *Class Factory Proxy* é criada na aplicação cliente e a criação do objeto prossegue como no caso não distribuído.

Para invocar uma função remota, o cliente faz uma chamada para o cliente *proxy* que empacota os parâmetros da chamada em uma mensagem de requisição e invoca um protocolo de transporte para transportar a mensagem para o servidor. No lado do servidor, o protocolo de transporte entrega a mensagem para o servidor *stub*, que então desempacota a mensagem de requisição e chama a função no objeto.

Um nome de servidor remoto pode ser especificado de duas formas no momento da instanciação. A primeira é através do arquivo de registro. Neste caso as informações sobre o servidor remoto ficam gravadas na máquina cliente. Para que isto aconteça, é preciso que cada componente remoto seja "registrado" na máquina cliente antes de poder ser chamado por alguma aplicação. A outra é através de parâmetros passados para as funções de criação de instância. Neste caso a localização do componente remoto é passada como parâmetro para a função que irá criar a instância do objeto.

Conforme já citado, a partir da tecnologia COM/DCOM diferentes programas aplicativos podem ser integrados. É o que ocorre entre os aplicativos do Microsoft Office, onde, por exemplo, se pode criar uma tabela do Microsoft Excel no Microsoft Word. Esses aplicativos são servidores COM e através de interfaces padrão, determinados objetos podem ser invocados em outro qualquer aplicativo, realizando as tarefas específicas, típicas daquele que está sendo conectado.

5 Projeto e Implementação da Interface

Este capítulo tem por objetivo descrever uma proposta de interface a fim de que o SIMOO-RT possa exportar dados para um sistema supervisório.

Para realizar essa tarefa duas etapas distintas devem ser executadas:

- Modificações no ambiente SIMOO-RT para especificar os elementos de visualização;
- Definição e implentação da interface e do código do objeto.

Como o SIMOO-RT é desenvolvido no ambiente Delphi, o trabalho de desenvolvimento da interface será realizado na mesma plataforma, no ambiente Windows 98.

É importante destacar que o modelo de objetos dos elementos de visualização (EVs) foi especificada por Tibola em [TIB00] e portanto não será alterada nesse trabalho. Cabe ao mesmo definir uma estratégia para disponibilizá-los, via interface, para um sistema supervisório. A seguir serão destacadas as atividades realizadas em cada uma dessas etapas citadas acima.

5.1 - Modificações no SIMOO-RT

O projetista pode definir, em tempo de projeto, os EVs que fazem parte do seu modelo. Estes EVs serão posteriormente associados às *Tags* no sistema supervisório, conforme descrito no capítulo 2, e serão usados no processo de supervisão da aplicação para informar ao usuário valores coletados em campo. As *Tags* permitem o controle dos alarmes e alterações na configuração do equipamento ao qual estarão associados.

Ressaltando o que foi citado no capítulo 3, no SIMOO-RT encontram-se disponíveis quatro tipos de EVs, que podem ser especificados no projeto: *gauge*(EVGauge), *display*(EVDisplay), *bitmap sequence* (EVBitAnim) e *bar graph* (EVBarGraph).

Os EVs são associados a atributos de usuários, os quais são definidos nas classes durante a construção do modelo. Cada atributo poderá ter um ou mais EVs, do mesmo tipo ou de tipos diferentes. Não é permitido dois EVs do mesmo tipo, referentes ao mesmo atributo e a mesma classe, com o mesmo nome.

Os EVs são definidos na página *Visualization*, da janela *Edit Methods and Attributes* da *unit* Editarinstância_s do SIMOO-RT, a qual foi modificada para permitir essa funcionalidade . Essa janela é acessada quando uma classe é selecionada através de um duplo clique com o mouse na ferramenta editora de modelos (MET). Na figura 5.1 observa-se a janela anterior a este trabalho e na figura 5.2 exibe-se como ficou a janela após as modificações que foram realizadas nessa etapa.

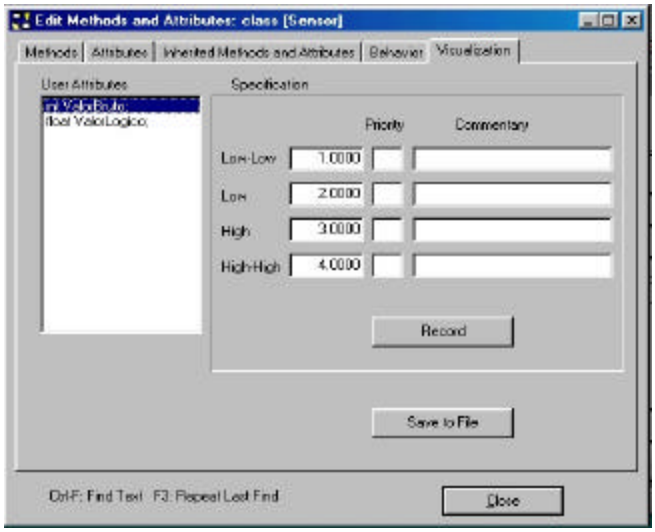

Figura 5.1 – Janela anterior à modificação

Inicialmente, a janela mostrava apenas os atributos. Quando um atributo era selecionado, eram visualizados os valores limites para ele, sendo possível editá-los e salvar os valores editados em um arquivo estruturado, associado ao nome da classe. Não era possível associar EVs aos atributos, nem especificar seus atributos.

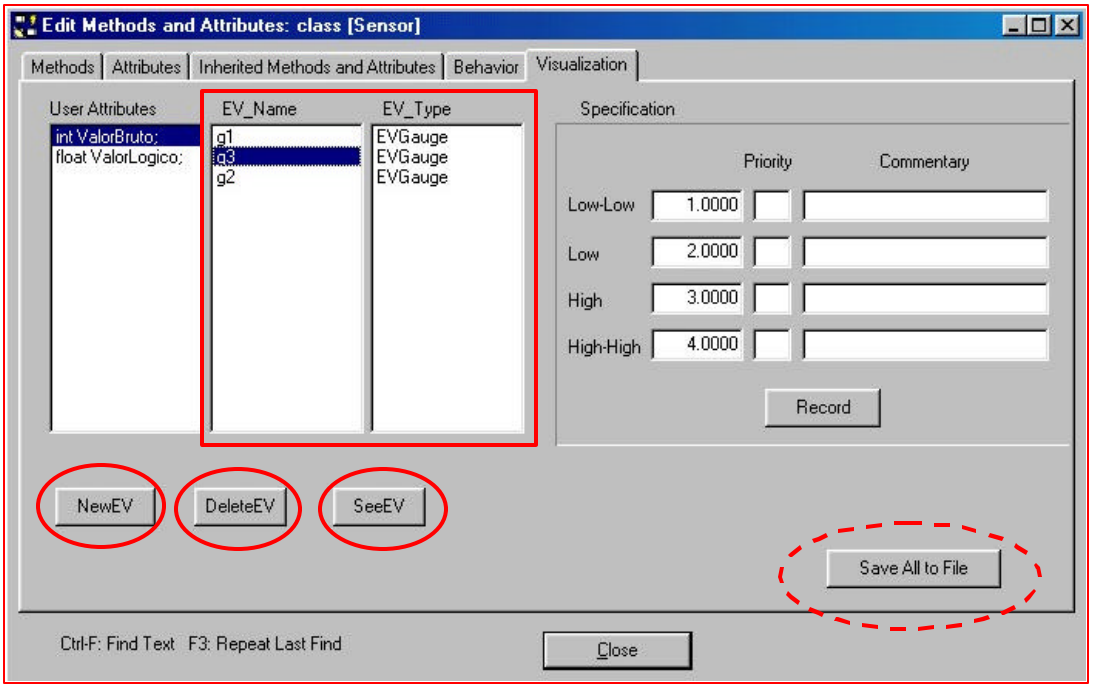

Figura 5.2 – Janela atual.

Na nova janela, foram então introduzidas duas caixas e três botões:

- **Caixa *EV_Name***: permite visualizar o nome dos EVs associadas ao atributo selecionado;
- **Caixa *EV_Type***: permite visualizar o tipo do EV referente a mesma posição da caixa anterior;

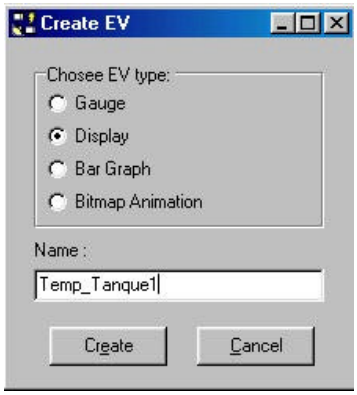

Figura 5.3 – Janela CreateEV

- **Botão *NewEv***: permite criar um novo EV associado ao atributo selecionado na caixa *Attributes*. Ele abre um formulário (figura 5.3) onde o usuário pode selecionar o tipo do EV e especificar o seu nome. Quando o usuário o preenche e seleciona botão *Create*, uma janela

específica para a determinação dos dados de cada EV será mostrada, conforme figura 5.4. Após o seu preenchimento, pressionando o botão OK, os dados editados são convertidos e armazenados em uma lista de *strings* (objeto Delphi TString). Por definição, toda vez que for definido, para um mesmo atributo, um EV com o mesmo nome e o mesmo tipo de um já existente, o anterior será apagado.

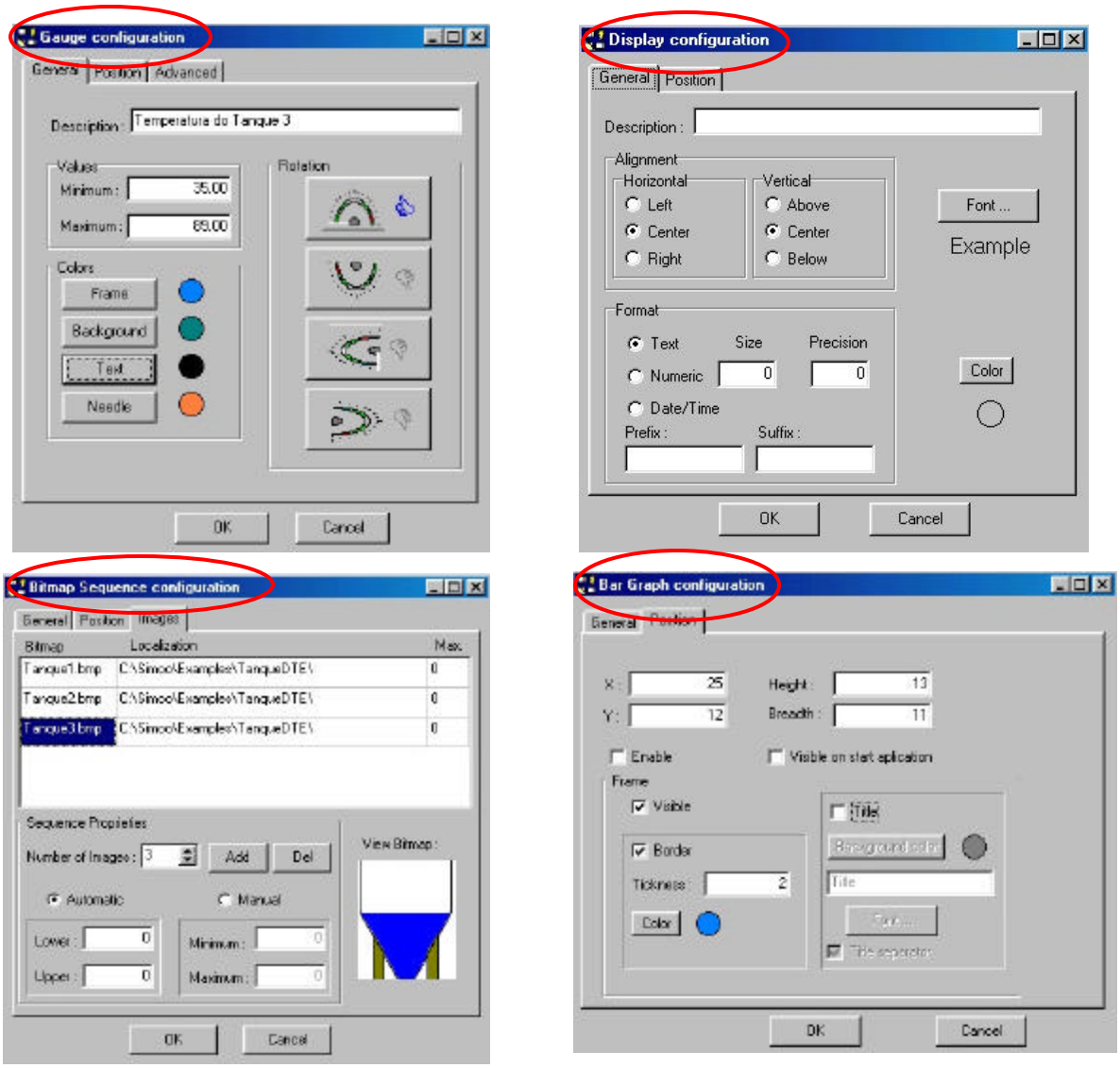

Figura 5.4 – Janelas de edição dos EVs (EVGauge, EVDisplay, EVBitAnim, EVBarGraph)

- **Botão *DeleteEV***: permite apagar um EV selecionado;
- **Botão *SeeEV***: permite visualizar os dados do EV selecionado, na janela de edição do EV (figura 5.4).

Finalmente um botão *Save All to File* permite salvar os dados dos EVs associados a todos os atributos de uma a classe em um arquivo tipo texto com o nome da classe e a extensão ". sup", o qual é denominado arquivo de definição dos EVs.

Quando a janela *Edit Methods and Visualization* é iniciada e um atributo é selecionado na janela *Attributes*, o arquivo de definição dos EVs é carregado na memória. A partir desse momento, todas as alterações são realizadas apenas em memória até que seja pressionado o botão citado acima.

Para dar suporte as tarefas citadas, foi construída uma *unit* denominada SalvaDadosEV que possui métodos para ler e salvar dados do arquivo, localizar EVs, carregar os dados do EV na memória, bem como métodos para retirar e enviar dados das janelas de edição dos EVs.

5.2 - Definição e implementação da interface e do objeto servidor.

Uma opção existente para que processos diferentes, rodando em endereços de memória distintos, como, por exemplo, dois arquivos executáveis, troquem dados entre si, é a utilização de um arquivo padrão que possa ser lido e gravado por ambos. Esta, porém, não é a alternativa que a maioria dos softwares usam, pois, entre outras desvantagens, é bastante limitante quanto à velocidade, havendo a necessidade de constantes operações de leitura/gravação. Uma solução adequada para este problema é usar mecanismos de comunicação que permitam a troca de informação durante a execução dos programas, com a transferência de dados já carregados em memória, usando recursos oferecidos pelo sistema operacional.

Assim foi adotada a tecnologia COM da Microsoft para construção da interface proposta, principalmente pelo fato deste ser um padrão bastante usado pelo mercado de software na atualidade. Este padrão certamente facilitará a comunicação do SIMOO-RT com diferentes sistemas supervisórios.

5.2.1 – Definição

Optou-se por organizar os dados dos EVs em lista de *strings*. A escolha de uma estrutura tipo lista deve-se ao fato dos EVs terem número diferentes de itens e o

EVBitAnim não ter número fixo de objetos, pois para ele podem ser especificadas várias imagens.

Até a posição 20 da lista todos os EVs possuem os mesmos atributos que foram organizados conforme figura 5.5. A partir dessa posição os dados são específicos de cada EV. A organização da parte complementar da lista de dados do EV pode ser observada nas figuras 5.6 (a), 5.6 (b) , 5.6 (c) e 5.6 (d)

DADOS
NomeEV
TipoEV,
Atributo
Classe
POSIÇÃO
Xpos
Ypos
Altura
Largura
VisivelIniciar
VisivelMoldura
VisivelBorda
EspessuraBorda
CorBorda
VisivelTitulo
CorFundoTitulo
TextoTitulo
FonteTitulo
SeparadorTitulo

Figura 5.5 – Objetos comuns aos Evs

5.2.2 –Implementação

O ambiente de programação Delphi 5 possui uma ferramenta denominada editor *Type Library* que esconde os detalhes de programação COM descritos no capítulo 4, fazendo geração automática de boa parte do código.

Uma *Type Library*, é uma biblioteca com conjunto de informações de tipo, com descrição de todos os elementos disponibilizados pelo servidor: a interface, os objetos e os tipos de dados, com código final independente de linguagem. Os elementos de tipo são definidos pelo OLE/COM como um subconjunto dos elementos padrão das linguagens de programação e qualquer ferramenta de desenvolvimento pode usá-los.

GERAL
GauMin
GauMax
GauCorMoldura
GauCorFundo
GauCorAgulha
GauRotacao
AVANCADO
GauVisivelMarcas
GauCorMarcas
GauVisivelSubMarcas
GauCorSubMarcas
GauTotalSubMarcas
GauVisivelValores
GauFonteValores
GauVisivelLegenda
GauVisivelLimLow
GauCorLimLow
GauVisivelLimLolo
GauCorLimLolo
GauVisivelLimHigh
GauCorLimHigh
GauVisivelLimHihi
EvGauge.GauCorLimHihi
GauCorNormal.

(a) EVGauge

GERAL
DisFonte
DisCor
DisAlinhaHoriz
DisAlinhaVerti
DisFormato
DisFormatoTamanho
DisFormatoPrecisao
DisSufixo

(b) EVDisplay

GERAL
BitTransparente
BitCorFundo
BitVisivelBorda
BitRefresh
IMAGENS
BitQtdeBitmaps
BitManualAuto
BitNomeBitmaps
BitLocalBitmaps
BitValorMin
BitValorMax.

(c) EVBitAnim

GERAL
BarCorFundo
BarCorBarra
BarLimSup
BarLimInf
BarOrientacao
BarFonteRegua
BarHabilitaRegua
BarCorFundo2
BarCorTexto
BarCorGrade
BarPosicaoRegua

(d) EVBarGraph

Figura 5.6 – Objetos específicos dos EVs

Para construir um servidor em um projeto existente, abre-se o projeto, seleciona-se *File | New | ActiveX | Automation Object*. Na caixa de diálogo *Automation Object Wizard* resultante introduz-se o nome da classe do servidor e selecionando-se OK será aberto o *Editor Type Library*. Nesse editor são definidos as interfaces COM, os métodos e as propriedades do objeto servidor, conforme pode ser observado na figura 5.7. Ao fechar esse editor e salvar o projeto, será gerando o arquivo de tipo (.tlb) com o mesmo nome do arquivo executável, o código fonte Object Pascal correspondente (*_TLB.pas) e a declaração do objeto servidor com todo o esqueleto do código dos métodos e/ou propriedades que foram definidos.

O arquivo da *Type Library* é conectado ao projeto usando-se uma instrução de inclusão de recurso {$*.TLB}, inserida no código fonte do arquivo de projeto.

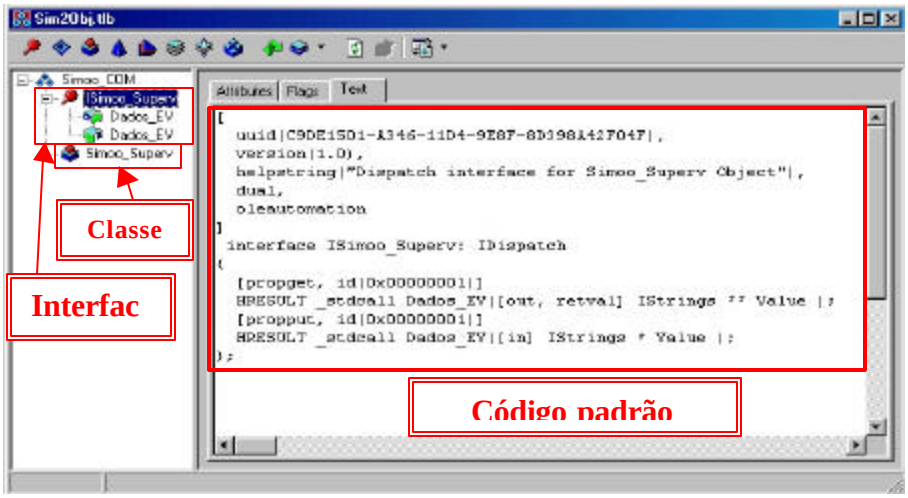

Figura 5.7 – Definição da Interface no editor de *Type Library*

Uma outra estratégia, que foi usada com sucesso, foi a elaboração de um projeto separado com a implementação de uma *Type Library* e de um objeto servidor. Posteriormente renomea-se o arquivo .tlb com o nome do arquivo executável e inclui-se a *unit* do objeto servidor no projeto existente.

Após compilar e gerar o arquivo executável do servidor deve-se registrá-lo para que seja acessível ao(s) cliente(s). Isso é possível porque a unidade que contém o objeto servidor possui, na seção *Intitialization*, a instrução

TAutoObjectFactory.Create (ComServer, TSimoo_Superv, CLASS_Simoo_Superv, ciMultiInstance, tmSingle)

que combinada com a chamada ao método `Initialize` do objeto *Aplication* que o delphi inclui no código fonte do projeto, permitem o registro do servidor no local *registry* do windows. O registro é feito executando-se o servidor na máquina destino, local onde o servidor será instalado, passando para ele o parâmetro `/regserver`. Para remover o servidor, executa-se o programa com o parâmetro `/unregserver`.

O objeto servidor foi definido com apenas uma propriedade, denominada Dados_EV, que é capaz de ler dados de um objeto e de escrever outro objeto da *unit* interface_s.

Todos os tipos de dados usados devem ser mapeados para tipo padrão COM, pois é o Sistema Operacional que movimenta os dados e a interface deve acessível por programas escritos em qualquer linguagem. O COM inclui tipos básicos como *Integer*, *SmallInt*, *byte*, etc. Para usar elementos complexos como fontes, listas de *strings* e

bitmaps defe-se fazer uso de interfaces fornecidas pelo Windows, tais como *IfontDisp*, *Istrings* e *IpictureDisp*.

Na implementação dessa propriedade foram usados dois utilitários COM, fornecido pela *unit* padrão AXCtrls denominados *GetOleStrings* e *SetOleStrings.*

O primeiro implementa um objeto *TStrings* como uma interface *IStrings* para ser usado por objetos OLE, possuindo o seguinte formato:

procedure GetOleStrings(Strings: TStrings; var OleStrings: IStrings)

sendo que o parâmetro Strings é implentado como uma interface *IStrings* e encaminhado para a propriedade `TStrings.StringsAdapter` que retorna *IString* interface como um parâmetro *OleStrings.*

O segundo usa uma interface *IStrings* para determinar o conteúdo de um objeto *Tstrings*, possuindo o seguinte formato:

procedure SetOleStrings(Strings: TStrings; OleStrings: IStrings)

onde o objeto *TStrings* corresponde às strings fornecidas por uma interface *IStrings*.

Na figura 5.8 podemos observar parte do código pascal que implementa os dois métodos da propriedade definida para o objeto servidor:

```
function TSimoo_Superv.Get_Dados_EV: IStrings;
begin
        GetOleStrings (FormInterface_s.ToInterface, Result);
end;
procedure TSimoo_Superv.Set_Dados_EV(const Value: IStrings);
begin
        SetOleStrings (FormInterface_s.FromInterface, Value);
end;
```

Figura 5.8 – Implementação da propriedade do objeto servidor

Conforme descrito por Cantù em [CAN00], não se deve incluir código relacionado à interface com o usuário dentro da classe do objeto servidor. O autor recomenda que se faça referência a um elemento de interface com o usuário, como uma classe formulário, e permitir que ele realize as ações, o que facilitará modificações futuras pois estas não afetarão o objeto servidor nem a interface COM. Foi seguindo essa recomendação que foi construímos a *unit* Interface_s, com uma classe formulário denominada FormInterface_s, que funcionará como interface para o usuário.

Pode-se observar na figura 5.8 que o objeto servidor faz apenas referências a métodos dessa *unit*, o que simplifica muito a sua implementação. É nesse formulário que um EV referente ao modelo pode ser selecionado no SIMOO-RT e seus dados ser acessados pelo sistema supervisório. Todas as tarefas de preparação dos dados dos EVs para envio ao sistema supervisório e o tratamento dos dados vindos do supervisório serão realizados por nesta *unit*.

Quando o sistema supervisório abre o servidor, o SIMOO-RT é carregado na memória. Deve-se abrir o arquivo do modelo que contenha os EVs cujos dados serão transferidos ao supervisório. Apartir de então, através de um botão localizado na barra de ferramentas do editor, conforme podemos ver na figura 5.9, o formulário de interface com o usuário pode ser aberto para selecionar os EVs que serão enviados ao cliente COM. Essa seleção deve ser feita individualmente.

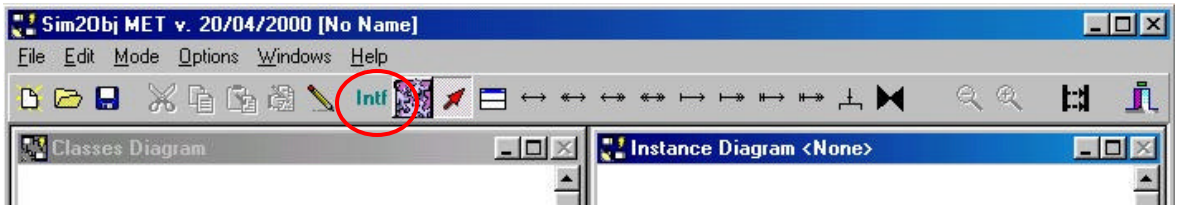

Figura 5.9 – Seleção da Interface com o usuário

Na figura 5.10 observa-se o formulário de interface com o usuário que é composto por quatro janelas. Na primeira janela, denominadas Classes, é mostrado, em forma de árvore, as classes definidas para o modelo, na ferramenta editora de modelos. Quando uma classe da árvore é selecionada, os seus atributos são mostrados na segunda janela, denominada Attributes.

Ao selecionar um atributo dessa janela, se existir o arquivo com EVs definidos para essa classe, conforme descrito em 5.1, o arquivo será aberto. Havendo EVs definidos para o atributo selecionado, seu(s) nome(s) será(ao) mostrado(s) na terceira janela com o correspondente tipo na quarta. Se não houver EVs definidos na classe, o arquivo não existe e, na primeira vez que um atributo da classe for selecionada, uma mensagem informa ao usuário que não há EV definido para essa classe.

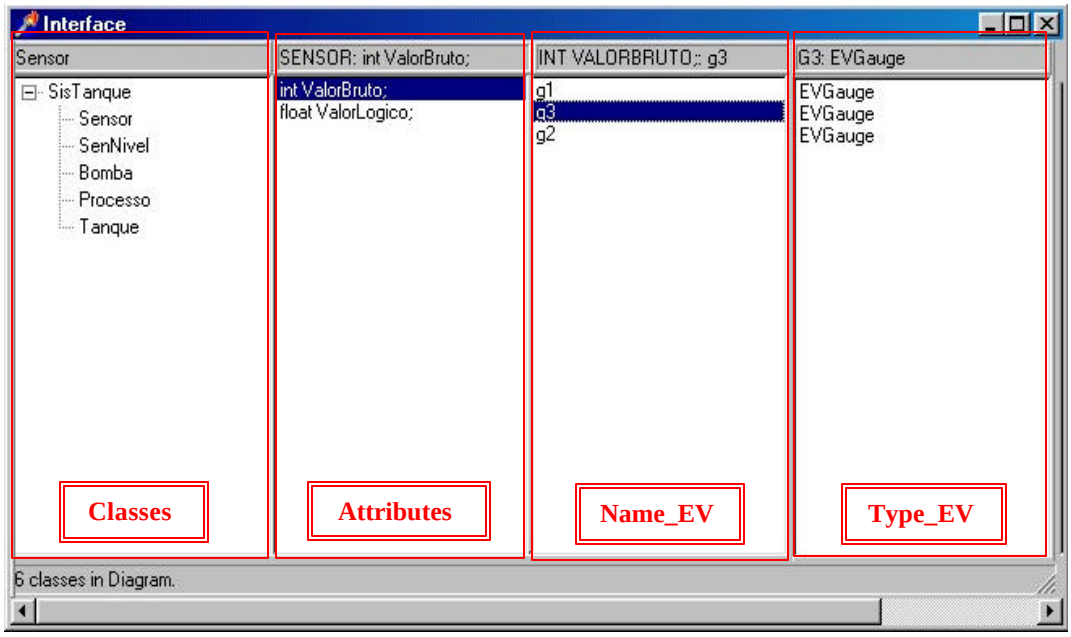

Figura 5.10 – Janela de Exportação dos EVs

Quando um EV for selecionado na janela *Name_EV* seus dados tornam-se disponíveis para serem acessados pelo objeto servidor, podendo ser transferido para o cliente.

Quando o usuário fechar essa janela os dados enviados pelo cliente serão salvos automaticamente no modelo, seguindo as definições descritas em 5.1. Se o cliente enviou várias vezes o mesmo EV, será armazenado no modelo as alterações que foram enviadas na última vez.

É nessa *unit* que se encontram os métodos para percorrer a estrutura de dados do SIMOO-RT e montar a árvore da janela Classes, para ler os atributos de classe e mostrá-los na janela *Attributes* e para mostrar os EVs e seus tipos nas janelas *Name_EV* e *Type_EV*. Nela estão também métodos que acessam e organizam dados do EV selecionado para que sejam acessados pelo objeto servidor e métodos que permitem salvar, no modelo, os dados alterados dos EVs que foram enviados pelo sistema supervisório.

6 Validação da Interface

O objetivo deste capítulo é realizar alguns procedimentos para validar as definições e implementações deste trabalho. Num primeiro momento será abordado a construção de um cliente COM, necessário para testar e validar a interface com o objeto servidor COM do SIMOO-RT. Posteriormente será abordado um estudo de caso que foi realizado para demonstrar as propriedades do ambiente desenvolvido.

6.1 Implementação do cliente COM

Como o cliente COM do servidor SIMOO-RT será um software de supervisão, foi implementado um software supervisório, com as funcionalidades mínimas necessárias para validar a interface descrita no item 5.2. A janela principal desse software pode ser vista na figura 6.1.

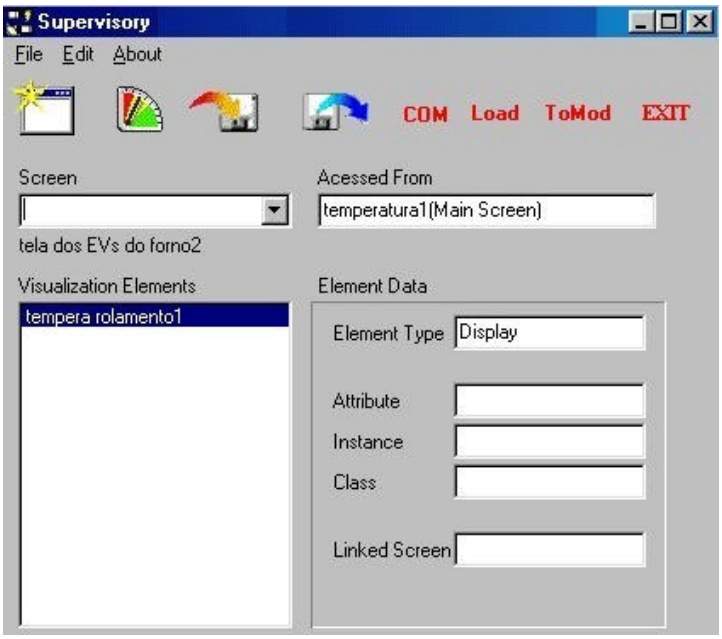

Figura 6.1 – Janela principal do software supervisório de teste

Na parte superior a janela apresenta oito botões. A seguir serão descritas as tarefas dos mesmos, da esquerda para a direita:

- **botão 1:** se um EV estiver selecionado na janela *Visualization Elements* permite que seja editada uma nova tela. Ao ser selecionado esse botão, a janela de edição de telas, mostrada na figura 6.2, é aberta. Quando for pressionado OK os dados dessa tela serão adicionados na estrutura de dados do supervisório que está sendo editado;
- **botão 2**: permite que sejam editados novos EVs para a tela atual. Quando esse botão é selecionado, é aberta a janela mostrada na figura 6.3 e posteriormente, de acordo com o tipo de EV selecionado, uma das janelas da figura 5.4;
- **botão 3:** grava os dados do supervisório editado em arquivo;
- **botão 4:** permite abrir um supervisório armazenado em arquivo;
- **botão 5:** abre o servidor COM, o SIMOO-RT, para receber e enviar dados de EVs de um modelo já definido;
- **botão 6:** permite que os dados de um EV selecionado no SIMOO-RT sejam incorporados na tela atual do arquivo de supervisório que está sendo editado;
- **botão 7:** os dados do EV selecionado na janela *Visualization Elements* são enviados ao SIMOO-RT para serem incorporados no modelo;
- **botão 8:** fecha o sistema supervisório.

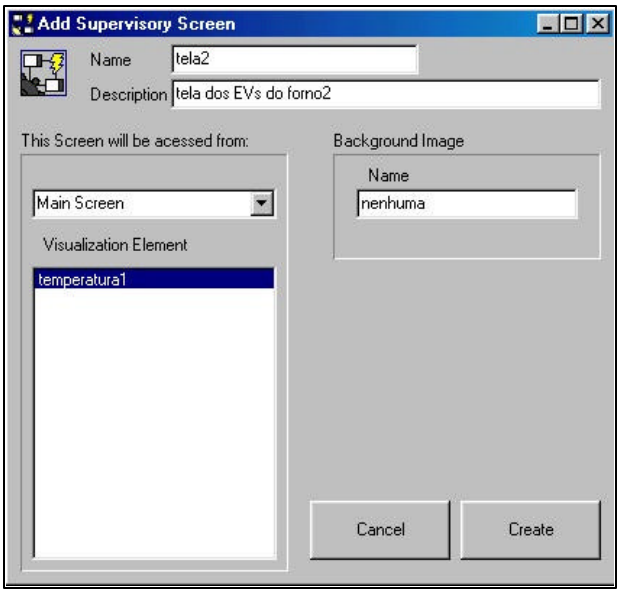

Figura 6.2 – Janela de edição de telas

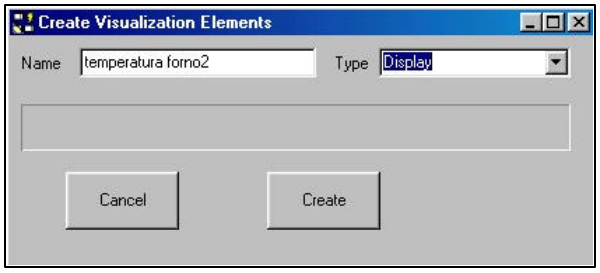

Figura 6.3 – Janela de criação de EVs

O formulário da figura 6.1 apresenta ainda as seguintes janelas:

- ***Screen*:** permite a seleção de uma tela já existente. Mostra a tela que está atualmente selecionada;
- ***Visualization Elements*:** mostra os EVs que foram editados na tela atual, ou seja, a tela que está selecionada na janela *screen;*
- ***Acessed From*:** mostra qual o EV que permite o acesso à tela atual e em que tela ele se encontra;
- ***Element Data*:** mostra alguns dados de um EV selecionado, relacionados com o modelo. Apenas EVs que tenham sido obtidos apartir do SIMOO-RT possuem todos os campos *attributes*, *instance* e *class* preenchidos e, por isso, podem ser reenviados ao modelo.Os EVs criados nos sistema supervisório, não podem ser incorporados ao modelo.

Através do menu da parte superior pode-se, além das tarefas executadas pelos botões, editar e apagar EVs ou telas selecionados.

O software supervisório acima apresenta apenas as funções básicas de edição de EVs e telas, sem qualquer preocupação com a estrutura de armazenagem e visualização dos dados e a comunicação com o processo externo, características típicas de um sistema de supervisão, conforme descrito no capítulo 2.

Para sua implementação foram aproveitadas diversas *units*, já desenvolvidas para implementação do SIMOO-RT, retirando-se qualquer referência a ferramenta de modelagem, a fim de caracterizar a situação proposta nesse trabalho, que é a comunicação do SIMOO-RT com um sistema supervisório, um arquivo executável independente da ferramenta de modelagem, utilizando-se recursos da tecnologia COM.

Após a implementação e teste das funções básicas de criação e edição de EVs e telas, o sistema descrito acima foi transformado em um cliente COM, para poder usar os recursos implementados na *Type Library*, a fim de poder receber e transferir EVs do SIMOO-RT.

O primeiro passo é referir-se ao arquivo pascal gerado pelo *editor Type Library* na cláusula *uses* do formulário. Se não estiver disponível o arquivo pascal do servidor o ambiente Delphi fornece uma ferramenta para importá-lo diretamente da *Type Library* (o arquivo .tlb) do servidor.

No lado do cliente não necessitamos de todo o código da biblioteca, apenas das declarações do código fonte do objeto servidor e da classe de criação, ou *CoClass*, que é usada para criar um objeto servidor local ou remotamente, cujo código pode ser observado na figura 6.4.

```
CoSimoo_Superv = class
        class function Create: ISimoo_Superv;
        class function CreateRemote(const MachineName: string): ISimoo_Superv;
end;
```

Figura 6.4 - Código da classe de criação de objetos COM

A classe CoSimoo_Superv fornece os métodos `Create` e `CreateRemote` para criar instâncias da Interface ISimoo_Superv disponibilizada pela CoClass Simoo_Superv. Estas funções devem ser usadas pelo cliente para instanciar os objetos da(s) CoClass(es) disponibilizada pelo servidor através de sua *Type Library*.

A primeira é usada para criar um objeto servidor e, posteriormente, iniciar o aplicativo servidor no mesmo computador e a Segunda, para criá-lo em outro computador, desde que sistema operacional ofereça suporte a DCOM.

Todo objeto COM é uma instância de uma *CoClass*, que é a classe que implementa uma ou mais interface COM, que fornece os serviços que são definidos pela sua interface *CoClass*, definida na *Type Library*. Esta classe é instanciada pela *class factory* (fábrica de classes). Quando o serviço de um objeto é requisitado pelo cliente, a fábrica de classes cria e registra uma instância do objeto para o cliente e, conforme pode ser observado no código acima, é devolvido para o cliente um ponteiro com o endereço do objeto instanciado.

O segundo passo então é instanciar o objeto servidor. Isso é feito quando o botão COM é pressionado. A função `Create` é chamada e o endereço do objeto servidor instanciado é fornecido a um objeto do tipo da interface. A propriedade que foi definida para o servidor pode ser acessada através do desse objeto, como se estivesse definida localmente, conforme pode-se ver na figura 6.5. Se o SIMMO-RT não estiver em memória ele será carregado no momento de instanciar o objeto.

Apartir de então, os dados dos EVs podem ser transferidos entre ambos os programas, como se residissem no mesmo endereçamento de memória e estivessem acessando variáveis comuns. O código da figura 6.5 mostra parte do código da implementação dos botões *Load* e *ToMod*, que permite a comunicação entre os dois softwares.

private

IServidor: ISimoo_Superv;

- **BOTÃO COM**

  if not Assigned (IServidor) then

  IServidor := CoSimoo_Superv.Create;

- **BOTÃO Load**

  TempIStrs := IServidor.Dados_EV;

  SetOleStrings (paramentro, TempIStrs);

- **BOTÃO ToMod**

  GetOleStrings (parametro, TempIStrs);

  IServidor.Dados_EV := TempIStrs;

Figura 6.5 - Código parcial dos botões *COM*, *Load* e *ToMod*

6.2 Estudo de Caso

Para teste das definições e implementações do trabalho foi realizado um estudo de caso com um modelo cujo diagrama de classes foi apresentado na figura 3.1. Este modelo apresenta cinco classes, derivadas da classe geral SisTanque: Sensor, SenNivel, Bomba, Processo e Tanque. Para esta classe foram definidos atributos e EVs, conforme pode ser observado na figura 6.6.

Classe	Atributos	Nome do EV			
		EVGauge	EVDisplay	EVBarGraph	EVBitAnim
Sensor	int ValorBruto	PosInic	PosInic	PosInic	PosInic
	float ValorLogico	PosInic	PosInic	PosInic	PosInic
SenNivel	float Posicao	PosInic	PosInic	PosInic	PosInic
Bomba	int Estado	---	---	---	---
Processo	---	---	---	---	---
Tanque	int NivelMin	Valor1	---	---	---
	int NivelMax	---	Valor2	---	---
	int Nivel	---	---	Valor3	---
	int Nivel1	---	---	---	Valor4

Figura 6.6 – Classes, atributos e EVs do estudo de caso

Pode-se observar na figura acima que para a classe Processo não foram definidos atributos, logo não há como definir EVs para ela, para a classe Bomba foi definido um atributo porém não foram definidos EVs e para os atributos das classes Sensor e SenNivel todos os EVs apresentam o mesmo nome.

As figuras 6.7 e 6.8 mostram partes página *Visualization* da janela *Edit Methods and Attributes*, onde pode ser visualizado o resultado da inserção dos dados acima.

Observa-se nas classes Sensor e SenNivel (figura 6.7) que todos os EVs possuem o mesmo nome, porém são de tipos diferentes. Isto demonstra o correto funcionamento da definição descrito em 5.1 de que dois EVs só serão iguais, aparecendo apenas uma vez na janela *Name_EV* e sendo sobrescritos no arquivo de definição dos EVs, se possuírem mesmo nome e estiverem associados ao mesmo atributo da mesma classe.

Após editar os EVs, os mesmos foram salvos em arquivo e a janela *Edit Methodos and Attribures* foi encerrada. Posteriormente ela foi reaberta para as diversas classes e cada atributo foi acessado novamente sendo todos os EVs visualizados (através do botão *SeeEV* – figura 5.2). Este procedimento permitiu conferir os dados de cada EV, carregados do arquivo conforme descrito no item 5.1, com os dados introduzidos no processo de edição, comprovando-se o correto funcionamento do procedimento de associação de EVs aos atributos especificados para as classes do modelo.

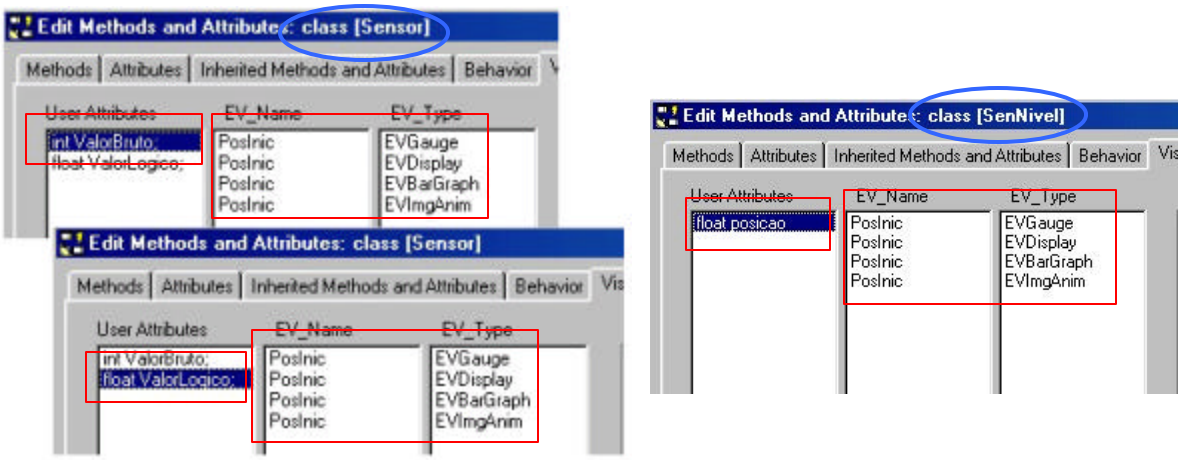

Figura 6.7 – Atributos e EVs (vista parcial das classes Sensor e SenNivel)

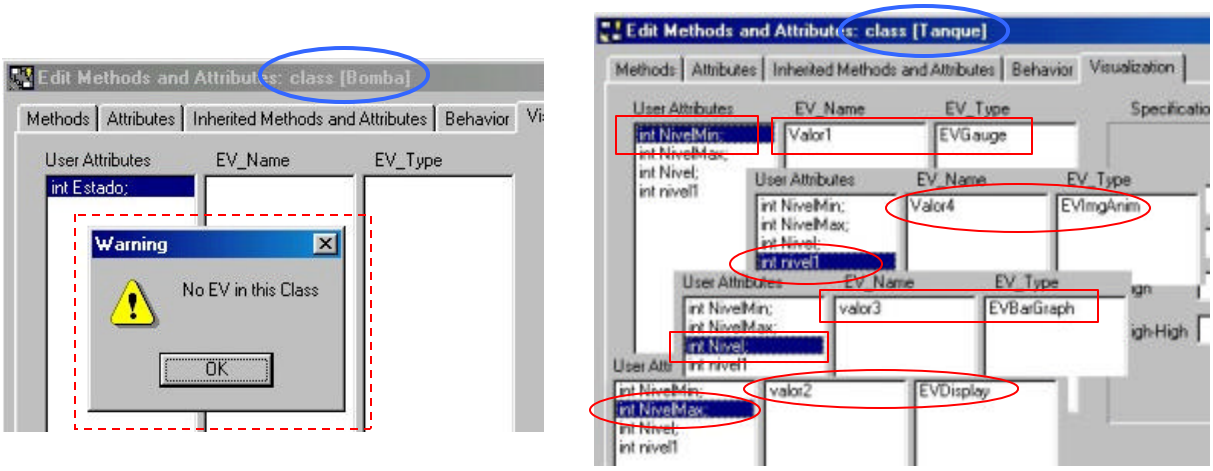

Figura 6.8 – Atributos e EVs (vista parcial das classes Bomba e Tanque)

O próximo passo é comprovar a passagem dos parâmetros dos EVs para o supervisório e deste para o SIMOO-RT, usando a interface COM implementada, conforme descrito em 5.2

Ao executar o supervisório e selecionar o botão COM, a ferramenta de modelagem automaticamente carregado na memória. O próximo passo é, no SIMOO-RT, abrir o arquivo de modelo que contenha os dados que se deseja transferir para o sistema supervisório. Com o arquivo do modelo em memória, o SIMOO-RT está pronto para transferir os dados para supervisório, que devem são selecionados através da janela de interface com o usuário, acessível a partir da ferramenta editora de modelos, conforme mostrado nas figura 5.9 e 5.10.

Na figura 6.9 pode-se ter a visão de todos os elementos envolvidos na transferência de dados do SIMOO-RT para o sistema supervisório

Nesse estudo de caso foi aberto o arquivo do modelo descrito acima e todos os EVs disponíveis na interface foram selecionados, um a um, e carregados no supervisório através do botão *Load*, conforme descrito em 6.1. O EV pode ser inserido em qualquer tela, bastando selecionar a tela em que se deseja inseri-lo antes de clicar o botão *Load*. Isto permite que o mesmo EV possa ser inserido em diferentes telas, conforme a necessidade de visualização do sistema que está sendo implantado.

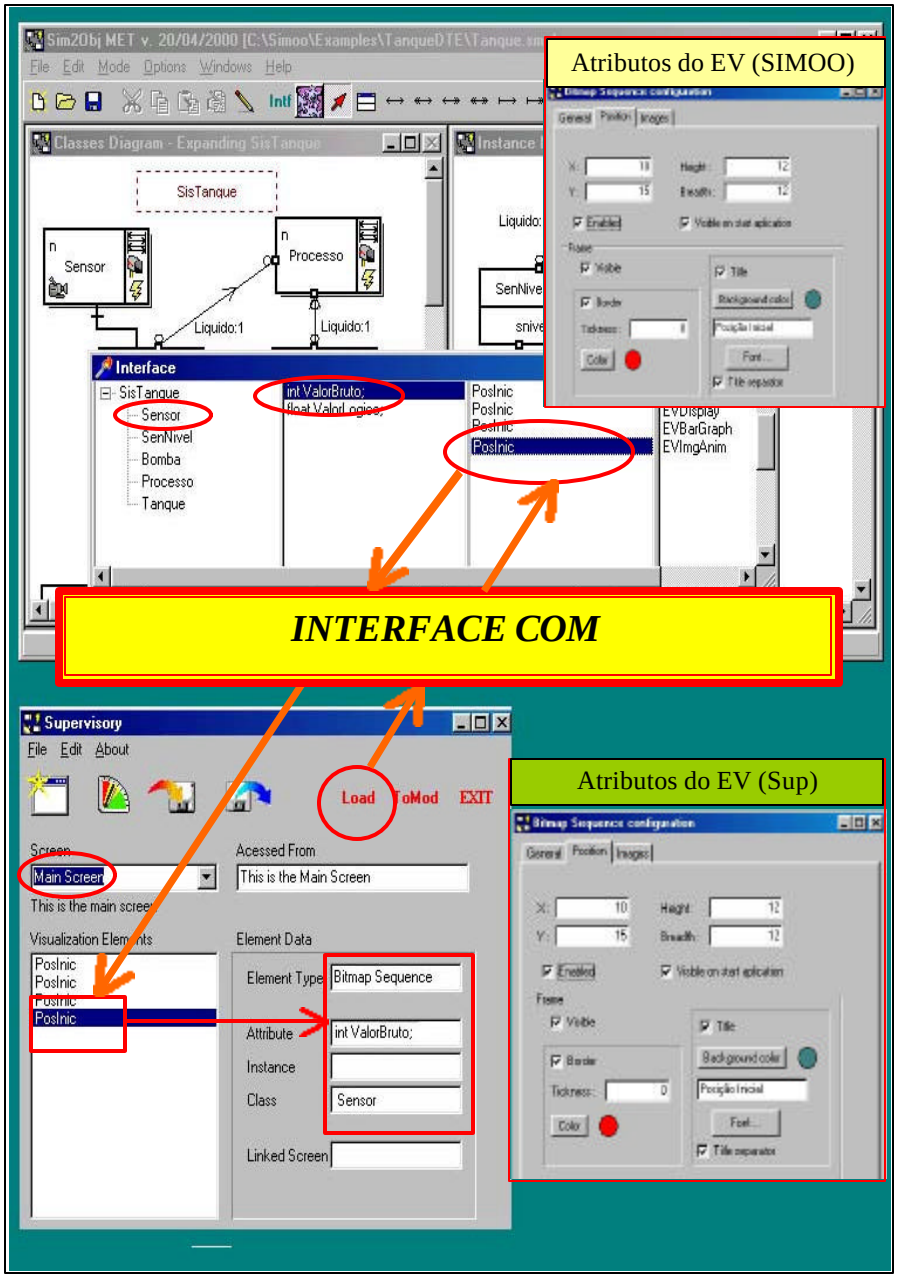

Figura 6.9 - Transferência de dados em o SIMOO-RT e o supervisório de teste

Os dados de cada EV ficam disponíveis no supervisório, podendo ser modificados e salvos de acordo com as definições do software supervisório, sem qualquer ligação com a ferramenta de modelagem.

Dados dos EVs também foram modificados e reenviados ao modelo. Para isso seleciona-se, no supervisório, o EV cujos dados devem ser alterado no modelo e aciona-se o botão *ToMod*. Quando a interface com o usuário for fechada, no SIMOO-RT, os dados dos EVs que foram enviados apartir do supervisório, poderão ser salvos nos arquivos correspondentes ao modelo. Importante salientar que todas as vezes que o mesmo EV foi enviado várias vezes, os dados salvos foram os correspondentes a última seleção, de acordo com a definição adotada.

Após salvar os EVs eles novamente podem ser visualizados na janela *Edit Methods and Attributes* do SIMOO-RT. Comprova-se, então, que aqueles EVs cujos objetos foram alterados no supervisório, apresentam os novos valores que foram enviados apartir desse software. Pode-se comprovar também que se um EV foi enviado várias vezes ao SIMOO-RT, são salvos, no modelo, os dados que correspondem ao último EV que foi enviado.

Este estudo consegue comprovar o funcionamento adequado de todas as definições implementadas nesse trabalho, comprovando a correta passagem dos dados dos elementos de visualização, definidos no SIMOO-RT, para um software supervisório externo e vice-versa, utilizando os recursos da Interface e do objeto servidor COM, implementados na ferramenta de modelagem.

7 Conclusões

A ferramenta SIMOO-RT é composta por diversas funcionalidades, contando atualmente com cerca de 140 módulos (*units*) 110 formulários (*forms*), onde estão definidas todas as classes, métodos e janelas que o compõem. Localizar os módulos onde estão definidas classes e métodos que interessam a realização do trabalho, entendendo o mecanismo de relação entre as classes, demandou muito mais tempo do que o previsto inicialmente no cronograma de execução do trabalho.

Embora as etapas de Definição e Implementação da Interface fosse o objetivo principal deste trabalho, as tarefas de especificação dos elementos de visualização e a construção do supervisório de teste ocuparam boa parte do tempo, pois eram importantes na definição e no teste da interface.

Considera-se que o objetivo do trabalho foi plenamente atingido, pois o SIMOO-RT já é um servidor COM e pode transferir e receber dados de softwares supervisórios, conforme ficou demonstrado no capítulo 6. A estrutura adotada permite que se façam novas especificações da interface com o usuário, alterando-se a *unit* Interface_s.pas, sem a necessidade de alterar-se a *Type Library* implementada. Entretanto, necessitando-se definir novas interfaces e objetos servidores em aplicações futuras, isso pode ser feito diretamente através do editor de *Type Library* acrescentando-se interfaces e classes na biblioteca atual e acrescentado os novos métodos do(s) objeto(s) servidor(es) na *unit* Sim_interf.pas.

Algumas modificações podem ser feitas para melhorar a definição adotada para a interface com o usuário:

- ter todos os EVs definidos em único arquivo estruturado, preferencialmente com o nome do arquivo do modelo;
- permitir a seleção múltipla de EVs na interface e o envio simultâneo de vários EVs ao sistema supervisório.

Outra necessidade será adequá-la ao *Supervisory Designer* quando o trabalho de implementação do mesmo estiver concluído.

Anexo 1 - Representação Gráfica da Estrutura Estática do Modelo

1.1 - Classes e subdiagramas

Cada classe é representada graficamente por um retângulo, o qual é sub-dividido em duas seções (ver figura A1.1):

- a seção do lado direito contém um conjunto de três ícones que indicam as abordagens que compõem o paradigma de simulação utilizado para descrever o comportamento da classe. A figura A1.2 apresenta a relação completa dos ícones possíveis: o primeiro ícone permite especificar a forma de recepção das mensagens, o segundo a abordagem de descrição do comportamento e o terceiro a abordagem utilizada para troca de mensagens.

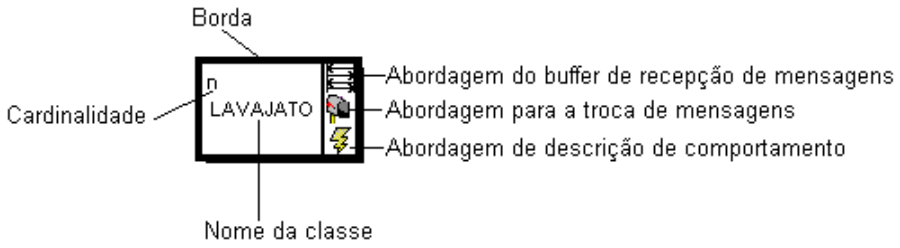

Figura A1.1 - Representação gráfica de uma classe

- do lado esquerdo do retângulo especifica-se, ao centro, o nome da classe e no canto superior esquerdo a cardinalidade da mesma. A cardinalidade permite definir restrições ao modelo limitando o número máximo de instâncias dessa classe no modelo.

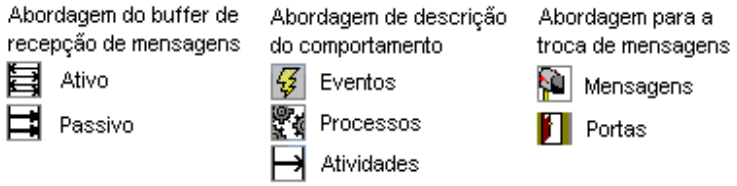

Figura A1.2 - Abordagens para a composição de um paradigma de simulação

Classes são detalhadas em subdiagramas (ver figura A1.1) onde descrevem-se as classes componentes das mesmas, caso estas existam. A identificação de um subdiagrama é feita pelo nome da classe sendo detalhada inserido em um retângulo de borda tracejada.

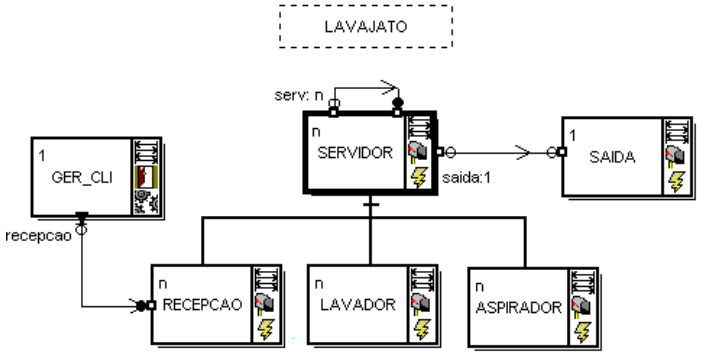

Figura A1.3 - Subdiagrama

1.2 - Relacionamento de conhecimento

Relacionamentos do tipo *conhecimento* denotam que um objeto conhece outro (e desta forma pode se comunicar com este). Um relacionamento do tipo *conhecimento* denota fraco acoplamento entre os objetos. A figura A1.4 detalha os tipos de relacionamento de conhecimento possíveis em SIMOO.

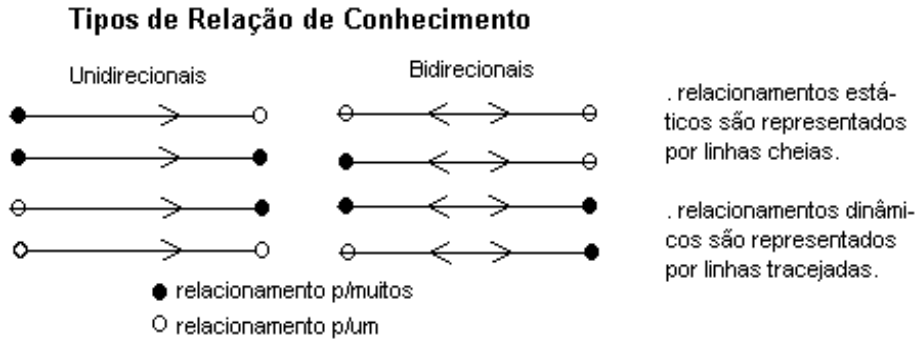

Figura A1.4 - Relacionamentos de conhecimento

Relações de *conhecimento* possuem sentido e cardinalidade. Quanto ao sentido podem ser uni ou bi-direcionais. Quanto à cardinalidade podem ser relacionamentos do tipo “um para um”, “um para muitos” ou “muitos para muitos”. A cardinalidade maior que 1 deve ser especificada. Isso pode ser feito de modo absoluto indicando-se numericamente a cardinalidade do relacionamento, ou de forma relativa utilizando-se

um asterisco. O asterisco irá indicar, para o caso dos relacionamentos estáticos, que a cardinalidade é determinada pela quantidade de instâncias no diagrama de instâncias. Nos relacionamentos dinâmicos o asterisco indica cardinalidade ilimitada. Relacionamentos estáticos são aqueles que devem ser resolvidos durante a geração do modelo executável. Relacionamentos dinâmicos, por outro lado, só podem ser resolvidos em tempo de execução do modelo. Durante a geração de código pode-se determinar apenas as variáveis que irão armazenar as futuras referências.

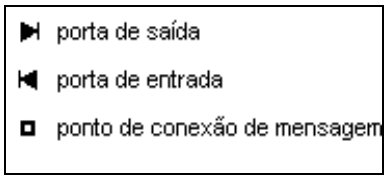

Figura A1.5 - Tipos de pontos de conexão

Os arcos que representam relações de conhecimento não são conectados diretamente às classes no diagrama. As conexões são feitas através de *pontos de conexão* (figura A1.5). Os pontos de conexão podem ser de dois tipos dependendo da abordagem de comunicação utilizada. Se a abordagem escolhida for *comunicação por mensagens*, então serão usados pontos de conexão de mensagens. Se a opção for comunicação por portas, os pontos de conexão representarão portas de entrada ou saída.

1.3 - Relacionamentos de criação

Relacionamentos do tipo *criação* representam a relação entre geradores de elementos da classe “X” e a própria classe “X”. Em modelos de simulação é comum a existência de uma entidade permanente que seja responsável por criar instâncias de entidades temporárias segundo uma distribuição de probabilidade. No exemplo da figura A1.6, uma entidade da classe GERADOR gera entidades (instâncias) da classe CLIENTE. Classes criadas a partir do relacionamento de criação não são consideradas, obrigatoriamente, componentes da classe em cujo subdiagrama a criação está representada. Assume-se que as classes criadas a partir de um relacionamento de criação são agregadas ao modelo.

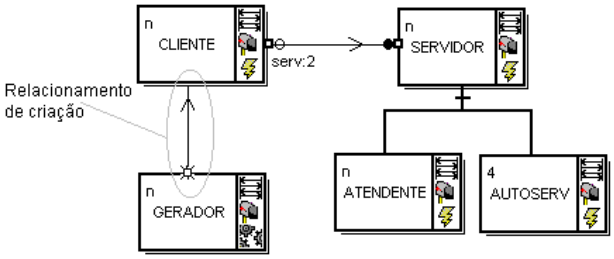

Figura A1.6 - Relacionamento de criação (GERADOR/CLIENTE)

1.4 - Elementos do diagrama de instâncias

No diagrama de instâncias, os objetos são representados por caixas divididas ao meio no sentido horizontal. Na parte superior da caixa apresenta-se o nome da classe e na parte inferior o nome da variável que contém a referência para a instância correspondente. Esta variável corresponde a um atributo da classe agregadora. O número no canto superior direito indica quantos objetos estão sendo referenciados pela mesma variável.

As relações de conhecimento estáticas devem ser explicitadas no diagrama de instanciamento. Cada ponto de conexão admite vários relacionamentos (independente do fato de ser ponto de conexão de mensagens ou portas), desde que sejam respeitados os limites de cardinalidade especificados no diagrama de classes. Pontos que não forem conectados serão considerados como relações nulas.

Os elementos gráficos podem ser observados na figura A1.7

Figura A1.7 – Elementos gráficos do diagrama de instâncias

Glossário

Atuadores: componentes que executam ações em um processo industrial (válvulas, relés, contatores, etc.)

Cliente: no contexto do COM, é o arquivo executável que usa os recursos de Interface implementados por um arquivo executável ou uma DLL.

Controlador Lógico Programável: é um equipamento microprocessado capaz de gerenciar o acionamento sensores e atuadores, que são ligados em placas de entradas e saídas, respectivamente, executando um software de controle armazenado em memória.

Drivers: no contexto dos sistemas SCADA, são módulos que permitem a comunicação com equipamentos industriais utilizando os serviços de mais baixo nível fornecidos por um driver do sistema operacional, por exemplo de uma placa serial.

Elemento de Visualização: objeto que representa a possibilidade de apresentação gráfica de um atributo, de um estado, de um método ou de uma classe.

Gauge: no contexto dos sistemas SCADA, são mostradores compostos por uma agulha que desloca-se sobre uma escala de fundo a qual pode ter diferentes regiões indicadas por cores específicas.

Marshiling: mecanismo de linearização dos parâmetros entre o cliente e o servidor COM, que consiste no processo ler uma informação da memória do processo cliente e passar esta informação para um buffer de memória, onde será lida pelo outro processo.

Script: linguagem de programação, geralmente é proprietária, embutida no sistema supervisório que permite programar ações para as situações ocorridas no processo que podem ser realizadas pelo sistema supervisório.

Sensores: componentes que fornecem sinais que indicam a condição de um processo industrial. Estes sinais podem ser digitais (posição, ligado/desligado, etc.) ou analógicos (temperatura, pressão, etc.).

Servidor: no contexto do COM, é o arquivo executável ou DLL que implementa uma ou mais interfaces.

Sistema de Tempo-real: são sistemas nos quais o resultado do processamento é válido apenas quando emitido dentro de um limite determinado de tempo.

Sistema Supervisório: categoria de sistemas que primariamente adquire dados para registro e análise, bem como provê supervisão e controle em aplicações sem requisitos estritos de tempo.

Tags: variáveis numéricas ou alfanuméricas envolvidas numa aplicação supervisória, podendo executar funções computacionais ou representar pontos de entrada e saída de dados do processo que está sendo controlado (correspondendo às variáveis como temperatura, vazão, nível, etc.).

Bibliografia

[BEC98] BECKER, Leandro Buss; PEREIRA, Carlos Eduardo. **Proposta de Ferramenta Orientada a Objetos para Modelagem, Simulação e Implementação de Sistemas Tempo Real Distribuídos**. Anais do Simpósio de Sistemas de Simulação e de Controle, Rio de Janeiro, 1998.

[BEC99a] BECKER, Leandro Buss. **Ambiente de modelagem e implementação de sistemas de tempo real utilizando o paradigma de orientação a objetos.** Dissertação de Mestrado - UFRGS, Porto Alegre, 1999.

[BEC99b] BECKER, Leandro Buss; PARDI Jr., Wilson & PEREIRA, Carlos Eduardo**. Proposal of an Integrated Object-Oriented Environment for the Design of Supervisory Software for Real-Time Industrial Automation Systems.** Workshop on Object-Oriented Real-Time Dependable Systems, Santa Barbara, 1999.

[BEC99c] BECKER, Gustavo Pizzini. **Programação com DCOM**. CPGCC/UFRGS, 1999. Disponível em < http://www.inf.ufrgs.br/ procpar/disc/cmp167/ trabalhos/sem99-1/T1/gustavo/DCOM.htm> Acesso em: 16 set. 2000.

[BRA99] BRAATZ, Alexandre. **Programação DCOM vs CORBA**. CPGCC/UFRGS, 1999. Disponível em < http://www.inf.ufrgs.br/ procpar/disc/cmp167/trabalhos/mp2000-1/AlexandreBraatz/ COMvsCORBA.htm > Acesso em: 16 set. 2000.

[CAN00] CANTÙ, Marco. **Dominando o Delphi 5 – A Bíblia**. São Paulo: Makron Books, 2000.

[COP97] COPSTEIN, B.. **SIMOO: Plataforma Orientada a Objetos para Simulação Discreta Multi-Paradigma**. Tese de Doutorado – UFRGS, Porto Alegre, 1997.

[ELI00] ELIPSE Software Ltda.. **ELIPSE SCADA 32 bits versão 2.19 (Windows 95/98; Windows NT)**. Versão de demonstração, atualizada em 03/05/2000, disponível em <http://elipse.com.br> Acesso em: 30 ago. 2000.

[ELI99] ELIPSE Software Ltda.. **Sistema de Supervisão e Controle: Manual do Usuário**. 1999, disponível em <http://elipse.com.br> Acesso em: 30 ago. 2000.

[HÖL00] HÖLTZ, Rudy Hamilton. **Estudo sobre a Utilização de Padrões de Comunicação para Interoperabilidade de Sistemas Computacionais em Automação Industrial.** PPGC/UFRGS, Porto Alegre, 2000.

[HOR97] HORSTMANN, Markus & KIRTLAND, Mary. **DCOM Architecture**. 1997. Disponível em <http://msdn.microsoft.com/library/backgrnd/html/msdn_dcomarch.htm> Acesso em: 10 set. 2000.

[LEÃ98] LEÃO, Marcelo**. Borland Delphi 4 – Curso Completo**. Rio de Janeiro: Axcel Books do Brasil, 1998.

[MIC00] MICROSOFT CORPORATION. **Distributed Component Object Model (DCOM): Downloads, Specifications, Samples, Papers, and Resources for Microsoft DCOM**. 2000. Disponível em: <http://www.microsoft.com/com/tech/dcom.asp> Acesso em: 10 set. 2000.

[MIC96] MICROSOFT CORPORATION. **DCOM Technical Overview**. 1996. Disponível em <http://msdn.microsoft.com/library/backgrnd/html/msdn_dcomtec.htm#dcomtec_arch> Acesso em: 10 set. 2000.

[PIN98] PINNOCK, Jonathan. **Professional DCOM Aplication Development**. Wrox Press: Birminham, 1998.

[RUM97] RUMBAUGH, James; BLAHA, Michael; PREMERLANI; William et al. **Modelagem e Projetos Baseado em Objetos**. Rio de Janeiro: Editora Campus, 1997.

[TIB00] TIBOLA, Leandro Rosniak. **Geração de Sistemas Supervisórios a partir de Modelos Orientados a Objetos de Aplicações Industriais.** Dissertação de Mestrado – UFRGS, Porto Alegre, 2000.

[TIB99] TIBOLA, Leandro Rosniak. **Geração de Sistemas Supervisórios a partir de Modelos Orientados a Objetos de Aplicações Industriais.** Anais da IV Semana Acadêmica do PPGC/ UFRGS. Porto Alegre, 1999.