Pós-Graduação em Ciência da Computação

GERENCIADOR DE OBJETOS PARA O AMBIENTE DE CONCEPÇÃO DE APLICAÇÕES EM BANCO DE DADOS

Por

Marcus Aurélio de Carvalho Macedo

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, JANEIRO/1991

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MARCUS AURÉLIO DE CARVALHO MACEDO

"GERENCIADOR DE OBJETOS PARA O AMBIENTE DE CONCEPÇÃO DE APLICAÇÕES EM BANCO DE DADOS"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR: PROF. DÉCIO FONSECA

RECIFE, JANEIRO/1991

Macedo, Marcus Aurélio de Carvalho
Gerenciador de objetos para o ambiente de Concepção de aplicações em banco de dados / Marcus Aurélio de Carvalho Macedo - Recife : O Autor, 1991.
vii, 165 folhas : il., fig.

Dissertação (mestrado) - Universidade Federal de Pernambuco. CIn. Ciência da Computação, 1991.

Inclui bibliografia e apêndices.

1. Engenharia de software. 2. Banco de dados - Ambiente - Aplicações. 3. Programação orientada a objetos. 4. Interface de comunicação. I. Título.

004.414.23	**CDU (2.ed.)**	**UFPE**
005.12	**CDD (21.ed.)**	**BC2003-399**

Dedico com amor e carinho a minha família:

Leda, Arthur, Julianne, Gabriella,

meus Pais e Irmãos.

AGRADECIMENTOS

Ao professor Décio Fonseca pela orientação e pelo apoio dados no decorrer deste trabalho.

Aos professores componentes da banca examinadora: Ana Carolina Salgado de Aguiar e Ulrich Schiel pelas sugestões e críticas feitas a este trabalho.

Aos professores do Departamento de Informática da UFPE pelos ensinamentos preciosos que contribuíram na minha formação acadêmica que direta e indiretamente levaram-me a elaboração deste trabalho.

Aos colegas do Mestrado: Mônica Bandeira, Eduardo, Mel, Carlinhos, Helmano, Alexandre, Max, Ana Cristina, Tereza, Tepedino, e outros, pela força e companheirismo.

A minha esposa Leda por todo amor, apoio emocional, compreensão e incentivo.

Aos meus filhos, Arthur, Juju e Gabi, pelo imenso amor e carinho sempre presentes.

Aos meus pais e irmãos que sempre me incentivaram, por seu carinho e afeto.

Em fim, a todos aqueles que contribuíram de alguma maneira para a realização deste trabalho.

S U M Á R I O

RESUMO

Existe hoje na evolutiva Ciência da Computação a intenção de tornar a relação homem-máquina mais flexível do ponto de vista de utilização dos recursos, desmistificando a idéia do "Cérebro Eletrônico" inacessível ao usuário. Novos softwares estão surgindo com esta filosofia, incorporando técnicas que facilitam o diálogo homem-computador, auxiliando-o no desenvolvimento de suas aplicações. Com esta visão, foi idealizado o Ambiente Interativo para Concepção de Aplicações em Banco de Dados, denominado POESIS, cuja finalidade é permitir que usuário, com diferentes níveis de conhecimento, possa criar interativamente suas aplicações em Banco de Dados.

Este trabalho descreve o GOLGO, parte do Ambiente POESIS responsável pelo gerenciamento dos objetos, gerenciamento de memória e de dispositivos. Constituído por uma estrutura flexível e apropriada à representação dos objetos componentes do sistema, o GOLGO visa prover o suporte a uma Interface orientada a objetos, mantendo um mecanismo para controle de acesso (sistema de segurança), além de um sistema para controle de versões das aplicações geradas no Ambiente Poesis. Foi criado com base na filosofia de Sistemas de Hipertexto e no Enfoque Orientado a Objetos.

ABSTRACT

Actually in the advanced Computation's Science there is the tendency to become the relationship between man-machine, from the point of view of resources more flexible, modifying the idea of "electronic brain" inaccessible for the great part of users. New softwares just now are arising with this philosophy, incorporating techniques that facilitate the dialogue user-computer, helping them in the development for theirs applications. In this line of thinking was created a interactive environment for conception of database applications, named POESIS whose purpose is to permit that users, with different and peculiar levels of knowledge, can create theirs database application in a interactive form.

This work describes the GOLGO, a part of the environment POESIS responsible by the management of objects, management of memory and of other computer devices. Constituted by an appropriate and flexible structure that makes possible the representation of components of the system, the GOLGO goals to support a objects interface oriented, maintaining an own control mechanism (safety / security system). Beyond this, the application offers a control system to be used like a control mechanism of versions for applications generated in the environment POESIS too. The GOLGO was created based in the concept of Hypertext Systems with approach object oriented.

CAPÍTULO 1

INTRODUÇÃO

Neste capítulo serão descritos os principais motivos que levaram o autor a realização deste trabalho. O projeto que deu origem a criação do Ambiente Poesis é apresentado com seus módulos componentes e respectivos objetivos. Os trabalhos realizados que contribuíram para o projeto são descritos também. A parte final apresenta a organização deste trabalho mostrando de forma resumida o conteúdo de cada um dos capítulos desta dissertação propiciando ao leitor uma visão geral do trabalho.

1.1 **MOTIVAÇÃO**

Estamos no limiar do ano 2000 e o que se pode constatar é que durante todos estes anos a tão falada crise do software nunca foi resolvida. O desenvolvimento da indústria de hardware avança a passos largos o que não acontece com o software. A tecnologia de microeletrônica está cada vez mais impulsionando o mercado de computadores no mundo. Nos últimos dois anos mais de uma dezena de novos microprocessadores de 32 bits invadiram o mercado a maioria seguindo a filosofia RISC que atualmente têm um desempenho de até 40 MIPs superando os da família Intel 386 e 486 com 5 a 10 MIPs. A indústria de periféricos vem acompanhando esta evolução com novos periféricos tais como os discos óticos tipo CD-ROM que armazenam novos tipos de dados tais como imagens e sons. Discos rígidos de alta capacidade estão surgindo, além de impressoras tipo laser com custo cada vez menor que farão parte do dia-a-dia de todos. Se for feita uma projeção para os próximos dez anos baseada nos dias atuais, o resultado é que o avanço da tecnologia de hardware vai chegar a um certo ponto, que nada valerá a pena se não houver a correspondente evolução do software.

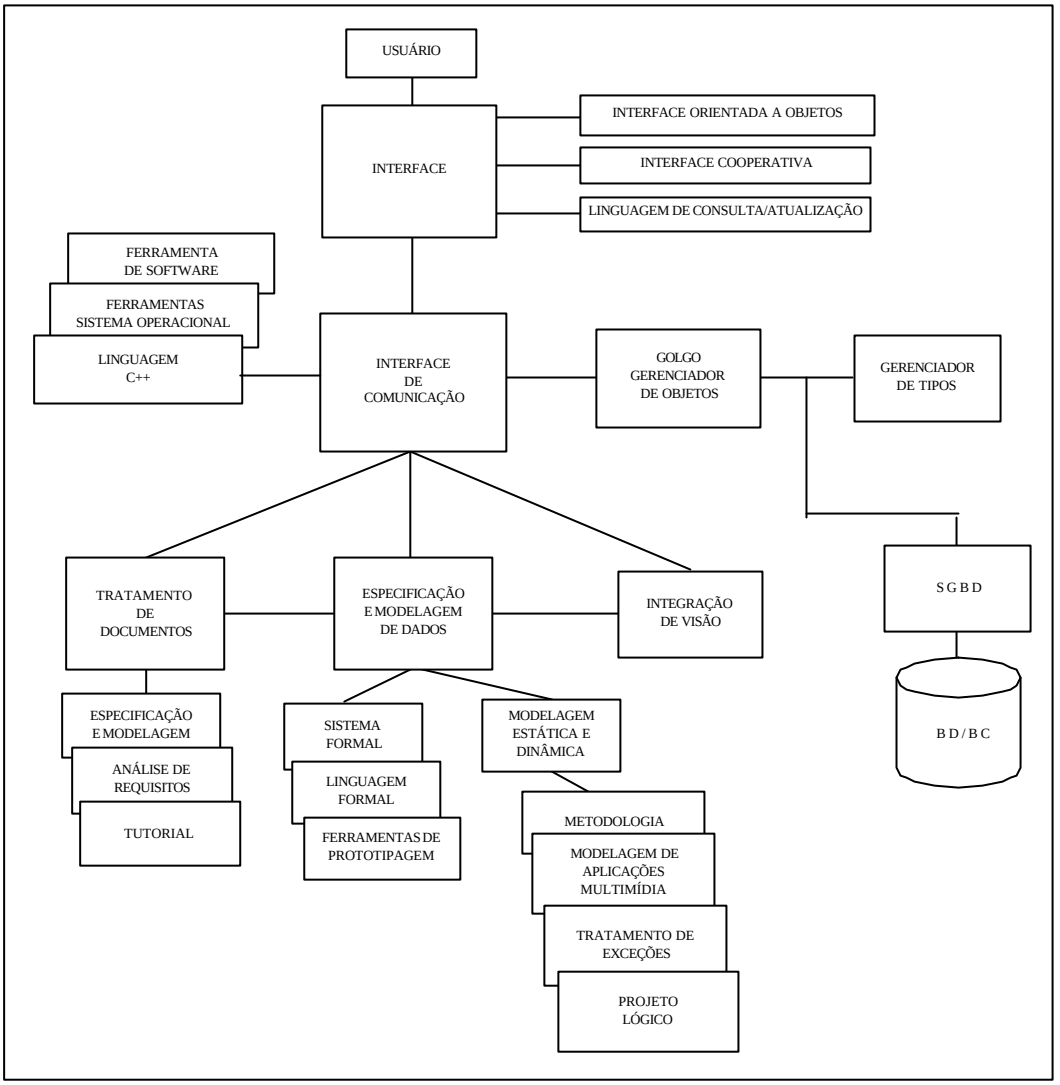
As pesquisas atuais têm convergido para uma nova realidade em termos de software e a tendência atual é a produção de ambientes de desenvolvimento de software com ferramentas rigorosas e eficazes que auxiliem o usuário na especificação, concepção, desenvolvimento e utilização de Sistemas de Informação de forma interativa e segura. Estes ambientes possuirão interfaces de simples utilização permitindo que pessoas de diferentes níveis de conhecimento possam projetar e manipular os seus Bancos de Dados sem a intermediação de especialistas.

Este trabalho pretende contribuir para a construção deste novo tipo de software um ambiente de desenvolvimento e manipulação integrado de aplicações multimídia.

1.2 **O PROJETO DE CRIAÇÃO DO AMBIENTE POESIS**

Um grupo de pesquisadores ligado a área de Banco de Dados do Departamento de Informática da Universidade Federal de Pernambuco motivados pelo desafio de construção de um protótipo de ambiente de desenvolvimento e manipulação de banco de dados multimídia concebeu um projeto de um ambiente denominado **POESIS** (palavra que em grego significa ***criação, concepção***). Este projeto integra as áreas de banco de dados, engenharia de software e inteligência artificial, com uma interação multidisciplinar de outras áreas de conhecimento, tais como: Lingüística Computacional, Psicologia Cognitiva, Educação, que contribuirão para o desenvolvimento de ferramentas que possibilitem a expressão e a solução de problemas clássicos em Banco de Dados de forma simples e segura. A figura 1 ilustra a arquitetura do Projeto de criação do Ambiente POESIS.

FIGURA 1 - ARQUITETURA DO PROJETO DO AMBIENTE POESIS



APRESENTAÇÃO DO PROJETO

O projeto é constituído dos seguintes temas:

1-Interface Orientada a Objetos (Cooperativa),
2-Tratamento de Documentos
- .Especificação e Modelagem de documentos,
- .Análise de Requisitos,
- .Tutorial

3-Integração de Visão,
4-Especificação e Modelagem de Dados
- .Modelo E/D
- .Metodologia
- .Especificação Formal
- .Modelagem de Aplicações Multimídia
- .Projeto Lógico
- .Tratamento de Exceções

5-Gerenciadores
- . Gerenciador de Objetos (GOLGO)
- . Gerenciador de Tipos

6-Linguagem de consulta e atualização de Banco de Dados.
7-Projeto Físico.

Os temas propostos são descritos em seguida.

1- Interface Orientada a Objetos

Consiste em desenvolver uma Interface Orientada a objeto baseada nos conhecimentos e experiências de outras áreas de pesquisa correlatas tais como: Engenharia de Software, Lingüística Computacional, Informática Educativa, Computação Gráfica e outras, que seguem basicamente as seguintes direções lógicas: a psicologia dos conceitos, os princípios de concepção e de organização do diálogo cooperativo, métodos de programação, especificação e concepção modular dos elementos da interface. No final do projeto ter-se-á uma **Ferramenta para a Construção de Interface Orientada a Objetos** e uma **Interface de Comunicação Cooperativa** cuja proposta consiste em uma Interface capaz de deduzir, através das ações realizadas pelo usuário dentro do ambiente, qual o objetivo que ele se propõe a alcançar e através da monitorização destas ações a interface o auxilia a atingir o objetivo final.

2 - Tratamento de Documentos

Neste tema serão pesquisados mecanismos para o processamento de documentos que irão permitir a **Especificação e Modelagem de Documentos** com a criação de um protótipo usando um Sistema de Hipertexto no nível conceitual auxiliando assim o processo de desenvolvimento de Sistemas de Informação que utilizam uma grande quantidade de documentos desde as primeiras etapas de concepção.

Outra pesquisa neste tema corresponde a **Análise de Requisitos,** onde se pretende contribuir para solucionar um dos grandes problemas no processo de desenvolvimento de software, correspondente ao fornecimento correto por parte do usuário das informações relevantes e precisas, e a identificação clara do problema a ser tratado.

Finalmente, a última pesquisa para este tema: **Tutorial,** no qual se pretende dotar o Ambiente Poesis de novos métodos de aprendizado substituindo os longos, tediosos, mal elaborados e desatualizados manuais tradicionais, por uma documentação interativa cujos objetivos são:

- aprendizado, por parte dos usuários que manipulam os Objetos do Ambiente, dos recursos disponíveis;
- A documentação de Aplicações Modeladas pelo usuário;
- A ajuda on-line solicitada pelos usuários do Ambiente.

3 - Integração de Visão

Este tema do projeto é uma linha de pesquisa que corresponde à modelagem de Visões e consiste na formalização dos resultados da análise de requisitos, onde se produz um esquema conceitual para representar de forma fiel as visões e assim eliminar as ambigüidades, imprecisões e incompreensões, decorrentes das formas tradicionais de análise de requisitos.

4 - Especificação e Modelagem de Dados

Este tema compreende as ferramentas de especificação e modelagem de dados propostas para o Ambiente Poesis, tais como o **Modelo E/D,** o primeiro modelo desenvolvido para ser utilizado no ambiente de POESIS, acrônimo de Estático/Dinâmico [BAND89].

Outra pesquisa ainda neste tema corresponde a **Especificação Formal de Aplicações**, cujo objetivo é o estudo de banco de dados orientado a objetos de forma integrada com a área de engenharia de software correspondente às linguagens de Especificação Formal, para se conseguir um nível formal no estudo dos modelos e assim ter seus conceitos formalmente especificados, sem ambigüidades e de forma independente das implementações.

A Modelagem de Aplicações Multimídia corresponde a outra linha de pesquisa a ser desenvolvida e aplicada ao ambiente.

Também neste tema temos o **Tratamento de Exceções** cujo objetivo é dar maior flexibilidade aos Sistemas de Informação através de mecanismos capazes de tratarem exceções de forma genérica durante a execução dos programas, e assim permitir que a continuidade normal de uma operação possa ser substituída por uma continuação excepcional na medida que uma exceção seja detectada.

5-Gerenciadores

Este tema compreende os seguintes projetos:

- **Gerenciadores de Objetos: Memória, Dispositivos e Ambiente** (GOLGO) descrito neste trabalho.

- **Gerenciador de Tipos** cujo objetivo é permitir a manipulação dinâmica de tipos de dados num contexto multimídia através de uma interface baseada nos conceitos de encapsulamento e herança de tipos.

6-Linguagem de consulta e atualização de Banco de Dados.

Neste tema será pesquisada uma linguagem flexível que permita efetuar consultas e atualizações em um Banco de Dados acoplado ao Ambiente Poesis.

7-Projeto Físico

Este tema compreenderá as pesquisas que visam a implementação das aplicações modeladas no Ambiente de forma semi-automática e otimizada.

CONSIDERAÇÕES FINAIS

O projeto apresentado tem como metas principais**:**

1) A avaliação e evolução do primeiro protótipo de Ambiente ;
2) Geração de protótipo incluindo os novos temas propostos;
3) Migração para Ambiente Unix.

A migração para Ambiente UNIX, em estações de trabalho do tipo Sun Sparc Station deverá ser realizada em paralelo com o projeto de pesquisa.

1.3 **ORGANIZAÇÃO DA DISSERTAÇÃO**

Esta seção apresenta o resumo dos capítulos que constituem este trabalho de dissertação.

O capítulo 2, denominado Gerenciador de Objetos: Conceitos e Características, descreve os requisitos básicos que determinaram o projeto do GOLGO (Gerenciador de Objetos Ligados por um Grafo Orientado) como um gerenciador para o Ambiente Poesis. Este capítulo define também as filosofias que influenciaram as suas características estruturais e funcionais: Hipertexto e Orientação a Objeto. Os elementos conceituais que compõem o GOLGO são descritos bem como as características gerais: composição de objetos, Interface Orientada a Objeto, Controle de acesso, Controle de versões, etc.

O capítulo 3, denominado Descrição do GOLGO, apresenta a arquitetura geral com os respectivos Módulos Gerenciadores. Em seguida, cada Módulo Gerenciador é descrito a partir da

especificação de cada classe de objetos que compõe. São especificados os objetos construtores, os objetos que gerenciam a memória, os objetos de gerenciamento do Ambiente, os objetos que gerenciam os dispositivos e concluindo este capítulo é feita a descrição da funcionalidade do GOLGO e do mecanismo de funcionamento.

O capítulo 4, denominado Objetos do Ambiente, apresenta a descrição de cada objeto constituinte do Ambiente Poesis, com as respectivas estruturas e operações.

O Capítulo 5, denominado Aspectos da Implementação do GOLGO, descreve o ambiente de implementação através das características da linha de equipamento e da ferramenta de programação escolhida. Em seguida é apresentada a arquitetura funcional e finalmente a análise e projeto de implantação.

O Capítulo final apresenta as considerações sobre o trabalho e as perspectivas de evolução do GOLGO.

CAPÍTULO 2

GERENCIADOR DE OBJETOS: CONCEITOS E CARACTERÍSTICAS

2.1 **INTRODUÇÃO**

O ambiente Poesis foi idealizado para prover seus usuários de diversas ferramentas para o desenvolvimento de suas aplicações, permanecendo sempre em constante evolução com a agregação de novas ferramentas baseadas em novas tecnologias e conceitos. Para a concepção deste ambiente adotou-se a filosofia de orientação a objeto visando torná-lo extensível e modular. Em conseqüência disto, o projeto do Gerenciador para este ambiente foi norteado segundo esta filosofia, o que de certo ponto de vista se revela promissor no tocante às facilidades decorrentes da utilização de seus conceitos tais como: modularidade, extensibilidade, herança, polimorfismo, etc. A partir desta determinação foi criado um gerenciador de objetos denominado GOLGO, cujo nome deriva das iniciais das palavras: Gerenciador de Objetos Ligados por um Grafo Orientado. A filosofia de concepção deste gerenciador será descrita no decorrer deste capítulo, apresentando os conceitos de sistemas baseados em Hipertexto e do Paradigma da Programação Orientada a Objeto. Os conceitos de Dicionário de Objetos, controle de versões, controle de acesso e restrições foram também contemplados neste projeto.

Este capítulo aborda basicamente os conceitos envolvidos na concepção de um Gerenciador de Objetos para fornecer ao Ambiente Poesis os subsídios necessários a sua implementação.

2.2 **CARACTERIZAÇÃO DO PROBLEMA**

O projeto de concepção de um ambiente orientado a objetos representa o desenvolvimento de um sistema complexo que necessita de um suporte estrutural para viabilizar a integração dos diversos módulos do ambiente POESIS. Este suporte estrutural deve ser fornecido por um mecanismo gerenciador (GOLGO) que efetue:

- Gerência de armazenamento e recuperação de informações;
- Controle de acesso;
- Controle de segurança e integridade;
- Intercâmbio entre as diversas ferramentas do ambiente.

A linha de desenvolvimento do Ambiente Poesis escolhida foi baseada nos conceitos de *Programação Orientada a Objetos* tem em vista o aproveitamento e modularidade decorrentes deste enfoque. Assim, nasceu a idéia do desenvolvimento de um Gerenciador de Objetos (GOLGO) para ser utilizado no controle e manipulação dos objetos a serem criados, que traduz a forma como os mesmos estão estruturados no referido ambiente.

2.3 **CONCEITOS ENVOLVIDOS NA CONCEPÇÃO DO GERENCIADOR DE OBJETOS**

A maneira pela qual foi idealizado o GOLGO teve como base duas linhas de pesquisas recentes: a filosofia de hipertexto e os conceitos do enfoque orientado a objetos. Da filosofia de Hipertexto e da orientação a objeto surgiu a idéia de implementar uma estrutura em forma de grafo

orientado, onde os nós seriam objetos do ambiente e estariam ligados por arcos que permitissem a navegação por parte do usuário. Cada objeto do ambiente nesta conceituação seria independente um do outro, possuindo também interfaces próprias, modulares e manipuláveis por ferramentas existentes no ambiente. Esta seção descreverá os conceitos envolvidos nestas duas áreas de pesquisa que estão associados às características do GOLGO.

2.3.1 **FILOSOFIA DE HIPERTEXTO**

O conceito de Hipertexto pode ser entendido como sendo um documento organizado de forma não linear em um grafo orientado onde os nós contém textos e os arcos denominados de links ou referências [CONK87, VASC88] fazem a conexão entre estes nós permitindo assim a manipulação do documento.

A idéia de hipertexto não é nova, surgiu a partir de 1945 de um artigo "As We May Think" publicado por Vannevar Bush que concebeu um sistema de armazenamento de textos, desenhos, fotografias, etc., baseados em um esquema de Índices, que permitiam a recuperação de informações, denominado *Memex* [CONK87]. O termo hipertexto foi adotado posteriormente na década de 60 por Ted Nelson na definição do sistema *Xanadu*. Existem quatro categorias de sistemas de hipertexto segundo a classificação de Jeff Conklin:

- Sistemas Macro-literários;
- Sistemas Exploratórios;
- Browsing System;
- Sistemas de propósito geral.

A categoria de Sistemas Macro-literários compreende os sistemas com o propósito de integração dinâmica de diversos volumes da literatura existente, em larga escala. Os Sistemas Exploratórios compreendem aqueles usados na estruturação e convergência de idéias para auxílio na solução de problemas. A categoria Browsing Systems compreendem aqueles sistemas voltados para o acesso rápido às informações, sendo semelhantes aos sistemas macro-literários porém em escala reduzida. Os sistemas de propósito geral são aqueles que usam o conceito de hipertexto, mas com objetivos diferentes das outras categorias.

O GOLGO utiliza a filosofia de hipertexto para prover o ambiente Poesis de uma estrutura dinâmica de manipulação de objetos e pode ser classificado na categoria de **sistemas de propósito geral**. A idéia básica é permitir ao usuário o acesso ao objeto desejado via navegação pela estrutura. Como também, permitir a inclusão ou remoção de objetos, revestindo assim o ambiente da característica fundamental: extensibilidade.

2.3.2 **ENFOQUE ORIENTADO A OBJETOS**

Os conceitos do enfoque de Orientação a Objeto podem ser compreendidos através do Paradigma de Orientação a Objetos que abrange dois aspectos [TAKA88]:

- O modelo de computação do objeto;
- A filosofia de desenvolvimento.

O conceito de objeto está ligado ao modelo de computação do mesmo. Neste modelo, um objeto é uma entidade semi-autônoma dotada de um comportamento representado por um conjunto de operações internas que o manipula, e um estado interno correspondente a uma memória local apenas acessada pelo objeto. Esta forma de ocultar o objeto é denominada de *Encapsulamento*. O processamento é ativado pelo envio de uma mensagem a um determinado objeto. A mensagem ativa uma operação interna do objeto. Neste modelo somente existem objetos e mensagens, e isto fornece a característica de uniformidade ao modelo.

A filosofia de desenvolvimento de sistemas com enfoque de Orientação a Objeto é feita com a

utilização de técnicas de abstração, tais como: *classificação, generalização e agregação.* A classificação é usada para a construção de ***Classes*** e torna possível agrupar objetos com características comuns produzindo assim o que se chama ***Hierarquia de Classes***. Outro conceito muito importante atrelado ao conceito de hierarquia de classes é o conceito de ***Herança.*** Herança é a capacidade que um objeto tem de herdar as características estruturais (dado) e as operações internas das classes superiores, ou super-classes. Esta organização dos objetos em uma hierarquia de Classes sugere a visualização desta estrutura pelos processos de generalização e especialização. A organização hierárquica de classes e objetos pode ser imaginada na forma de árvore. O processo de generalização corresponderia a uma navegação para cima buscando características comuns a diversos objetos para a criação de uma classe superior (super-classe).

Outras áreas de pesquisa contribuíram para consolidar os conceitos da Orientação a Objeto tais como: Especificação Algébrica e Tipos Abstratos de Dados, com a idéia de separar a parte de especificação do dado da sua implementação; projeto Conceitual em Base de Dados, com as técnicas de abstração: Classificação, Generalização, Agregação, Instanciação, Especialização e Refinamento.

2.4 **CARACTERÍSTICAS DO GERENCIADOR DE OBJETOS - GOLGO**

Nesta seção serão descritas as características e conceitos usados na concepção do **GOLGO** que dará suporte a implementação do ambiente para concepção de aplicações em Banco de Dados (Poesis). O GOLGO tem por objetivos:

- Criar uma estrutura flexível para representação dos objetos necessários ao Ambiente Poesis;
- Suportar uma interface orientada a objetos;
- Elaborar e manter um sistema de controle de acesso visando o aspecto de segurança;
- Controlar as versões de aplicações modeladas pelo usuário.

2.4.1 **CONCEITO DE NÓ**

O GOLGO utiliza uma estrutura de grafo orientado composto por nós e arcos com semânticas próprias para permitir a representação e manipulação dos objetos do ambiente. Existem basicamente três tipos de nós na estrutura do GOLGO:

- NÓ DE CLASSE,
- NÓ DE INSTÂNCIA, e
- NÓ DE COMPOSIÇÃO.

O **Nó de Classe** é usado para representar as classes de objetos existentes no ambiente. A cada Nó de Classe está associado um conjunto de informações que representam as características de uma determinada classe de objetos permitindo sua manipulação. Este conjunto é composto pelas seguintes informações:

- **Endereço de Localização** usado para acesso e recuperação de uma determinada classe;
- **Status**, corresponde ao atributo do Controle de Acesso que permite restringir o uso dos objetos do ambiente;
- **Identificador** e **Descritor** usados para a representação externa do objeto que constituem o Dicionário de Objetos do Sistema;
- **Conjunto de Referências de Navegação** usado para permitir a navegação no sistema, que permitirá o acesso aos demais nós conectados;
- **Conjunto de Operações Internas** usado para ativação dos métodos de uma determinada classe de objetos;
- **Conjunto de Definições Visuais** usados para a definição da interface de cada objeto do ambiente (janelas, ícones, etc.);
- **Conjunto de Restrições** que limita a formação de instâncias do objeto;
- **Conjunto de Referências de Processos** usado para ativação de operações entre objetos de classes diferentes (operações externas);

- **Conjunto de Referências de Composição** que permitirá a composição de objetos.

O **Nó de Instância** é utilizado para representar as instâncias de objetos do ambiente. É portanto considerado um nó genérico, pois sua característica estrutural dependerá da estrutura do objeto a que pertence. Associadas à cada Nó de Instância existem as seguintes informações:

- **Endereço de Localização** usado para permitir a recuperação de uma determinada instância;
- **Status**, corresponde ao atributo do Controle de Acesso usado para restringir o acesso a um dado Nó de Instância.

O **Nó de Composição** foi criado para implementar o conceito de ***Composição de Objetos***. É usado para representação de instâncias de objetos denominados **Componentes** e **Compostos**. Um objeto é denominado *composto* quando utiliza as características estruturais e comportamentais de outros objetos para compor suas próprias características, sem necessariamente ocorrer uma hierarquização e consequentemente herança destas propriedades. Um objeto é considerado *componente* quando participa da composição de um determinado objeto *composto*. Isto ocorre com os objetos do Modelo E/D [BAND89], onde o Objeto Atributo é usado para compor semanticamente o Objeto Entidade e o Objeto Relacionamento. O Nó de Composição possui algumas informações associadas, tais como:

- **Endereço de Localização** usado para acesso e recuperação de um determinado Nó de Composição;
- **Status**, corresponde ao atributo do Controle de Acesso;
- **Identificador** e **Descritor** usados para a representação no Dicionário de Objetos do Sistema;
- **Conjunto de Referências de Composição** que permitirá a composição de objetos;
- **Endereço de Localização** do Nó de Instância associado.

2.4.2 **CONCEITO DE ARCO**

No GOLGO existem três tipos de arcos: Links de Navegação, Links de Processos e Links de Composição.

Os *Links de Navegação* constituem o ***Conjunto de Referências de Navegação*** correspondente ao Nó de Classe e são usados para a navegação no grafo. Eles permitem que a partir de um determinado Nó de Classe (Origem), seja possível acessar um nó associado (Destino), que poderá ser um outro Nó de Classe, um Nó de Instância ou um Nó de Composição. Desta forma, torna-se possível a navegação através dos diversos objetos do ambiente POESIS. Os links de navegação poderão ser visualizados através de Definições Visuais definidas para cada Objeto do Ambiente, e contém basicamente o Endereço de Localização do nó destino e o Status associado, que permite o controle do acesso aos objetos.

Os *Links de Processos* correspondem ao ***Conjunto de Referências de Processos*** associado a cada Nó de Classe. Estes Links definem as operações consideradas externas, por envolverem a manipulação de objetos de classes diferentes, e portanto não sendo consideradas Operações Internas de um objeto específico do ambiente. Os Links de Processo permitem a ativação por parte do usuário dos diversos processos existentes, visualizados sob a forma de uma Definição Visual. Um Link de Processos contém basicamente um Identificador do Processo, o Status associado e um apontador que permitirá a ativação do processo. Os processos internos são os métodos associados a cada classe de objetos componentes do ambiente e são ativados no decorrer da utilização das instâncias dos mesmos.

Os *Links de Composição* correspondem ao ***Conjunto de Referências de Composição*** associado a um determinado Nó de Classe ou a um Nó de Composição. Foram criados para implementar o conceito de *Composição de Objetos* (seção 2.4.3) e podem ser entendidos como sendo apontadores que associam os ***objetos compostos*** aos respectivos ***objetos componentes***. Este link possui uma semântica diferente dos outros citados acima. Existe um atributo para a classificação dos *links de composição*. O atributo indica: Link forte e Link fraco. Um link de composição é

considerado como *Link Forte* quando o objeto componente é obrigatório à composição do *objeto composto*. Por outro lado, o *Link Fraco* indica que o *objeto componente* poderá ser opcional (existir ou não) na composição do *objeto composto*.

2.4.3 **COMPOSIÇÃO DE OBJETOS**

Uma outra característica do GOLGO muito importante, consiste na capacidade de explicitar a composição de objetos. Isto é realizado a partir do Nó de Composição e do Link de Composição. Estes dois elementos reunidos permitem determinar a composição de objetos com um grau de flexibilidade tal, que permite ao usuário modificar a composição de acordo com as características semânticas dos objetos envolvidos. São muitas as aplicações para esta propriedade, e vão desde a implementação do Modelo E/D, no qual o usuário participa ativamente da composição de seus objetos a partir da interface de Modelagem de Dados, até aplicações gráficas das mais variadas, tais como: Sistema CAD, onde as características de acoplamento de peças estariam definidas para a composição de um objeto;

2.4.4 **CONTROLE DE ACESSO**

O controle de acesso está ligado ao aspecto de segurança e diz respeito à privacidade das informações dos usuários. Um sistema deverá oferecer um controle de acesso que no mínimo possua estes dois aspectos.

O GOLGO possui um controle de acesso que permite a implementação de um mecanismo de visão muito simplificado, porém bastante eficiente no tocante às características deste gerenciador de objetos e ao que ele se propõe. A idéia básica deste controle de acesso é permitir que determinados usuários possam criar seus objetos, instâncias, etc., tornando possível a ocultação destes elementos para os demais usuários do sistema. Além disso, é de interesse do Ambiente Poesis [BAND89] permitir o acesso aos objetos, métodos, etc., àqueles usuários que tenham um certo grau de conhecimento para utilização do sistema. Quanto ao grau de conhecimento um determinado usuário poderá ser:

- Leigo,
- Intermitente,
- Especialista e
- Superusuário.

O usuário LEIGO é aquele que possui apenas conhecimento da aplicação que trabalha. A este tipo de usuário é dado um tratamento especial pela Interface, auxiliando-o através de interações mais simples. O usuário INTERMITENTE é aquele que se enquadra na faixa situada entre usuários Leigos e Especialista. São aqueles que possuem algum conhecimento de Modelagem de Dados, etc., mas que com um certo grau de limitação que o impede de classificá-lo como Especialista. É o usuário intermediário que terá uma visão intermediária e uma iteração com a Interface conforme o seu nível. O usuário ESPECIALISTA é aquele que possui conhecimentos bastante solidificados nos conceitos envolvidos na manipulação do Ambiente, tendo portanto, mais opções oferecidas pelo sistema. O SUPERUSUÁRIO é aquele que possui o acesso total e irrestrito a todos os objetos do sistema. Se enquadram nesta categoria aqueles usuários responsáveis pela administração do sistema e os projetistas do Ambiente. Existe portanto, uma hierarquia de tipos de usuário.

A hierarquia baseia-se no conhecimento e utilização do sistema da seguinte forma:

1. Leigo - pouco conhecimento (ou nenhum);
2. Intermitente - possui algum conhecimento;
3. Especialista - detém conhecimento dos recursos do ambiente;
4. Superusuário - Alto grau de conhecimento.

O Superusuário é o único que possui o acesso a qualquer componente do sistema sem qualquer restrição.

Além do controle de acesso limitado pelo grau de conhecimento do usuário, o sistema permite um outro controle que se baseia nos conceitos de programação orientada a objetos, com uma abordagem voltada para o direito de privacidade das informações. O sistema permite que um determinado conjunto de instâncias sejam acessadas somente pelo seu usuário criador. Neste controle de acesso um determinado componente do sistema (objeto, instância, método, etc.) poderá ser classificado em:

- Público ou
- Privado.

Como o próprio nome indica, um dado componente é considerado PÚBLICO quando seu acesso é permitido a qualquer usuário. E PRIVADO é aquele componente que somente poderá ser acessado pelo usuários responsável por sua criação no ambiente.

O STATUS DE UM COMPONENTE DO SISTEMA

São considerados componentes do sistema todos os Objetos de Gerenciamento definidos para o GOLGO (ver seção 3.2.2.2). Os Links de Navegação, por exemplo, é um componente do sistema que permite a navegação através dos objetos do sistema. É necessário entretanto, que este processo de navegação possua uma informação que permita a navegação limitada apenas aos objetos do sistema permitidos a um determinado usuário. Esta informação é o **STATUS** que define os parâmetros de acesso a um determinado componente. O STATUS é uma informação composta dos seguintes parâmetros:

- CÓDIGO INTERNO DO USUÁRIO;
- GRAU DE CONHECIMENTO;
- TIPO DE ACESSO.

O Código Interno do Usuário é um valor inteiro positivo armazenado na representação hexadecimal que está associado a cada usuário e gerado quando da implantação de cada usuário no Ambiente.

O Tipo de Acesso indica se um determinado componente é Público ou Privado. Se um determinado componente é Público o parâmetro assume o valor 0. Se for Privado assume o valor 1.

Desta forma, qualquer componente do sistema terá seu STATUS definido conforme a necessidade de acesso. Estes parâmetros podem ser incluídos quando da criação de uma instância de Objetos de Gerenciamento. Podem até ser modificados através da Interface, para se adequar às necessidades dos usuários que irão manipular o sistema.

FUNCIONALIDADE DO CONTROLE DE ACESSO

Este tópico irá descrever o funcionamento do controle de acesso que será implementado em cada Objeto de Gerenciamento. O procedimento é único e segue a seguinte estratégia:

- O Superusuário tem acesso a todo e qualquer Objeto de Gerenciamento definido no sistema. Seu código interno é 000.
- Os parâmetros do Status somente poderão ser alterados pelo Superusuário.
- Apenas o parâmetro TIPO DE ACESSO é que poderá ser alterado pelo usuário criador do componente.
- O acesso a um determinado componente será feito primeiro, verificando o TIPO DE ACESSO. Se o tipo de acesso for PRIVADO apenas o usuário que criou o componente é que poderá visualizá-lo e manipulá-lo. Se o tipo de acesso for PÚBLICO, então é feita a verificação do parâmetro Grau de Conhecimento.

2.4.5 **INTERFACE ORIENTADA A OBJETO**

No projeto do GOLGO foi considerada a necessidade primordial de fornecer ao Ambiente um suporte ao desenvolvimento de uma Interface Orientada a Objeto com as seguintes propriedades:

- Flexibilidade;
- Modularidade;
- Extensibilidade.

Estas propriedades somente são obtidas quando se implementa a Interface como parte encapsulada do objeto. Assim, cada objeto possui em seu interior as várias maneiras de interação com o usuário que poderá optar pela Interface mais adequada ao grau de conhecimento. Assim, um mesmo objeto poderia interagir para um certo tipo de usuário através de janelas com ícones, etc., e com outro através de janela de menus para escolha. Desta forma, estaria sendo implementado o ***princípio reativo*** proposto no Paradigma da Orientação a Objeto. Caberia então ao sistema a tarefa de conduzir o usuário através dos objetos existentes (Navegação) até que sua escolha fosse realizada ficando a partir daí a responsabilidade de interação com o objeto escolhido.

As diversas formas de apresentação, ou seja, a Interface de cada objeto, corresponde no GOLGO às Definições Visuais. Um determinado objeto poderá possuir um número quase ilimitado de formas de apresentação que serão ativadas a medida que forem sendo requisitadas.

2.4.6 **CONTROLE DE VERSÕES**

O desenvolvimento de aplicações em um ambiente é uma tarefa mais agradável de se executar, quando existem facilidades de manipulação das ferramentas, com uma Interface "amigável" e com certos dispositivos que permitam manter versões de classes de objetos e instâncias de objetos no ambiente. O Controle de Versões tem por objetivo:

- permitir a manipulação de novas versões de instâncias criadas durante o processamento no ambiente;
- permitir a manipulação de versões de objetos do ambiente.

As versões são administradas e identificadas no Dicionário de Objetos, sendo também controladas pela data de criação e pela identificação do usuário responsável pela modificação/criação. A criação de uma versão é opcional sendo ativada pelo usuário através da Interface Orientada a Objetos.

O controle de versões é realizado pelo objeto componente do gerenciador de objetos denominado Versões e será posteriormente descrito na seção 3.2.2.2 .

2.5 **ARQUITETURA COM OBJETOS DO AMBIENTE POESIS**

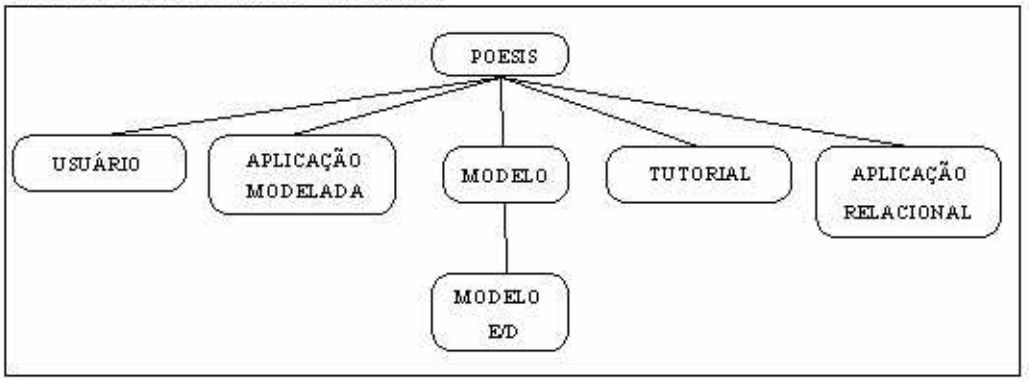
Nesta arquitetura que poderá ser visualizada a partir da figura 2, existem os Objetos do Ambiente que serão manipulados pelo usuário para o desenvolvimento de suas aplicações. O Objeto inicial denominado POESIS pode ser compreendido como sendo um Nó de Classe (figura 2) que é a raiz de todo o sistema. A partir do objeto POESIS através da navegação poderão ser acessados os demais objetos usados pelo sistema. Os principais objetos, especificados no capítulo 4, são: USUÁRIO, APLICAÇÃO MODELADA, MODELO, TUTORIAL e APLICAÇÃO RELACIONAL.

O objeto USUÁRIO irá conter e manipular informações sobre os usuários que utilizarão o sistema. A partir das operações (métodos) definidos para esta classe de objetos, será possível efetuar o controle dos usuários que utilizarão o Sistema. Cada usuário no Sistema possuirá uma SENHA, um CÓDIGO interno, e outras informações necessárias ao Sistema.

O objeto APLICAÇÃO MODELADA irá conter todas as aplicações geradas a partir do

modelo selecionado. A esta classe de objeto estará associada o Controle de Versões realizado por um Objeto de Gerenciamento descrito no seção 3.2.2.2. Este objeto permitirá ao usuário do Sistema, representar a sua Aplicação definida a partir do processo de Modelagem associado a cada Modelo utilizado. Serão utilizados os Nós de Composição para a representação de instâncias que herdam a Composição definida no Modelo utilizado.

FIGURA 2 - Arquitetura Inicial do Ambiente Poesis.



O objeto MODELO irá conter todos os modelos implementados no sistema e que serão usados para a geração das instâncias do objeto APLICAÇÃO MODELADA. Este objeto permitirá ao usuário escolher que modelo de dados será usado para desenvolver suas aplicações. Cada modelo de dados implementado no ambiente possuirá associado ao objeto MODELO seus respectivos Nós de Classe. A figura 2 ilustra o grafo do ambiente onde pode ser observada a existência do Nó de Classe correspondente ao Modelo E/D. A partir do Nó de Classe de cada modelo de dados, serão implementadas as classes referentes aos objetos que constituem o modelo, e onde serão representados com sua sintaxe e semântica para permitir a definição conceitual de qualquer aplicação. Este objeto utilizará o conceito de Composição de Objetos para efetuar a Modelagem onde um determinado objeto do Modelo poderá ser construído a partir da composição de outros.

O objeto TUTORIAL irá controlar informações como textos, imagens e questões visando o aprendizado dos usuários. Cada aplicação modelada poderá ter um tutorial correspondente funcionando como documentação e aprendizado ao mesmo tempo, sendo criada pelo usuário especialista da aplicação.

O objeto APLICAÇÃO RELACIONAL irá conter todas as aplicações modeladas e transformadas para o modelo relacional que serão *exportadas* para a implementação posterior em um SGBD relacional a ser definido. Podemos visualizar na figura 2 os links de navegação que permitem a partir do objeto POESIS a navegação para os demais Objetos do Ambiente.

CAPÍTULO 3

DESCRIÇÃO DO GOLGO

3.1 **INTRODUÇÃO**

Este capítulo apresenta o projeto de concepção e desenvolvimento do GOLGO, especificando sua Arquitetura Geral com detalhes dos principais módulos gerenciadores: Memória, Objetos e Dispositivos. A composição de cada módulo gerenciador é definida em termo dos objetos componentes, sua integração e funcionamento. Na primeira parte é apresentado o Gerenciador de Memória com sua filosofia. A segunda parte é dedicada à descrição do Gerenciador de Objetos é especificada a hierarquia de classes, sendo resultante da análise dos objetos e técnicas de abstração de dados tais como: classificação, fatoração, generalização e especialização. São descritos também a categoria de Objetos Construtores usados na implementação da hierarquia de classes e os denominados Objetos de Gerenciamento. A terceira parte descreve o módulo Gerenciador de Dispositivos com os objetos da classe Dispositivo e os objetos auxiliares. Em seguida, é apresentada a funcionalidade do GOLGO e finalmente a descrição do mecanismo de funcionamento.

3.2 **ARQUITETURA GERAL**

A arquitetura do GOLGO é baseada em objetos independentes entre si, com funções características bem diferenciadas que favorece futuras extensões. Entretanto, há uma interdependência entre eles, pois existem superfícies de conexão entre estes objetos tal como se fossem componentes de uma engrenagem mecânica, sendo que no caso de objetos elas ocorrem a nível de troca de mensagens no decorrer do processamento do ambiente.

A arquitetura geral do GOLGO pode ser definida em partes funcionais denominadas de Módulos Gerenciadores. Um módulo Gerenciador tem por objetivo efetuar o controle e a execução de um conjunto de procedimentos peculiares a uma determinada tarefa no ambiente. Os Módulos Gerenciadores são:

- Gerenciador de Memória;
- Gerenciador de Objetos;
- Gerenciador de Dispositivos.

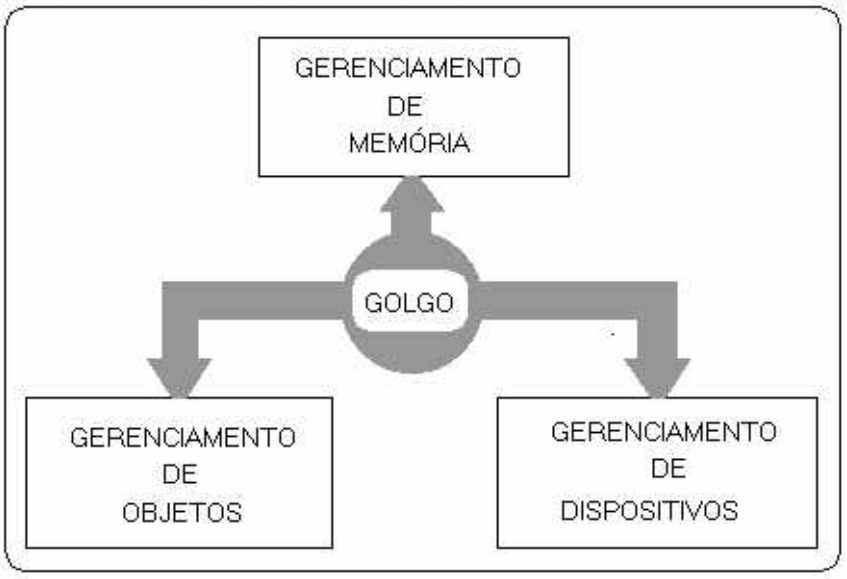
O módulo Gerenciador de Memória tem por objetivo efetuar o gerenciamento da memória principal (memória RAM) e da memória secundária.

O módulo Gerenciador de Objetos tem por objetivo controlar e manipular os diversos objetos que compõem o ambiente de forma a implementar os conceitos de objeto descritos no capítulo 2.

O módulo Gerenciador de Dispositivos (especificado na seção 3.2.3) tem por finalidade implementar os objetos que irão manipular a Interface Orientada a Objeto.

A figura 3 apresenta um diagrama ilustrando os Módulos Gerenciadores que compõem o projeto do GOLGO. Serão abordados a seguir cada um destes módulos gerenciadores com a especificação dos objetos componentes e seu funcionamento.

FIGURA 3 - Arquitetura Geral do GOLGO



3.2.1 **GERENCIAMENTO DE MEMÓRIA**

Os Objetos de Gerenciamento do GOLGO e suas instâncias são manipulados durante o processamento do sistema em memória RAM. Isto contribui para diminuir o tempo de acesso às estruturas internas dos objetos quando da manipulação de suas instâncias. Entretanto, a memória RAM não é tão extensa a ponto de permitir que o processamento se realize com todas as instâncias armazenadas nesta memória. Devido ao fator espaço de armazenamento, o gerenciamento da memória orientado a objeto deve permitir a manipulação parcial ou total de todas as instâncias dos Objetos de Gerenciamento associadas ao objeto do sistema que está sendo processado. A manipulação de instâncias em dispositivos de armazenamento persistente é outro aspecto importante no gerenciamento de memória e deve permitir a rápida recuperação das instâncias dos Objetos de Gerenciamento, bem como administrar as lacunas ou espaços de memória decorrentes da remoção de instâncias.

Na seção seguinte serão descritas as características do Gerenciamento de Memória, através da especificação da filosofia do projeto, sua arquitetura e funcionamento.

3.2.1.1 - **CARACTERÍSTICAS DO OBJETO MEMÓRIA**

O gerenciamento de memória no GOLGO é realizado pelo Objeto Memória que consiste de uma classe de objetos componentes possuindo cada um deles atribuições específicas, ou seja, estes objetos executam uma função específica dentro do Objeto Memória, sendo coordenados por um outro objeto que possui a função de controle dos demais. O Objeto Memória se propõe a gerenciar as alocações de instâncias dos diversos objetos do ambiente POESIS tanto em memória principal quanto na memória secundária. Além disso, permite a transferência de instâncias entre estes dois tipos de memória administrando o espaço existente de forma otimizada, possibilitando o reuso de blocos de memória utilizados. Outra função muito importante é o tratamento de overflow que torna possível o uso da memória principal sem interrupção no processamento global do ambiente.

Para uma melhor diferenciação será usado a expressão MEMÓRIA REAL para identificar a memória principal ou RAM e a expressão MEMÓRIA PERSISTENTE para identificar a Memória Secundária.

3.2.1.1.1 **FILOSOFIA DE ENDEREÇAMENTO**

Tanto na Memória Real quanto na Memória Persistente torna-se necessário o estabelecimento de uma forma padronizada de endereçamento, haja visto que é de fundamental importância que uma determinada instância de um objeto qualquer possua as mesmas características estruturais de quando manipulada na Memória Real. Esta preocupação justifica-se pela necessidade primordial de redução do tempo de processamento na transferência entre estas memórias.

Apesar dos tipos de memória possuírem características muito peculiares ambas possuem, entretanto, uma característica comum que permitirá tratá-las, do ponto de vista de endereçamento , com um enfoque único. Este enfoque, vem do fato de que certos objetos usados no ambiente possuem estruturas com encadeamento lógico como é o caso de listas, árvores, etc., o que impõe uma certa disciplina no manuseio das instâncias destes objetos.

Uma instância de um objeto qualquer na Memória Real pode ser vista como um agrupamento de bytes dispostos de forma seqüencial a partir de um endereço que será denominado ENDEREÇO BASE, possuindo um tamanho finito. Para o Objeto Memória o conteúdo não é relevante pois o controle das informações ali contidas somente poderá ser manipulado pelo objeto ao qual pertence a instância. Esta mesma instância na Memória Persistente é também vista como um conjunto de bytes localizados em um arquivo a partir de um ENDEREÇO BASE VIRTUAL e com um tamanho pré-definido. Do ponto de vista do gerenciamento de memória uma instância possui então:

- Endereço Base e
- Tamanho.

A partir destas duas informações será possível o tratamento de uma determinada instância nas respectivas memórias de forma padronizada permitindo assim a transferência entre elas, bem como sua manipulação por seus objetos correspondentes.

Para instâncias que possuem encadeamento lógico, no caso, das estruturas encadeadas, o endereçamento interno é feito usando o conceito de ENDEREÇO RELATIVO. Um Endereço Relativo é a posição ocupada por um fragmento de memória encadeado segundo as características estruturais do objeto, a partir do Endereço Base considerado como sendo a posição relativa 0 (Zero). Desta forma, seja qual for o tipo de memória em que a instância estiver localizada, é sempre mantido o encadeamento lógico.

Com isto, percebe-se que o objeto MEMÓRIA trata as instâncias de forma encapsulada, não interferindo em sua composição interna. Conforme será visto, no decorrer deste trabalho, esta maneira de manipular a memória é muito simples e flexível permitindo a manipulação dos mais diversificados tipos de instâncias.

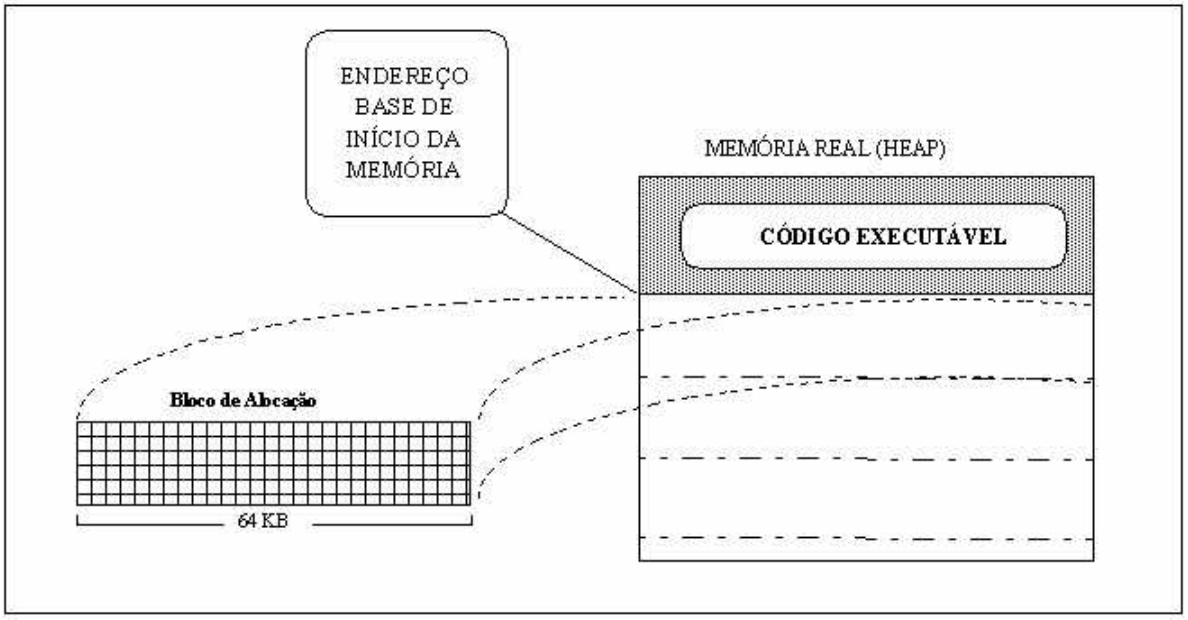
3.2.1.1.2 **FORMAS DE ALOCAÇÃO**

O Objeto Memória utiliza toda a região de Memória Real disponível para organizar o armazenamento do sistema. Esta região denominada de Área de Armazenamento Real (AAR) ou simplesmente Heap possui dois parâmetros básicos:

- O Endereço Real de início;
- Tamanho (em bytes).

O Endereço Real de Início ou Endereço Base da Área de Armazenamento Real irá variar conforme o equipamento, bem como o Tamanho que irá variar conforme a configuração física de memória. Estes dois parâmetros serão largamente usados no controle de alocação.

FIGURA 4 - Região da Memória Real utilizada para alocação em Blocos de Alocação de 64KBytes.



Embora a Heap compreenda toda a memória RAM disponível sua alocação é feita de forma parcial, isto é, são realizadas pré-alocações de conjuntos contíguos de bytes denominados Blocos de Alocação. Cada Bloco de Alocação possui um tamanho fixo parametrizado pelo sistema a partir de 64 KBytes. O objeto Memória utiliza, então, o Bloco de Alocação para efetuar as alocações de instâncias dos Objetos de Gerenciamento do GOLGO. A figura 4 ilustra a utilização dos Blocos de Alocação. Com isso, existe um controle das alocações de memória por Bloco de Alocação. Quando ocorre a alocação total de um Bloco, um novo Bloco é criado para permitir novas alocações. Da mesma maneira, a medida que são feitas desalocações de todas as instâncias de um Bloco este é desalocado da Memória Real.

Esta forma de alocação permitirá o uso mais flexível da memória tornando possível a execução de outros softwares e a utilização mais diversificada do Ambiente Poesis.

As alocações de memória solicitadas pelos Objetos de Gerenciamento que desejam manipular suas instâncias são feitas a partir do envio da mensagem ALOCAÇÃO ao Objeto Memória. Esta solicitação dará origem ao Processo de Alocação (PA). O Processo de Alocação é uma tupla:

PA = < IDP, IDO, ERV, EB, T, ISO, IB>

onde:

IDP é o identificador do Processo de Alocação;
IDO é o identificador do objeto de gerenciamento;
ERV é o endereço relativo virtual onde se encontra armazenada a instância em Memória Persistente;
EB é o endereço base de localização na Memória Real;
T é o tamanho da Memória Real alocada;
ISO é o indicador de estado de swap;
IB é o indicador de bloqueio da instância.

O objetivo do Processo de Alocação é registrar todas as alocações realizadas na Memória Real solicitadas ao Objeto Memória. Qualquer processo de alocação gerado pelo Objeto Memória é identificado pelo IDP não podendo existir dois IDP's iguais. Os parâmetros IDO e ERV são utilizados quando da transferência de uma instância de um determinado objeto identificado pelo IDO localizada no Endereço ERV da Memória Persistente para a Memória Real, os quais permitirão o

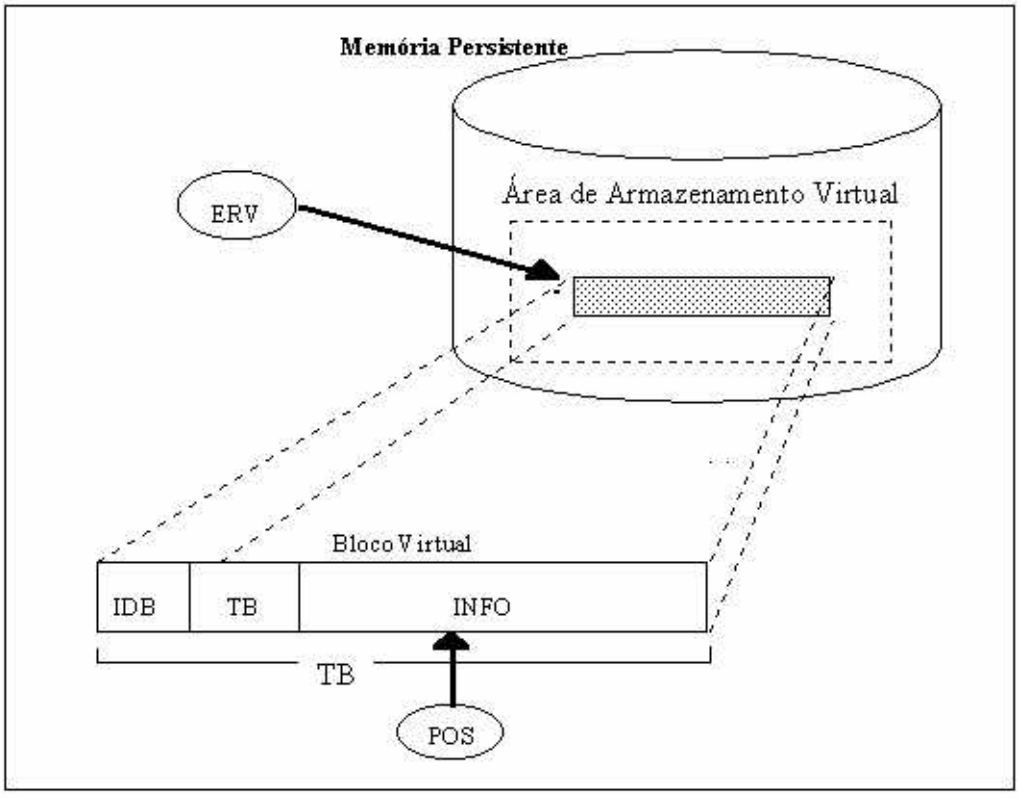
controle de transferências entre as memórias (ver maiores detalhes na seção 3.2.1.3.4). Os parâmetros EB e T são aqueles necessários a alocação de qualquer instância conforme especificado no início desta subseção. Os parâmetros ISO e IB são utilizados para indicar o estado do Processo de Alocação para o tratamento de Overflow da Memória Real conforme será visto na seção 3.2.1.1.3.

Quando uma instância de um dado objeto de gerenciamento é alocada em uma porção da Memória Real, esta alocação recebe um IDP que deverá ser utilizado por este objeto de gerenciamento toda vez que necessitar o acesso à memória e quando for necessário expandir o tamanho de sua instância. Em outras palavras pode-se dizer que o IDP é o elo de ligação entre um objeto de gerenciamento e sua instância armazenada na Memória Real, assim como a tupla <IDO, ERV> identifica esta mesma instância na Memória Persistente.

Na Memória Persistente a alocação de memória considerada permanente é realizada de outra forma. Cada objeto de gerenciamento possui uma região de Memória Persistente denominada de Área de Armazenamento Virtual que é o equivalente a um arquivo de dados. Uma instância na Memória Persistente é identificada por dois parâmetros básicos conforme indicado no parágrafo anterior:

IDO é o identificador do objeto de gerenciamento. Sua função básica é identificar a Área de Armazenamento Virtual no qual serão armazenadas todas as suas instância;
ERV é o Endereço Relativo Virtual onde a instância foi armazenada na Área de Armazenamento Virtual.

FIGURA 5 - Estrutura de armazenamento na Memória Persistente



A instância é então armazenada em um BLOCO VIRTUAL reconhecível pelo Objeto Memória que o localiza através do IDO e ERV. Um Bloco Virtual é a tupla :

<IDB, TB, INFO>, onde:

IDB é um caracter especial cuja finalidade é a identificação do início do Bloco Virtual;
TB é o tamanho (em bytes) ocupado pela instância;
INFO é a informação ou o conteúdo da instância.

A figura 5 ilustra a composição do Bloco Virtual numa Área de Armazenamento Virtual.

Alguns objetos têm a necessidade de manipular fragmentos de uma informação contida num Bloco Virtual e para isto o Objeto Memória permite o acesso ao início deste fragmento para que seja capturado e trazido à Memória Real para o processamento. Para isto existem algumas mensagens que manipulam estas frações do Bloco Virtual que necessitam de dois parâmetros fundamentais:

POS que indica a posição relativa de início do fragmento de instância;
TAM que indica o tamanho deste fragmento.

Estes dois parâmetros possuem uma única restrição dada pela expressão:

$$0 < (POS + TAM) \leq TB$$

que indica que a soma de POS e TAM não pode ser maior que o tamanho do Bloco Virtual (TB).

3.2.1.1.3 **FUNÇÕES BÁSICAS DO OBJETO MEMÓRIA**

O Objeto Memória foi criado para dar suporte aos Objetos de Gerenciamento do GOLGO no que tange a administração dos recursos de memória disponível para o ambiente. O controle destes recursos de memória pode ser caracterizado a partir da especificação das funções básicas por ele desempenhadas que são as seguintes:

a) Controle de Alocação e mecanismo de proteção;
b) Administração de espaços vazios ou desalocados;
c) Tratamento de Overflow;
d) Manipulação total ou parcial de instâncias.

a) CONTROLE DE ALOCAÇÃO E MECANISMO DE PROTEÇÃO

O objeto Memória administra todas as alocações solicitadas pelos Objetos de Gerenciamento a partir da manipulação do Processo de Alocação (PA) conforme especificado na seção 3.2.1.1.2 . O conjunto de informações contidas no Processo de Alocação permitem a identificação, localização e manipulação de qualquer instância armazenada na Memória Real. Entretanto, para que seja criado o Processo de Alocação é necessário que o Objeto Memória efetue algumas verificações:
(1) Ocupação total do Bloco de Alocação - Consiste em verificar se o Bloco de Alocação foi totalmente utilizado. Em caso afirmativo é efetuada uma alocação de outro bloco.
(2) Existência de Espaço Desalocado - Consiste em verificar se existe uma região contínua de Memória Real com tamanho suficiente para a alocação. Em caso afirmativo esta região é utilizada.
(3) Ocorrência de Overflow da Memória Real - A partir da ocupação total do Bloco de Alocação (verificação 1), sem que haja espaço desalocado suficiente para alocar a instância (verificação 2) e nem espaço disponível para a alocação na Memória Real, ocorre o que se chama de Overflow da Memória Real. Em caso afirmativo, o Objeto Memória administra a situação criando uma solução para o problema (ver item **c**).

Em qualquer uma destas situações é sempre criado o Processo de Alocação, ponto de partida para que a instância do objeto seja manipulada na Memória Real. As mesmas verificações ocorrem quando um objeto de gerenciamento necessita expandir sua alocação na memória.

No caso da expansão de uma instância e como o Objeto Memória considera que qualquer instância se encontra numa região contínua de memória torna-se necessário a execução dos seguintes procedimentos:

- Localizar uma nova região de Memória Real cujo tamanho seja igual ou superior ao novo tamanho da instância;

- Efetuar a transferência da informação da região original para a nova região;
- Atualizar o Processo de Alocação com as novas informações de EB e T;
- Liberar a antiga região para o administrador de espaços desalocados.

Na Memória Persistente o Objeto Memória efetua suas alocações seguindo a seguinte estratégia:

- Primeiramente verifica-se a existência de um "espaço vazio" ou seja, uma região de memória desalocada, cujo tamanho seja igual ou superior ao que se deseja armazenar. Esta verificação é realizada por um ***objeto*** com a função de administrar espaços desalocados. No caso de existência desta região, realiza-se sua realocação e em seguida uma transferência física da Memória Real para a Memória Persistente (salva) da instância. A antiga região ocupada fica então à disposição do objeto que administra os espaços desalocados.
- Para o caso de não existir esta região desalocada, a alocação será realizada então no próximo endereço livre disponível que no caso encontra-se no final da Área de Armazenamento Virtual.

Em qualquer uma das situações apresentadas acima, o Objeto Memória sempre irá fornecer, ao término da execução da alocação, o novo Endereço Relativo Virtual para que o objeto de gerenciamento possa manuseá-lo para futuros acessos àquela instância armazenada.

O Objeto Memória utiliza um mecanismo muito simples de proteção de instâncias em Memória Real: o Indicador de Bloqueio (IB). Quando torna-se necessária a proteção de uma determinada instância o Objeto Memória permite que seja ativado o Indicador de Bloqueio através do envio de uma mensagem com o IDP do Processo de Alocação desejado. Da mesma forma que permite desativar este indicador. O resultado disso é que o Processo de Alocação em questão fica protegido não podendo ser desalocado nem removido da Memória Real. Na Memória Persistente uma determinada instância somente pode ser removida quando solicitada pelo objeto ao qual pertence, havendo pois uma proteção total já que para cada objeto de gerenciamento existe uma correspondente Área de Armazenamento Virtual que contém informações restritas àquele objeto.

b) ADMINISTRAÇÃO DE ESPAÇOS DESALOCADOS

A reutilização de espaços desalocados é de fundamental importância no gerenciamento de memória. Em geral, alguns sistemas utilizam algoritmos que administram estes espaços ou "lixo" e são denominados de algoritmos de ***Garbage Colletion*** [COHE81, MCEN87]. Entretanto, para um ambiente orientado a objeto esta função pode ser realizada por objetos especializados. Logo, na concepção do Objeto Memória foram criados os objetos Tabela de Espaços Desalocados e Lista de Espaços Vazios, que administram respectivamente os espaços desalocados na Memória Real e na Memória Persistente. Estes objetos que serão especificados na próxima seção utilizam simplesmente dois parâmetros:

EBD que é o Endereço Base da região de memória desalocada e
TD que é o Tamanho desta região.

O Objeto Memória através deste dois objetos mapeam toda a memória disponível no ambiente para uma realocação, mantendo assim o controle e permitindo que estas áreas sejam reutilizadas a medida que forem sendo requisitadas.

Mesmo sendo administrados para posterior reutilização os espaços desalocados podem na sua totalidade, atingir um nível crítico de tamanho que de certa forma pode consumir recursos físicos do ambiente. Isto geralmente ocorre pela fragmentação das regiões desalocadas, gerando porções cujo tamanho não ofereça as condições de reuso. Neste caso, torna-se necessária a execução de uma COMPACTAÇÃO da memória. A Compactação consiste no rearranjo de todas as instâncias armazenadas, eliminando assim, os espaços desalocados. No caso, da Memória Real este procedimento realiza-se a partir do Processo de Alocação através da alteração do Endereço Base (EB) de cada instância após seu reposicionamento. Já no caso da Memória Persistente, o processo é mais

complexo, pois existe uma interligação de instâncias de todos os Objetos de Gerenciamento através do Endereço Relativo Virtual. Portanto, ao ser atingido o nível crítico de uma determinada Área de Armazenamento Virtual imediatamente realiza-se a Compactação Virtual a partir do objeto de gerenciamento Nó de Classe.

O nível crítico é parametrizado para cada tipo de memória. Esta parametrização permite o ajuste visando atingir um melhor rendimento do sistema. O nível crítico é dado em percentual sobre toda a área ocupada, visando não limitar o sistema a qualquer modelo de equipamento no qual for implantado. Portanto existe um nível crítico para a Memória Real e para cada Área de Armazenamento Virtual da Memória Persistente.

c) TRATAMENTO DE OVERFLOW DE MEMÓRIA

A condição de Overflow de memória ocorre quando toda a memória disponível encontra-se ocupada e existe a necessidade de alocar mais memória, gerando assim um impasse (deadlock) no sistema. Alguns sistemas possuem mecanismos de manipulação que permitem novas alocações de memória quando ocorre a situação de overflow. Estes mecanismos possuem uma estratégia voltada intrinsecamente para suas próprias características de funcionamento.

O Objeto Memória foi projetado para também administrar esta situação e tratá-la a partir de uma estratégia adequada a realidade de um ambiente onde manipulam-se instâncias de diversos tipos de objetos. A estratégia de tratamento de overflow do Objeto Memória que denomina-se SWAP e consiste basicamente da permuta ou troca de espaço de armazenamento entre a instância que deseja ser alocada na Memória Real e uma ou mais instâncias já alocadas na Memória Real. Em outras palavras, ocorre uma desalocação de uma ou mais instâncias armazenadas na Memória Real que são transferidas temporariamente para uma região da Memória Persistente denominada de EXTENSÃO da Memória Real, para que a instância que solicita uma alocação ao Objeto Memória possa ser armazenada. As instâncias que foram removidas temporariamente da Memória Real podem ser trazidas novamente a medida que seus objetos respectivos solicitarem o acesso a elas. Neste caso, se a condição de overflow ainda persistir, a mesma estratégia será usada para realocar esta instância. O processo de remoção temporária de uma instância por ocasião de um overflow é denominado de SWAP-OUT e o processo de realocação de uma instância removida por swap-out é denominado de SWAP-IN. Na seção 3.2.1.3 será descrito toda a mecânica de funcionamento destes dois processos e dos objetos envolvidos.

A escolha das instâncias que deverão ser removidas pelo processo de Swap-out faz parte também da estratégia de solução do problema de Overflow da Memória Real. Este processo de escolha denominado VARREDURA (Scan) utiliza para esta finalidade o Processo de Alocação (PA) no qual obtém dados acerca do tamanho da instância e sua disponibilidade. A disponibilidade é representada pelo Indicador de Bloqueio (IB) o qual protege a instância do processo de Swap-out. Quando uma instância é removida por Swap-out o Processo de Alocação registra esta ocorrência através do Indicador de Swap-out (ISO) que também será utilizado para indicar sua disponibilidade ao processo de Varredura. O processo de Varredura inicia uma pesquisa em todos os Processos de Alocação usando a seguinte estratégia:

- Inicialmente são pesquisados os Processos de Alocação encontrados no último Bloco de Alocação existente e a medida que não seja encontrado o PA ideal esta pesquisa retroage até o primeiro Bloco de Alocação;
- O PA ideal é aquele cujo tamanho da instância seja maior ou igual ao tamanho da instância que se deseja alocar;
- Caso não seja encontrado o PA ideal, é realizado o Swap-out dos últimos PA's do último Bloco de Alocação existente.

Cada Processo de Alocação removido por Swap-out é registrado para posterior realocação com informações de sua localização na Extensão da Memória Real através do Processo de Alocação Virtual (PAV). O Processo de Alocação Virtual é a tupla:

< IDP, EBe> , onde:

IDP é o identificador do Processo de Alocação Virtual que é o mesmo do PA;
EBe é o Endereço Base da instância na Extensão de Memória Real.

O processo de Swap-out ao transferir a instância para a Extensão da Memória Real desaloca aquela região da memória informando ao objeto administrador de espaços desalocados sua localização e tamanho. Após o processo de Swap-out, a alocação da nova instância ocorre normalmente estando a Memória Real com o espaço necessário para alocá-la.

d) MANIPULAÇÃO TOTAL OU PARCIAL DE INSTÂNCIAS

A manipulação flexível do Objeto Memória consiste em proporcionar o acesso e recuperação total ou parcial de instâncias na Memória Persistente. Com essa finalidade foram criados métodos que acessam parcialmente e totalmente instâncias nas Áreas de Armazenamento Virtual.

Conforme já especificado, um Bloco Virtual é a unidade de acesso a Memória Persistente que pode ser localizado através do Endereço Relativo Virtual (ERV). A partir do ERV um determinado fragmento pode ser recuperado através de sua Posição Relativa no Bloco Virtual (POS) e de seu tamanho (TAM). Usando estes parâmetros os métodos que serão especificados na seção 3.2.1.2 podem:

- Recuperar parte de uma instância trazendo-a para à Memória Real para processamento;
- Rearmazenar uma parte de uma instância alterada na Memória Real;
- Acrescentar uma nova parte a uma instância armazenada na Memória Persistente;
- Recuperar toda a instância armazenada na Memória Persistente;
- Armazenar toda uma instância na Memória Persistente.

Com esta filosofia o Objeto Memória permite que outros objetos possuam novas formas de manipulação o que representa de certa forma uma economia de recursos do ambiente. Por exemplo, um objeto do tipo árvore binária poderá ser implementado com métodos que permitam a manipulação parcial de suas instâncias não necessitando o armazenamento de toda a instância em Memória Real.

3.2.1.2 - **ARQUITETURA DO OBJETO MEMÓRIA**

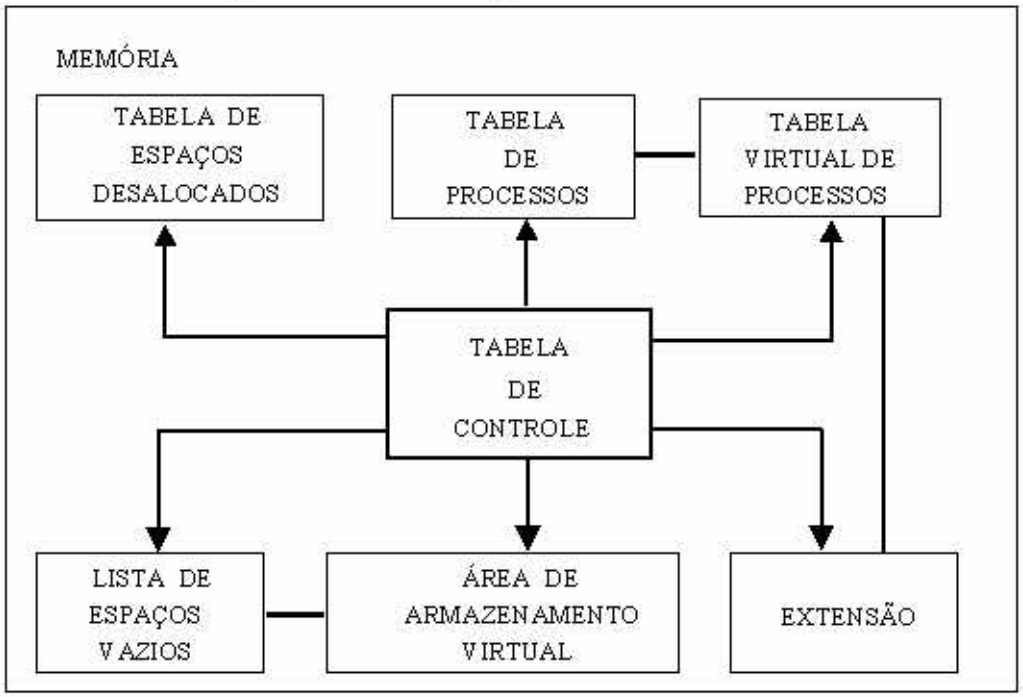
O Objeto Memória foi desenvolvido baseado na propriedade fundamental das Linguagens de Programação Orientada a Objeto: a Modularidade. Cada uma das funções básicas e estruturas básicas que dão suporte ao seu funcionamento estão sendo desempenhadas por objetos componentes, cada um realizando suas funções de controle e se comunicando através de mensagens trocadas entre eles. Nesta seção será apresentada a arquitetura do Objeto Memória e todos estes objetos com suas respectivas funções no sistema.

A arquitetura do Objeto Memória pode ser visualizada a partir da figura 6. Nela podemos visualizar os seguintes objetos componentes:

- Tabela de Controle;
- Tabela de Processos;
- Tabela Virtual de Processos;
- Tabela de Espaços Desalocados;
- Extensão;
- Área de Armazenamento Virtual;
- Lista de Espaços Vazios.

Cada um destes objetos que serão especificados a seguir funciona de maneira coordenada e interativa desempenhado a função de gerenciamento de memória para o ambiente POESIS.

FIGURA 6 - Diagrama da Arquitetura do Objeto Memória. As setas indicam o fluxo de mensagens enviadas durante o processamento.



3.2.1.2.1 **TABELA DE CONTROLE**

Este objeto é responsável pela coordenação dos outros objetos componentes do Objeto Memória. Seu objetivo básico é manter o controle do funcionamento dos demais objetos fornecendo e manipulando informações fundamentais ao gerenciamento da memória. Por ser um objeto indispensável ao funcionamento do Objeto Memória, a Tabela de Controle foi implementada como uma estrutura interna do Objeto Memória.

ESTRUTURA INTERNA

A Tabela de Controle (TC) pode ser compreendida como sendo a tupla:

TC = < EIM, NBA, TBA, PEL, CP, NCR, TAAV> onde:

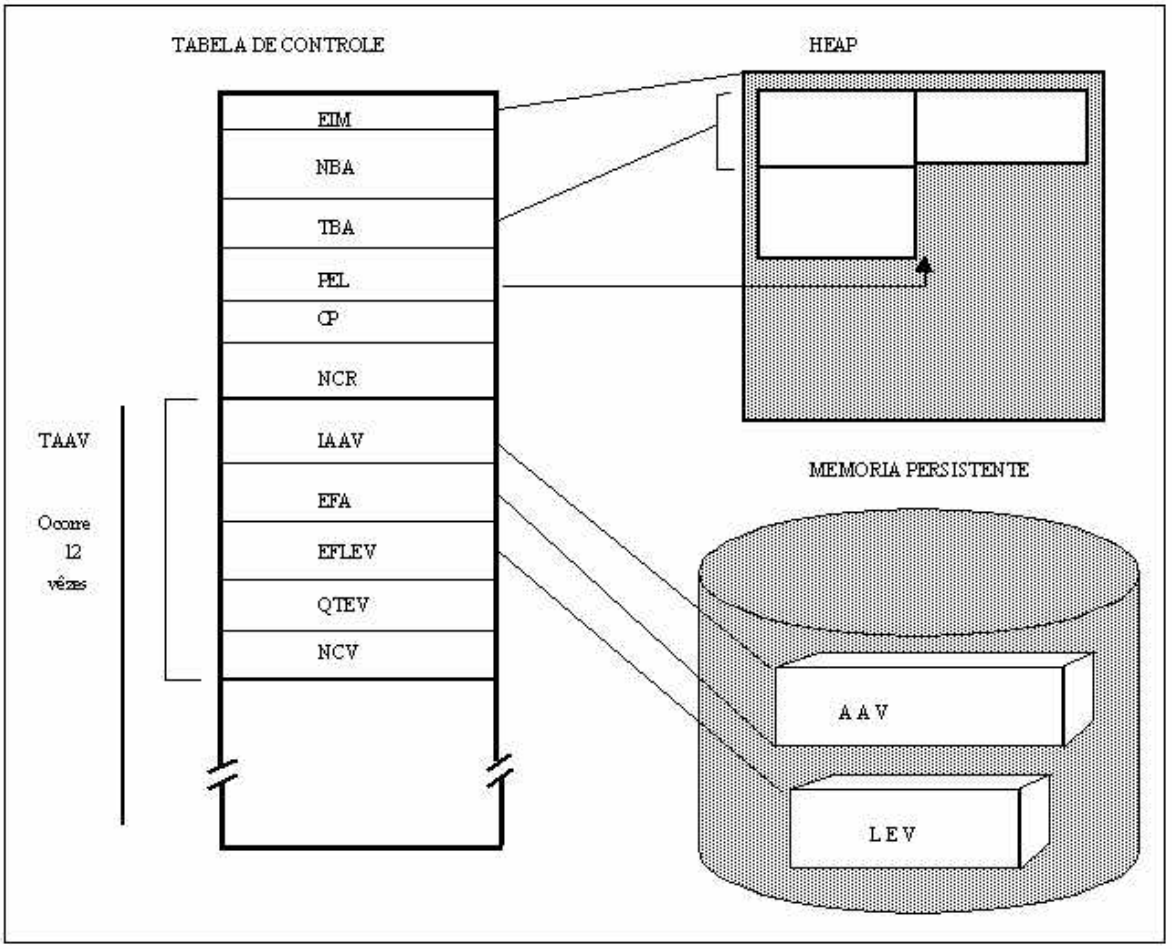
EIM é o Endereço de Início da Área de Armazenamento Real (Heap);
NBA é o Número de Blocos Alocados;
TBA é o Tamanho do Bloco de Alocação
PEL é o Próximo Endereço Livre na Heap;
CP é o contador de Processos de Alocação;
NCR é o nível crítico da Memória Real;
TAAV é a Tabela de Controle das Áreas de Armazenamento Virtual.

Estas informações são fornecidas e atualizadas a medida que o sistema entra em funcionamento e os demais objetos componentes do Objeto Memória executam suas funções. O parâmetro EIM é inicializado a partir de solicitação ao Sistema Operacional. Os parâmetros NBA e TBA são usados para o controle dos Blocos de Alocação. O parâmetro PEL indica a próxima posição de Memória Real disponível para alocação do Bloco de Alocação. O parâmetro CP é usado para fornecer a numeração seqüencial de IDP's usados pelos Processos de Alocação. NCR é usado para checar o nível crítico de espaços desalocados para ativação automática da Compactação Real e consiste de um percentual (Default 25%). A TAAV tem por objetivo controlar as Áreas de Armazenamento Virtual do ambiente e compreende 12 ocorrências de um conjunto de informações representado pela tupla:

TAAV = < IAAV, EFA, EFLEV, QTEV, NCV> onde:

IAAV é o identificador da Área de Armazenamento Virtual;
EFA é o Endereço correspondente ao Final da Área de Armazenamento Virtual;
EFLEV é o Endereço Final da Lista de Espaços Vazios;
QTEV é a Quantidade Total de Espaços Vazios;
NCV é o Nível Crítico Virtual.

FIGURA 7 - Ilustração da Tabela de Controle.



O parâmetro IAAV identifica uma Área de Armazenamento Virtual através do nome correspondente a um arquivo de dados para o Sistema Operacional e consequentemente segue as mesmas regras para nomeação de arquivos. O parâmetro EFA é um ponteiro que localiza a próxima posição na Área de Armazenamento Virtual na qual pode ser armazenada um determinada instância. O parâmetro EFLEV indica a posição final da Lista de Espaços Vazios que se encontra armazenada em um arquivo de dados. QTEV indica a quantidade de espaços vazios correspondente àquela Área de Armazenamento Virtual e NCV indica o nível crítico de espaços vazios usado para ativação automática da Compactação Virtual o qual é dado em forma de percentual e varia conforme as características da Área de Armazenamento Virtual. As 12 ocorrências da Tabela de Controle das Áreas de Armazenamento Virtual correspondem respectivamente aos 12 tipos de Objetos de Gerenciamento existente no ambiente. A indexação desta tabela é feita a partir do parâmetro IDO (Identificador do Objeto) que sempre é fornecido pelas mensagens enviadas para o Objeto Memória. A figura 7 ilustra a Tabela de Controle e os parâmetros que a compõe.

A Tabela de Controle por conter informações fundamentais para o funcionamento do Objeto Memória, é armazenada em um arquivo sempre que se finaliza o processamento do Ambiente. Da

mesma forma, sempre que é feita a inicialização do sistema, sua única instância é transferida deste arquivo para a posição inicial da Heap. O arquivo utilizado para armazenar a instância da Tabela de Controle possui a seguinte identificação TABELA.CON sendo protegido de qualquer ação inconseqüente do usuário.

OPERAÇÕES INTERNAS

A Tabela de Controle conforme foi descrito no início desta seção não foi implementada como sendo um objeto com suas respectivas operações internas. Sua estrutura faz parte da composição do Objeto Memória e consequentemente as operações internas que manipulam sua estrutura são as operações internas do Objeto Memória. Sendo assim, serão relacionadas a seguir algumas operações internas do Objeto Memória.

CONSTRUTOR - Este método efetua a inicialização do Objeto Memória através da criação do primeiro Bloco de Alocação, carga para região inicial de Memória Real da instância da Tabela de Controle armazenada em arquivo, criação das instâncias do objeto Área de Armazenamento Virtual, inicializa a Tabela de Processos, a Tabela Virtual de Processos e a Tabela de Espaços Desalocados, insere os Processos de Alocação correspondentes às Tabelas citadas anteriormente.
VERIFICA-OVR - Este método verifica a ocorrência de overflow da Memória Real.
VERIFICA-OVV - Este método verifica a ocorrência de overflow da Memória Persistente.
BLOQUEIA - Este método permite a manipulação do Indicador de Bloqueio para que um determinado Processo de Alocação seja bloqueado.
DESBLOQUEIA - Este método permite a manipulação do Indicador de Bloqueio para que um determinado Processo de Alocação seja desbloqueado.
SWAP-IN - Este método executa os procedimentos de Swap-in que corresponde a realocação de um Processo de Alocação transferido da Memória Real por ocasião de um overflow da memória.
SWAP-OUT - Este método executa os procedimentos necessários à transferência de um Processo de Alocação por ocasião de um overflow na Memória Real.
SALVA - Este método efetua a transferência de uma instância de um determinado objeto de gerenciamento para sua respectiva Área de Armazenamento Virtual.
SOBRESALVA - Este método efetua a transferência de frações de instâncias de Objetos de Gerenciamento da Memória Real para a posição correspondente na respectiva Área de Armazenamento Virtual.
SALVA-PARCIAL - Este método permite a inclusão de frações de instâncias no final daquelas armazenadas numa determinada Área de Armazenamento Virtual. Com este método é possível acrescentar partes a uma instância sem necessariamente ter que manipulá-la em Memória Real.
DESALOCA-VIRTUAL - Este método possibilita a desalocação de um Bloco Virtual.
ALOCAÇÃO - Este é o principal método do Objeto Memória. Sua finalidade é permitir a alocação de Memória Real a partir da criação dos Processos de Alocação e manipulação das principais situações envolvidas tais como overflow da memória.
DESALOCAÇÃO - Este método efetua a desalocação de uma determinada região de memória alocada a partir da remoção do Processo de Alocação correspondente e envio de informações ao objeto Tabela de Espaços Desalocados a cerca do novo espaço disponível para realocação.
CARGA - Este método efetua a transferência de uma determinada instância localizada na Memória Persistente para a Memória Real previamente alocada.
OBTER-EALOC - Este método obtém a localização na Memória Real de um determinado Processo de Alocação efetuando inclusive o disparo do processo de Swap-in caso o objeto não esteja fisicamente alocado na Memória Real.
OBTER-TALOC - Este método obtém o tamanho da região alocada para um determinado Processo de Alocação.
DUMP - Este método exibe toda a região ocupada por um determinado Processo de Alocação no formato hexadecimal.

3.2.1.2.2 **TABELA DE PROCESSOS**

Este objeto componente do Objeto Memória tem por objetivo básico controlar e manipular os

Processos de Alocação (PA). Para isto, sua estrutura foi idealizada na forma de um array onde cada elemento é um Processo de Alocação. O tamanho deste array é fixo e igual a 100 elementos, ou 100 Processos de Alocação. A medida que os Objetos de Gerenciamento solicitam alocações ao Objeto Memória, Processos de Alocação vão sendo gerados e armazenados na Tabela de Processos. Quando ocorre o preenchimento total da Tabela de Processos existe um mecanismo de paginação que permite o armazenamento da Tabela em um arquivo, e uma nova página é inicializada na Memória Real. Da mesma forma, quando é necessário o acesso a um determinado Processo de Alocação que não se encontra na página que está na Memória Real, ocorre a permuta de páginas, ou seja, a pagina que estava na Memória Real é transferida para o arquivo e a página que estava no arquivo é transferida para a Memória Real. O arquivo no qual são armazenadas as páginas da Tabela de Processos, considerado temporário (sua utilização somente ocorre durante o processamento do ambiente sendo depois removido), possui a seguinte identificação: TABPROC.TMP. As páginas são armazenadas em posições fixas determinadas a partir do identificador da página PG da seguinte forma:

EP = PG * Tpag, onde:

EP é o Endereço Relativo da Página;
Tpag é o Tamanho da Página da Tabela de Processos.

O tamanho da página é constante e igual a cem vezes o tamanho de memória ocupada por um Processo de Alocação.

Esta filosofia foi idealizada visando a otimização no uso de memória propiciando assim uma economia, pois a Tabela de Processos poderá crescer a medida que as alocações do ambiente forem sendo feitas, e não haverá em conseqüência disso uma alocação de memória adicional.

ESTRUTURA INTERNA

A estrutura interna do objeto Tabela de Processos (TP) pode ser compreendida através da tupla:

< PAG, TOPO, TPROC>, onde:

PAG é o Controlador do número de páginas;
TOPO indica a próxima ocorrência livre na Tabela de Processos;
TPROC é o array com 100 ocorrências.

O parâmetro PAG conforme foi descrito no início permite o controle das páginas da Tabela de Processos criadas durante o processamento. O parâmetro TOPO é usado para informar a posição livre na Tabela de Processos para a inserção do Processo de Alocação. TPROC é o array de 100 ocorrências onde cada elemento é um Processo de Alocação cuja estrutura descrita na seção 3.2.1.1.2 é a tupla:

PA = < IDP, IDO, ERV, EB, T, ISO, IB> , onde:

IDP é o identificador do Processo de Alocação;
IDO é o identificador do objeto de gerenciamento;
ERV é o endereço relativo virtual onde se encontra armazenada a instância em Memória Persistente;
EB é o endereço base de localização na Memória Real;
T é o tamanho da Memória Real alocada;
ISO é o indicador de estado de swap;
IB é o indicador de bloqueio da instância.

A figura 8 apresenta uma ilustração do objeto Tabela de Processos.

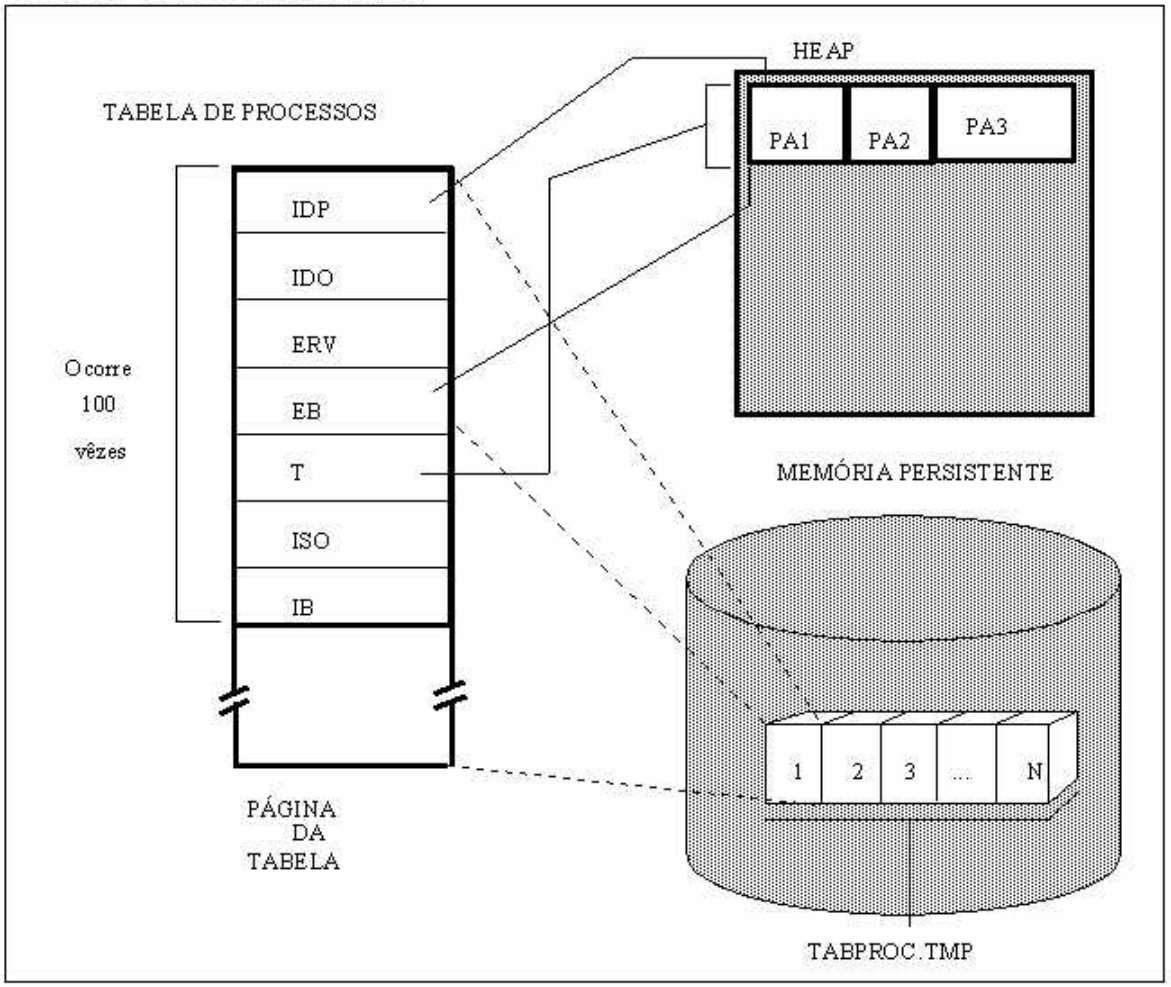
OPERAÇÕES INTERNAS

O Objeto Tabela de Processos é manipulado pelas seguintes operações internas:

CONSTRUTOR - Este método efetua a inicialização da instância deste objeto, bem como a criação do arquivo de dados onde serão armazenadas as páginas desta Tabela;
SALVAR - Este método efetua a transferência de uma determinada página da Tabela de Processos da Memória Real para o arquivo de dados correspondente;
CARREGAR - Este método ao contrário do anterior transfere a página do arquivo onde estava armazenada para a Memória Real;
ZERAR - Este método inicializa todos os elementos da tabela atribuído o valor zeros aos parâmetros;
INSERIR - Este método efetua a inserção de um elemento na Tabela de Processos na posição indicada por TOPO;
REMOVER - Este método efetua a remoção de um determinado elemento da Tabela através da cópia do último elemento inserido para a localização deste elemento;
OBTER-EREAL - Este método obtém o valor correspondente à posição na Tabela de um Processo de Alocação a partir dos parâmetros IDO e ERV fornecendo também o status deste elemento;
PESQUISAR-IDP - Este método procura na Tabela de Processos a localização ou posição de um elemento correspondente ao IDP fornecido como parâmetro;
ALTERAR-ISO - Este método possibilita a alteração do Indicador de Swap-out de um determinado Processo de Alocação;
ALTERAR-IB - Este método efetua a alteração do parâmetro Indicador de Bloqueio;
ALTERAR-IDO - Este método permite a alteração do parâmetro Identificador do Objeto de um determinado Processo de Alocação;
ALTERAR-ERV - Este método efetua a alteração do parâmetro Endereço Relativo Virtual de um Processo de Alocação;
ALTERAR-AMR - Este método possibilita a alteração do Endereço Base de localização da instância na Memória Real;
ALTERAR-TMO - Este método efetua a alteração do parâmetro Tamanho da Memória Real ocupada por um Processo de Alocação;
OBTER-IDP - Este método obtém o valor correspondente ao Identificador do Processo de Alocação de um determinado elemento da Tabela;
OBTER-IDO - Este método obtém o valor correspondente ao Identificador do Objeto;
OBTER-ERV - Este método obtém o valor correspondente ao Endereço Relativo Virtual do Processo de Alocação de um determinado elemento da Tabela;
OBTER-AMR - Este método informa o valor correspondente ao Endereço Base de localização da instância na Memória Real;
OBTER-TMO - Este método obtém o Tamanho da Memória Real ocupada por um Processo de Alocação;
OBTER-ISO - Este método obtém o valor correspondente ao Indicador de Swap-out do Processo de Alocação de um determinado elemento da Tabela;
OBTER-IB - Este método obtém o valor correspondente ao Indicador de Bloqueio;
ENCONTRAR - Este método procura um intervalo de Memória Real cujo tamanho seja igual ou superior ao informado via parâmetro e que o(s) Processo(s) de Alocação que ocupa(m) esta região não esteja(m) bloqueado(s);

OBTER-TOPO - Este método obtém o valor correspondente ao TOPO da Tabela de Processos;
OBTER-PAGINA - Este método obtém o valor correspondente à página corrente da Tabela de Processos;

FIGURA 8 - TABELA DE PROCESSOS.



3.2.1.2.3 **TABELA VIRTUAL DE PROCESSOS**

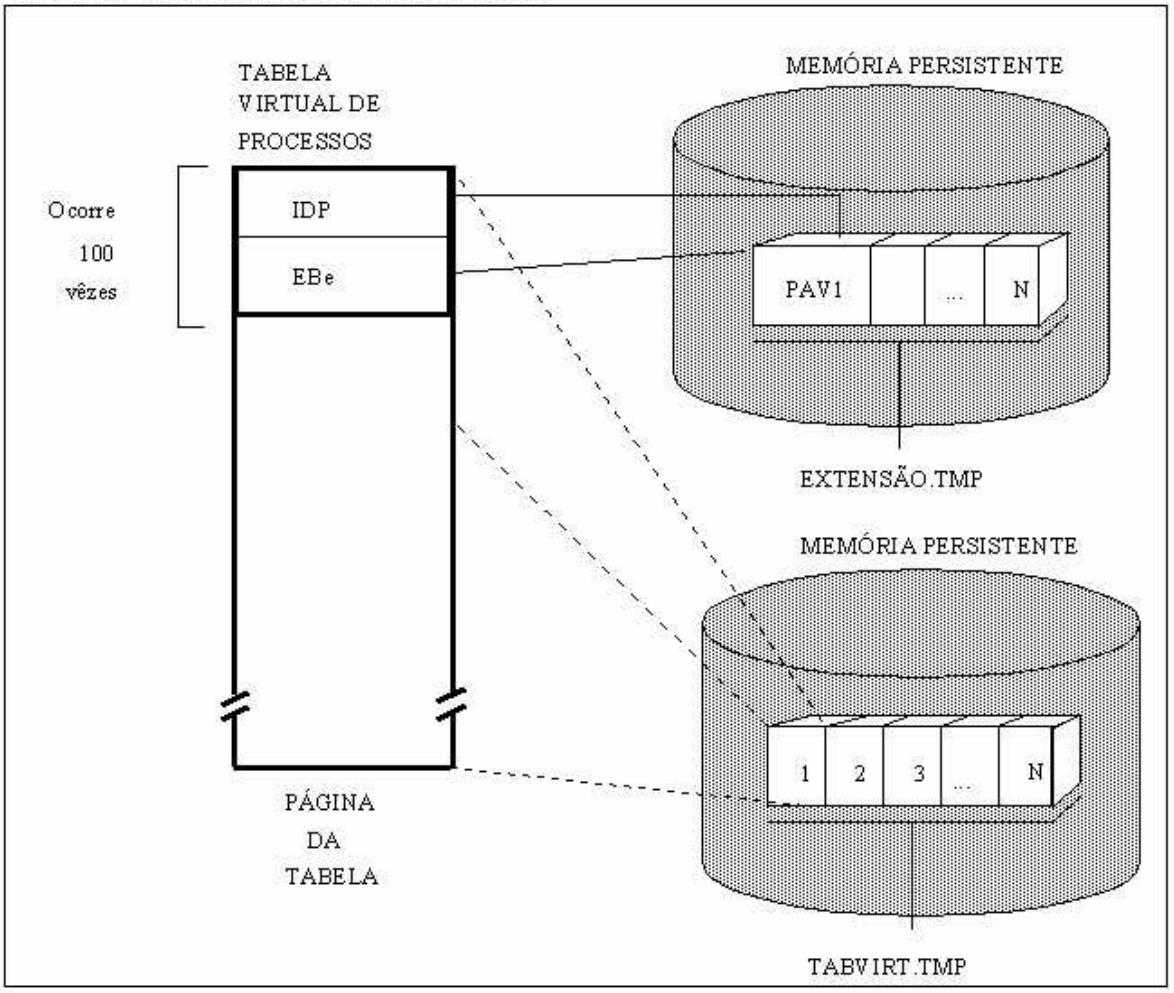
A Tabela Virtual de Processos é o objeto componente do Objeto Memória responsável pela manipulação dos Processos de Alocação Virtuais (PAV) que foi descrito na seção 3.2.1.1.3. Seu objetivo é controlar os Processos de Alocação que foram transferidos e removidos da Memória Real pelo método de Swap-out, mantendo um conjunto de informações acerca de cada Processo de Alocação necessárias à sua recuperação e realocação.

Da mesma maneira que o objeto Tabela de Processos, foi implementada a mesma filosofia de paginação para o objeto Tabela Virtual de Processos. O mecanismo funciona então, de forma similar ao descrito na seção 3.2.1.2.2. O arquivo onde as páginas são armazenadas possui a seguinte identificação: TABVIRT.TMP. O endereço relativo virtual de cada página é dado pela expressão:

ERVP = PAG * TPAG onde:

PAG indica o número da página corrente e pode variar de 0 até o limite de armazenamento da Memória Persistente;
TPAG é o tamanho de uma página da Tabela Virtual de Processos dado a partir do número de ocorrências multiplicado pela tamanho (em bytes) do Processo de Alocação Virtual.

FIGURA 9 - TABELA VIRTUAL DE PROCESSOS.



ESTRUTURA INTERNA

A Tabela Virtual de Processos (TVP) consiste de um conjunto de informações representado pela tupla:

TVP = < TV_{PAG}, TV_{TOPO}, TVPROC> , onde:

TV_{PAG} é o indicador do número de páginas usadas;
TV_{TOPO} indica a próxima posição livre na Tabela Virtual de Processos;
TVPROC é a Tabela Virtual de Processos.

O parâmetro TV_{PAG} permite o controle das páginas da Tabela Virtual de Processos criada durante o processamento. O parâmetro TV_{TOPO} é usado para informar a posição livre na Tabela Virtual de Processos para a inserção do Processo de Alocação Virtual. TVPROC é o array de 100 ocorrências onde cada elemento é um Processo de Alocação Virtual (PAV) cuja estrutura descrita na seção 3.2.1.1.2 é a tupla:

PAV = < IDP, EBe> , onde:

IDP é o identificador do Processo de Alocação Virtual que é o mesmo do PA;
EBe é o Endereço Base da instância na Extensão de Memória Real.

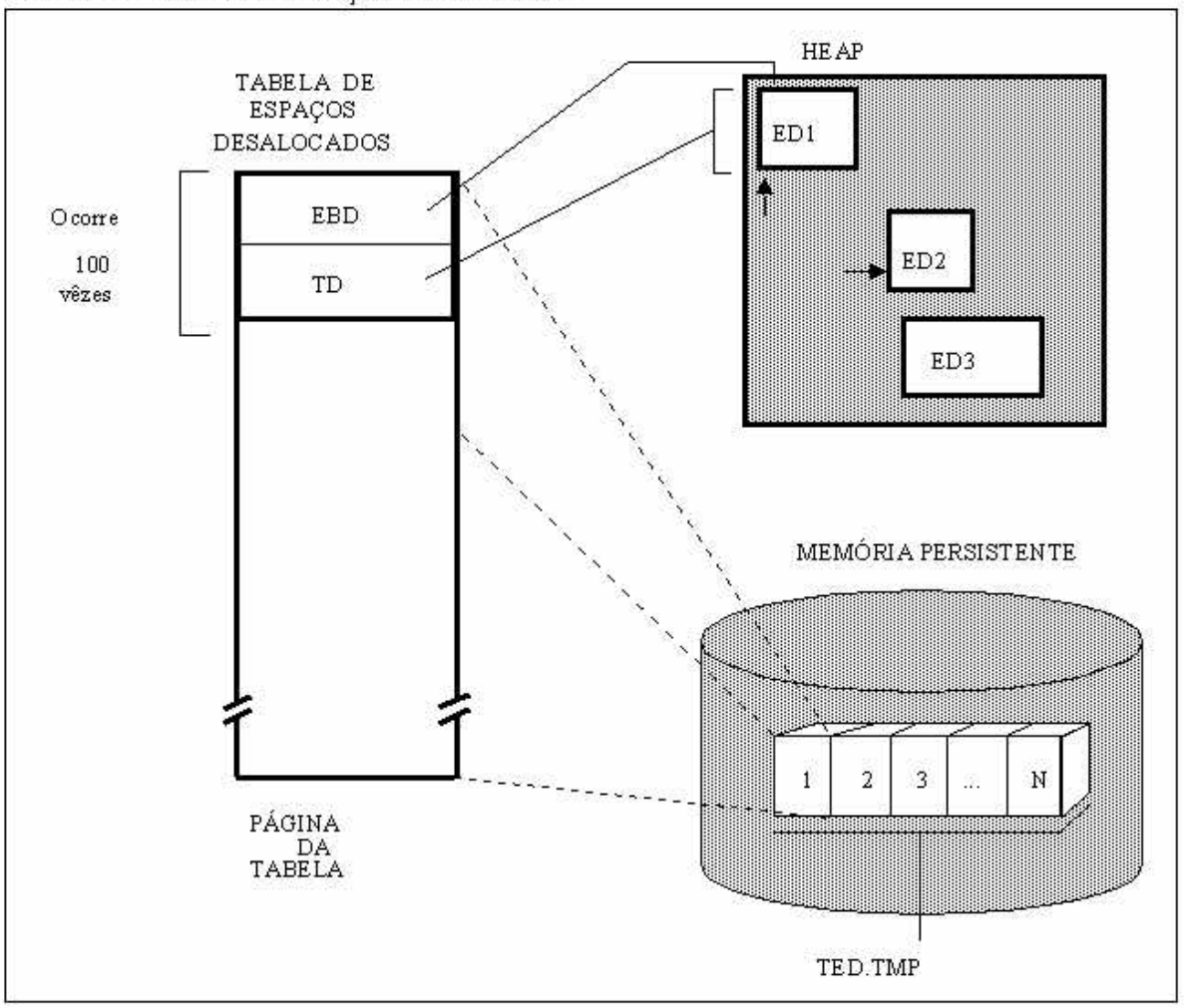
A figura 9 apresenta a estrutura descrita acima ilustrando também a utilização do mecanismo de paginação.

OPERAÇÕES INTERNAS

O objeto Tabela Virtual de Processos é manipulado por um conjunto de operações internas (métodos) os quais possibilitam o processamento dos Processos de Alocação Virtual. Estes métodos são os seguintes:

CONSTRUTOR - Este método permite a inicialização da instância deste objeto, e a criação do arquivo de dados onde serão armazenadas as páginas desta Tabela;
SALVAR - Este método efetua a transferência de uma determinada página da Tabela Virtual de Processos da Memória Real para o arquivo de dados correspondente;
CARREGAR - Este método ao contrário do anterior transfere a página do arquivo onde estava armazenada para a Memória Real;
ZERAR - Este método inicializa todos os elementos da tabela atribuído o valor zeros aos parâmetros;
INSERIR - Este método efetua a inserção de um elemento na Tabela na posição indicada por TV_{TOPO} ;
REMOVER - Este método efetua a remoção de um determinado elemento da Tabela através da cópia do último elemento inserido para a localização deste elemento;
OBTER-ERP - Este método obtém o valor correspondente ao Endereço Relativo do Processo armazenado na instância do objeto Extensão;
PESQUISAR-IDP - Este método procura um elemento na Tabela cujo IDP corresponda ao informado através de parâmetro.

FIGURA 10 - TABELA DE ESPAÇOS DESALOCADOS.



3.2.1.2.4 **TABELA DE ESPAÇOS DESALOCADOS**

Este objeto foi criado para administrar os espaços ou porções de memória pertencentes à instâncias dos Objetos de Gerenciamento que foram desalocadas durante o processamento no

Ambiente. A administração destes espaços visa sua reutilização possibilitando o uso mais eficiente e produtivo da Memória Real.

O objeto Tabela de Espaços Desalocados foi implementado com a filosofia de tabela ou array de tamanho fixo da mesma forma que os outros objetos anteriormente descritos. Esta forma de implementação justifica-se pela necessidade de se manter constante o uso da Memória Real para estes objetos, já que seria inconveniente o crescimento de suas instâncias as quais tem por objetivo o controle da Memória Real, e que consumiriam recursos de memória destinado às instâncias dos Objetos de Gerenciamento. Sendo assim, para permitir o crescimento da instância dos objetos implementados com a filosofia de tabela foi criado o mecanismo de paginação conforme descrito na seção 3.2.1.2.2.

O objeto Tabela de Espaços Desalocados é usado no momento em que ocorre uma desalocação de Memória Real de uma determinada instância e quando o sistema necessita efetuar uma nova alocação. No primeiro momento, é enviada uma mensagem à Tabela de Espaços Desalocados informando o Endereço Base Desalocado e o Tamanho deste espaço. O objeto verifica então se aquela região informada é imediatamente contínua a uma outra já desalocada. Se for contínua ocorre então apenas uma atualização do tamanho da região anteriormente existente. Caso contrário, aquele espaço é inserido na tabela. No segundo momento, uma mensagem é enviada para que o objeto Tabela de Espaços Desalocados efetue uma pesquisa em sua instância buscando uma região de memória cujo tamanho, informado através de parâmetro, seja compatível com a quantidade que se deseja alocar.

ESTRUTURA INTERNA

O objeto Tabela de Espaços Desalocados (TED) possui um conjunto de informações necessárias a sua manipulação que pode ser compreendido a partir da tupla:

$< TED_{PAG} , TED_{TOPO} , TED_{TAB} >$ onde:

TED_{PAG} controla o número de páginas existentes da tabela;
TED_{TOPO} indica a próxima posição livre na tabela;
TED_{TAB} é o array que constitui a tabela propriamente dita.

Os parâmetros TED_{PAG} e TED_{TOPO} são usados para a manipulação da instância, enquanto que TED_{TAB} é usado para armazenar as informações acerca dos espaços desalocados que podem ser entendidas através da tupla:

< EBD, TD> onde:

EBD é o Endereço Base da Região Desalocada, e
TD é o Tamanho (em bytes) desta região.

A figura 10 ilustra a estrutura do objeto Tabela de Espaços Desalocados mostrando ainda o mecanismo de paginação deste objeto.

OPERAÇÕES INTERNAS

As operações internas do objeto Tabela de Espaços Desalocados permite a manipulação de sua instância permitindo assim o gerenciamento dos espaços desalocados de Memória Real. Este conjunto de operações internas compreende os seguintes métodos:

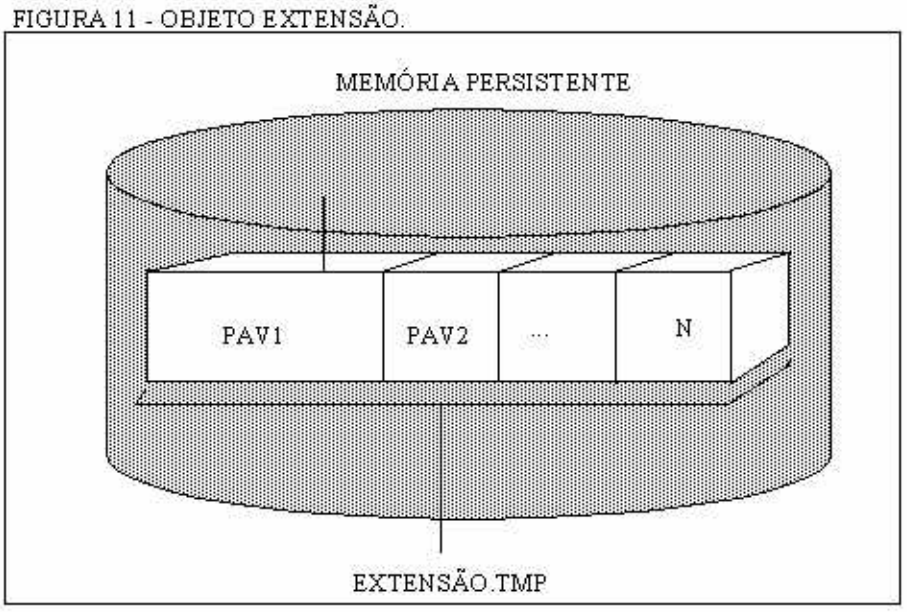
CONSTRUTOR - Este método possibilita a inicialização da instância deste objeto, bem como a criação do arquivo de dados onde serão armazenadas as páginas desta Tabela;
SALVAR - Este método efetua a transferência de uma determinada página da Tabela de Espaços Desalocados da Memória Real para o arquivo de dados correspondente;
CARREGAR - Este método ao contrário do anterior transfere a página do arquivo onde estava

armazenada para a Memória Real;
ZERAR - Este método inicializa todos os elementos da tabela atribuído o valor zeros aos parâmetros;
INSERIR - Este método efetua a inserção de um elemento na Tabela na posição indicada por TED_{TOPO} ;
REMOVER - Este método efetua a remoção de um determinado elemento da Tabela através da cópia do último elemento inserido para a localização deste elemento;
ALTERAR-EBD - Este método possibilita a alteração do Endereço Base de localização do espaço desalocado;
ALTERAR-TD - Este método efetua a alteração do parâmetro Tamanho do Espaço Desalocado;
OBTER-EBD - Este método obtém o valor correspondente ao Endereço Base do Espaço Desalocado de um determinado elemento da Tabela;
OBTER-TD - Este método obtém o valor correspondente ao Tamanho do Espaço Desalocado;
PESQUISA-ÚLTIMO - Este método procura um elemento da Tabela cuja posição final corresponda à posição inicial da área a ser desalocada verificando assim se existe a continuidade de memória;
PESQUISAR - Este método procura um elemento na Tabela cujo tamanho seja igual ou superior ao desejado.

3.2.1.2.5 **EXTENSÃO**

Este objeto componente do Objeto Memória tem por objetivo o armazenamento temporário de instâncias que foram transferidas e removidas durante o processo de Swap-out conforme descrito na seção 3.2.1.2. Este armazenamento é feito em um arquivo ou Área de Armazenamento Virtual específica para esta finalidade, conforme poderá ser visto na figura 11. O arquivo usado para esta finalidade possui a seguinte identificação: EXTENSAO.TMP.

FIGURA 11 - OBJETO EXTENSÃO.



ESTRUTURA INTERNA

O objeto Extensão possui apenas uma única informação que constitui sua composição interna. Esta informação denominada de APE é um apontador para o arquivo onde são armazenadas todas as instâncias transferidas.

OPERAÇÕES INTERNAS

Este objeto possui as seguintes operações internas:

SALVA - Este método permite a transferência da instância da Memória Real para um determinado endereço relativo virtual disponível no arquivo Extensão;
CARGA - Este método efetua a transferência da instância do arquivo Extensão para a Memória Real;
REMOVER - Efetua a remoção do arquivo Extensão da Memória Persistente.

A primeira execução do método SALVA efetua a criação da instância do Objeto Extensão ou seja, a criação do arquivo na Memória Persistente.

3.2.1.2.6 **ÁREA DE ARMAZENAMENTO VIRTUAL**

O objeto Área de Armazenamento Virtual tem um importante função no gerenciamento de memória secundária do sistema. Este objeto é usado para o armazenamento e recuperação das instâncias dos Objetos de Gerenciamento na Memória Persistente. O controle do armazenamento é compartilhado com o objeto Tabela de Controle o qual administra as informações das diversas instâncias do objeto Área de Armazenamento Virtual. Uma Área de Armazenamento Virtual consiste de um arquivo de dados cuja função é armazenar instâncias de um determinado tipo de objeto.

O número de instâncias do objeto Área de Armazenamento Virtual corresponde ao número de Objetos de Gerenciamento criados para o GOLGO (ver seção 3.2.2.2). Sendo assim, existe na instância do objeto Tabela de Controle um conjunto de informações referentes à cada uma das 12 instâncias do objeto Área de Armazenamento Virtual. Entre estas informações temos: o nome externo da Área de Armazenamento Virtual e um ponteiro indicando o próximo Endereço Relativo Virtual disponível para uma alocação. Uma alocação virtual corresponde a criação de um Bloco Virtual conforme especificado na seção 3.2.1.1.2. O nome que identifica uma Área de Armazenamento Virtual é composto da seguinte maneira:

XXXXXXXX . AAV onde:

XXXXXXX é o nome principal do arquivo definido de acordo com o nome do objeto de gerenciamento;

AAV é o nome secundário que no caso é sempre constante.

Quando algum objeto de gerenciamento solicita que uma determinada instância seja armazenada, ele o faz através do envio de uma mensagem ao Objeto Memória. Por conseguinte, o Objeto Memória envia uma mensagem ao objeto Área de Armazenamento Virtual para efetuar a transferência física entre a Memória Real e a Memória Persistente. Da mesma forma acontece quando um determinado objeto de gerenciamento deseja recuperar uma instância armazenada em sua Área de Armazenamento Virtual somente sendo alterado o sentido de transferência, ou seja, da Memória Persistente para a Real. A figura 12 apresenta uma Área de Armazenamento Virtual e sua composição.

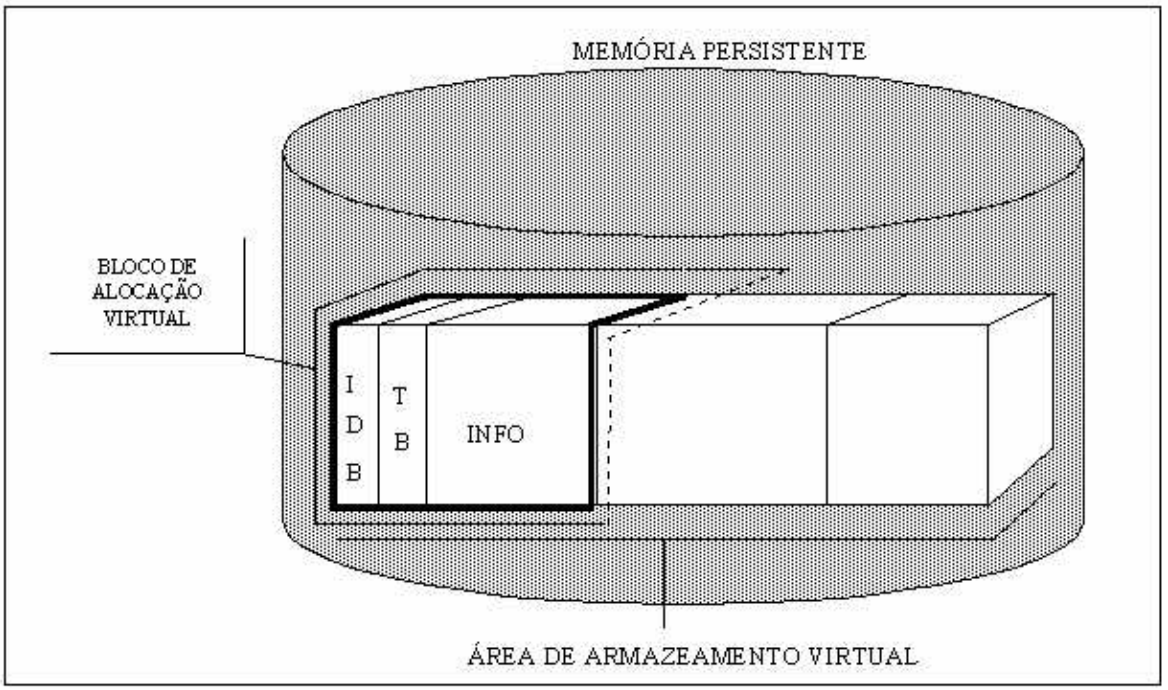
ESTRUTURA INTERNA

O objeto Área de Armazenamento Virtual da mesma forma que o objeto Extensão possui apenas uma única informação que constitui sua composição interna. Esta informação denominada de APAA é um apontador para o arquivo de dados correspondente a instância da Área de Armazenamento Virtual onde são armazenadas todas as instâncias transferidas.

OPERAÇÕES INTERNAS

O objeto Área de Armazenamento Virtual possui um conjunto de operações internas utilizadas para a manipulação de suas instâncias. Este conjunto é composto pelos seguintes métodos:

FIGURA 12 - OBJETO ÁREA DE ARMAZENAMENTO VIRTUAL



INSERIR-BLOCO - Este método tem por objetivo efetuar a alocação virtual, que implica na criação do Bloco Virtual e a transferência física da Memória Real para a Área de Armazenamento Virtual informada através do IDO (identificador do objeto de gerenciamento);
SALVAR-BLOCO - Este método efetua a transferência física para um Bloco Virtual já existente;
OBTER-DIM - Este método acessa um determinado Bloco Virtual retornando com o valor correspondente ao tamanho deste Bloco;
ALTERAR-DIM - Este método permite a alteração do tamanho (TB) de um dado Bloco Virtual;
REMOVER - Este método efetua a remoção de uma determinada instância do objeto Área de Armazenamento Virtual, ou seja, remove o arquivo correspondente;
CARGA - Este método tem por objetivo efetuar a transferência física de uma determinada instância armazenada num dado Bloco Virtual para a Memória Real.

3.2.1.1.7 **LISTA DE ESPAÇOS VAZIOS**

As instâncias de objeto Área de Armazenamento Virtuais são usadas para o armazenamento de todas as instâncias dos respectivos Objetos de Gerenciamento do GOLGO. Entretanto, durante o processamento do Ambiente, podem surgir nestas Áreas Virtuais espaços provenientes da desalocação virtual de algumas instâncias. A Memória Persistente possui um objeto capaz de administrar estes vazios de memória existentes para cada instância de Área de Armazenamento Virtual. O objeto responsável então por esta função é a Lista de Espaços Vazios.

O objeto Lista de Espaços Vazios consiste de uma lista implementada em um arquivo de dados cuja identificação corresponde de uma certa forma a identificação de uma Área de Armazenamento Virtual. O nome que identifica uma Lista de Espaços Vazios é composto da seguinte maneira:

XXXXXXXX . LEV onde:

XXXXXXX é o nome principal do arquivo definido de acordo com o nome do objeto de

gerenciamento;

LEV é o nome secundário que no caso é sempre constante.

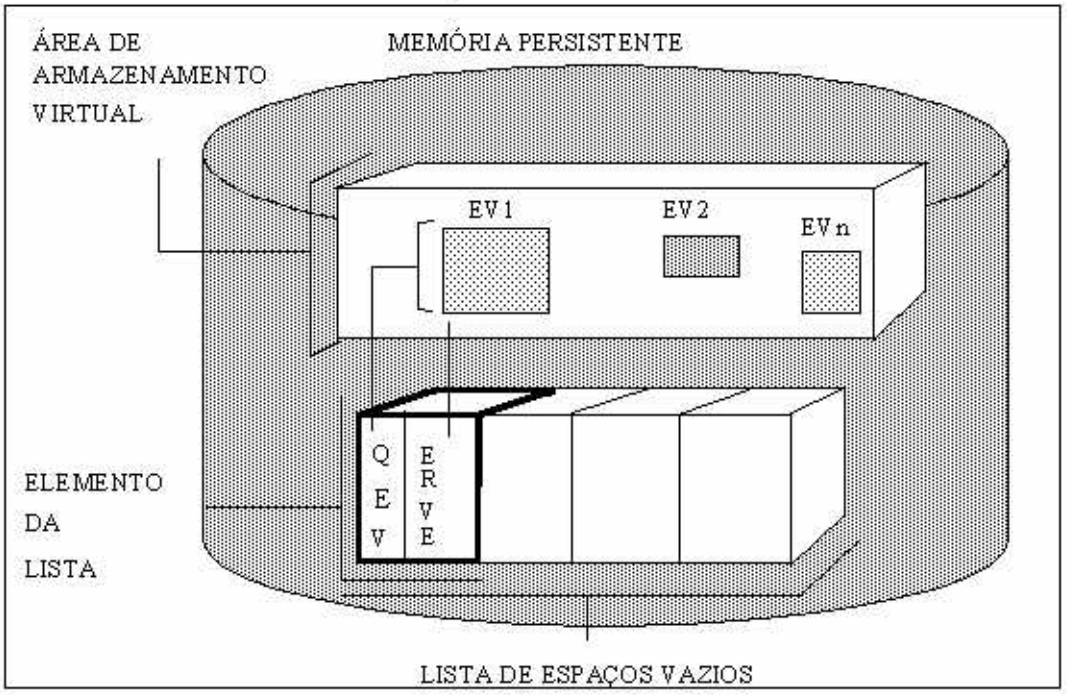
A Lista de Espaços Vazios é manipulada de forma parcial, ou seja, apenas um único elemento é trazido à Memória Real de cada vez para fornecer as informações acerca das regiões desalocadas na Área de Armazenamento Virtual correspondente. Esta forma de processamento faz com que haja uma economia de memória no processamento. A utilização da lista ocorre em dois momentos: o primeiro, quando da desalocação de um Bloco Virtual, e a segunda quando torna-se necessário uma alocação de um novo Bloco Virtual. No primeiro caso, o objeto Lista de Espaços Vazios verifica na sua instância se existe a continuidade entre o Bloco desalocado virtualmente com uma região já desalocada e vice versa. Este procedimento visa manter as regiões contínuas de Memória Persistente representadas por um único elemento. Se não existe a continuidade é feita a inserção do novo elemento à Lista. No segundo caso, ocorre uma pesquisa na Lista em busca de uma quantidade de Memória Persistente desalocada de tamanho igual ou superior a que se deseja alocar. Caso a quantidade seja superior, é feita a alocação para os primeiros bytes da instância e o restante é mantido na lista com a quantidade debitada daquela usada para a alocação. Se a quantidade for igual à requisitada, é feita a remoção daquele elemento da Lista.

O objeto Tabela de Controle mantém dois parâmetros que coordenam a manipulação de cada instância da Lista de Espaços Vazios, que são os seguintes:

EFLEV que é o Endereço Final da Lista de Espaços Vazios e
QTEV que é a Quantidade Total de Espaços Vazios.

Estes parâmetros são fornecidos ao objeto Lista de Espaços Vazios para que ele os atualize mantendo assim o correto funcionamento deste objeto. A figura 13 ilustra a composição e utilização do Objeto Lista de Espaços Vazios.

FIGURA 13 - OBJETO LISTA DE ESPAÇOS VAZIOS



ESTRUTURA INTERNA

O objeto Lista de Espaços Vazios (LEV) possui uma estrutura interna que poderá ser compreendida a partir da seguinte tupla:

< AEV, E> onde:

AEV é o apontador para o arquivo onde se encontra armazenada a lista;
E é o elemento constituinte da lista.

O parâmetro E que representa um elemento qualquer da Lista de Espaços Vazios é constituído dos seguintes parâmetros identificados na tupla:

< QEV, EREV>, onde:

QEV indica a quantidade de bytes que compõe o espaço vazio;
EREV é o Endereço Relativo Virtual onde inicia-se o espaço vazio.

A manipulação dos elementos E da Lista de Espaços Vazios é feita usando o mesmo conceito de endereçamento descrito na seção 3.2.1.1.1, ou seja, cada elemento se encontra num determinado endereço dado a partir da expressão:

$ER_E = (N - 1) * T_E$, onde:

ER_E é o endereço relativo virtual do elemento;
N é o identificador de seqüência do elemento;
T_E é o tamanho (em bytes) da memória ocupada pelo elemento.

Quando um determinado elemento é removido da instância da Lista de Espaço Vazio o endereço relativo deste elemento é imediatamente ocupado pelo último elemento da lista. Isto impede a perda de seqüência dos elementos que é característica deste objeto já que ele não mantém apontadores de um elemento para o outro.

OPERAÇÕES INTERNAS

O objeto Lista de Espaços Vazios dispõe de alguns métodos que são usados para manipulação de suas instâncias. Os métodos são os seguintes:

INSERIR - Este método permite a inserção de um elemento na Lista, bem como a criação do arquivo de dados quando o elemento a ser inserido é o primeiro da instância deste objeto;
REMOVER - Este método efetua a remoção lógica do elemento a partir da copia do último elemento para o Endereço Relativo onde se encontra este elemento;
PESQUISAR - Este método efetua uma busca na instância de uma Lista de Espaços Vazios a procura de um elemento que satisfaça a pesquisa, ou seja, cuja quantidade de espaços vazios seja igual ou superior àquela requisitada.

3.2.1.3 **DESCRIÇÃO DO FUNCIONAMENTO DO OBJETO MEMÓRIA**

A partir das especificações definidas na seção anterior torna-se mais compreensiva a composição do Objeto Memória revelando a modularidade nele existente. Por ser composto de partes independentes, o Objeto Memória precisa coordenar estas partes numa sincronia indispensável ao funcionamento do gerenciamento de memória do ambiente. Nesta seção será descrito o funcionamento de algumas ações realizadas pelo Objeto Memória consideradas como fundamentais.

As principais ações realizadas pelo Objeto Memória ocorre de duas formas:

- Pela solicitação externa, através do recebimento de uma mensagem;
- Por ocasião da ocorrência de situações extras que requerem procedimentos especiais.

As principais ações desempenhadas pelo Objeto Memória são:

- Inicialização da memória;
- Alocação;
- Desalocação;
- Carga de Instâncias;
- Salvamento (gravação) de Instâncias;
- Overflow.

3.2.1.3.1 **INICIALIZAÇÃO DA MEMÓRIA**

Esta ação é de fundamental importância para o funcionamento do Ambiente. Consiste basicamente de um conjunto de ações que visam a inicialização de todos os objetos que compõe o Objeto Memória bem como a alocação inicial do primeiro bloco de Memória Real. Este conjunto de ações é constituído dos seguintes procedimentos:

(1) - Obtenção do EIM ou Endereço Base Inicial da Área de Armazenamento Real (Heap);

(2) - Alocação do primeiro Bloco de Memória Real;
(3) - Carga da instância do objeto Tabela de Controle a partir do Endereço Base inicial da Heap. Caso não exista esta instância, são realizados os seguintes procedimentos:
(3.a) - Inicialização dos parâmetros básicos: Tamanho do Bloco de Alocação (TBA), Nível Crítico da Memória Real (NCR), e Tabela de Áreas de Armazenamento Virtual (TAAV);
(3.b) - Salvamento (gravação) da instância no respectivo arquivo de dados;
(4) - Atualização dos parâmetros da Tabela de Controle: EIM com o valor respectivo, NBA com o valor 1, PEL com o valor correspondente ao primeiro Endereço disponível após a Tabela de Controle e CP com o valor zero;
(5) - Criação da instância do objeto Tabela de Processos;
(6) - Inserção do Processo de Alocação correspondente à instância da Tabela de Controle;
(7) - Inserção do Processo de Alocação correspondente à alocação da instância da Tabela de Processos;
(8) - Criação da instância do objeto Tabela Virtual de Processos;
(9) - Inserção do Processo de Alocação correspondente à alocação da instância da Tabela Virtual de Processos;
(10) - Criação da instância do objeto Tabela de Espaços Desalocados;

(11) - Inserção do Processo de Alocação correspondente à alocação da instância da Tabela de Espaços Desalocados;

(12) Em cada um dos procedimentos (6), (7), (9) e (11) é incrementado o Contador de Processos (CP).

A partir da execução dos procedimentos acima descritos, o Objeto Memória estará apto a efetuar o gerenciamento de memória do ambiente.

3.2.1.3.2 **ALOCAÇÃO**

A alocação de memória tanto Real quanto Persistente é uma das mais importantes funções do Objeto Memória. Conforme a especificação existente na seção 3.2.1.1.2 uma alocação na Memória Real consiste na criação de um Processo de Alocação que tem por objetivo manter um conjunto de informações sobre desta alocação. Entretanto, na criação de um Processo de Alocação estão envolvidos alguns procedimentos que compõem a Alocação na Memória Real. Estes procedimentos são os seguintes:

(1) - Se o IDP fornecido através do envio da mensagem for igual a zero significa que é uma alocação inicial, caso contrário, corresponde a uma expansão de memória solicitada pelo objeto de gerenciamento.

(2) - Tanto no primeiro quanto no segundo caso, o Objeto Memória necessita efetuar uma nova alocação de memória. Uma mensagem é então enviada ao objeto Tabela de Espaços Desalocados solicitando uma região de tamanho informado via parâmetro do método.
(3) Caso exista esta região o objeto Tabela de Espaços Desalocados fornece o Endereço Base de localização ao Objeto Memória para a atualização das informações necessárias a inserção ou atualização do Processo de Alocação na instância do objeto Tabela de Processos.
(4) Caso não exista espaço desalocado suficiente para a reutilização, torna-se necessário a alocação de uma nova região de Memória Real. Neste caso, verifica-se a ocorrência de overflow do Bloco de Alocação, ou seja, a total utilização deste Bloco. Se for detectada esta situação, o Objeto Memória providencia um novo Bloco de Alocação para novas alocações.
(5) Se entretanto ocorrer uma situação de Overflow da Memória Real, durante a tentativa de alocação de um novo Bloco, o Objeto Memória executa os procedimentos necessários para contornar esta situação, que serão descritos na seção 3.2.1.3.6.
(6) Mesmo ocorrendo Overflow, o Objeto Memória sempre fornecerá a região de memória solicitada. No caso de ser uma alocação inicial é feita a inserção de um Processo de Alocação na Tabela de Processos, onde o novo IDP é obtido a partir de uma incrementação do Contador de Processos (CP). Já no caso de um expansão de memória, são atualizados os parâmetros: EB (Endereço Base), e T

(Tamanho alocado) com os novos valores obtidos.

Desta forma, todas as alocações de Memória Real são administradas e controladas permitindo assim o funcionamento do Objeto Memória.

3.2.1.3.3 **DESALOCAÇÃO**

A desalocação de Memória Real ocorre quando um determinado objeto de gerenciamento não necessita mais manipular sua instância em memória. Neste caso, o objeto de gerenciamento envia uma mensagem ao Objeto Memória informando esta situação e será então imediatamente efetuada a desalocação desta instância.

A desalocação de Memória Real consiste de um conjunto de procedimentos os quais serão descritos a seguir.

(1) Para a desalocação, o objeto de gerenciamento precisa informar o IDP correspondente à instância armazenada. A partir deste identificador, o Objeto Memória irá localizar na Tabela de Processos as informações necessárias à localização e manipulação deste Processo de Alocação.

(2) Após a localização do Processo de Alocação, é então enviada uma mensagem ao objeto Tabela de Espaços Desalocados fornecendo os dados necessários a inserção desta nova região desalocada.

(3) O objeto Tabela de Espaços Desalocados verifica se esta região a ser desalocada é contínua a uma outra existente na Tabela ou vice versa. Caso exista a continuidade, é feita apenas uma atualização dos parâmetros: EBD (Endereço Base do espaço Desalocado) e TD (Tamanho do espaço Desalocado).

(4) No caso de não existência da continuidade, é feita então a inserção de um novo elemento na Tabela contendo as informações correspondentes à nova região desalocada.

A partir dos procedimentos descritos acima, o Objeto Memória irá administrar através do objeto Tabela de Espaços Desalocados todas as porções de Memória Real já utilizadas para um posterior reaproveitamento.

3.2.1.3.4 **CARGA DE INSTÂNCIAS**

A carga de instâncias é a operação responsável pela transferência de instâncias armazenadas em Áreas de Armazenamento Virtual para regiões previamente alocadas na Memória Real. O Objeto Memória executa alguns procedimentos necessários a esta transferência e utiliza para isto os objetos Tabela de Controle, Tabela de Processos e Área de Armazenamento Virtual. Os procedimentos para a carga de uma determinada instância são os seguintes:

(1) Ao ser enviada a mensagem de carga ao Objeto Memória, são enviados também alguns parâmetros tais como: IDP (Identificador do Processo de Alocação), IDO (Identificador do Objeto de Gerenciamento), ERV (Endereço Relativo Virtual da Instância), POS (Posição Relativa do Segmento de Instância), TAM (Tamanho do Segmento da Instância).

(2) O método CARGA através do IDP envia uma mensagem ao objeto Tabela de Processos para obter o Endereço Base (EB) da Memória Real para onde será transferida a instância.

(3) É enviada também uma mensagem ao objeto Tabela de Controle para obtenção do Identificador da Área de Armazenamento Virtual (nome do arquivo) através do parâmetro IDO.

(4) Após estes procedimentos iniciais, a Área de Armazenamento Virtual é então aberta, sendo realizada o primeiro acesso a instância, localizada pelo ERV, visando verificar se o ERV corresponde realmente ao início do Bloco Virtual. Caso não corresponda, o método é finalizado sendo atribuída à variável global de erro do sistema o código correspondente ao erro de acesso.

(5) Caso seja localizado o Bloco Virtual, é realizada então a transferência a partir da posição POS, sendo transferidos TAM bytes. Quando se deseja transferir toda a instância basta apenas informar o parâmetro POS com o valor zero.

(6) Após a execução da transferência o método envia mensagens para o objeto Tabela de Processos para alteração dos valores correspondentes aos parâmetros da Tabela: IDO e ERV. Este procedimento final garantirá o controle acerca das instâncias em processamento. Este controle permitirá ao Objeto Memória localizar com exatidão o Endereço Base de Memória Real aonde foi armazenada a instância transferida da Memória Persistente.

Desta forma, qualquer instância armazenada na Memória Persistente poderá ser recuperada para processamento na Memória Real, bem como algumas instâncias poderão localizar facilmente na Memória Real o endereço de localização de uma determinada instância que foi transferida também para processamento.

3.2.1.3.5 **SALVAMENTO (GRAVAÇÃO) DE INSTÂNCIAS**

O método de Salvamento visa preservar determinadas instâncias nas Áreas de Armazenamento Virtual para um posterior processamento. Consiste basicamente em transferir um conjunto de bytes localizados em um Endereço Base (EB) da Memória Real, para um Endereço Relativo Virtual (ERV) da Memória Persistente. O funcionamento destes métodos baseia-se na execução dos seguintes procedimentos:

(1) Ao ser enviada a mensagem SALVA, são enviados algumas informações necessárias a execução do método, tais como: IDP, IDO e IG (Indicador de gravação).

(2) Usando o IDP este método envia uma mensagem ao objeto Tabela de Processos para obter as informações referentes a instância armazenada na Memória Real, tais como: EB (Endereço Base) e T (Tamanho).

(3) Usando o IDO o método envia uma mensagem ao objeto Tabela de Controle para obtenção das informações sobre a Área de Armazenamento Virtual a ser utilizada, tais como: IAAV (Identificador da Área de Armazenamento Virtual), EFA (Endereço Final da Área de Armazenamento Virtual), EFLEV (Endereço Final da Lista de Espaços Vazios) e QTEV (Quantidade total de Espaços Vazios).

(4) O parâmetro IG é utilizado para informar se a instância já existe ou não. Esta informação permitirá que uma mesma instância não seja armazenada em duplicidade quando tratar-se apenas de uma atualização.

(5) Se uma determinada instância já se encontrava anteriormente armazenada em sua respectiva Área de Armazenamento Virtual e sendo ***T*** o Tamanho da instância armazenada em Memória Real e ***To*** o Tamanho original armazenado na Área de Armazenamento Virtual, este método prevê os seguintes casos:

(A) T < To ,
ou seja, a instância diminuiu de tamanho após o último processamento;
(B) T = To ,
ou seja, a instância permanece com o mesmo tamanho original;
(C) T > To ,
ou seja, a instância em processamento aumentou o seu tamanho original.

No caso (A), o método executará a transferência da instância para o ERV informado, sendo que a parte da região de Memória Persistente anteriormente ocupada é considerada "espaço vazio" sendo então enviada ao objeto Lista de Espaços Vazios. No caso (B), simplesmente o método efetuará a transferência para o mesmo ERV original. Já no caso (C), a instância aumentou de tamanho e pode

não ser possível armazená-la no mesmo local de origem. Existe apenas uma situação em que é possível que uma instância que tenha sido expandida seja armazenada no mesmo ERV original. É no caso desta instância está armazenada no final da Área de Armazenamento Virtual, ou seja, não existir nenhuma outra instância armazenada consecutivamente. Quando não acontece esta situação, o método irá providenciar uma nova região para esta instância através dos seguintes procedimentos:

(5.a) Primeiramente é efetuada a verificação na instância do objeto Lista de Espaços Vazios correspondente à Área de Armazenamento Virtual da existência de um Espaço Vazio cujo tamanho seja igual ou superior ao tamanho requerido. Caso seja encontrado, o elemento correspondente da LEV é removido e a região se tornará disponível para armazenamento. Caso contrário, o método SALVA irá efetuar o armazenamento no final da Área de Armazenamento Virtual.

(5.b) Após este procedimento, a antiga região ocupada é enviada à Lista de Espaços Vazios para inserção do elemento com as informações correspondentes.

(6) Quando a instância está sendo armazenada pela primeira vez o método executará os mesmos procedimentos descritos no item (5.a) acima.

(7) Após a transferência da instância para a Memória Persistente, o parâmetro ERV é fornecido ao objeto de gerenciamento que enviou a mensagem SALVA, para que o mesmo seja utilizado por este objeto.

Utilizando basicamente os mesmos procedimentos acima descritos, existem outros métodos utilizados pelo Objeto Memória para salvamento (gravação) de instâncias em Memória Persistente. São eles:

SOBRESALVA - Este método efetua a transferência de frações de instâncias de Objetos de Gerenciamento da Memória Real para a posição correspondente na respectiva Área de Armazenamento Virtual.

SALVA-PARCIAL - Este método permite a inclusão de frações de instâncias no final daquelas armazenadas numa determinada Área de Armazenamento Virtual. Com este método é possível acrescentar partes a uma instância sem necessariamente ter que manipulá-la totalmente em Memória Real.

3.2.1.3.6 **OVERFLOW**

O tratamento da situação de Overflow de Memória Real é realizada por alguns métodos de vários objetos envolvidos, tais como: Tabela de Controle, Tabela de Processo, Tabela Virtual de Processos, Tabela de Espaços Desalocados e Extensão.

Conforme o que está descrito na seção 3.2.1.1.3 item (c) a ocorrência de overflow decorre de um impasse gerado quando a Memória Real está totalmente preenchida e torna-se necessário alocar mais memória. Para solucionar este problema o Objeto Memória precisa desalocar temporariamente alguns Processos de Alocação visando com isso conseguir a porção de memória solicitada. Entretanto, estes Processos de Alocação temporariamente desalocados precisam ser transferidos para a Memória Persistente para que não se perca o conteúdo da informação, sendo mantido o controle sobre eles. A região de memória onde eles estavam alocados é considerada então um espaço desalocado e será fornecida para a alocação da nova instância. Este método que retira da Memória Real instâncias ainda em processamento para possibilitar novas alocações é denominado de SWAP-OUT. Com esta estratégia, o Objeto Memória sempre conseguirá efetuar novas alocações.

Quando o objeto de gerenciamento cuja instância foi transferida para a Memória Persistente precisa manipular novamente esta instância é necessário que a mesma seja novamente alocada. Este método de trazer de volta a instância é denominado SWAP-IN. A mecânica de funcionamento do tratamento de Overflow baseia-se então na execução dos métodos de SWAP-OUT e SWAP-IN, e serão descritos a seguir os procedimentos (métodos) envolvidos em cada um destes processos, bem

como os objetos que deles participam.

SWAP-OUT

Participam deste processo os objetos: Tabela de Controle (TC), Tabela de Processos (TP), Tabela Virtual de Processos (TVP), Tabela de Espaços Desalocados (TED) e Extensão (EXT). O funcionamento do método Swap-Out pode ser descrito pelos seguintes procedimentos:

(1) Ao ser detectada a situação de Overflow pelo método ALOCAÇÃO, ocorre o envio da mensagem SWAP-OUT que dará início a execução deste método. Inicialmente é enviado ao objeto Tabela de Processos a mensagem ENCONTRAR que tem por objetivo localizar um ou mais Processos de Alocação cuja região total ocupada corresponda a região requerida. A forma como isso é realizado corresponde a estratégia descrita 3.2.1.1.3 item (c) (Processo de Varredura).

(2) Ao detectado o Processo de Alocação a ser transferido, o método Swap-out envia a mensagem SALVA ao objeto Extensão que executa a transferência da instância identificada pelas informações contidas no Processo de Alocação, para o arquivo de dados controlado por este objeto.

(3) Em seguida, é enviado a mensagem INSERIR ao objeto Tabela Virtual de Processos cuja finalidade é inserir as informações referentes ao Processo de Alocação transferido.

(4) Após a execução deste método, o Objeto Memória envia a mensagem ALTERAR-ISO ao objeto Tabela de Processos que efetua a alteração do parâmetro ISO (Identificador de Swap-out) que irá registrar no Processo de Alocação a ocorrência de Swap-out.

(5) Finalmente é enviado ao objeto Tabela de Espaços Desalocados a mensagem INSERIR cuja finalidade é registrar a região de Memória Real desalocada pelo processo de Swap-out.

SWAP-IN

Os seguintes objetos participam do processo de Swap-in: Tabela de Controle , Tabela de Processos, Tabela Virtual de Processos, Tabela de Espaços Desalocados e Extensão. O funcionamento do método Swap-in pode ser descrito através dos seguintes procedimentos:

(1) O método Swap-in é executado quando um objeto de gerenciamento solicita o acesso ao respectivo Processo de Alocação identificado pelo IDP, através dos métodos: OBTER-EALOC ou OBTER-TALOC, e verifica-se que a instância correspondente não se encontra em memória por ter sido transferida por SWAP-OUT. Primeiramente, o método Swap-in a partir do IDP envia a mensagem PESQUISAR-IDP ao objeto Tabela Virtual de Processos para obter a informação referente à instância: EBe (Endereço Base na Extensão).

(2) A partir do envio da mensagem OBTER-TMO ao objeto Tabela de Processos é obtido o Tamanho de memória ocupada (TMO) pela instância.

(3) Em seguida é realizada a realocação do Processo de Alocação para uma nova localidade na Memória Real com base no parâmetro TMO, sendo alterado o novo Endereço Base (EB) através do envio da mensagem ALTERAR-AMR ao objeto Tabela de Processos.

(4) Após a execução dos procedimentos constantes no item (3) é realizada a transferência da instância armazenada na Extensão, através da mensagem CARGA, para o novo Endereço Base.

(5) A mensagem REMOVER é enviada ao objeto Tabela Virtual de Processos que efetua a remoção do elemento desta Tabela identificado pelo IDP.

(6) Finalmente é enviada a mensagem ALTERAR-ISO ao objeto Tabela de Processos que desativa o Indicador de Swap-Out (ISO).

3.2.1.4 **PERSPECTIVAS DE EXPANSÃO DO OBJETO MEMÓRIA**

O Objeto Memória foi projetado inicialmente para atender as demandas básicas do Ambiente Poesis no que se refere a administração de memória principal e secundária permitindo uma flexibilidade e modularidade resultante do enfoque orientado a objeto, no processamento dos mais diferentes tipos de objetos que constitui o GOLGO.

O próximo passo evolutivo do ambiente Poesis é a manipulação de informações multi-mídia, ou seja, mais precisamente a manipulação de imagens, entre outras. Sendo assim, será necessário modificar o Objeto Memória para incluir a manipulação deste tipo de informação. Isto se tornará relativamente fácil, tendo em vista a maneira como este objeto gerenciador foi projetado. Ou seja, a propriedade de modularidade irá facilitar sobremaneira a implementação de outras formas de dispositivos de armazenamento, bem como sua manipulação. Da mesma maneira, será facilitado o processo de implantação em outros tipos de equipamentos.

3.2.2 GERENCIAMENTO DE OBJETOS

O Gerenciamento de Objetos do Ambiente Poesis é a parte fundamental do sistema, pois permitirá que os usuários manipulem as diversas ferramentas existentes para alcançar o objetivo desejado. Estas ferramentas no enfoque orientado para objetos, são manipuladas através dos objetos disponíveis no Ambiente. Entretanto, para permitir esta manipulação de forma mais flexível torna-se necessário uma estrutura que proporcione uma flexibilidade ao Ambiente, que também possibilite uma evolução de seus objetos. O gerenciamento de objetos consiste em:

- Permitir a manipulação das instâncias dos Objetos do Ambiente, a partir da ativação de suas operações internas disponíveis;
- Proporcionar a interação direta entre o usuário e o Objeto do Sistema, através da implementação de Interfaces personalizadas em cada um deles;
- Manter um controle de acesso às operações internas, instâncias e processamentos desenvolvidos no Ambiente.

Para implementar as funções básicas descritas acima, foi criado o GOLGO cuja estrutura em forma de grafo permite ao usuário a navegação através do Ambiente possibilitando o acesso controlado à cada Objeto do Ambiente, através de suas Interfaces.

Durante o processo de implementação do GOLGO, foram sendo criados alguns objetos necessários a implantação das características descritas no capítulo 2 deste trabalho classificados em duas categorias:

- Objetos Construtores;
- Objetos de Gerenciamento.

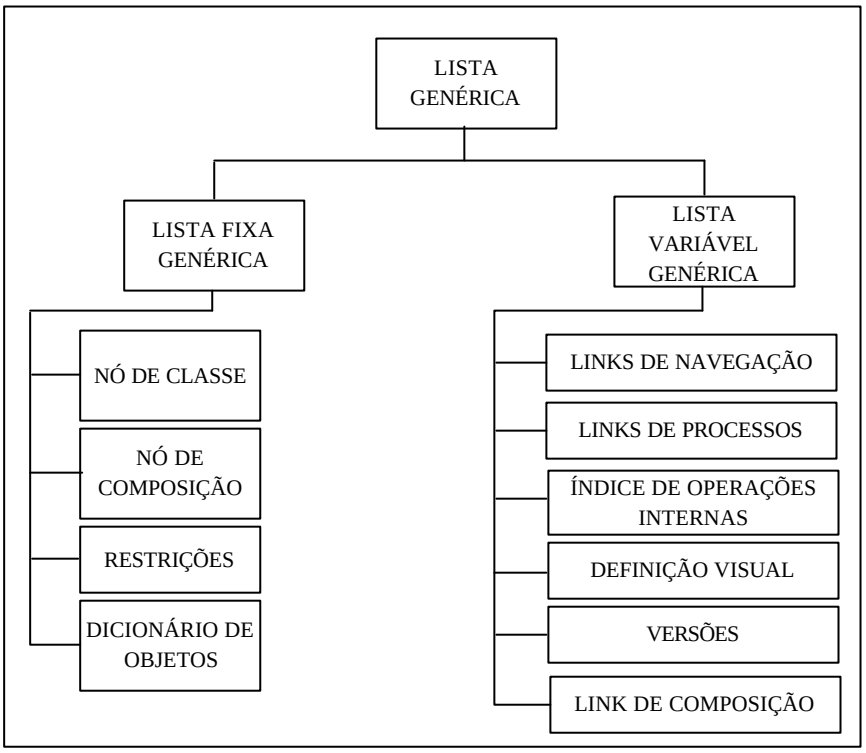
Os Objetos construtores foram criados para implementar a Hierarquia de Classes do GOLGO representada pela figura 14 visando otimizar o desenvolvimento do Ambiente, a partir da utilização dos recursos das Linguagens de Programação Orientada a Objetos: herança e Polimorfismo. Os Objetos de Gerenciamento são responsáveis pelo funcionamento do Ambiente sendo construídos a partir dos Objetos Construtores conforme pode ser visto na figura 14.

Estes duas categorias de objetos que participam da implementação do GOLGO serão especificados a seguir.

3.2.2.1 OBJETOS CONSTRUTORES

A partir do processo de especificação dos objetos do gerenciamento algumas similaridades foram detectadas no que se refere ao comportamento dos objetos, ou seja, a maneira pela qual os objetos manipulam suas instâncias, a forma como são armazenadas e recuperadas suas instâncias. Estas similaridades foram paulatinamente sendo tratadas e através do processo de ***especialização*** e ***generalização*** foram identificadas algumas super-classes e subclasses, o que se conclui que seriam necessário a utilização de objetos genéricos que permitiriam a construção dos objetos de gerenciamento do GOLGO. Estes objetos foram então denominados de objetos construtores e serão especificados a seguir.

FIGURA 14 - HIERARQUIA DA CLASSE LISTA GENÉRICA.



Os objetos construtores compreendem um conjunto de classes utilizadas para a construção de novos objetos. A idéia de sua utilização surgiu a partir da análise das características estruturais e comportamentais dos objetos de gerenciamento e tendo em vista ainda a possibilidade de uso do mecanismo de herança para redução do esforço de programação. Utilizando os processos de classificação, especialização e generalização, conceitos básicos de Programação Orientada a Objetos, os objetos de gerenciamento que possuíam semelhanças estruturais e comportamentais foram agrupados em classes construtoras. As classes de objetos construtores identificadas e utilizadas pelo sistema são:

- Lista genérica;
- Lista Fixa genérica;
- Lista Variável genérica;
- Árvore-B genérica;

A figura 14 ilustra a hierarquização destas classes. A classe Lista genérica é uma super-classe que agrupa as classes: Lista Fixa e Lista Variável. A classe Lista Fixa possui propriedades estruturais e comportamentais que a difere da classe Lista Variável, sendo pois necessária a diferenciação destas classes que possuem apenas uma característica comum entre elas: o fato de serem Listas genéricas, herdando portanto, as propriedades da super-classe Lista genérica.

Nas próximas subseções serão descritas estas classes que compõem o conjunto dos objetos construtores do GOLGO.

3.2.2.1.1 **CLASSE: LISTA GENÉRICA**

Ao analisar as características estruturais e comportamentais dos objetos de gerenciamento algumas semelhanças foram percebidas tais como a forma como as instâncias são manipuladas e armazenadas. Inicialmente, a primeira fase do processo de desenvolvimento consistiu da especificação dos objetos de gerenciamento. Após uma análise das características estruturais e comportamentais destes objetos vislumbrou-se a possibilidade de implementação da maioria deles como sendo Listas Encadeadas cujas instâncias ou eram elementos destas listas ou até mesmo eram listas. A partir desta constatação surgiu a necessidade da criação da Classe Lista Genérica que será descrita a seguir.

DESCRIÇÃO

A classe Lista Genérica possui o mesmo conceito conhecido do tipo abstrato de dados: Lista. É caracterizada como sendo o conjunto representado pela expressão:

$$\mathbf{L = \{ e_1, e_2, ..., e_n \}} \text{ , onde:}$$

cada $\mathbf{e_i}$ $\mathbf{(1 \leq i \leq n)}$ é um elemento genérico componente da lista e possui um Endereço Relativo associado que permite a sua localização;

n é o número de elementos da Lista genérica.

Um elemento genérico pode ser compreendido como sendo a tupla

$$\mathbf{e = < T, EP, INF > ,} \text{ onde:}$$

T é o tamanho de cada elemento;
EP é o apontador para o próximo elemento;
INF é a informação contida no elemento.

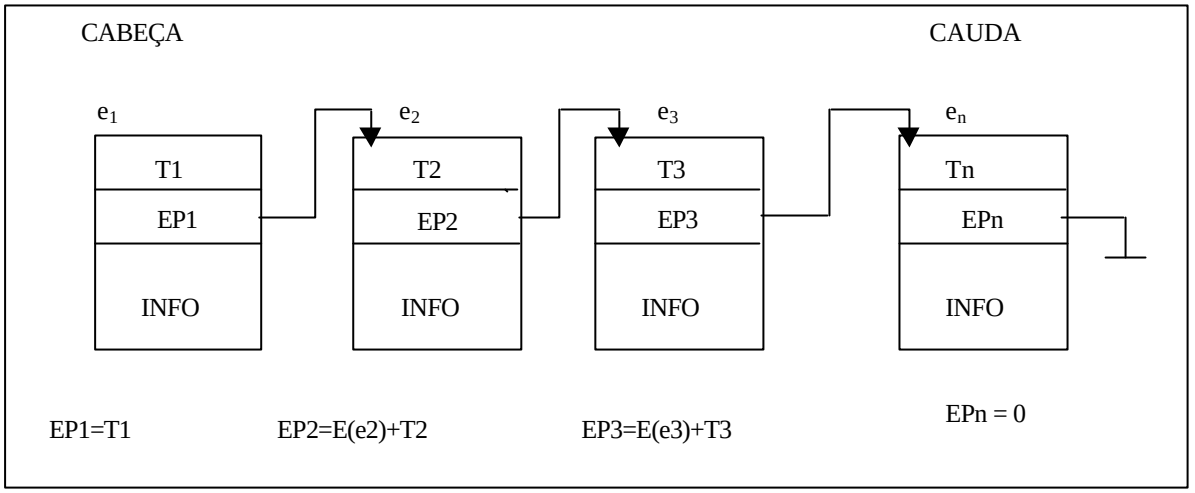
A definição acima indica que a classe Lista Genérica é uma implementação de um tipo especial de lista: a Lista Encadeada, ou seja, os seus elementos genéricos contêm um apontador que localiza o endereço do próximo elemento genérico. Dois outros conceitos são fundamentais na classe Lista Genérica. Estes conceitos são: **CABEÇA** e **CAUDA** da lista. A **CABEÇA** de uma lista é um apontador que localiza o primeiro elemento: $\mathbf{e_1}$. A **CAUDA** de uma lista é definida como sendo um apontador para o último elemento da lista: e_n . A cauda de uma lista genérica L poderá também ser identificada da seguinte forma:

Dado um elemento e_i de uma Lista Genérica L, composta por n elementos genéricos, dizemos que e_i é a Cauda de L, se e somente se, e_i . EP = 0 e i = n. A figura 15 ilustra uma instância de Lista Genérica.

Quanto às suas características comportamentais uma Lista Genérica possui operações que serão descritas mais adiante. As variáveis Cabeça e Cauda são utilizadas em todas as operações internas da Classe. A inserção na Lista Genérica é sempre realizada no final da lista, sendo o elemento inserido considerado a nova Cauda da Lista Genérica. A remoção de um elemento genérico poderá ser feita para qualquer elemento da lista. A busca é sempre realizada a partir da Cabeça da lista.

A classe Lista Genérica, possui um ***objeto membro*** denominado de Elemento Genérico, conforme especificado acima. O objeto é considerado genérico porque seu objetivo é armazenar informações que poderão ter um formato e um tamanho não determinado, sendo portanto de uso genérico.

FIGURA 15 - Lista Genérica com n elementos.



ESTRUTURA DA CLASSE LISTA GENÉRICA

A classe Lista Genérica foi implementada usando a definição anterior e sua estrutura pode ser visualizada pela tupla:

< CABEÇA, CAUDA, ELEM>, onde:

CABEÇA é o Endereço Base da Lista;
CAUDA é o Endereço Relativo de localização do último elemento;
ELEM corresponde a implementação do Elemento Genérico.

A variável **CABEÇA** é um apontador do tipo Elemento Genérico que contem o endereço base da Lista Genérica, e pela definição acima, aponta sempre para o primeiro elemento. A variável **CAUDA** contem o endereço relativo do último elemento da Lista genérica. Uma Lista Genérica é considerada vazia, ou seja, que não possui elementos genéricos, quando o conteúdo de **CABEÇA** for igual a zero. O **Elemento Genérico** representado pela variável **ELEM** definido anteriormente corresponde a implementação dos elementos que compõem a estrutura de Lista. Este objeto é composto pelas seguintes variáveis definidas na tupla:

< T, EP, INF>, onde:

T é o tamanho em bytes da instância do elemento;
EP é o endereço relativo do próximo elemento;
INF é a informação contida no elemento genérico.

Na figura 15 é apresentada a composição do objeto membro Elemento Genérico. A variável **INF** irá conter a estrutura interna das classes dos objetos construtores Lista Fixa e Lista Variável, implementados como subclasses da classe Lista Genérica. A variável **T** justifica-se pela necessidade de se determinar o endereço do próximo elemento, já que por ser genérico um determinado elemento poderá ter um tamanho que varia conforme a necessidade. O endereço do próximo elemento é definido da seguinte forma:

$EP(e_i) = ENDEREÇO(e_i) + T(e_i)$, onde:

e_i é um elemento genérico, e $0 < i < n$;

ENDEREÇO (e_i) é o endereço relativo do elemento e_i. Conforme especificação contida na seção 3.2.1 (Gerenciamento de Memória) a implementação dos objetos requer o uso da filosofia de endereçamento. Portanto, um elemento genérico é localizado num Endereço Relativo à posição inicial correspondente à Cabeça da Lista.

OPERAÇÕES INTERNAS DA CLASSE LISTA GENÉRICA

A Classe Lista Genérica possui um conjunto de operações que manipulam suas instâncias. Estas operações internas, ou métodos, que serão descritos a seguir, permitem a manipulação das instâncias tanto em memória real quanto em memória persistente.

CRIAÇAO - Esta operação interna permitirá que seja criado o primeiro elemento da lista, através da alocação de memória necessária a este elemento.
CARGA PARCIAL - Esta operação interna permitirá que seja transferido da memória virtual para a memória real apenas um elemento de uma lista genérica.
DESALOCAÇÃO DA INSTÂNCIA - Esta operação permitirá que uma instância de uma lista genérica seja desalocada da memória real conforme descrito na seção 3.2.1.
INSERÇÃO PARCIAL - Esta operação interna permitirá a inserção de um elemento na lista genérica, não sendo necessário que toda a lista esteja em memória real.
CARGA TOTAL - Esta operação permite que todos os elementos de uma Lista Genérica sejam transferidos da respectiva Área de Armazenamento Virtual para a memória real.
INSERÇÃO REAL - Esta operação permite a inserção de novo elemento em uma instância de Lista Genérica armazenada na memória real.
SOBRECARGA - Esta operação interna permite que um determinado elemento de uma lista seja transferido da memória persistente para a memória real sem necessariamente haver uma nova alocação de memória, ou seja, ocorre simplesmente a transferência física para uma determinada região identificada pelo IDP do Processo de Alocação.
SOBRESALVA - Esta operação permite a atualização de uma instância da Lista Genérica, ou de um único elemento localizado na Área de Armazenamento Virtual. Neste caso ocorre uma modificação do conteúdo da instância em memória persistente sem necessariamente haver uma expansão desta instância.
SALVA - Esta operação interna permite a transferência para a memória persistente de uma instância de uma Lista Genérica localizada em memória real possibilitando com isso o armazenamento permanente desta instância.
REMOÇÃO - Esta operação interna permite a remoção de um elemento da Lista Genérica armazenada em memória real.
OBTER-ER-PRÓXIMO - Esta operação tem por finalidade a obtenção do Endereço Relativo do próximo elemento a partir das informações contidas no elemento genérico.
OBTER-ER-CAUDA-VIRTUAL - Esta operação interna obtém o Endereço Relativo do elemento genérico correspondente à Cauda da Lista através do acesso a sua localização em memória persistente. Esta operação é realizada sem a necessidade da carga total da lista em memória real.
OBTER-ER-CAUDA-REAL - Esta operação interna tem por objetivo obter o Endereço Relativo do elemento genérico correspondente à Cauda da Lista quando a mesma encontra-se totalmente armazenada em memória real.
TRANSFERÊNCIA LÓGICA - Esta operação interna efetua uma compactação de uma instância de Lista Genérica. A partir da estrutura original é feita uma *varredura lógica* em todos os elementos da lista. Cada elemento é copiado para uma região de memória real (buffer) alocada para esta finalidade. Isto permitirá que apenas os elementos "ativos" sejam copiados, ou seja, os elementos que não foram removidos durante o processamento desta instância. Esta operação é efetuada no momento anterior à transferência da memória real para a memória persistente (SALVA).
OBTER-TAMANHO - Esta operação interna tem por objetivo recuperar a informação contida em um determinado elemento genérico que indica o tamanho (em bytes) ocupado pelo mesmo.
OBTER-ENDEREÇO-REAL - Este método solicita ao Objeto Memória que forneça o Endereço

Base Real onde foi armazenada uma determinada instância de um objeto de gerenciamento. Corresponde a invocação do método OBTER-EREAL do Objeto Memória.

SALVA-PARCIAL - Esta operação interna tem por finalidade permitir que um novo elemento genérico de determinada instância de Lista Genérica seja inserido no final desta instância armazenada em memória persistente. A inserção corresponde à transferência deste elemento para o final do Bloco Virtual.

BLOQUEIA - Esta operação interna efetua o bloqueio da instância correspondente a uma determinada Lista Genérica solicitando ao Objeto Memória que ative o Indicador de Bloqueio do Processo de Alocação identificado pelo IDP.

DESBLOQUEIA - Esta operação interna solicita ao Objeto Memória que desative o Indicador de Bloqueio do Processo de Alocação identificado pelo IDP correspondente a uma determinada instância de Lista Genérica.

3.2.2.1.2 **SUBCLASSE: LISTA FIXA GENÉRICA**

O processo de generalização permitiu que os objetos de gerenciamento fossem agrupados em uma única classe: a Lista Genérica. Muito embora tenha sido um avanço para implementação destes objetos, ainda haviam características muito peculiares entre alguns objetos de gerenciamento que justificavam a criação de uma subclasse de Lista Genérica: a subclasse LISTA FIXA GENÉRICA. O processo de criação desta subclasse denominado especialização permitiu a montagem desta hierarquia de classes de fundamental importância na implementação do GOLGO. Nesta seção serão apresentadas as características estruturais e comportamentais desta subclasse.

DESCRIÇÃO

A subclasse Lista Fixa Genérica corresponde a um caso particular da Classe Lista Genérica que se caracteriza por possui homogeneidade de elementos genéricos no que se refere ao tamanho, ou seja todos os elementos de uma determinada instância de Lista Fixa possuem o mesmo tamanho. Uma outra característica desta subclasse corresponde ao fato de que os elementos genéricos que a compõe jamais podem ser removidos da lista, ou seja, uma vez criado, um dado elemento genérico permanece na lista e mesmo que tenha sido removido logicamente pelo GOLGO a sua posição permanece na Lista e consequentemente o tamanho da lista permanece inalterado. Esta característica justifica-se pelo fato de que cada elemento genérico é identificado pelo Endereço Relativo que ocupa durante toda a sua vida útil. A Lista Fixa foi criada visando implementar certos objetos de gerenciamento cujas instâncias são localizadas pelo Endereço Relativo e que devem permanecer com esta identificação até que sejam "removidos" logicamente do ambiente. Uma Lista Fixa pode ser considerada então como sendo uma lista de instâncias de um objeto de gerenciamento.

Para se manter uma lista com tais características de forma otimizada, torna-se necessário identificar os vazios provenientes da remoção lógica de instâncias para deixá-los disponíveis a uma próxima utilização. Como o tamanho de cada elemento não varia basta apenas localizar cada vaga disponível. Visando permitir a implementação da estratégia acima descrita foi criada uma estrutura interna a esta subclasse denominada Lista de Vagas Disponíveis. Embora possa parecer complicado o fato da existência de duas listas em uma mesma instância de Lista Fixa, sua implementação é muito simples e será especificada a seguir.

ESTRUTURA DA CLASSE LISTA FIXA

Sendo uma subclasse de Lista Genérica, a classe Lista Fixa herdará as propriedades estruturais e comportamentais da super-classe. Será herdada a estrutura caracterizada pelo conjunto dado pela expressão:

$\mathbf{L = \{ e_1, e_2, ..., e_n \}}$, onde:

cada $\mathbf{e_i}$ $\mathbf{(0 < i < n)}$ é um elemento genérico componente da lista e possui um Endereço Relativo associado que permite a sua localização;

n é o número de elementos da Lista genérica.

O elemento genérico, também herdado, correspondente a tupla:

e = < T, EP, INF > , onde:

T é o tamanho de cada elemento;
EP é o apontador para o próximo elemento;
INF é a informação contida no elemento.

Uma Lista Fixa contém então todos as variáveis descritas acima herdadas da Super-classe Lista Genérica.

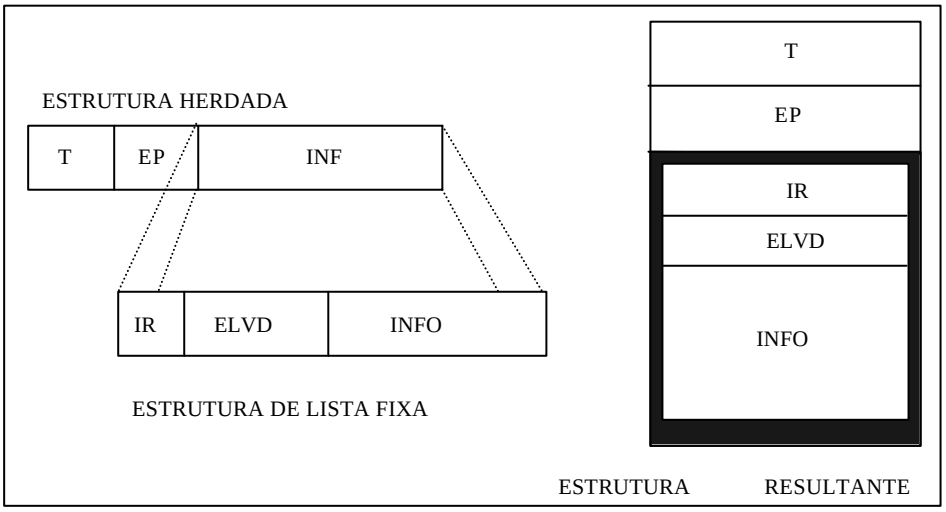
As características estruturais da Classe Lista Fixa foi implementada a partir da variável **INF.** O componente **INF**, que representa a informação contida no elemento genérico para a classe Lista Fixa genérica pode ser compreendido a partir da tupla:

INF = < IR, ELVD, INFO>, onde:

IR é o identificador de remoção lógica do elemento;
ELVD é o Endereço do próximo elemento da Lista de Vagas Disponíveis;
INFO corresponde a informação contida no elemento;

A variável IR identifica que elementos foram removidos logicamente pelo GOLGO e que podem ser reutilizados. A variável ELVD contém o Endereço Relativo do próximo elemento genérico pertencente à Lista de Vagas Disponíveis. A figura 16 ilustra a composição de estruturas da classe Lista Fixa.

FIGURA 16 - Composição da estrutura da classe Lista Fixa Genérica.



REUTILIZAÇÃO DE ELEMENTOS DA LISTA FIXA - LISTA DE VAGAS DISPONÍVEIS.

A Lista de Vagas Disponíveis ou LVD é formada a partir do elemento genérico corresponde a CAUDA da lista. Quando não existe a LVD, o valor contido na variável ELVD de CAUDA possuirá o valor negativo representado por -1, caso contrário o valor contido nesta variável irá corresponder ao primeiro elemento da LVD. Para tornar isto mais compreensível pode-se imaginar a seguinte situação:

Seja $\mathbf{e_k}$ um elemento qualquer de uma lista fixa

$LF = \{ e_1 , e_2 , ... , e_{k-1} , e_k , e_{k+1} , ... , e_n \}$.

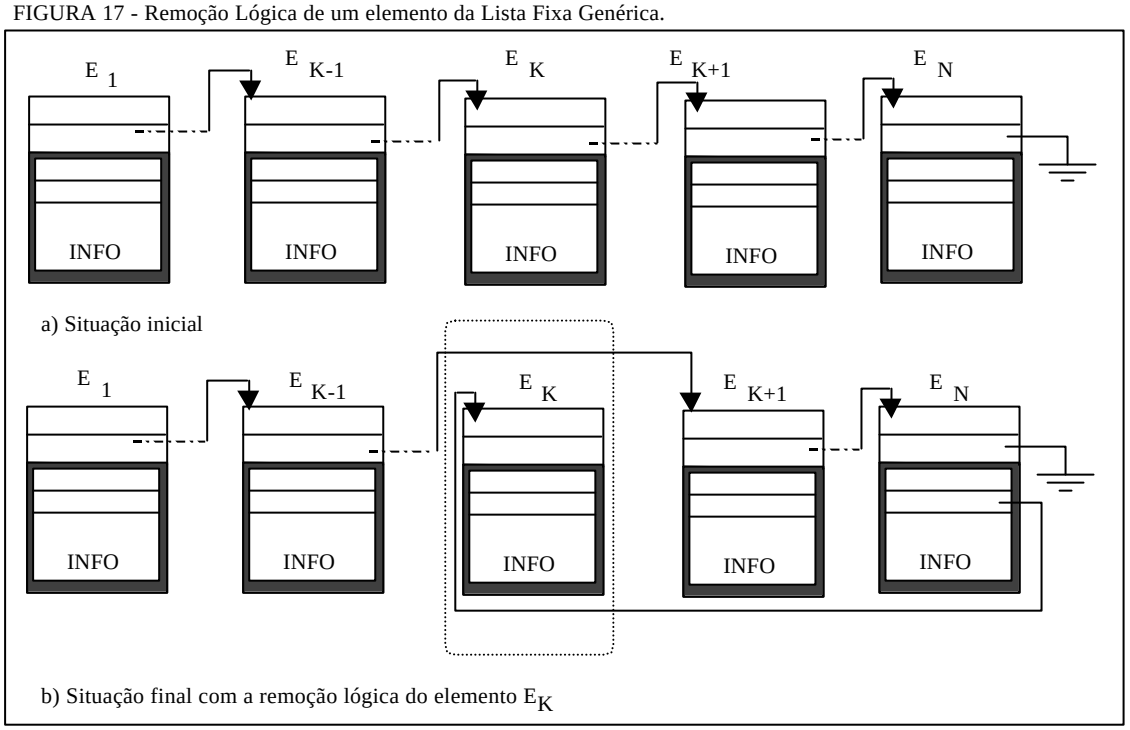
Ao se remover logicamente o elemento $\mathbf{e_k}$ ocorre os seguintes eventos:

(1) O conteúdo da variável EP do elemento anterior a $\mathbf{e_k}$ (e_{k-1}) irá conter o valor correspondente ao conteúdo da variável EP de $\mathbf{e_k}$, ou seja o Endereço Relativo de e_{k+1} , o que corresponde a expressão:

$$\mathbf{e_{k-1} . EP = e_k . EP}$$

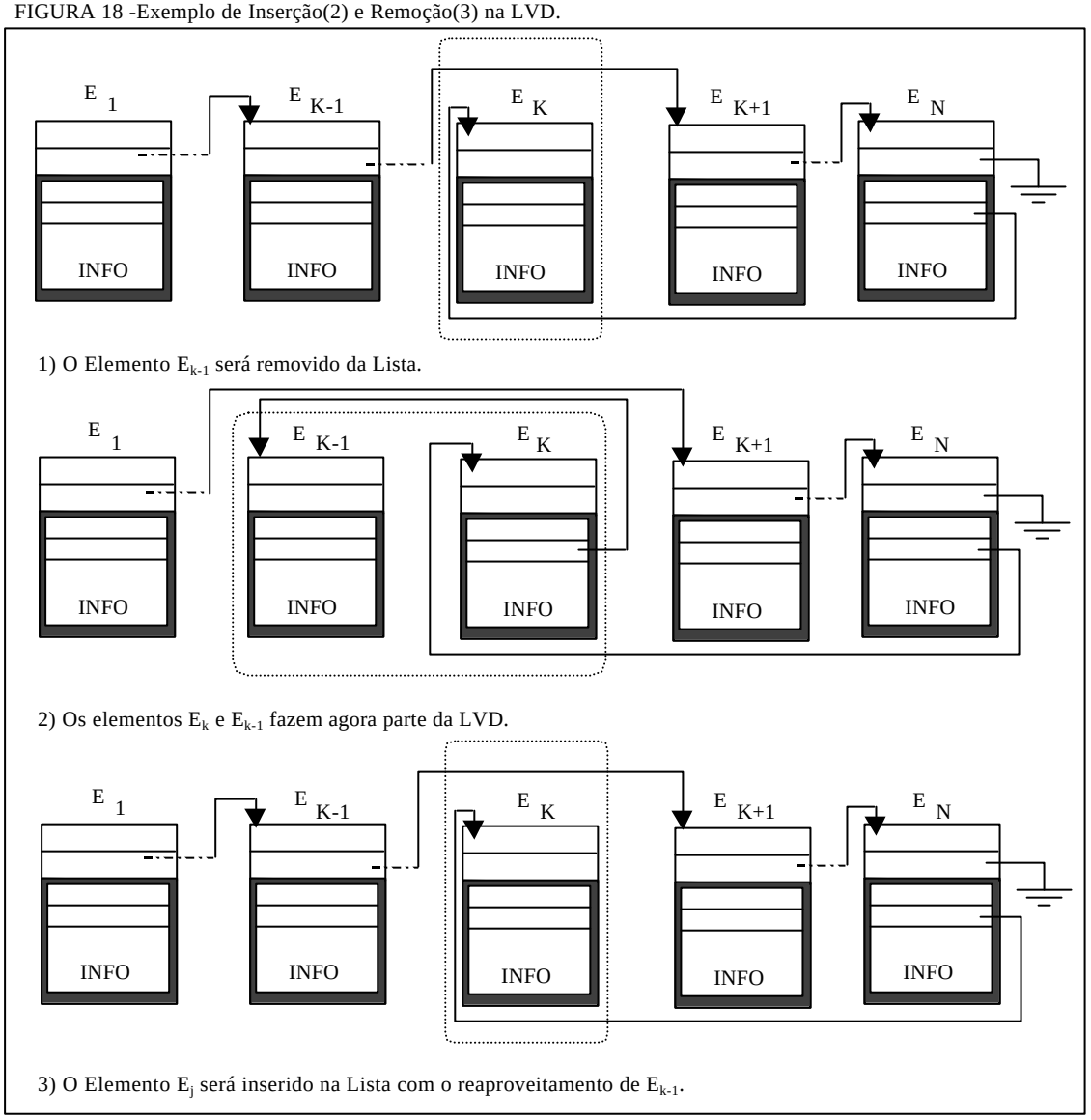
(2) O Endereço Relativo do elemento $\mathbf{e_k}$ será atribuído à variável ELVD do elemento CAUDA (e_n) iniciando assim a LVD. A figura 17 ilustra a remoção lógica descrita acima.

FIGURA 17 - Remoção Lógica de um elemento da Lista Fixa Genérica.



Na Lista de Vagas Disponíveis a inserção dos elementos é realizada sempre no final da lista. O final de uma LVD é identificado pelo conteúdo da variável ELVD que no caso conterá o valor -1 já que zero (0) corresponde ao Endereço Relativo do primeiro elemento genérico da lista.
Sendo assim torna-se necessário efetuar uma pesquisa para localizar o último elemento da LVD. O ponto de partida para efetuar a busca é sempre o elemento genérico CAUDA da Lista Fixa. Ao ser localizado, a inserção se realiza alterando o conteúdo da variável ELVD deste último elemento que conterá o Endereço Relativo do novo elemento da LVD. A variável ELVD deste novo elemento conterá a partir deste momento o valor negativo (-1) indicando assim o novo último elemento da LVD.

FIGURA 18 -Exemplo de Inserção(2) e Remoção(3) na LVD.



A remoção de um elemento da LVD é feito também a partir de uma pesquisa do último elemento. Sempre o último elemento é a vaga disponível para reutilização. Ao ser localizado, o Endereço Relativo deste último elemento da LVD é fornecido ao método da classe Lista Fixa, e ocorre então a atualização do antepenúltimo elemento da LVD cuja variável ELVD conterá o valor negativo (-1). A figura 18 ilustra uma inserção e uma remoção na LVD.

HERANÇA DE OPERAÇÕES

A Classe Lista Fixa ao ser implementada como uma subclasse da Classe Lista Genérica herdou as propriedades estruturais e comportamentais desta super-classe. As propriedades comportamentais correspondem a parte dinâmica do objeto, ou seja suas operações internas. Desta forma, todas as operações internas descritas na seção 3.2.2.1.1 podem ser usadas pela classe Lista Fixa. A utilização das operações pode ser feita de duas maneiras:

- Usando diretamente as operações da Classe Lista Genérica;
- Usando na implementação dos métodos da Classe Lista Fixa.

OPERAÇÕES INTERNAS DA CLASSE LISTA FIXA GENÉRICA

A Classe Lista Fixa Genérica possui um conjunto de operações internas, ou métodos, que

serão descritos a seguir, e que permitem a manipulação das instâncias tanto em memória real quanto em memória persistente.

INSERÇÃO PARCIAL - Esta operação interna tem por finalidade efetuar a inserção de elementos genéricos na Lista Fixa sem a necessidade de manipular toda a lista em memória real. Em sua implementação foram utilizados métodos herdados da classe Lista Genérica tais como: Criação, Carga-Parcial, Inserção-Parcial, Obtem-ER-Cauda-Virtual, entre outros. Este método também manipula a LVD para verificar a possibilidade de reuso de vagas disponíveis.
REMOÇÃO PARCIAL - Esta operação corresponde a implementação da remoção lógica descrita anteriormente. Neste caso, a remoção lógica ocorre sem a necessidade de manipulação de toda a lista em memória real. O elemento removido logicamente é colocado na LVD para posterior reutilização e é realizada uma modificação da variável IR (Indicador de Remoção Lógica) que passará a indicar de fato a remoção. Este método utiliza também métodos herdados da classe Lista Genérica.
INSERÇÃO REAL - Esta operação possui o mesmo objetivo que a Inserção Parcial, ou seja, a inserção de um elemento genérico na lista. A diferença é que neste método toda a lista foi armazenada em memória real para este processamento. Da mesma maneira, é realizada uma verificação na LVD visando o reuso de vagas de elementos disponíveis. Este método também utiliza alguns métodos herdados da classe Lista Genérica na sua implementação.
REMOÇÃO REAL - Este método efetua a remoção lógica de um elemento genérico informado através de seu Endereço Relativo, colocando a sua posição em disponibilidade na LVD e alterando a variável IR da mesma forma que a Remoção Parcial. A diferença está na alocação parcial ou total da lista. No caso da Remoção Real toda a lista está armazenada em memória real.
PESQUISA-LVD-PARCIAL - Esta operação, solicitada durante uma Inserção Parcial, tem por objetivo efetuar uma pesquisa na LVD buscando o Endereço Relativo correspondente a última vaga de elemento disponível inserida. Esta operação caracteriza-se por ser efetuada sem a necessidade de alocação de toda a instância da Lista Fixa em memória real.
PESQUISA-LVD-REAL - Esta operação realizada de maneira similar a Pesquisa Parcial, é solicitada durante uma Inserção Real, tendo por objetivo também efetuar uma pesquisa na LVD buscando o Endereço Relativo correspondente a última vaga de elemento disponível inserida. Esta operação caracteriza-se por ser efetuada com toda a instância da Lista Fixa alocada em memória real.
OBTER INDICADOR DE REMOÇÃO - Este método tem por finalidade fornecer o conteúdo da variável IR, ou Indicador de Remoção Lógica de um elemento genérico qualquer da Lista Fixa, cujo Endereço Relativo é informado via parâmetro.
OBTER ENDEREÇO DO PRÓXIMO ELEMENTO DA LVD - Esta operação permite obter o conteúdo da variável ELVD de um elemento genérico qualquer da Lista Fixa.

3.2.2.2.3 SUBCLASSE: LISTA VARIÁVEL GENÉRICA

A análise dos objetos de gerenciamento cujas características estruturais e comportamentais se assemelhavam à listas encadeadas propiciou a descoberta da subclasse Lista Fixa Genérica. Os demais objetos não considerados pertencentes à Classe Lista Fixa Genérica, pois possuíam elementos que não satisfaziam às propriedades desta subclasse, deram origem a uma nova classe: a subclasse LISTA VARIÁVEL GENÉRICA. A combinação dos processos de generalização e especialização tornou possível identificar estas propriedades particulares à nova subclasse permitindo assim a construção desta hierarquia da Classe Lista Genérica. Nesta seção serão apresentadas as características estruturais e comportamentais desta nova subclasse, bem como as características por ela herdadas.

DESCRIÇÃO

A subclasse Lista Variável Genérica compreende um tipo particular de Lista Genérica cujas principais características são:

- Possui elementos cujo tamanho varia conforme o tipo de informação que armazena;
- Seus elementos podem ser removidos fisicamente da lista;
- A inserção de um novo elemento ocorre sempre no final da lista (característica herdada da superclasse);
- Possui dois tipos de elementos distintos: Header e Componente. O Header é um tipo de elemento que sempre ocupará a posição inicial da lista. O Componente representa os demais elementos genéricos. Em geral, o Header é usado pelos objetos de gerenciamento para armazenar informações de controle da lista, enquanto que o Componente tem a finalidade exclusiva de armazenamento das informações da lista.
- Numa mesma instância de Lista Variável poderão existir elementos do tipo Componente de tamanhos variados;
- Uma instância de Lista Variável é sempre totalmente alocada em memória real para manipulação, sendo que ao ser solicitada a sua gravação em memória persistente é realizada a TRANSFERÊNCIA LÓGICA. A Transferência Lógica é uma forma de compactação de instâncias deste tipo de lista.

ESTRUTURA DA CLASSE LISTA VARIÁVEL

A classe Lista Variável Genérica implementada como subclasse de Lista Genérica tornou possível a partir do mecanismo de herança existente na POO a utilização das características estruturais. Desta forma uma Lista Variável pode ser compreendida como sendo o conjunto:

$\mathbf{L = \{ e_1, e_2, ..., e_n \}}$, onde:

cada $\mathbf{e_i}$ **(0 < i < n)** é um elemento genérico componente da lista e possui um Endereço Relativo associado que permite a sua localização;

n é o número de elementos da Lista genérica.

O elemento genérico, também herdado, correspondente a tupla:

e = < T, EP, INF > , onde:

T é o tamanho de cada elemento;
EP é o apontador para o próximo elemento;
INF é a informação contida no elemento.

As características estruturais próprias da Classe Lista Variável foram implementadas a partir da variável INF, que representa as informações armazenadas pelo elemento genérico. A estrutura da subclasse Lista Variável pode ser compreendida pela tupla:

INF = < TE, INFO> , onde:

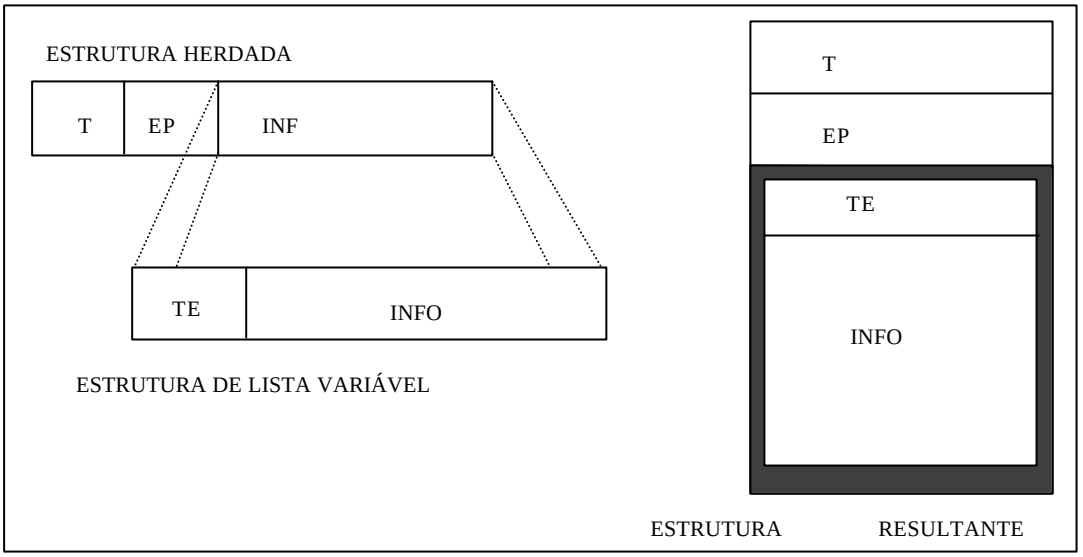
TE indica o tipo de elemento genérico;
INFO corresponde a informação contida no elemento.

A variável TE é usada para identificar o tipo de elemento que compõe a lista e assume dois valores:

'H' que indica que o elemento é do tipo Header;
'C' que indica que o elemento é um Componente.

A variável INFO corresponde a informação propriamente dita, armazenada pelo elemento. A especificação de INFO será dada a partir da especificação de cada classe de objeto de gerenciamento implementado como subclasse de Lista Variável. A figura 19 mostra como é feita a composição das estruturas herdadas e próprias da Classe Lista Variável.

FIGURA 19 - Composição da estrutura da classe Lista Variável Genérica.



HERANÇA DE OPERAÇÕES

Ao ser implementada como subclasse de Lista Genérica, a classe construtora Lista Variável Genérica tem a possibilidade de herdar todas as operações internas de sua Super-classe. Desta maneira, é possível manipular instâncias de Lista Variável como sendo instâncias de Lista Genérica. Assim, todas as operações descritas na seção 3.2.2.1.1 estão disponíveis para esta classe. A utilização destas operações foi feita de duas formas: pelo uso direto, que torna possível invocar um método definido para a super-classe Lista Genérica diretamente; pelo utilização na construção das operações próprias da classe Lista Variável.

OPERAÇÕES INTERNAS DA CLASSE LISTA VARIÁVEL

A classe Lista Variável Genérica possui um conjunto de operações internas que tornam possível a manipulação de suas instâncias. Serão descritos a seguir esta operações, ou métodos, desenvolvidos para esta classe.

CRIAÇÃO - Esta operação interna tem por finalidade inicializar uma instância de Lista Variável a partir da criação do elemento inicial: o Header. Esta operação utiliza operações internas da super-classe Lista Genérica.
INSERÇÃO - Esta operação interna permite a inserção de elementos do tipo Componente em uma instância já inicializada. Este método também utiliza operações internas da super-classe Lista Genérica.
REMOÇÃO HEADER - Esta operação tem por objetivo efetuar a remoção do elemento Header. Neste caso, somente será possível após serem removidos todos os elementos do tipo Componente da lista.
REMOÇÃO COMPONENTE - Esta operação efetua a remoção de elementos do tipo Componente.

3.2.2.2.4 CLASSE: ÁRVORE-B GENÉRICA

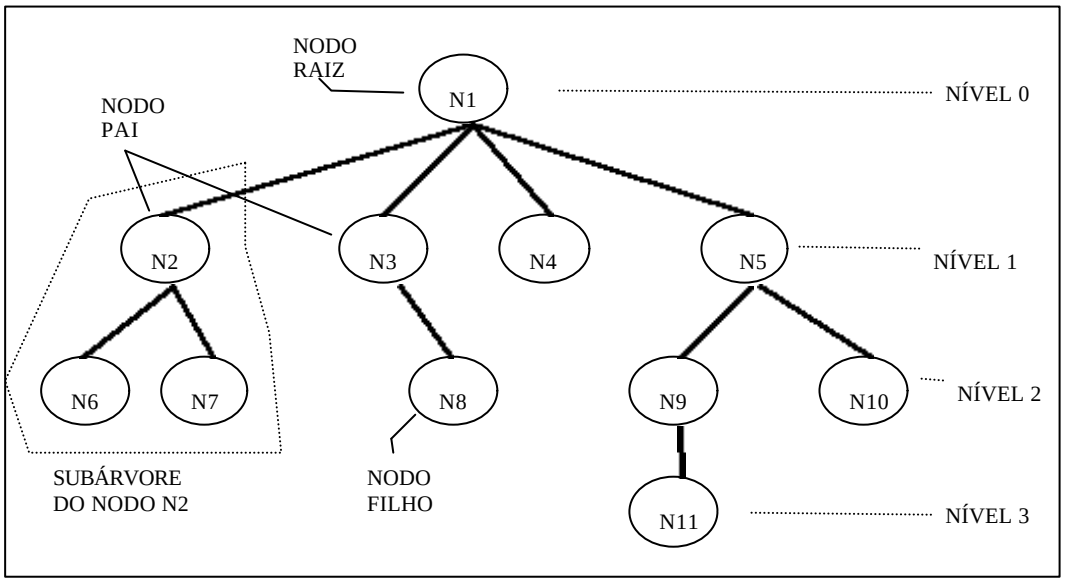
Um outro tipo de classe de objetos de gerenciamento identificado no processo de classificação foi a Classe Árvore-B. Alguns objetos de gerenciamento necessitavam de uma implementação que permitisse um acesso mais rápido para obtenção de informações armazenadas em suas instâncias. Um tipo de estrutura muito comum em implementações que possuem estas características é a árvore-B. Foi criada então a super-classe Árvore-B Genérica para permitir a implementação de diversos tipos de

objetos. Através do mecanismo de herança da POO torna-se mais rápida a implementação de muitos objetos como sendo subclasses desta classe Árvore-B Genérica.

DESCRIÇÃO

A Árvore-B Genérica implementada no Ambiente consiste de um caso especial muito conhecido da estrutura de dados: árvore de pesquisa binária, ou simplesmente árvore-B. Em geral, a estrutura de dados árvore é formada por elementos denominados **nodos**. Cada nodo, exceto um nodo chamado RAIZ, por ser o nodo inicial da árvore, possui um nodo **pai** (antecedente) e diversos (zero ou mais) nodos **filhos** (descendentes). Um nodo que não tem filhos é denominado de FOLHA, ou nodo terminal. São denominados de nodos internos aqueles que não são do tipo folha. Um outro conceito muito importante é o **nível** de um nodo, e consiste da seqüência de seus nodos antecedentes a partir do nodo RAIZ cujo valor é zero. Denomina-se **subárvore** de um nodo **n** o conjunto formado por este nodo e todas as subárvores dos seus descendentes (o nodo n, os nodos filhos de n, os nodos filhos dos filhos, etc.). A figura 20 apresenta um exemplo de árvore.

FIGURA 20 - Exemplo de uma árvore com 11 nodos.



Uma árvore de pesquisa é um tipo de árvore usada para conduzir de forma mais rápida uma pesquisa de uma determinada informação através de uma variável denominada ***chave de acesso*** ou ***chave de pesquisa*** que caracteriza-se por conter valores não repetidos. Esta propriedade de unicidade de chaves de pesquisa tem por finalidade garantir a integridade no processo de pesquisa. As árvores de pesquisa são muito utilizadas em SGBD´s e outros sistema de armazenamento de informações para implementação de Índices Multiníveis Dinâmicos. Uma árvore de pesquisa de ordem **k** é uma árvore cujos nodos contém pelo menos **k - 1** chaves de pesquisa e **k** apontadores na seguinte ordem dada pela tupla:

$< Ap_1, C_1, Ap_2, C_2, \ldots, Ap_{m-1}, C_{m-1}, Ap_k >$, onde:

m é a quantidade de subárvores existentes ($m \leq k$);
cada Ap_i é uma apontador para um nodo filho da subárvore (ou caso não exista é nulo);
cada C_i é uma chave de pesquisa;
$i \leq k$.

O processo de pesquisa inicia a partir de um valor de pesquisa V pelo nodo RAIZ cujo conteúdo contém as variáveis descritas na tupla acima. A navegação através das subárvores do nodo RAIZ é realizada pelos apontadores Ap_i´s que deverão satisfazer as condições:

1) $C_{i-1} < V < C_i$, para $1 < i < m$;

2) $V < C_i$, para i = 1;

3) $V > C_i$, para i = m;

A navegação é necessária quando se realiza uma pesquisa, uma inclusão e uma remoção de nodos na árvore. Esta estratégia visa garantir que sempre um nodo cujo valor da chave de acesso V satisfazendo umas das condições acima, permaneça em sua posição como nodo interno ou terminal.

Uma árvore-B é uma estrutura do tipo árvore de pesquisa descrita acima com algumas características a mais que tornam mais rápida a recuperação de informações. Cada nodo de uma árvore-B possui uma estrutura descrita de acordo com a tupla:

$< Ap_1, <C_1, P_1>, Ap_2, <C_2, P_2>, \ ... \ , Ap_{m-1}, <C_{m-1}, P_{m-1}>, Ap_k >$, onde:

k indica a ordem da árvore
m é a quantidade de subárvores existentes ($m \leq k$);
cada Ap_i é uma apontador para um nodo filho da subárvore (ou caso não exista é nulo);
cada par $<C_i , P_i >$ contem uma chave de pesquisa C_i e um apontador P_i para a localização de memória onde a informação representada por C_i está armazenada;
$i \leq k$.

Outra característica que diferencia uma árvore-B é o processo de ***balanceamento*** implementado nas operações de inserção e remoção de nodos. Este processo tem por finalidade manter a estrutura da árvore-B uniforme, ou seja, cada nodo pai deverá permanecer sempre com igual número de subárvores.

A árvore-B implementada no Ambiente é de ordem igual a 2, ou seja , cada nodo possui apenas uma chave de pesquisa, e duas subárvores: esquerda e direita, sendo então constituído pela tupla:

< ESE, CHAVE, INFO, ESD> , onde:

ESE é o Endereço da sub-árvore da esquerda;
CHAVE é variável de identificação do nodo;
INFO é a informação armazenada;
ESD é o Endereço da sub-árvore da direita.

Para se manter uma Árvore-B de forma otimizada, foi implementada a mesma filosofia de reuso de elementos disponíveis desenvolvida para a classe Lista Fixa Variável. Esta filosofia consiste em não remover fisicamente o elemento, sendo necessário identificar os vazios provenientes da remoção lógica de nodos para deixá-los disponíveis a uma próxima utilização. Independente do tamanho de cada elemento é utilizada uma variável que visa localizar cada vaga disponível a partir de uma estrutura interna a esta subclasse denominada Lista de Vagas Disponíveis. Embora possa parecer complicado o fato da existência de uma lista em uma instância de Árvore-B, esta estratégia visa diminuir o tempo de processamento na transferência de uma instância da memória real para a memória persistente, sendo que sua implementação é muito simples e será especificada a seguir.

ESTRUTURA DA CLASSE ÁRVORE-B GENÉRICA

A classe Árvore-B Genérica implementada para o GOLGO é composta pela estrutura interna dada pela tupla:

< RAIZ, NODO>, onde:

RAIZ é o Endereço Base para localização do nodo Raiz da árvore-B;
NODO corresponde ao elemento componente da árvore-B.

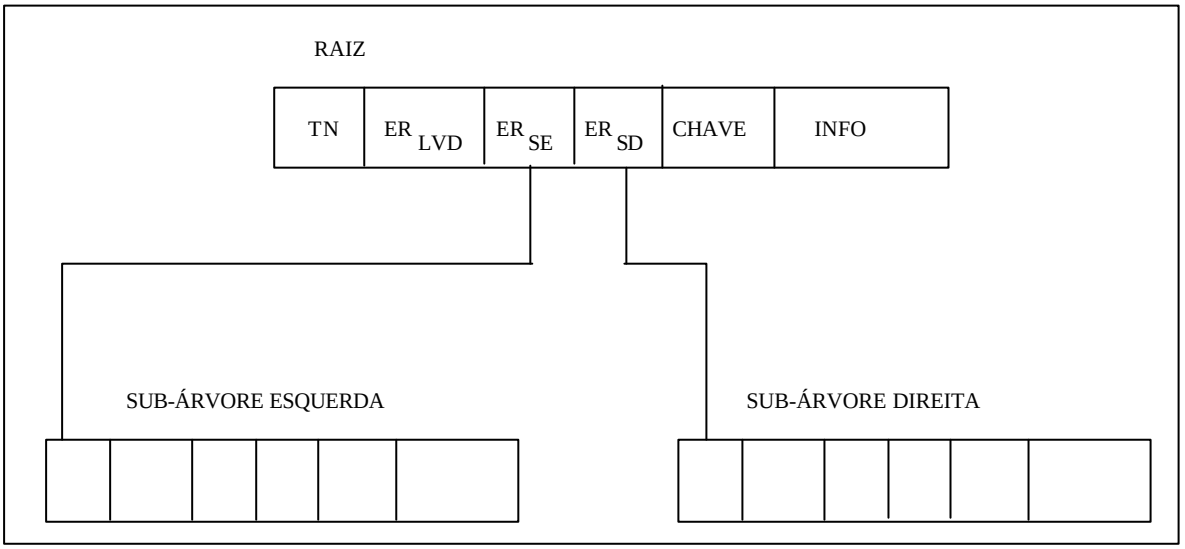
A variável NODO é a implementação do conceito de nodo descrito acima e pode ser melhor compreendido a partir da tupla:

$< T_N, ER_{LVD}, ER_{SE}, ER_{SD}$, CHAVE, INFO > , onde:

T_N é o tamanho de memória (em bytes) ocupado pelo nodo;
ER_{LVD} é o Endereço Relativo da Lista de Vagas Disponíveis;
ER_{SE} é o Endereço Relativo da sub-árvore da esquerda do nodo;
ER_{SD} é o Endereço Relativo da sub-árvore da direita do nodo;
CHAVE é a variável de acesso que identifica o nodo;
INFO é a informação armazenada.

A figura 21 ilustra a estrutura descrita acima. A variável RAIZ irá conter o Endereço Base de localização na memória real do nodo Raiz. A variável CHAVE é do tipo string de tamanho variável. A variável INFO irá conter a estrutura interna das classes dos objetos de gerenciamento implementados como subclasses de Árvore-B Genérica.

FIGURA 21 - Estrutura da Classe Árvore-B Genérica.



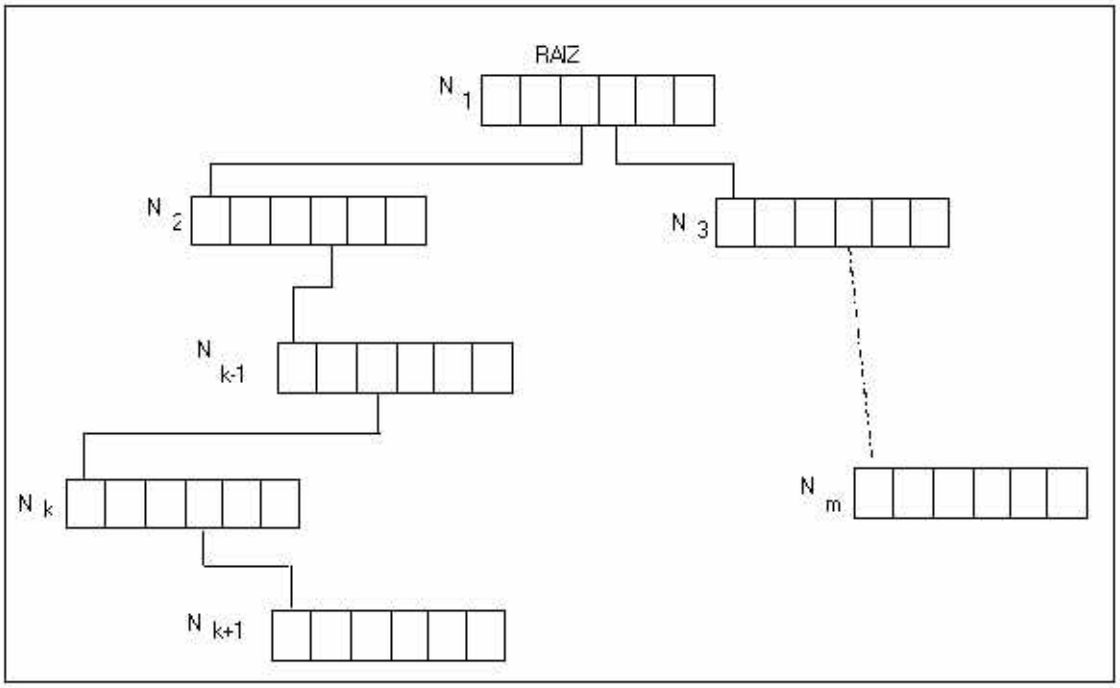
REUTILIZAÇÃO DE NODOS DA ÁRVORE-B - LISTA DE VAGAS DISPONÍVEIS.

A Lista de Vagas Disponíveis ou LVD é formada a partir do nodo corresponde a RAIZ da árvore-B. Quando não existe a LVD, o valor contido na variável ER_{LVD} da Raiz possuirá o valor negativo representado por -1, caso contrário o valor contido nesta variável irá corresponder ao primeiro elemento da LVD. Para tornar isto mais compreensível pode-se imaginar a seguinte situação:

Seja $\mathbf{n_k}$ um nodo qualquer de uma árvore-B

$A= \{ n_1 , n_2 , ... , n_{k-1} , n_k , n_{k+1} , ... , n_m \}$ representada pela figura 22.

FIGURA 22 - EXEMPLO DE UMA ÁRVORE-B GENÉRICA



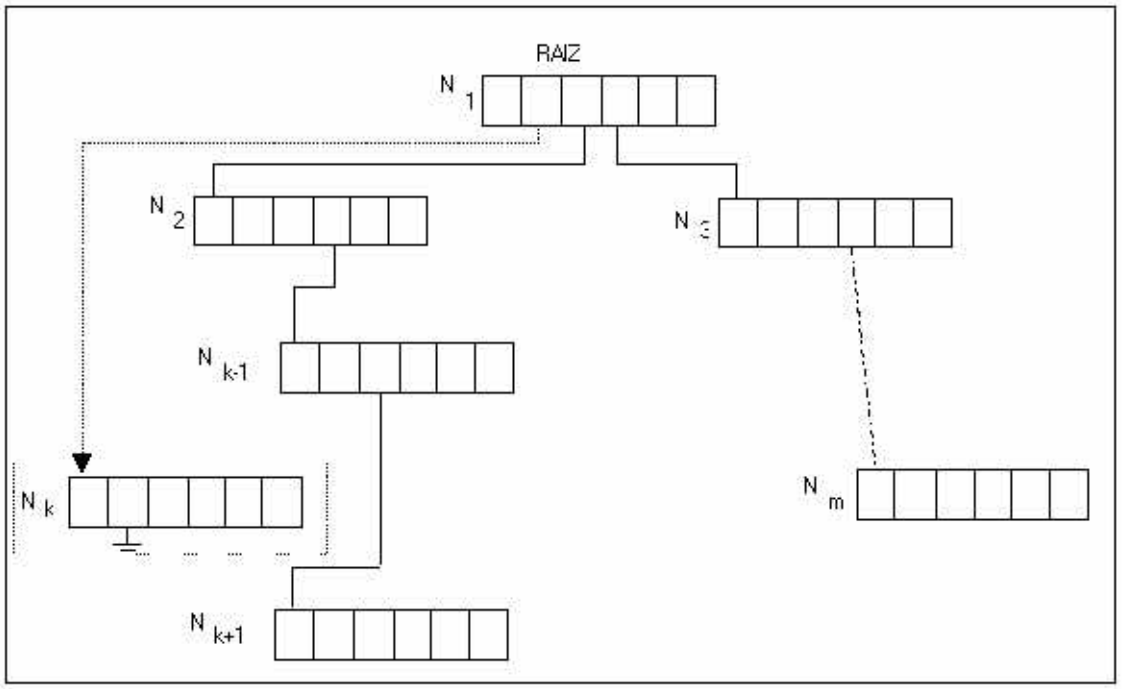
Ao ser remover logicamente o nodo $\mathbf{n_k}$ ocorre os seguintes eventos:

(1) O conteúdo da variável ER_{SE} do nodo pai de $\mathbf{n_k}$ (n_{k-1}) irá conter o valor correspondente ao conteúdo da variável ER_{SD} de $\mathbf{n_k}$.

(2) O Endereço Relativo do nodo $\mathbf{n_k}$ será atribuído à variável ER_{LVD} do nodo Raiz (n_1) iniciando assim a LVD. A figura 23 ilustra a remoção lógica descrita acima.

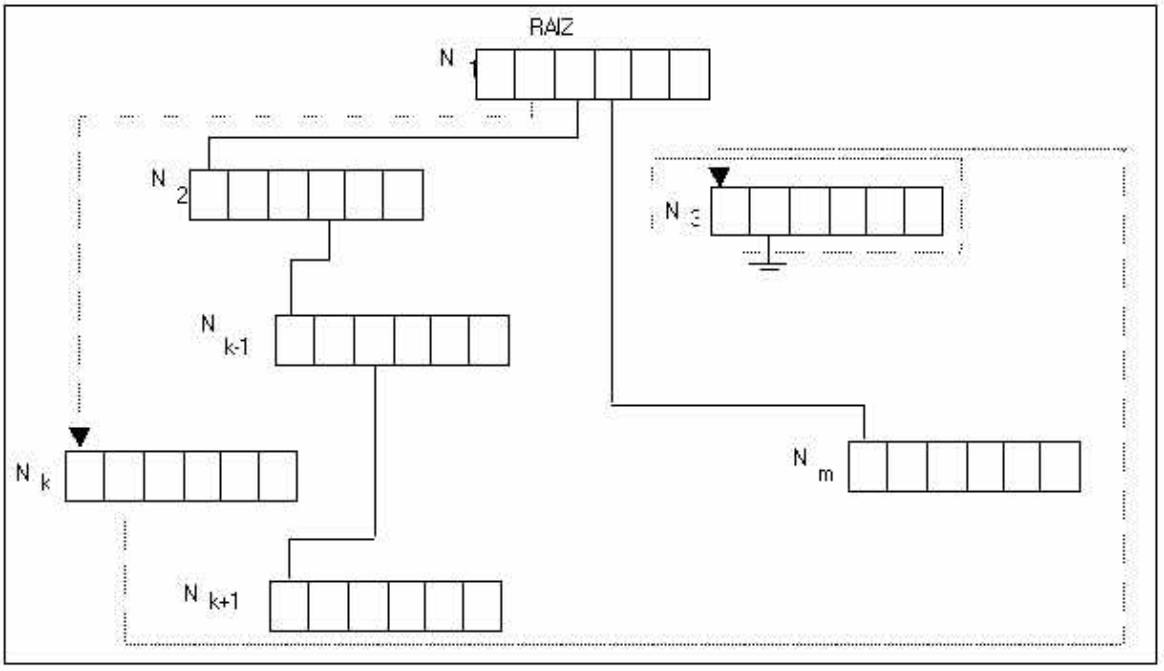
Na Lista de Vagas Disponíveis a inserção dos elementos é realizada sempre no final da lista. O final de uma LVD é identificado pelo conteúdo da variável ELVD que no caso conterá o valor -1 já que zero (0) corresponde ao Endereço Relativo do primeiro nodo genérico da árvore-B. Sendo assim torna-se necessário efetuar uma pesquisa para localizar o último elemento da LVD. O ponto de partida para efetuar a busca é sempre o nodo genérico Raiz da Árvore-B. Ao ser localizado, a inserção se realiza alterando o conteúdo da variável ELVD deste último elemento que conterá o Endereço Relativo do novo elemento da LVD. A variável ELVD deste novo elemento conterá a partir deste momento o valor negativo (-1) indicando assim o novo último elemento da LVD.

FIGURA 23 - REMOÇÃO LÓGICA DO NODO Nk DE UMA ÁRVORE-B



A remoção de um elemento da LVD é feito também a partir de uma pesquisa do nodo Raiz. Sempre o último elemento da LVD corresponde à vaga disponível para reutilização. Ao ser localizado, o Endereço Relativo deste último elemento da LVD é fornecido ao método da classe Árvore-B, e ocorre então a atualização do antepenúltimo elemento da LVD cuja variável ER_{LVD} conterá o valor negativo (-1). As figuras 24 e 25 ilustram uma inserção e uma remoção na LVD.

FIGURA 24 - REMOÇÃO LÓGICA DO NODO N3 DE UMA ÁRVORE-B E SUA INSERÇÃO NA LVD APÓS O NODO N8.



OPERAÇÕES INTERNAS DA CLASSE ÁRVORE-B GENÉRICA

Para permitir a manipulação de instâncias desta classe foram implementados algumas operações internas, ou métodos, que serão descritos a seguir.

CRIAÇÃO - Esta operação interna tem por objetivo inicializar a instância de Árvore-B criando o nodo Raiz. A criação no nodo Raiz é realizada após o envio da mensagem ALOCAÇÃO ao Objeto Memória solicitando a região necessária.

INSERÇÃO - Esta operação interna efetua a inclusão de um novo nodo na instância de Árvore-B totalmente armazenada em memória real, a partir da respectiva Chave de Acesso fornecida via parâmetro. Nesta operação foi implementada a filosofia de balanceamento descrita no início desta seção. A Lista de Vagas Disponíveis é pesquisada para obtenção de um Endereço Relativo e assim permitir o reuso do espaço de armazenamento.

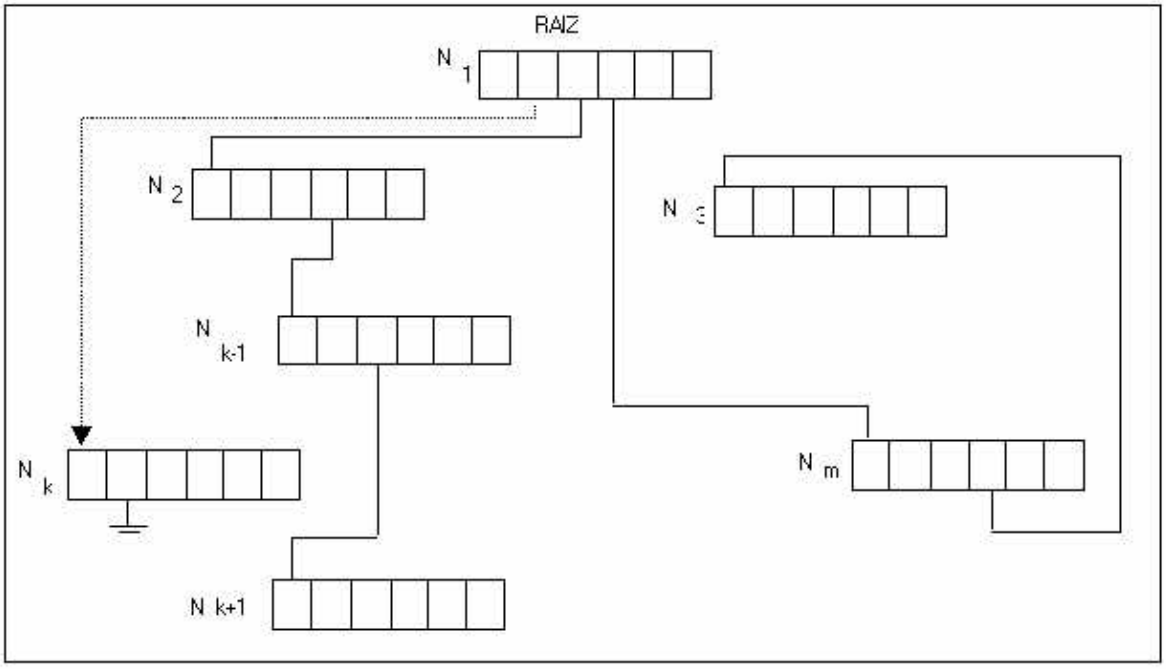
REMOÇÃO - Esta operação interna, implementada com a filosofia de balanceamento descrita no início desta seção, efetua a remoção lógica de um nodo da Árvore-B totalmente armazenada em memória real, inserindo na LVD seu Endereço Relativo.

PESQUISA - Este método tem por finalidade efetuar uma pesquisa na instância de Árvore-B totalmente armazenada em memória real, em busca do Endereço Relativo de um nodo identificado pela Chave de Acesso informada via parâmetro.

INSERÇÃO-LVD - Esta operação interna efetua a inclusão na Lista de Vagas Disponíveis do Endereço Relativo de um nodo removido logicamente da instância de Árvore-B totalmente armazenada em memória real.

REMOÇÃO-LVD - Esta operação interna efetua a remoção de um elemento da LVD da Árvore-B totalmente armazenada em memória real, que será reutilizado durante a execução do método Inserção.

FIGURA 25 - INSERÇAO DE UM NOVO NODO COM O REAPROVEITAMENTO DO NODO N$_3$ REMOVIDO DA LVD.



PESQUISA-LVD - Este método tem por finalidade efetuar uma pesquisa na LVD de uma instância de Árvore-B totalmente armazenada em memória real, em busca do Endereço Relativo do último nodo removido logicamente.

OBTEM-CHAVE - Este método obtém o valor correspondente ao conteúdo da variável CHAVE de um determinado nodo da Árvore-B.

OBTEM TAMANHO - Este método obtém o valor correspondente ao conteúdo da variável T_N de

um determinado nodo da Árvore-B.

CARGA - Esta operação interna tem por finalidade a transferência de uma instância identificada pelo Endereço Relativo Virtual da Árvore-B da Área de Armazenamento Virtual para a memória real.

CARGA PARCIAL - Esta operação interna tem por finalidade a transferência de um determinado nodo identificado pelo Endereço Relativo Virtual e sua Posição na instância da Árvore-B, da Área de Armazenamento Virtual para a memória real.

SOBRECARGA - Esta operação interna tem por finalidade a transferência de um determinado nodo de uma instância da Árvore-B identificada pelo Endereço Relativo Virtual da Área de Armazenamento Virtual para uma região da memória real já alocada e inicialmente ocupada por outro nodo.

SALVA - Esta operação interna efetua a transferência de uma instância da Árvore-B armazenada em memória real para a respectiva Área de Armazenamento Virtual.

SALVA PARCIAL - Esta operação efetua a transferência de um determinado nodo localizado na memória real para o final do Bloco Virtual onde se encontra armazenada a instância da Árvore-B na respectiva Área de Armazenamento Virtual cuja localização é dada pelo Endereço Relativo Virtual.

SOBRESALVA - Esta operação efetua a transferência de um determinado nodo localizado na memória real para a localização dada pelo Endereço Relativo Virtual e sua Posição na respectiva Área de Armazenamento Virtual já previamente ocupada pelo nodo da instância da Árvore-B.

INSERÇÃO PARCIAL - Esta operação interna efetua a inclusão de um novo nodo na instância de Árvore-B parcialmente armazenada em memória real, a partir da respectiva Chave de Acesso fornecida via parâmetro. Nesta operação foi também implementada a filosofia de balanceamento descrita no início desta seção. A Lista de Vagas Disponíveis é pesquisada para obtenção de um Endereço Relativo e assim permitir o reuso do espaço de armazenamento.

REMOÇÃO PARCIAL - Esta operação interna, implementada com a filosofia de balanceamento , efetua a remoção lógica de um nodo da Árvore-B parcialmente armazenada em memória real, inserindo na LVD seu Endereço Relativo.

PESQUISA PARCIAL - Este método realiza uma pesquisa na instância de Árvore-B parcialmente armazenada em memória real, em busca do Endereço Relativo de um nodo identificado pela Chave de Acesso informada via parâmetro.

INSERÇÃO-LVD PARCIAL - Esta operação interna tem por objetivo efetuar a inclusão na Lista de Vagas Disponíveis do Endereço Relativo de um nodo removido logicamente da instância de Árvore-B que está parcialmente armazenada em memória real.

REMOÇÃO-LVD PARCIAL - Esta operação interna tem por finalidade efetua a remoção de um elemento da LVD da Árvore-B que se encontra parcialmente armazenada em memória real, o qual será reutilizado durante a execução do método Inserção Parcial.

PESQUISA-LVD PARCIAL - Este método efetua uma pesquisa na LVD de uma instância de Árvore-B que se encontra parcialmente armazenada em memória real, em busca do Endereço Relativo do último nodo removido logicamente.

3.2.2.2 OBJETOS DE GERENCIAMENTO DO GOLGO

O GOLGO foi criado com o objetivo de gerenciar objetos do Ambiente Poesis através de uma estrutura em forma de grafo orientado composta por um conjunto de objetos que tornam possível a implementação do enfoque descrito no capítulo 2. Os objetos que permitem gerenciar o Ambiente Poesis denominados Objetos de Gerenciamento, desempenham funções específicas no processamento do sistema. Apesar de serem, do ponto de vista de implementação independentes, característica obtida quando se desenvolve no enfoque orientado para objetos, existe uma interdependência no funcionamento global do ambiente. O conjunto de objetos de gerenciamento é composto pelas seguintes classes:

- Nó de Classe;
- Nó de Composição;
- Nó de Instância;
- Link de Navegação;
- Link de Processos;
- Índice de Operações Internas;

- Link de Composição;
- Índice do Dicionário de Objetos;
- Dicionário de Objetos;
- Versões;
- Restrições;
- Definição Visual.

Nesta seção serão especificadas as características estruturais e comportamentais destes objetos que tornam possível a manipulação dos Objetos do Sistema durante o processamento no Ambiente Poesis.

A estrutura das respectivas classes de objetos aqui descritos encontra-se especificada no apêndice I.

3.2.2.2.1 **CLASSE: NÓ DE CLASSE**

A Classe de Objetos denominada Nó de Classe foi criada para implementar no enfoque orientado para objetos o conceito de Nó de Classe. O Nó de Classe é um Objeto de Gerenciamento usado para representar as Classes e Subclasses dos Objetos do Sistema (Ver capítulo 4) permitindo que estes objetos sejam manipulados segundo a filosofia criada para o GOLGO.

O Nó de Classe é considerado o principal objeto de gerenciamento por controlar o conjunto de informações que tornam possível o acesso às instâncias dos demais objetos de gerenciamento. Este conjunto de informações consiste dos Endereço Relativos Virtuais das instâncias dos outros objetos usados para implementar as características do GOLGO referentes aos conceitos de links verticais, links horizontais, definição visual, restrições, dicionário, etc.

A classe Nó de Classe foi implementada como subclasse de Lista Fixa Genérica herdando desta maneira suas características estruturais e comportamentais. Esta hierarquia de classe obtida a partir do processo de generalização efetuado sobre os objetos de gerenciamento permitiu uma rápida implementação deste objeto. A figura 14 (seção 3.2.2.1) permite visualizar a hierarquia de classes implementada no GOLGO.

No Ambiente o objeto Nó de Classe utiliza algumas informações que permitem sua manipulação e seu relacionamento com o Objeto Memória. Estas informações são as seguintes:

Identificador do Objeto (IDO): 00 (zero);
Identificação da AAV (IAAV): NCLASSE.AAV;
Identificação da LEV: NCLASSE.LEV.

DESCRIÇÃO DO FUNCIONAMENTO DO NÓ DE CLASSE

A forma de implementação do Nó de Classe na hierarquia descrita anteriormente, determinou a manipulação de suas instâncias. Esta manipulação consiste basicamente em duas modalidades de funcionamento: Parcial e Total. A modalidade Parcial consiste em alocar e manipular as instâncias a medida que for sendo necessário o acesso a elas. A modalidade Total consiste em alocar todas as instâncias em Memória Real. Esta flexibilização do uso das instâncias do objeto Nó de Classe justifica-se pela necessidade de otimizar o uso da memória e pelo fato de que nem sempre todas as instâncias deste objeto deverão ser alocadas quando do uso normal do Ambiente.

O funcionamento deste objeto na modalidade Parcial em geral compreende basicamente a realização das seguintes ações:

(1) Transferência da instância (Carga) da memória persistente para uma região previamente alocada na memória real;

(2) Manipulação da instância através do envio de mensagens invocando as operações internas disponíveis (operações da classe e operações herdadas);

(3) Transferência da instância (Salva) da região alocada na memória real para a memória persistente;

A execução das ações descritas acima, é realizada sempre que for necessário o acesso a uma determinada instância, ou seja, quando o usuário através da "navegação" pelo sistema, proporcionado pela interface e o objeto Link de Navegação, deseja acessar um determinado objeto do sistema. O acesso ao Nó de Classe que representa o objeto do sistema permitirá o acesso a todas as outras instâncias de objetos de gerenciamento associadas, que tornam possível ao usuário interagir com o objeto do sistema desejado.

O funcionamento na modalidade Total compreende a realização das seguintes ações:

(1) Transferência de todas as instâncias de Nó de Classe (Carga Total) da memória persistente para uma região previamente alocada na memória real;

(2) Manipulação das instância através do envio de mensagens invocando as operações internas disponíveis (operações da classe e operações herdadas);

(3) Transferência das instâncias (Salva Total) da região alocada na memória real para a memória persistente;

As ações descritas acima são realizadas quando é necessário o acesso a todas as instância em Memória Real. Em geral ocorre no momento de execução da Compactação Virtual. O acesso ao Nó de Classe que representa do objeto do sistema, permitirá a realização desta operação que tem por objetivo a eliminação dos espaços vazios provenientes da remoção de instâncias dos demais objetos de gerenciamento.

A Compactação Virtual é realizada sempre usando o objeto Nó de Classe, porque todas as instâncias de qualquer objeto de gerenciamento podem ser recuperadas, já que este objeto possui apontadores para suas respectivas localizações em memória persistente. A Compactação Virtual é feita de forma automática, a partir da constatação pelo sistema do atingimento do Nível Crítico por parte de uma determinada Área de Armazenamento Virtual. Em geral, consiste na execução das seguintes ações:

(1) Alocação Total das instâncias do objeto Nó de Classe;

(2) Varredura de todas as instâncias ativas de Nó de Classe visando obter o conteúdo da variável Endereço Relativo Virtual do objeto de gerenciamento cuja Área de Armazenamento Virtual está sendo compactada;

(3) Carga das instâncias do objeto de gerenciamento em Memória Real a partir da varredura;

(4) Salvamento ou transferência das instâncias do objeto de gerenciamento para seus novos Endereços Relativos Virtuais na memória persistente;

(5) Atualização para cada instância salva dos respectivos novos Endereços Relativos Virtuais na instância de Nó de Classe correspondente;

(6) Remoção da instância de Lista de Espaços Vazios da memória persistente.

3.2.2.2.2 **CLASSE: NÓ DE COMPOSIÇÃO**

A classe de objetos denominada Nó de Composição foi criada especialmente para implementar o conceito de Nó de Composição descrito no capítulo 2 deste trabalho. O Nó de Composição é um Objeto de Gerenciamento usado para representar instâncias especiais das Classes definidas para os Objetos do Sistema especificados para o Modelo E/D [BAND89] criado para o Ambiente Poesis (Ver capítulo 4). Esta forma de implementação permitiu que estes objetos fossem manipulados segundo a filosofia criada para o GOLGO.

O Nó de Composição é considerado como sendo uma implementação resumida do Nó de Classe sendo seu principal objetivo controlar o conjunto de informações que tornam possível o acesso às instâncias dos objetos de gerenciamento utilizados na implementação do Modelo E/D. Este conjunto de informações consiste dos Endereço Relativos Virtuais das diversas instâncias dos objetos de gerenciamento que tornam possível a utilização dos conceitos de composição de objetos definido do capítulo 2.

A classe Nó de Composição foi também implementada como subclasse de Lista Fixa Genérica herdando desta maneira suas características estruturais e comportamentais. A hierarquia de classe visualizada na figura 14 (seção 3.2.2.1) foi obtida pelo mesmo processo descrito para o objeto Nó de Classe permitindo também uma rápida implementação deste objeto.

No Ambiente o objeto Nó de Composição utiliza informações que permitem sua manipulação e seu relacionamento com o Objeto Memória. Estas informações são as seguintes:

Identificador do Objeto (IDO): 01;
Identificação da AAV (IAAV): NCOMPOS.AAV;
Identificação da LEV: NCOMPOS.LEV.

DESCRIÇÃO DO FUNCIONAMENTO DO NÓ DE COMPOSIÇÃO

O objeto Nó de Composição é usado no GOLGO para representar as instâncias dos objetos componentes do Modelo E/D. Estas instâncias estão associadas a uma Aplicação desenvolvida pelo usuário a partir deste Modelo Conceitual implementado no Ambiente. Desta forma, as instâncias do objeto Nó de Composição são agrupadas em blocos cuja característica comum é o fato de pertencerem a uma Aplicação Modelada. A manipulação das instâncias do Nó de Composição é realizada sempre na modalidade Total, ou seja, todas as instâncias do bloco pertencentes a uma determinada aplicação são alocadas em Memória Real. A localização de cada instância na Memória Real é feita pelo Endereço Relativo ou Posição Relativa ocupada pela instância a partir do Endereço Base de Localização..

O funcionamento do Nó de Composição na modalidade Total compreende a realização das seguintes ações:

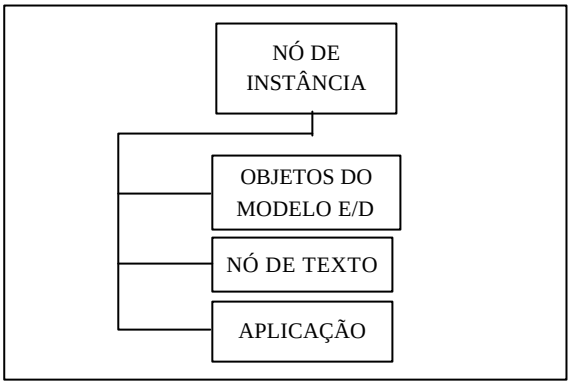
(1) Transferência de todas as instâncias de Nó de Composição (Carga Total) da memória persistente para uma região previamente alocada na memória real;
(2) Manipulação das instância através do envio de mensagens invocando as operações internas disponíveis (operações da classe e operações herdadas);
(3) Transferência das instâncias (Salva Total) da região alocada na memória real para a memória persistente;

3.2.2.2.3 **CLASSE: NÓ DE INSTÂNCIA**

Nó de Instância é classe de objetos criada para implementar no enfoque orientado para objetos o conceito de Nó de Instância. O Nó de Instância é um Objeto de Gerenciamento usado para representar as todas as instâncias dos Objetos do Sistema permitindo que estes objetos sejam manipulados segundo a filosofia criada para o GOLGO.

Para permitir que a classe Nó de Instância fosse capaz de manipular as instâncias dos objetos do sistema foi necessário a sua implementação como uma super-classe cujas subclasses corresponderiam aos objetos do sistema permitindo assim a herança de suas características estruturais e comportamentais. Esta hierarquia de classe pode ser visualizada na figura 26.

FIGURA 26 - HIERARQUIA DA CLASSE NÓ DE INSTÂNCIA



No Ambiente o objeto Nó de Instância utiliza algumas informações que permitem sua manipulação e seu relacionamento com o Objeto Memória. Estas informações são as seguintes:

Identificador do Objeto (IDO): 11;
Identificação da AAV (IAAV): NINSTANC.AAV;
Identificação da LEV: NINSTANC.LEV.

DESCRIÇÃO DO FUNCIONAMENTO DO NÓ DE INSTÂNCIA

As instâncias do objeto Nó de Instância são de tipos genéricos e portanto manipuladas como um conjunto de bytes armazenado em Memória Real. A manipulação lógica irá depender de cada subclasse correspondente ao vários tipos de objetos do sistema.

O funcionamento deste objeto ocorre sempre na modalidade Total e consiste basicamente na realização das seguintes ações:

(1) Transferência da instância (Carga) da memória persistente para uma região previamente alocada na Memória Real;
(2) Manipulação da instância através do envio de mensagens invocando as operações internas disponíveis (operações da classe e operações herdadas);
(3) Transferência da instância (Salva) da região alocada na memória real para a memória persistente;

A execução das ações descritas acima, é realizada sempre que for necessário o acesso a uma determinada instância, ou seja, quando o usuário através da "navegação" pelo sistema desejar acessar uma instância de um determinado objeto do sistema.

3.2.2.2.4 **CLASSE: LINK DE NAVEGAÇÃO**

A classe de objetos Link de Navegação foi criada para implementar o conceito de Link de Navegação no GOLGO. O Link de Navegação é um Objeto de Gerenciamento usado para a navegação na estrutura em forma de grafo correspondente a estrutura do Ambiente Poesis, que permite o acesso às classes, subclasses e instâncias de objetos. Consiste basicamente numa lista encadeada cujos elementos contém informações que tornam possível localizar as instâncias de outros objetos tais como: Nó de Classe, Nó de Composição e Nó de Instância. O Link de Navegação permite também controlar o acesso à determinados objetos do sistema, criando assim um sistema de segurança descrito no capítulo 2.

Uma determinada instância de Link de Navegação está sempre associada a uma instância de um Nó de Classe, e pode manter ***n*** ligações com outras instâncias de Nó de Classe, Nó de Composição ou Nó de Instância. Na conceituação de grafos sua descrição corresponde ao conceito de arco cuja função é efetuar a ligação entre 2 nodos.

A classe Link de Navegação foi implementada no GOLGO como uma subclasse de Lista Variável Genérica. Suas características estruturais e comportamentais facilitou o desenvolvimento desta classe e a hierarquia decorrente permitiu que este objeto de gerenciamento herdasse as estruturas e operações necessárias à manipulação de suas instâncias. A figura 14 (seção 3.2.2.1) possibilita a visualização desta hierarquia de classes.

No Ambiente o objeto Link de Navegação utiliza algumas informações que tornam possível sua manipulação e seu relacionamento com o Objeto Memória. Estas informações são as seguintes:

Identificador do Objeto (IDO): 04;
Identificação da AAV (IAAV): LNAVEG.AAV;
Identificação da LEV: LNAVEG.LEV.

DESCRIÇÃO DO FUNCIONAMENTO DO LINK DE NAVEGAÇÃO

O objeto define os caminhos de acesso aos objetos do sistema permitindo ao usuário a localização do objeto ou instância desejada. O processo de navegação consiste basicamente na realização das seguintes ações:

(1) Obtenção do Endereço Relativo Virtual da instância de Link de Navegação associada a um determinado Nó de Classe. Esta operação é realizada pelo objeto Nó de Classe.

(2) Transferência da instância (Carga) da memória persistente para uma região previamente alocada na memória real.

(3) Manipulação da instância através do envio de mensagens invocando as operações internas disponíveis (operações da classe e operações herdadas) visando deixar disponível os caminhos a serem escolhidos.

(4) Transferência da instância (Salva) da região alocada na memória real para a memória persistente;

Durante a execução da ação descrita no item (3) poderá ocorrer uma expansão ou uma diminuição do tamanho da instância do objeto Link de Navegação, caso seja incluído ou removido algum elemento Componente. Na execução da ação (4) ocorre uma operação interna herdada da super-classe Lista Genérica, que corresponde a Transferência Lógica descrita na seção 3.2.2.1.1. A Transferência Lógica visa compactar a instância por ocasião de seu armazenamento na memória persistente.

3.2.2.2.5 **CLASSE: LINK DE PROCESSOS**

A classe Link de Processos foi implementada a partir do conceito de Link de Processos criado para o GOLGO. A semântica deste objeto compreende a execução de procedimentos não implementáveis a nível de operação interna de classes. São rotinas que manipulam diversas classes de objetos do sistema para execução de procedimentos especiais entre estas classes. Alguns Links Horizontais tais como: ***Modelagem***, que corresponde ao processo de criação dos objetos componentes de uma determinada aplicação através da manipulação dos objetos implementados para o Modelo E/D; ***Transformação***, que efetua a geração de uma aplicação no Modelo Relacional a partir da Aplicação modelada pelo Modelo E/D. Estes "procedimentos especiais" serão especificados no capítulo 4.

O objeto de Gerenciamento Link de Processos consiste basicamente numa lista encadeada cujos elementos contém informações que tornam possível a ativação dos procedimentos desejados. O Link de Processos permite também controlar o acesso à determinados procedimentos especiais do sistema, criando assim um sistema de segurança descrito no capítulo 2.

Uma determinada instância de Link de Processos está sempre associada a uma instância de um Nó de Classe, e pode manter ***n*** ocorrências de procedimentos especiais associados.

A classe Link de Processos foi implementada como uma subclasse de Lista Variável Genérica. Suas características estruturais e comportamentais determinadas a partir desta hierarquia de classes decorrente permitiu que este objeto de gerenciamento herdasse as estruturas e operações necessárias à manipulação de suas instâncias. A figura 14 (seção 3.2.2.1) possibilita a visualização desta hierarquia de classes.

No Ambiente o objeto Link de Processos utiliza algumas informações que permitem sua manipulação e relacionamento com o Objeto Memória. Estas informações são as seguintes:

Identificador do Objeto (IDO): 05;
Identificação da AAV (IAAV): LPROCES.AAV;
Identificação da LEV: LPROCES.LEV.

DESCRIÇÃO DO FUNCIONAMENTO DO LINK DE PROCESSOS

A utilização do objeto Link de Processos ocorre quando o usuário através da Interface decide executar um determinado procedimento especial no Ambiente Poesis. Este objeto a partir da escolha realizada pelo usuário informa ao sistema o Índice correspondente no array de apontadores, à função que deverá ser executada. A manipulação de uma instância de Link de Processos consiste basicamente na realização das seguintes ações:

(1) Obtenção do Endereço Relativo Virtual da instância de Link de Processos associada a um determinado Nó de Classe. Esta operação é realizada pelo objeto Nó de Classe.

(2) Transferência da instância (Carga) da memória persistente para uma região previamente alocada na memória real.

(3) Manipulação da instância através do envio de mensagens invocando as operações internas disponíveis (operações da classe e operações herdadas) visando deixar disponível os procedimentos especiais existentes no Ambiente, ou efetuar a manutenção da instância através da inclusão de novos elementos.

(4) Transferência da instância (Salva) da região alocada na memória real para a memória persistente;

Durante a execução da ação descrita no item (3) poderá ocorrer uma expansão ou uma diminuição do tamanho da instância do objeto Link de Processos, caso seja incluído ou removido algum elemento

Componente. Na execução da ação (4) ocorre uma operação interna herdada da super-classe Lista Genérica, que corresponde a Transferência Lógica descrita na seção 3.2.2.1.1. A Transferência Lógica visa compactar a instância por ocasião de seu armazenamento na memória persistente.

3.2.2.2.6 **CLASSE: ÍNDICE DE OPERAÇÕES INTERNAS**

O conceito de Índice de Operações Internas descrito no capítulo 2 consiste de associar a um determinado Nó de Classe, que representa uma classe de objetos, o seu correspondente conjunto dos métodos implementados com a função de manipulá-la. A classe Índice de Operações Internas foi implementada com esta finalidade. O objetivo desta classe é permitir que o usuário tenha acesso aos métodos implementados para um determinado objeto do sistema (ver capítulo 4) visando executá-los.

O objeto Índice de Operações Internas consiste de uma lista encadeada cujos elementos contém informações sobre os métodos disponíveis para execução de cada objeto do sistema. O Índice de Operações Internas permite também controlar o acesso à determinados métodos, criando assim um sistema de segurança a nível de métodos de objetos do sistema conforme descrito no capítulo 2. Uma determinada instância de Índice de Operações Internas está sempre associada a uma instância de um Nó de Classe, e pode manter ***n*** ocorrências de métodos, ou operações internas associadas.

A classe Índice de Operações Internas foi implementada como uma subclasse de Lista Variável Genérica (ver figura 14 - seção 3.2.2.1), ou seja, suas características estruturais e comportamentais foram determinadas a partir desta hierarquia de classes, o que permitiu a este objeto de gerenciamento a herança das estruturas e operações necessárias à manipulação de suas instâncias.

O objeto Índice de Operações Internas utiliza algumas informações que permitem sua manipulação e relacionamento com o Objeto Memória. Estas informações são as seguintes:

Identificador do Objeto (IDO): 06;
Identificação da AAV (IAAV): INDOPER.AAV;
Identificação da LEV: INDOPER.LEV.

DESCRIÇÃO DO FUNCIONAMENTO DO ÍNDICE DE OPERAÇÕES INTERNAS

A utilização do objeto Índice de Operações Internas ocorre quando o usuário através da Interface decide executar um determinado método de um Objeto do Sistema no Ambiente Poesis. O usuário ao acessar o Objeto do Sistema desejado, terá disponível através de sua Interface todos os métodos a que tem acesso. Após a escolha do método realizada pelo usuário o objeto Índice de Operações Internas informa ao sistema o Índice correspondente, no array de apontadores, à operação interna que deverá ser executada. A manipulação de uma instância de Índice de Operações Internas consiste basicamente na realização das seguintes ações:

(1) Obtenção do Endereço Relativo Virtual da instância de Índice de Operações Internas que está associada a um determinado Nó de Classe. Esta operação é executada pelo objeto Nó de Classe.

(2) Transferência da instância (Carga) do objeto Índice de Operações Internas, da memória persistente para uma região previamente alocada na Memória Real.

(3) Manipulação desta instância através do envio de mensagens invocando as operações internas disponíveis (operações da classe e operações herdadas) visando deixar disponível os métodos referentes a um determinado Objeto do Sistema existente no Ambiente, ou efetuar a manutenção da instância através da inclusão de novos elementos ou remoção de métodos existentes.

(4) Transferência da instância (Salva) da região alocada na memória real para a memória persistente;

Durante a execução da ação descrita no item (3) poderá ocorrer uma expansão ou uma diminuição do tamanho da instância do objeto Índice de Operações Internas, caso seja incluído ou removido algum elemento Componente. Na execução da ação (4) ocorre a execução de uma operação interna herdada da super-classe Lista Genérica, que corresponde a Transferência Lógica descrita na seção 3.2.2.1.1. A Transferência Lógica visa compactar a instância por ocasião de seu armazenamento na memória persistente.

3.2.2.2.7 **CLASSE: LINK DE COMPOSIÇÃO**

Os Links de Composição foram criados para permitir a manipulação do processo de composição de objetos usado pelo Modelo E/D [BAND89]. Os Links de Composição tornam explícitas as ligações existentes entre o Objeto Composto e os respectivos Objetos Componentes. O objetivo deste enfoque é dotar o sistema de um mecanismo flexível de associação entre objetos criados para representação do modelo conceitual de dados. Com essa finalidade a classe de objetos Link de Composição foi criada para implementar o conceito de Link de Composição no GOLGO conforme descrito no capítulo 2.

Uma determinada instância de Link de Composição está associada a uma instância de um Nó de Composição ou de um Nó de Classe, e pode manter ***n*** ligações (Links) com outras instâncias de Nó de Classe ou Nó de Composição. O objeto Link de Composição mantém desta forma uma lista de referências que partem dos objetos compostos (representados por instâncias de Nó de Composição, ou Nó de Classe) para os objetos componentes (também representados por instâncias Nó de Composição, ou Nó de Classe). Para permitir o controle do uso do processo de composição foram criadas dois tipos de instâncias do objeto Link de Composição: Link de Composição Normal e Link de Composição Invertida. Os Links de Composição Normal são aqueles que representam o conceito descrito anteriormente, ou seja, são Links de Composição que associam o Objeto Composto aos Objetos Componentes. Os Links de Composição Invertida efetuam a associação entre cada Objeto Componente e os Objetos Compostos nos quais este objeto participa no processo de composição. Os Links de Composição Invertida mantêm um conjunto de ligações (links) que permitem localizar todos os Objetos Compostos que utilizam um determinado Objeto Componente em sua composição.

A classe Link de Composição foi implementada no GOLGO como uma subclasse de Lista Variável Genérica. Suas características estruturais e comportamentais se assemelhavam às características desta super-classe e consequentemente passou a fazer parte da hierarquia de Classe (ver figura 14 - seção 3.2.2.1) o que permitiu ao objeto de gerenciamento Links de Composição a herança das estruturas e operações necessárias à manipulação de suas instâncias.

No Ambiente o objeto Link de Composição utiliza algumas informações necessárias ao relacionamento com o Objeto Memória e à manipulação de suas instância. Estas informações são as seguintes:

Identificador do Objeto (IDO): 08;
Identificação da AAV (IAAV): LCOMPOS.AAV;
Identificação da LEV: LCOMPOS.LEV.

DESCRIÇÃO DO FUNCIONAMENTO DO LINK DE COMPOSIÇÃO

As instâncias do objeto de gerenciamento Link de Composição são manipuladas durante a execução do ***procedimento especial*** do Ambiente Poesis denominado **Modelagem**. Este procedimento especial executado a partir da ativação do Link de Processos correspondente, efetua a criação de instâncias dos objetos do Modelo E/D e a composição de objetos através da criação de instâncias de Link de Composição. A manipulação de uma instância de Link de Composição consiste da realização das seguintes ações:

(1) Obtenção do Endereço Relativo Virtual da instância de Link de Composição que está associada a

um determinado Nó de Classe ou Nó de Composição. Esta operação é executada conforme o caso, pelo objeto Nó de Classe ou pelo objeto Nó de Composição.

(2) Transferência da instância (Carga) do objeto Link de Composição, da memória persistente para uma região previamente alocada na memória real.

(3) Manipulação desta instância através do envio de mensagens invocando as operações internas disponíveis (operações da classe e operações herdadas).

(4) Transferência da instância (Salva) da região alocada na memória real para a memória persistente;

Durante a execução da ação descrita no item (3) poderá ocorrer uma expansão ou uma diminuição do tamanho da instância do objeto Link de Composição, caso seja incluído ou removido algum elemento Componente. Na execução da ação (4) ocorre a execução de uma operação interna herdada da super-classe Lista Genérica, que corresponde a Transferência Lógica descrita na seção 3.2.2.1.1. A Transferência Lógica visa compactar a instância por ocasião de seu armazenamento na memória persistente.

3.2.2.2.8 **CLASSE: ÍNDICE DO DICIONÁRIO DE OBJETOS**

A Classe de Objetos denominada Índice do Dicionário de Objetos foi criada para otimizar o uso do Dicionário de Objetos do Ambiente Poesis. Esta otimização consiste basicamente em reduzir o tempo de pesquisa no Dicionário de Objetos, facilitando sua manutenção. A pesquisa ao Dicionário de Objetos é sempre realizada quando se deseja incluir, remover, ou simplesmente consultar informações referentes aos Objetos do Sistema.

O Índice do Dicionário de Objetos funciona em total integração com o objeto de gerenciamento Dicionário de Objetos. Seu objetivo principal é manter o controle das informações armazenadas pelo objeto Dicionário de Objetos visando garantir a integridade dos dados. Cada Objeto que compõe o Ambiente possui um identificador representado pelo nome do objeto e caracteriza-se por sua unicidade. Associado ao identificador do objeto existem informações destinadas a documentação e ajuda (Help) ao usuário. O objeto Índice do Dicionário de Objetos é o responsável pela manutenção da unicidade do identificador dos objetos sendo intermediário no acesso ao Dicionário de Objetos.

A classe Índice do Dicionário de Objetos foi implementada como subclasse de Árvore-B Genérica herdando suas características relativas a estrutura e funcionamento. Esta maneira de implementar resulta da observação do comportamento deste objeto que necessita de uma estrutura que permita uma busca rápida e manipulação flexível, peculiares à super-classe Árvore-B Genérica. A hierarquia de classe apresentada na figura 14, permitiu desta forma, uma rápida implementação deste objeto

Para tornar possível sua manipulação através do relacionamento com o Objeto Memória, a classe Índice do Dicionário de Objetos utiliza algumas informações especiais, tais como:

Identificador do Objeto (IDO): 10;
Identificação da AAV (IAAV): DOINDICE.AAV;
Identificação da LEV: DOINDICE.LEV.

DESCRIÇÃO DO FUNCIONAMENTO DO ÍNDICE DO DICIONÁRIO DE OBJETOS

O Índice do Dicionário de Objetos foi criado para permitir a manipulação das informações contidas no Dicionário de Objetos tendo em vista garantir a integridade dos dados. Sua estrutura em

forma de Árvore-B herdada pela implementação como subclasse de Árvore-B Genérica permite um acesso mais rápido as informações referentes a um determinado Objeto do Sistema. A super-classe Árvore-B Genérica dispõe de duas modalidades de processamento de suas instâncias: Parcial e Total. A modalidade Parcial consiste em alocar e manipular os nodos da Árvore-B a medida que for sendo necessário o acesso a eles. A modalidade Total consiste em alocar toda a instância da Árvore-B em Memória Real. Esta flexibilização do uso das instâncias do objeto Índice do Dicionário de Objetos, herdada da classe Árvore-B Genérica, justifica-se pela necessidade de otimizar o uso da memória devido ao fato de que uma instância deste objeto deverá crescer em proporções que dificultarão sua manipulação total durante o processamento normal do Ambiente.

O funcionamento deste objeto realiza-se quase sempre na modalidade Parcial que compreende basicamente a realização das seguintes ações:

(1) Transferência da parte da instância referente a um nodo(Carga Parcial) da memória persistente para uma região previamente alocada na memória real;

(2) Manipulação desta fração de instância através do envio de mensagens invocando as operações internas disponíveis (operações da classe e operações herdadas);

(3) Transferência da instância (Salva Parcial) da região alocada na memória real para a memória persistente;

A execução das ações descritas acima, realiza-se sempre que for necessário o acesso a uma determinada instância, ou seja, quando o algum objeto de gerenciamento enviar uma mensagem solicitando o acesso a um determinado nodo.

O funcionamento na modalidade Total compreende a realização das seguintes ações:

(1) Transferência de toda a instância de Índice do Dicionário de Objetos (Carga Total) da memória persistente para uma região previamente alocada na memória real;

(2) Manipulação da instância através do envio de mensagens invocando as operações internas disponíveis (operações da classe e operações herdadas);

(3) Transferência da instância (Salva Total) da região alocada na Memória Real para a memória persistente;

3.2.2.2.9 **CLASSE: DICIONÁRIO DE OBJETOS**

A classe Dicionário de Objetos foi criada visando implementar os conceitos de identificação e descrição de objetos no âmbito do Ambiente Poesis. O Dicionário de Objetos é um Objeto de Gerenciamento usado para armazenar as informações relativas aos Objetos do Sistema cujo objetivo é documentar para consulta e ajuda on-line. Funciona integrado ao objeto Índice do Dicionário de Objetos, descrito na seção anterior, cujo objetivo principal é manter a integridade das instâncias armazenadas no Dicionário do Ambiente.

A classe Dicionário de Objetos foi implementada na hierarquia de classes do GOLGO, como subclasse de Lista Fixa Genérica herdando desta forma suas características estruturais e comportamentais. Esta hierarquia de classe apresentada pela figura 14 (seção 3.2.2.1) foi obtida a partir do processo de especialização efetuado sobre os objetos de gerenciamento permitiu uma rápida implementação deste objeto.

O Objeto Memória se relaciona no Ambiente com o objeto Dicionário de Objetos através de algumas informações fornecidas como parâmetros em alguns métodos desta classe os quais permitem sua manipulação. Estas informações são as seguintes:

Identificador do Objeto (IDO): 02;
Identificação da AAV (IAAV): DOBASE.AAV;
Identificação da LEV: DOBASE.LEV.

DESCRIÇÃO DO FUNCIONAMENTO DO DICIONÁRIO DE OBJETOS

A classe Dicionário de Objetos é utilizada no Ambiente para inserir, alterar, remover ou consultar as informações relativas aos Objetos do Sistema. A maneira pela qual esta classe foi implementada, conforme descrito anteriormente, determinou a manipulação de suas instâncias que consiste basicamente em duas modalidades de funcionamento: Parcial e Total. A modalidade Parcial, muito usada para consulta ao Dicionário de Objetos, consiste em alocar e manipular as instâncias a medida que for sendo necessário o acesso a elas. A modalidade Total consiste em alocar todas as instâncias em Memória Real. Esta duas formas de manipular as instâncias do Dicionário de Objetos propiciando uma flexibilidade no uso deste objeto, justifica-se pela necessidade de otimização do uso da Memória Real que será compartilhada por todas as instâncias dos demais objetos de gerenciamento durante o processamento normal do Ambiente.

Apesar destas modalidades de manipulação total e parcial, usarem métodos diferentes, seu funcionamento é idêntico e consiste da execução das seguintes ações:

(1) Transferência da instância (Carga) da memória persistente para uma região previamente alocada na memória real;

(2) Manipulação da instância através do envio de mensagens invocando as operações internas disponíveis (operações da classe e operações herdadas);

(3) Transferência da instância (Salva) da região alocada na memória real para a memória persistente;

(4) Desalocação da memória ocupada pela instância.

A ação descrita no item (2) poderá ser realizada pelos métodos que obtêm o valor das variáveis, ou mesmo os que alteram estes valores.

3.2.2.2.10 **CLASSE: VERSÕES**

As aplicações criadas, denominadas de Aplicações Modeladas, segundo o Modelo E/D, precisam ser constantemente manipuladas por seus idealizadores, já que o processo de concepção, ou modelagem possui uma dinâmica que permite uma evolução destas aplicações. Tendo por objetivo dar suporte a evolução de Aplicações desenvolvidas no Ambiente Poesis, foi criada a Classe Versões. A classe Versões tem por finalidade básica manter o controle das versões de Aplicações Modeladas pelos usuários conforme descrito no capítulo 2.

Devido ao fato de que as características estruturais e comportamentais da Classe Versões se assemelham às características da classe Lista Variável Genérica, Versões foi implementada como subclasse desta super-classe, e consequentemente a classe Versões passou a fazer parte da hierarquia de Classes do GOLGO apresentada na figura 14. Assim, o objeto de gerenciamento Versões herdou as estruturas e operações necessárias à manipulação de suas instâncias, reduzindo o tempo de desenvolvimento desta classe.

Uma determinada instância do objeto Versões é uma lista associada a uma instância de um Nó de Classe representante de uma Aplicação Modelada (ver capítulo 4), cujos elementos mantêm informações referentes as diversas versões de aplicação implementadas pelo usuário. Este elementos que compõem a instância do objeto Versões possuem os Endereços Relativos Virtuais das instâncias de Nó de Classe referentes às versões. Em outras palavras, o objeto Versões mantém uma lista de referências que partem dos objetos identificadores da Aplicação Modelada (representados por

instâncias de Nó de Classe) para os objetos que representam as várias versões (também representados por instâncias de Nó de Classe).

No Ambiente o objeto Versões utiliza algumas informações necessárias ao relacionamento com o Objeto Memória e à manipulação de suas instância. Estas informações são as seguintes:

Identificador do Objeto (IDO): 08;
Identificação da AAV (IAAV): VERSOES.AAV;
Identificação da LEV: VERSOES.LEV.

DESCRIÇÃO DO FUNCIONAMENTO DO OBJETO VERSÕES

As instâncias do objeto de gerenciamento Versões são manipuladas em dois momentos do Ambiente:

- quando da criação de novas versões;
- durante a navegação para acesso a uma versão previamente criada.

Em cada um destes momentos a manipulação de uma instância de Versões consiste da realização das seguintes ações:

(1) Obtenção do Endereço Relativo Virtual da instância do objeto Versões que está associada a um determinado Nó de Classe. Esta operação é executada pelo objeto Nó de Classe.

(2) Transferência da instância (Carga) do objeto Versões, da memória persistente para uma região previamente alocada na memória real.

(3) Manipulação desta instância através do envio de mensagens invocando as operações internas disponíveis (operações da classe e operações herdadas).

(4) Transferência da instância (Salva) da região alocada na memória real para a memória persistente;

Durante a execução da ação descrita no item (3) poderá ocorrer uma expansão ou uma diminuição do tamanho da instância do objeto Versões, caso seja incluído ou removido algum elemento Componente. Na execução da ação (4) ocorre a execução de uma operação interna herdada da super-classe Lista Genérica, que corresponde a Transferência Lógica descrita na seção 3.2.2.1.1. A Transferência Lógica visa compactar a instância por ocasião de seu armazenamento na memória persistente.

3.2.2.2.11 **CLASSE: RESTRIÇÕES**

A classe de objetos denominada Restrições foi criada especialmente para implementar o conceito de Restrições descrito no capítulo 2 deste trabalho. O Objeto de Gerenciamento Restrições é usado para manipular o uso das instâncias de outros objetos de gerenciamento estabelecendo limites de controle para criação. Nem todos objetos de gerenciamento possuem parâmetros de controle armazenados nas instâncias do objeto Restrições. Os parâmetros existentes controlam o número máximo de instâncias permitidas dos seguintes objetos de gerenciamento:

- Nó de Composição;
- Nó de Instância;
- Link de Navegação;
- Link de Composição;
- Versões.

A utilização do objeto Restrições é opcional, podendo ser usado para alguns Objetos do Sistema que precisam de um controle mais rigoroso sobre o número de instâncias a serem armazenadas no Ambiente.

A classe Restrições foi implementada no Ambiente Poesis, como uma subclasse de Lista Fixa Genérica. Em conseqüência disto, esta classe herdou as características estruturais e comportamentais desta super-classe. A hierarquia de classe apresentada pela figura 14 permite a compreensão das diversas classes que participam na construção da classe Restrições. O resultado imediato obtido, foi a rápida implementação deste Objeto de Gerenciamento, já que o mecanismo de herança proveniente do enfoque orientado para objetos torna possível a redução do esforço de programação geralmente desprendido no desenvolvimento de um sistema deste porte.

Algumas informações são muito importantes para a manipulação das instâncias do objeto Restrições e seu relacionamento com o Objeto Memória. Estas informações são as seguintes:

Identificador do Objeto (IDO): 03;
Identificação da AAV (IAAV): RESTRIC.AAV;
Identificação da LEV: RESTRIC.LEV.

DESCRIÇÃO DO FUNCIONAMENTO DO OBJETO RESTRIÇÕES

O objeto Restrições é usado no GOLGO para informar os limites máximos para criação de instâncias de alguns objetos de gerenciamento. A manipulação das instâncias do Restrições é realizada sempre na modalidade parcial, havendo entretanto, possibilidade de manipulação Total de instâncias em Memória Real.

Tanto na modalidade parcial, quanto na modalidade total, o funcionamento do objeto Restrições caracteriza-se sempre pela realização das seguintes ações:

(1) Obtenção do Endereço Relativo Virtual da instância do objeto Restrições que está associada a um determinado Nó de Classe. Esta operação é executada pelo objeto Nó de Classe.

(2) Transferência da(s) instância(s) de Restrições (Carga Parcial/Total) da memória persistente para uma região previamente alocada na memória real;

(3) Manipulação da(s) instância(s) através do envio de mensagens invocando as operações internas disponíveis (operações da classe e operações herdadas);

(4) Transferência da(s) instância(s) (Salva Parcial/Total) da região alocada na memória real para a memória persistente;

3.2.2.2.12 **CLASSE: DEFINIÇÃO VISUAL**

Uma das características fundamentais do GOLGO consiste em dotar os Objetos do Sistema da capacidade de interação com o usuário. Esta capacidade introduz uma nova abordagem para o conceito de Interfaces Orientadas para Objetos, pois baseia-se no princípio reativo dos objetos, onde cada Objeto do Sistema controla e manipula sua própria interface. Cada Objeto do Sistema poderá possuir diversas formas de visualização no Ambiente Poesis, que propicia uma flexibilidade na implementação de Interfaces Evolutivas. Estas formas de visualização poderão ser ampliadas bem como poderão ser introduzidos novos objetos no Ambiente sem necessariamente ser preciso a manutenção do Ambiente, pois a parte da Interface nova é criada no momento da inserção destes objetos. Logo do ponto de vista de interface, o Ambiente permitirá através da navegação pela estrutura do GOLGO, que cada Objeto do Sistema realize sua própria interação com o usuário pela manipulação das Definições Visuais.

A classe Definição Visual foi criada para o GOLGO com o objetivo de controlar um conjunto de informações necessárias a geração da Interface de um determinado Objeto do Sistema. Estas informações contidas em cada instância do objeto Definição Visual, descrevem a parte estática da

definição visual bem como sua dinâmica de funcionamento destas, e serão fornecidas aos objetos responsáveis pela execução da Interface: a classe Dispositivos e as classes auxiliares compostas pelos objetos Mapa de Teclas e Contexto Sensitivo (ver seção 3.2.3).

No GOLGO, a classe Definição Visual foi implementada como uma subclasse de Lista Variável Genérica, e portanto, suas instâncias são listas cujos elementos contém os parâmetros necessários a execução da Interface de um Objeto do Sistema qualquer. A figura 14 apresenta a hierarquia de classes que constitui os objetos componentes do GOLGO. Os parâmetros que fazem parte do Objeto de Gerenciamento Definição Visual que tornam possível a ativação da interface do objeto, são:

- Formato;
- Mapa de Teclas e/ou Contexto Sensitivo.

O Formato define que objeto da classe Dispositivos será usado, e também como será feito, através de parâmetros especiais que permitem gerar janelas, ícones, etc. A parte da Definição Visual que determina o funcionamento da Interface do objeto é definida pelo Mapa de Teclas e/ou Contexto Sensitivo. O Mapa de Teclas contém informações sobre as funções das teclas envolvidas e o tipo de ação que elas produzem. O Contexto Sensitivo, usado pelo Mouse, define as regiões sensíveis ao click do Mouse e o tipo de ação que será ativada. O Formato é sempre obrigatório para a Definição Visual representada no elemento da instância, e tanto o Mapa de Teclas quanto o Contexto Sensitivo são opcionais. O detalhamento destas informações será feito na seção 3.2.3 deste trabalho.

No Ambiente o objeto Definição Visual utiliza algumas informações necessárias ao relacionamento com o Objeto Memória e à manipulação de suas instância. Estas informações são as seguintes:

Identificador do Objeto (IDO): 07;
Identificação da AAV (IAAV): DEFVIS.AAV;
Identificação da LEV: DEFVIS.LEV.

DESCRIÇÃO DO FUNCIONAMENTO DO OBJETO DEFINIÇÃO VISUAL

As instâncias do objeto de gerenciamento Definição Visual são manipuladas basicamente quando é feita a criação de novas Definições Visuais e quando é necessária a ativação das mesmas durante a navegação e processamento no Ambiente. Em cada uma destas situações a manipulação de uma instância de Definição Visual consiste da realização das seguintes ações:

(1) Obtenção do Endereço Relativo Virtual da instância do objeto Definição Visual que está associada a um determinado Nó de Classe. Esta operação é executada pelo objeto Nó de Classe.

(2) Transferência da instância (Carga) do objeto Definição Visual, da memória persistente para uma região previamente alocada na memória real.

(3) Manipulação desta instância através do envio de mensagens invocando as operações internas disponíveis (operações da classe e operações herdadas).

(4) Transferência da instância (Salva) da região alocada na memória real para a memória persistente;

Durante a execução da ação descrita no item (3) poderá ocorrer uma expansão ou uma diminuição do tamanho da instância do objeto Definição Visual, caso seja incluído ou removido algum elemento Componente. Na execução da ação (4) ocorre a execução de uma operação interna herdada da super-classe Lista Genérica, que corresponde a Transferência Lógica descrita na seção 3.2.2.1.1. A Transferência Lógica visa compactar a instância por ocasião de seu armazenamento na memória persistente.

3.2.3 **GERENCIAMENTO DE DISPOSITIVOS**

A criação de um Ambiente para concepção de aplicações envolve todo um processo de elaboração das estruturas internas para permitir gerenciamento de memória, gerenciamento dos objetos e o gerenciamento dos dispositivos de Input/Output e com isso tornar possível sua utilização. O gerenciamento de dispositivos de Input/Output é uma das partes mais importantes de um projeto que tem por objetivo fornecer um suporte ao desenvolvimento de Interfaces Orientada para Objetos.

Com a evolução da eletrônica, novos dispositivos estão surgindo para integração a sistemas computacionais, e com isso permitindo ao homem moderno desfrutar de ferramentas cada vez mais sofisticadas que vão deste o processamento de imagens e sons em estações de trabalho até os mais recentes dispositivos sensoriais, tais como VIEW (Virtual Interface Environment Workstation), que permite o usuário explorar um ambiente virtual de imagens de 360 graus (Stereoscopic Display System) controlado pela posição, voz, e por dispositivos do tipo Hand-Tracking que permitem ao usuário a percepção sensorial de forma de objetos através de sensações táteis [EGLO90, FISH90]. Esta tendência com o passar dos anos proporcionará o surgimento de dispositivos ainda não imagináveis acoplados a computadores ultra-modernos.

Com relação a parte de software, existem hoje pesquisas direcionadas a produzir sistemas para controle de dispositivos não-convencionais. Estes sistemas vão desde o armazenamento e recuperação de informações "especiais" (do tipo imagem e som), em Sistema Gerenciadores de Banco de Dados, até Interfaces para a manipulação destas informações pelo usuário. Estes sistemas são denominados Sistemas de Multimídia e tornou-se nos últimos anos uma área fascinante de pesquisa, pois seus resultados estão produzindo mecanismos para sistemas com aplicação em diversas áreas do conhecimento humano.

Visando dotar o Ambiente Poesis de uma estrutura capaz de implementar o controle e a manipulação de dispositivos convencionais e futuramente multimídia, a serem usados na manipulação da Interface Orientada para Objetos, foi concebido o gerenciamento de dispositivos que será especificado nesta seção através de suas características, arquitetura e funcionamento.

3.2.3.1 **CARACTERÍSTICAS DO GERENCIAMENTO DE DISPOSITIVOS**

A parte do GOLGO responsável pelo gerenciamento de dispositivos compreende um conjunto de objetos associados a cada tipo de dispositivo existente na configuração do equipamento. Os objetos do gerenciamento de dispositivos são responsáveis pela execução do interfaceamento de cada Objeto do Ambiente Poesis com o usuário, através das respectivas Definições Visuais associadas. Este enfoque de orientação para objetos adotado para implementação deste gerenciador tem por objetivos:

1 - Permitir que cada dispositivo existente possua características estruturais e comportamentais próprias;
2 - Permitir a manipulação das Definições Visuais (ver seção 3.2.2.2) associadas a cada objeto existente no Ambiente;
3 - Possibilitar a evolução de diferentes tipos de Definição Visual a medida que o Ambiente evolua produzindo novas ferramentas;
4 - Permitir a manipulação do conceito de Interface Orientada para Objetos (ver capítulo 2).

O gerenciamento de dispositivos compreende basicamente duas categorias de objetos:

- Objetos Dispositivos;
- Objetos Auxiliares.

Os Objetos Dispositivos correspondem a implementação dos diversos tipos de dispositivos de Input/Output e suas respectivas formas de interação com o usuário. Os Objetos Dispositivos podem ser encarados como sendo manipuladores dos dispositivos através de tipos de definições associadas, e

produzem a captação e o fornecimento de informações entre o meio externo e o Ambiente. Os Objetos Auxiliares compreendem aqueles usados pela Interface do Ambiente para a manipulação dinâmica das várias definições visuais existentes através da associação destas com as respectivas ações a serem processadas, quando forem ativadas pelo usuário. Estas duas categorias serão abordadas em detalhes na seção 3.2.3.2.

PROPRIEDADES DO GERENCIAMENTO DE DISPOSITIVOS

Alguns ambientes que implementaram o gerenciamento de dispositivos com o enfoque orientado para objetos tais como o ORION [WOEL87] para sistemas multimídia possuem propriedades que aprimoram a eficiência deste tipo de gerenciamento. Para o GOLGO, foram consideradas as seguintes propriedades na concepção do gerenciamento de dispositivos:

a) **Modularidade** - Esta propriedade é "herdada" quando se utiliza o enfoque orientado para objetos para o desenvolvimento de um sistema, e corresponde a implementação das funções do gerenciamento de dispositivos, em objetos com funções pré-definidas. A modularidade é obtida desta forma encapsulando as funções em objetos e integrando-os ao sistema através do compartilhamento de mensagens trocadas.

b) **Extensibilidade** - Corresponde basicamente a 3 aspectos: a capacidade de expansão (expansibilidade), de ser o mais genérico possível (generabilidade) e de permitir modificações que não causem impacto ao projeto como um todo (modificabilidade).

c) **Flexibilidade** - Esta propriedade é decorrente das anteriores e consiste em dotar o Ambiente de estruturas (objetos) de fácil manipulação.

O GERENCIAMENTO DE DISPOSITIVOS E O OBJETO DEFINIÇÃO VISUAL

Com o objetivo de implementar o conceito de Interface Orientada para Objetos (descrita no capítulo 2) que consiste basicamente em dotar os Objetos do Ambiente de interfaceamento individual, foi criado o Objeto de Gerenciamento denominado Definição Visual (mais detalhes na seção 3.2.2.2.12). O objeto Definição Visual, permite associar a um determinado Objeto do Ambiente, um conjunto de Definições Visuais que irão constituir a Interface própria deste objeto. Com este conceito, um determinado objeto poderá possuir diversos tipos de interfaces que serão ativadas conforme o tipo de usuário que está manipulando o Ambiente Poesis.

Uma determinada Definição Visual é composta pelas seguintes informações dada pela tupla:

< FORMATO, MAPA-DE-TECLAS, CONTEXTO-SENSITIVO >.

A informação denominada de FORMATO contém parâmetros que permitem gerar no dispositivo apropriado a definição visual pretendida. Corresponde a parte estática da Definição Visual e varia conforme o tipo de dispositivo para o qual foi criada.

A informação MAPA-DE-TECLAS corresponde a um conjunto de elementos que associa o acionamento de "teclas especiais" a ações a serem realizadas para uma determinada Definição Visual. Esta informação não é obrigatória, podendo existir determinadas Definições Visuais que não a possuem.

A informação CONTEXTO-SENSITIVO corresponde a um conjunto de elementos que associa regiões do vídeo sensíveis ao click do mouse cujo acionamento determina a execução de ações a serem realizadas para uma determinada Definição Visual. Esta informação também não é obrigatória, podendo existir determinadas Definições Visuais que não a possuem.

As informações MAPA-DE-TECLAS e CONTEXTO-SENSITIVO implementam a parte dinâmica de uma determinada Definição Visual, e são utilizadas pelos ***Objetos Auxiliares*** do

gerenciamento de dispositivos. A informação FORMATO é fornecida aos ***Objetos Dispositivos*** para a geração e manipulação de uma determinada Definição Visual.

3.2.3.2 **ARQUITETURA DO GERENCIAMENTO DE DISPOSITIVOS**

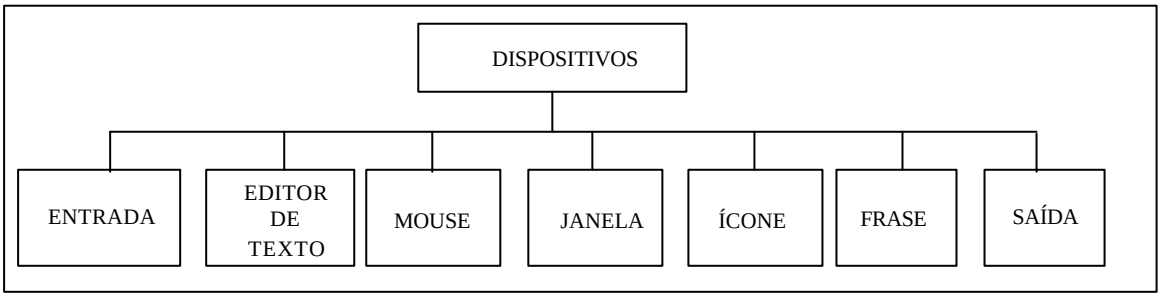
Conforme foi descrito na seção 3.2.3.1 o gerenciamento de dispositivos corresponde a duas categorias: Objetos Dispositivos e Objetos Auxiliares. Os Objetos Dispositivos foram implementados segundo a hierarquia de classes descrita pela figura 27. Nesta figura observa-se a existência de uma Super-classe denominada DISPOSITIVOS, composta pelas seguintes subclasses:

- ENTRADA;
- EDITOR DE TEXTO;
- JANELA;
- ÍCONE;
- FRASE;
- SAÍDA;
- MOUSE;

A subclasse ENTRADA é responsável pela geração de Definições Visuais que captam informações do tipo Campo de Dados, efetuando a crítica automática para campos numéricos e de outros tipos como data e hora. A subclasse EDITOR DE TEXTO é utilizada para gerar e manipular a Definição Visual que permite ao usuário a edição de textos no Ambiente.

A subclasse JANELA é responsável pela geração e manipulação de Definições Visuais do tipo Janela (windows), bem como a composição de outros tipos de Definições Visuais (Ícones e Frases) para interação com o usuário. A subclasse ÍCONE é usada para a geração e manipulação de Definições Visuais do tipo Ícone visando tornar as Interfaces dos objetos mais "amigáveis". A subclasse FRASE é usada para exibir palavras ou textos do tipo constante, existente em Definições Visuais do tipo Janela. A subclasse SAÍDA é usada para a geração de Definições Visuais do tipo Saída que exibem campos de dados com máscara de edição associada.

FIGURA 27 - HIERARQUIA DE CLASSES DO GERENCIAMENTO DE DISPOSITIVOS DO GOLGO.



A subclasse MOUSE é responsável pela manipulação do dispositivo mouse associado ao sistema permitindo assim a interação entre o usuário e as Definições Visuais ativas no dispositivo Vídeo.

Outra categoria de objetos do gerenciamento de dispositivo é a correspondente aos Objetos Auxiliares usados na manipulação dinâmica das diversas Interfaces dos Objetos do Ambiente. Esta categoria é formada pelas seguintes classes de objetos:

- MAPA-TECLAS;
- CONTEXTO-SENSITIVO;
- PILHA-MAPA-TECLAS;
- PILHA-CONTEXTO-SENSITIVO.

A classe MAPA-TECLAS é responsável pela manipulação da informação Mapa de Teclas das diversas Definições Visuais implementadas no Ambiente.

A classe CONTEXTO-SENSITIVO efetua a manipulação da informação Contexto Sensitivo das Definições Visuais no Ambiente.

A classe PILHA-MAPA-TECLAS é usada para empilhar diversas instâncias do objeto Mapa-Teclas que estão simultaneamente sendo manipuladas no dispositivo vídeo.

A classe PILHA-CONTEXTO-SENSITIVO efetua o empilhamento de diversas instâncias do objeto Contexto-Sensitivo que estão simultaneamente sendo manipuladas no dispositivo vídeo.

Serão descritas a seguir cada uma destas classes que compõem o gerenciamento de dispositivos. As especificações das estruturas das classes estão descritas no apêndice II.

3.2.3.2.1 **CLASSE DISPOSITIVOS**

A classe Dispositivo corresponde a super-classe da hierarquia de dispositivos implementada no Ambiente Poesis. A esta classe estão subordinadas todas as classes que implementam os diversos dispositivos manipulados durante o processamento da Interface Orientada a Objetos. Na figura 27 pode ser visualizada esta hierarquia composta pelas seguintes subclasses:

- ENTRADA;
- EDITOR DE TEXTO;
- JANELA;
- ÍCONE;
- FRASE;
- SAÍDA;
- MOUSE;

A medida que o Ambienta evolua com a inclusão de novas ferramentas a hierarquia de dispositivos poderá ser expandida com a inclusão de novas subclasses que permitirá o controle dos respectivos periféricos no enfoque orientado para objetos.

CLASSE ENTRADA

A classe Entrada representa o processador genérico do tipo de Definição Visual denominada Entrada associada aos dispositivos Teclado ou console, e Vídeo, acoplado ao sistema. Esta classe manipula estes dispositivos de modo a oferecer ao Ambiente um tipo de Definição visual que permita a entrada de campos de dados para os mais diversos fins. Responsável ainda pela integridade dos dados introduzidos via teclado, esta classe controla tipos de campos, tamanho, atributos especiais, entre outros, permitindo a transferência das informações entre os objetos do sistema. Na hierarquia de dispositivos apresentada pela figura 27, esta classe foi implementada como uma subclasses da classe Dispositivo.

CLASSE EDITOR DE TEXTO

A classe Editor de Texto implementa um processador genérico para o tipo de Definição Visual denominada Editor de Texto associada aos dispositivos Teclado ou console, e Vídeo, acoplado ao sistema. Esta classe através de seus métodos efetuam a manipulação destes dispositivos de modo a oferecer ao Ambiente Poesis uma ferramenta que permite a edição de textos com todas as funções pertinentes. O editor de textos implementado nesta classe é da categoria de editores baseados em

comandos (tais como o WordStar), ou seja, o usuário através de combinação de teclas especiais (CTRL) ativa ou desativa os comandos sempre no modo de edição, evidentemente funcionando no modo direto através do uso das teclas de direção entre outras. Todas as funções de edição de textos estão presentes neste objeto, tais como:

- Controles do Cursor;
- Inserção ou Deleção;
- Manipulação de Blocos de textos;
- Busca e troca;
- Finalização.

Na hierarquia de dispositivos apresentada pela figura 27, esta classe foi implementada como uma subclasses da classe Dispositivo.

CLASSE JANELA

A classe Janela representa o processador genérico do tipo de Definição Visual denominada Janela associada ao dispositivo Vídeo, acoplado ao sistema. Esta classe manipula este dispositivo de modo a oferecer ao Ambiente um tipo de Definição visual que permita a exibição de Janelas para os mais variados fins. A Definição Visual Janela corresponde ao conceito de Windows definido em [SHNE87] usada para a comunicação entre o usuário e o sistema concentrando em uma região do vídeo a atenção sobre as informações que estão sendo mostradas/captadas em uma moldura retangular que poderá ser manipulada pelo usuário. Na hierarquia de dispositivos apresentada pela figura 27, esta classe foi implementada como uma subclasses da classe Dispositivo.

CLASSE ÍCONE

A classe Ícone representa o processador genérico do tipo de Definição Visual denominada Ícone associada ao dispositivo Vídeo, acoplado ao sistema. Esta classe manipula este dispositivo de modo a oferecer ao Ambiente um tipo de Definição visual que permita a exibição de Ícones para os mais variados fins. A Definição Visual Ícone corresponde a representação de um determinado objeto com um gráfico representando a figura que permite ao usuário sua identificação. O conceito de Ícone [COX86] usado no Ambiente Poesis visa a construção de uma Interface Orientada para Objetos de caráter amigável, fornecendo ao usuário condições necessárias a identificação dos objetos e operações através de gráficos que associam visualmente o objeto representado pelo Ícone ao Objeto do Sistema. Na hierarquia de dispositivos apresentada pela figura 27, esta classe foi implementada como uma subclasses da classe Dispositivo.

CLASSE FRASE

A classe Frase é responsável pela manipulação e ativação do tipo de Definição Visual denominada Frase associada ao dispositivo Vídeo, acoplado ao sistema. Esta classe manipula este dispositivo de modo a oferecer ao Ambiente um tipo de Definição visual que permita a exibição de Frases que na sua maior parte são utilizadas na construção de Definições Visuais do tipo Janela. A Definição Visual Frase corresponde a representação de um determinado string de caracteres usado para compor as informações de uma determinada Janela. Na hierarquia de dispositivos apresentada pela figura 27, esta classe foi implementada como uma subclasses da classe Dispositivo.

CLASSE SAÍDA

A classe Saída corresponde ao processador genérico do tipo de Definição Visual denominada Saída associada ao dispositivo Vídeo acoplado ao sistema. Esta classe manipula este dispositivo de modo a oferecer ao Ambiente um tipo de Definição visual que permita a exibição de campos de dados

para os mais diversos fins. A visualização de campos é também de fundamental importância no projeto de uma Interface, e portanto não poderia deixar de existir uma classe responsável pela manipulação de variáveis com exibição conforme o tipo de campo, tamanho, atributos especiais, máscaras de edição, entre outros. Na hierarquia de dispositivos apresentada pela figura 27, esta classe foi implementada como uma subclasses da classe Dispositivo.

CLASSE MOUSE

A classe Mouse encapsula o processamento do dispositivo Mouse acoplado ao sistema permitindo seu uso na manipulação da Interface Orientada para objetos implementada no Ambiente Poesis. Para o GOLGO o dispositivo Mouse é considerado uma Definição Visual já que é permitido associar um ou mais ícones que irá representar o cursor correspondente no dispositivo Vídeo para melhor ser visualizado pelo usuário. A forma clássica de exibição do cursor é uma seta que varia em algumas interfaces assumindo diversas formas como lápis, pincéis, ampulhetas, relógios, etc. sempre em função do tipo de aplicação e interface em uso. No Ambiente Poesis a forma de apresentação do cursor do Mouse poderá também variar de acordo o tipo de interação com o usuário. Na hierarquia de dispositivos apresentada pela figura 27, esta classe foi implementada como uma subclasses da classe Dispositivo.

3.2.3.2.2 **OBJETOS AUXILIARES**

Para que o gerenciamento dos dispositivos e definições visuais fosse implementado outros objetos foram sendo necessários para tornar possível a manipulação da Interface Orientada para Objetos do Ambiente Poesis. Estes objetos denominados Objetos Auxiliares efetuam o controle da manipulação dinâmica das diversas Interfaces dos Objetos do Ambiente. Esta categoria é formada pelas seguintes classes de objetos:

- MAPA-TECLAS;
- CONTEXTO-SENSITIVO;
- PILHA-MAPA-TECLAS;
- PILHA-CONTEXTO-SENSITIVO.

Serão apresentados a seguir a descrição de cada um destes objetos auxiliares.

CLASSE MAPA-TECLAS

A classe MAPA-TECLAS é responsável pela manipulação da informação Mapa de Teclas das diversas Definições Visuais implementadas no Ambiente. Este objeto efetua o controle sobre as teclas especiais acionadas e encaminha a ação correspondente para o processamento da Interface. Este tipo de objeto permite a utilização do teclado para a manipulação mais rápida das opções disponíveis na Interface de cada Objeto do Sistema.

CLASSE CONTEXTO-SENSITIVO

A classe CONTEXTO-SENSITIVO tem por objetivo executar a manipulação da informação Contexto Sensitivo das Definições Visuais no Ambiente. Este objeto efetua o controle sobre as regiões do video sensíveis ao pressionamento do botão de seleção do Mouse e o encaminhamento da ação correspondente para o processamento da Interface.

CLASSE PILHA-MAPA-TECLAS

A classe Pilha-Mapa-Teclas é usada no decorrer do processamento do Ambiente Poesis quando diversas Definições Visuais estiverem ativas no video. A estrutura em forma de pilha permitirá que o Mapa de Teclas da última Definição Visual ativa seja manipulado e assim evitar conflitos quando duas ou mais Definições Visuais possuírem a mesma Tecla especial. Da mesma maneira, a medida que as Definições Visuais forem sendo desativadas os respectivos Mapa de Teclas são removidos da Pilha-Mapa-Teclas.

CLASSE PILHA-CONTEXTO-SENSITIVO

A classe Pilha-Contexto-Sensitivo é usada conjuntamente com a Pilha-Mapa-Teclas no decorrer do processamento do Ambiente Poesis quando diversas Definições Visuais estiverem ativas no video. A estrutura em forma de pilha permitirá que o Contexto-Sensitivo da última Definição Visual ativa seja manipulado e assim evitar conflitos quando duas ou mais Definições Visuais possuírem a mesma Região Sensitiva. Da mesma maneira, a medida que as Definições Visuais forem sendo desativadas os respectivos Contextos-Sensitivo são removidos da Pilha-Contexto-Sensitivo.

3.2.3.3 DESCRIÇÃO DO FUNCIONAMENTO

O gerenciamento de dispositivos implementado através das diversas classes de objetos criadas para permitir ao usuário uma Interface para cada Objeto do Sistema tem seu funcionamento executado de forma bastante simples. Cada Objeto do Sistema é portador de sua forma de apresentação e interação com o usuário próprias. Além disso, estes objetos podem ter mais de uma forma de apresentação selecionáveis e ajustáveis de acordo com o tipo de usuário. O objeto de gerenciamento responsável pelo controle e manipulação destas formas de apresentação que constituem a Interface de cada objeto é denominado de Definição Visual. Quando o usuário através do processo de navegação no Ambiente atinge um certo Objeto do Sistema, a Definição Visual default deste objeto é recuperada e enviada ao objeto da classe Dispositivo responsável pela sua ativação. Neste caso, as seguintes etapas podem ser observadas:

Etapa 1 - Recuperação da Definição Visual a ser ativada;
Etapa 2 - Empilhar Mapa de Teclas (se houver) na instância do objeto Pilha-Mapa-Teclas;
Etapa 3 - Empilhar Contexto-Sensitivo (se houver) na instância do objeto Pilha-Contexto-Sensitivo.
Etapa 4 - Ativar dispositivo com o Formato;

A etapa 4 corresponde a ativação de uma ou mais subclasses de Dispositivos descritas na seção 3.2.3.2 e seu funcionamento poderá ser visualizado através da figura 28.

Na figura 28 a seqüência de execução obedece as seguintes ações:

A1 - Montagem da Definição Visual desejada;
A2 - Interação com o usuário;

Dependendo da escolha do usuário o fluxo de processamento segue 3 caminhos possíveis:

(a) Verificação da Pilha-Contexto-Sensitivo, para o caso de haver sido usado o Mouse para acionar uma região sensitiva;

(b) Verificação da Pilha-Mapa-Teclas, para o caso de haver sido pressionada alguma tecla especial (ou várias);

(c) Transferência do controle para o GOLGO após a execução da Definição Visual.

As ações (a) e (b) podem entretanto, se forem executadas poderão produzir uma nova ação que poderá ser:

- A transferência para o GOLGO informando o Indicador de Ação (ver seção 3.2.3.3);

- A ativação de uma nova Definição Visual "escrava", seguindo novamente os as etapas de 1 a 4.

FIGURA 28 - FLUXO DE EXECUÇÃO DA ETAPA 4 DE ATIVAÇÃO DO DISPOSITIVO.

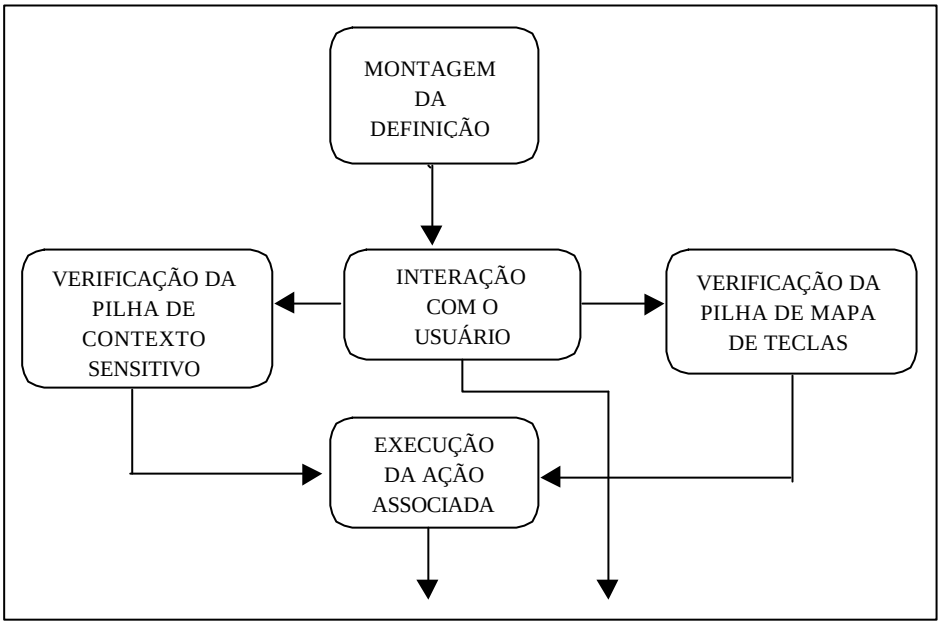
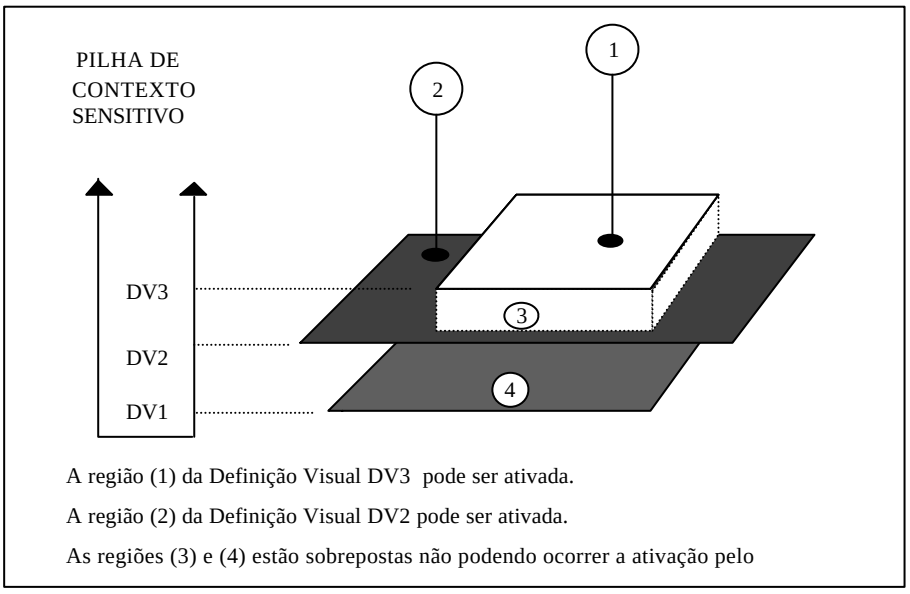


FIGURA 29 - ILUSTRAÇÃO DO FUNCIONAMENTO DA PILHA-CONTEXTO-SENSITIVO

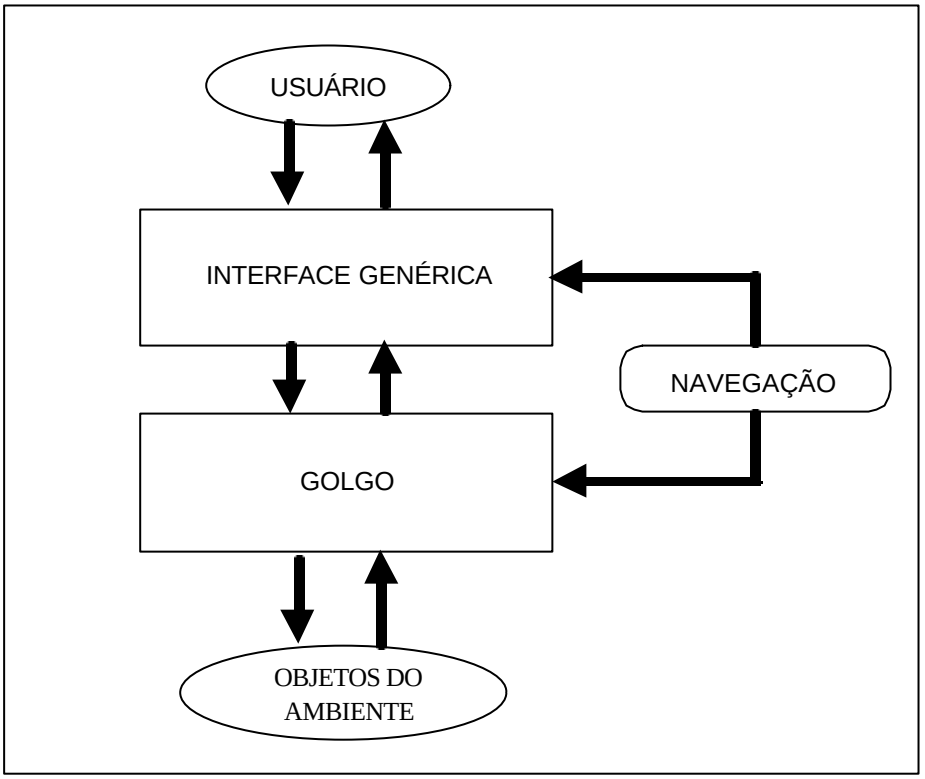


O funcionamento de cada objeto do gerenciamento de dispositivos segue basicamente o que foi descrito nas seções 3.2.3.2 e 3.2.3.3 e consiste em gerar para cada dispositivo associado ao sistema uma forma de apresentação com características definidas pelos parâmetros que são fornecidos pelo FORMATO existente em cada Definição Visual.

Os objetos auxiliares são também fundamentais a todo o processamento controlando a

dinâmica de utilização dos recursos da Interface através do acionamento de Teclas Especiais ou a utilização do Mouse para selecionar as opções desejadas. As pilhas de Mapa de Teclas e Contexto-Sensitivo são de muita importância no momento em que diversas Definições Visuais forem sendo ativadas. A figura 29 ilustra o uso da Pilha-Contexto-Sensitivo no caso da ativação de várias Definições Visuais do tipo Janela. Cada nova Definição Visual ativada sobrepõe as regiões sensitivas das Definições Visuais anteriores.

FIGURA 30 - DIAGRAMA DOS MÓDULOS FUNCIONAIS DO AMBIENTE POESIS.



3.3 **FUNCIONALIDADE DO GOLGO**

O ambiente Poesis pode ser visto como a composição de 3 partes que se inter-relacionam: os Objetos do Sistema, o GOLGO e a Interface Genérica. A figura 30 ilustra esta composição mostrando as linhas de comunicação existente através de troca de mensagens entre as partes.

Nesta seção será descrita a composição funcional do GOLGO, a composição funcional da Interface Genérica, e o relacionamento existente entre estas partes. Os Objetos do Ambiente estão definidos no capítulo 4 deste trabalho.

3.3.1 **DESCRIÇÃO FUNCIONAL DO GOLGO**

O principal objetivo do GOLGO é gerenciar os objetos do sistema de acordo com a filosofia descrita no capítulo 2. No aspecto funcional podemos caracterizar as seguintes funções básicas:

- Armazenamento;
- Recuperação;
- Navegação;
- Gerenciamento de Classes e Instâncias;
- Controle de Dispositivos;
- Controle de acesso;
- Controle de Versões;

- Composição de Objetos;

ARMAZENAMENTO e RECUPERAÇÃO

Esta função desempenhada pelo Objeto Memória possibilita o armazenamento de classes, instâncias de Objetos de Gerenciamento, bem como Objetos do Sistema tanto em memória principal, como em memória secundária permitindo inclusive a recuperação destas instâncias da memória secundária.

NAVEGAÇÃO

A filosofia de navegação existente no Ambiente Poesis e controlada pelo GOLGO, através do objeto Link de Navegação, permite ao usuário a busca de qualquer Objeto do Sistema, bem como suas instâncias, de forma interativa e respeitando a política de acesso e segurança.

GERENCIAMENTO DE CLASSES E INSTÂNCIAS

Esta função realizada por objetos de gerenciamento do GOLGO: Nó de Classe, Nó de Instância e Nó de Composição, possibilita a implementação de uma ambiente realmente orientado para objetos, com manipulação de métodos de classes e instâncias.

CONTROLE DE DISPOSITIVOS

O GOLGO através da classe Dispositivos permite uma manipulação dos Dispositivos acoplados ao sistema através de objetos que geram Definições Visuais que constituem a Interface de cada objeto.

CONTROLE DE ACESSO

O controle de acesso é das funções que permite restringir o acesso àquelas instâncias protegidas, bem como permite que o usuário defina níveis de proteção para as instâncias particulares. O controle de acesso é executado pelo objeto Link de Navegação.

CONTROLE DE VERSÕES

Alguns objetos do Ambiente Poesis necessitam do controle sobre as versões de instâncias geradas. É o caso das aplicações modeladas que poderão ser controladas através do Objeto de Gerenciamento Versões.

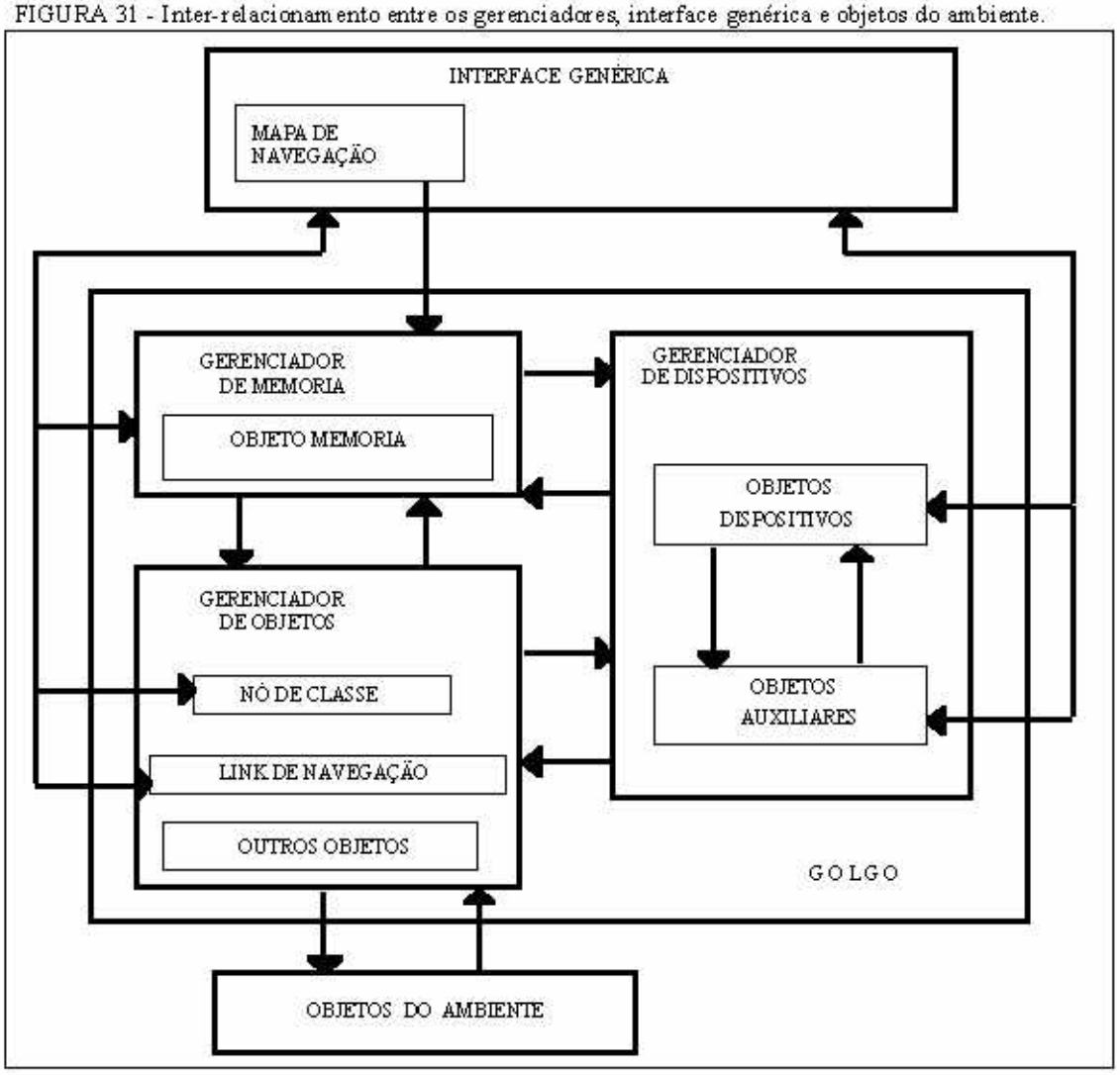
COMPOSIÇÃO DE OBJETOS

O GOLGO através dos objetos Nó de Composição e Link de Composição permite a implementação de objetos que necessitam de controle sobre a composição de suas instâncias.

3.3.2 **INTERFACE GENÉRICA**

No Ambiente Poesis cada objeto possui sua própria Interface sendo ativada a medida que o usuário o acessa, através do processo de Navegação. Entretanto, torna-se necessário uma Interface em um nível maior que permita o usuário efetuar a navegação e posterior acesso ao objeto que se deseja. Esta Meta-Interface é denominada no Ambiente Poesis de INTERFACE GENÉRICA. A figura 31 ilustra a Interface Genérica e seu inter-relacionamento com o GOLGO.

FIGURA 31 - Inter-relacionamento entre os gerenciadores, interface genérica e objetos do ambiente.



A Interface Genérica é considerada como elo de ligação entre o Objeto Definição Visual e os objetos da classe Dispositivos, sendo responsável pela ativação das Definições Visuais que definem a Interface de cada Objeto do Sistema.

3.3.2.1 **DESCRIÇÃO FUNCIONAL DA INTERFACE GENÉRICA**

A Interface Genérica possui as seguintes funções no processamento do Ambiente Poesis:

- Ativação de Definições Visuais;
- Encaminhamento das Ações para o GOLGO;
- Captura e Apresentação de Dados;
- Navegação.

ATIVAÇÃO DE DEFINIÇÕES VISUAIS

A Interface Genérica participa da Ativação de Definições Visuais de cada Objeto do Sistema através da transferência dos parâmetros armazenados em cada Definição Visual para os respectivos objetos da classe Dispositivo, solicitando em seguida que efetuem a ativação da Definição.

ENCAMINHAMENTO DE AÇÕES

Algumas Definições Visuais possuem elementos dinâmicos associados que são: Mapa de Teclas e Contexto Sensitivo. Tanto o Mapa de Teclas quanto o Contexto Sensitivo possuem ações associadas a cada tecla especial ou região sensitiva. A Interface Genérica identifica cada ação encaminhando-a para processamento pelo GOLGO.

CAPTURA e APRESENTAÇÃO DE DADOS

A Interface Genérica é responsável pela transferência dos buffers de dados a serem exibidos ou captados pelas respectivas Definições Visuais Saída e Entrada.

NAVEGAÇÃO

Considerada como sua função principal a Navegação através dos objetos do sistema é de fundamental importância para o processamento do Ambiente Poesis. A Interface Genérica efetua todo o processamento através da comunicação entre os objetos auxiliares dos dispositivos: Mapa-Teclas, Contexto-Sensitivo, Pilha-Mapa-Teclas, Pilha-Contexto-Sensitivo. Mantém ainda um controle de Navegação que permite delinear todo o caminho percorrido através do Mapa de Navegação.

3.3.2.2 **OBJETOS DA INTERFACE GENÉRICA**

A Interface Genérica utiliza alguns objetos usados para a execução de suas funções. Os objetos que são manipulados durante o processamento do Ambiente Poesis pela Interface Genérica são:

- Objetos auxiliares dos Dispositivos (Mapa-Teclas, Contexto-Sensitivo, Pilha-Mapa-Teclas, Pilha-Contexto-Sensitivo);
- Objetos da classe Dispositivo (ver seção 3.2.3.2);
- Mapa de Navegação;

MAPA DE NAVEGAÇÃO

O objeto Mapa de Navegação tem por finalidade manter todo o caminho percorrido pelo usuário durante o processo de navegação. Este objeto armazena as informações relativas às instâncias dos objetos Nó de Classe e Nó de Instância visitadas em uma estrutura em forma de pilha. Cada elemento da pilha compreende um conjunto de informações que permitem à Interface Genérica a recuperação das informações. Este elemento do Mapa de Navegação pode ser compreendido a partir da tupla:

< TIPO, ERV, IDP >, onde:

TIPO é o parâmetro que indica o tipo de Nó visitado podendo assumir os seguintes valores:

0 Nó de Classe;
1 Nó de Composição;
2 Nó de Instância.

ERV é a variável que informa o Endereço Relativo Virtual da instância visitada dos objetos Nó de Classe, Nó de Composição ou Nó de Instância, que permite recuperá-la na memória persistente;

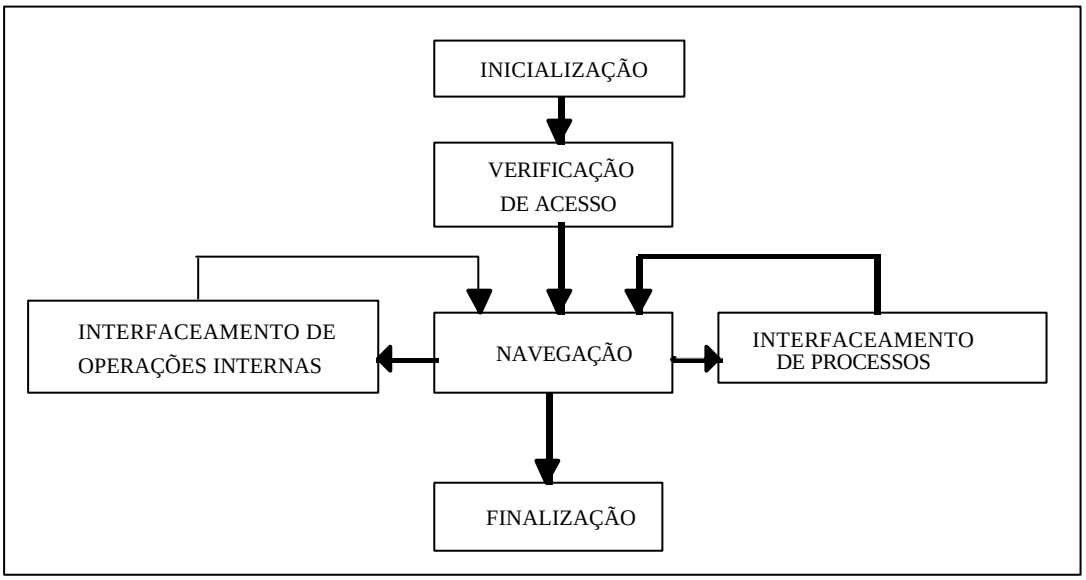
IDP é o Identificador do Processo de Alocação que permitirá recuperar a instância na memória real.

OPERAÇÕES INTERNAS DO OBJETO MAPA DE NAVEGAÇÃO

O objeto Mapa de Navegação manipula suas instâncias através de um conjunto de operações internas, ou métodos, que serão descritos a seguir.

CONSTRUTOR - Este método efetua a inicialização da instância ajustando a variável TOPO para o valor zero.
INSERÇÃO - Este método executa a inserção de um determinado elemento na pilha sendo que os valores das variáveis são informados via parâmetro.
REMOÇÂO - Este método retira o elemento localizado na posição indicada por TOPO.
OBTER-TIPO, OBTER-ERV e OBTER-IDP - Estes métodos fornecem respectivamente o conteúdo das variáveis: TIPO, ERV e IDP do elemento da pilha referenciado por TOPO.
DESTRUTOR - Este método finaliza o processamento do Objeto Mapa de Navegação retirando da memória real a instância correspondente.

FIGURA 32 - DIAGRAMA DE MÓDULOS QUE COMPÕEM O FUNCIONAMENTO DO SISTEMA.



3.4 DESCRIÇÃO DO MECANISMO DE FUNCIONAMENTO DO GOLGO

O GOLGO como foi descrito nas seções anteriores é constituído por um conjunto de objetos com funções específicas que se comunicam a partir de troca de mensagens executando cada um a sua tarefa no processamento do Ambiente Poesis. A filosofia de funcionamento segue basicamente os conceitos do Paradigma da Orientação para Objetos [TAKA88]. Cada objeto possui sua independência no que se refere ao processamento pelo qual é responsável, não havendo interferência do meio externo. As operações internas implementadas em cada objeto são ativadas pelo envio de uma mensagem ao objeto que efetua então o processamento. Entretanto, torna-se necessária a coordenação do funcionamento de todos os objetos de forma a permitir a integração de todos os objetos do sistema. Esta integração é realizada pelo módulo principal composto de submódulos com funções características. O módulo principal do sistema denominado POESIS possui os seguintes módulos:

- INICIALIZAÇÃO;
- VERIFICAÇÃO DE ACESSO;
- NAVEGAÇÃO;
- INTERFACEAMENTO DE OPERAÇÕES INTERNAS;
- PROCESSOS;
- FINALIZAÇÃO;

A figura 32 apresenta um diagrama de relacionamento destes módulos bem como a seqüência de execução. Cada um destes módulos será descrito a seguir.

3.4.1 **INICIALIZAÇÃO**

Este módulo tem por objetivo efetuar a inicialização do Ambiente Poesis através da execução dos seguintes procedimentos:

- Inicializar o Objeto Memória;
- Verificar periféricos;
- Inicializar objetos auxiliares do gerenciador de dispositivos;
- Inicializar o objeto Mapa de Navegação;
- Efetuar a carga para a memória real das instâncias do objeto Nó de Classe representantes dos Objetos do Ambiente;
- Efetuar a carga para a memória real das instâncias do objeto Link de Navegação de todos os Objetos do Ambiente;
- Efetuar a carga para a memória real da instância do objeto Definição Visual referente ao Objeto do Ambiente denominado POESIS;
- Ativar a Definição Visual default do objeto inicial POESIS;

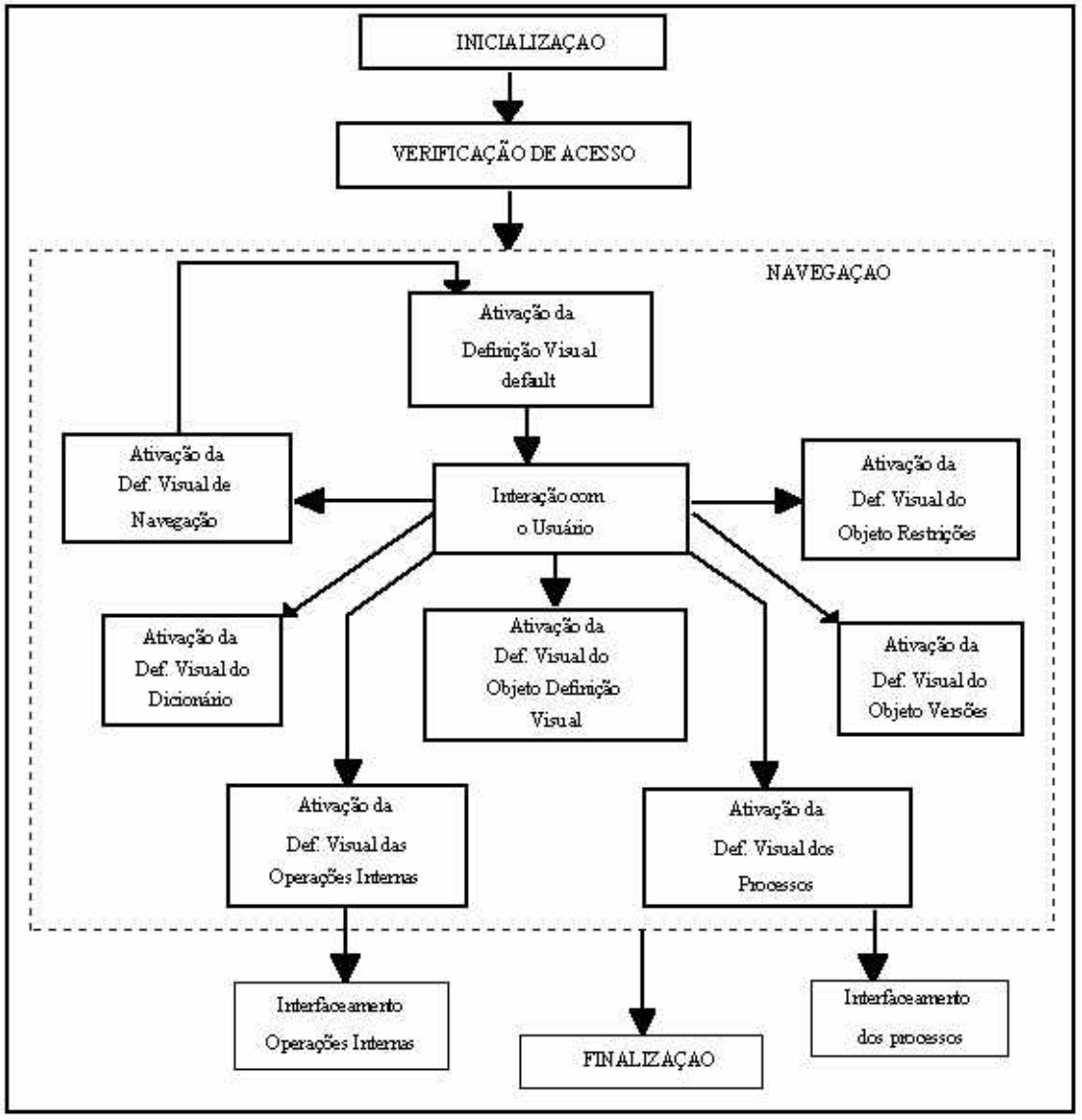
Após a execução do último procedimento o Ambiente estará pronto para a execução do módulo seguinte.

3.4.2 **VERIFICAÇÃO DE ACESSO**

O Ambiente Poesis possui um protocolo inicial de funcionamento que consiste em verificar se o usuário que está pretendendo acessá-lo tem ou não a permissão, através da verificação de seu código e senha. Este módulo efetua esta verificação a partir da execução dos seguintes procedimentos:

- Acessar a instância do objeto do ambiente denominado USUÁRIO (ver seção 4.1)para ativação da Definição Visual correspondente a solicitação de código e senha;
- Após a interação com o usuário, enviar a mensagem PESQUISA ao objeto USUÁRIO para que o mesmo realize a busca da informação relativa ao usuário;
- Se houver qualquer erro na pesquisa ou mesmo se a senha não conferir o sistema executa o módulo de finalização do Ambiente;
- Caso contrário será então efetuada a ativação da Definição Visual que permitirá o processo de navegação do sistema.

FIGURA 33 - Composição do Módulo Navegação do Ambiente Poesis



3.4.3 **NAVEGAÇÃO**

Este módulo corresponde a implementação da Interface Genérica e sua interação com o GOLGO, sendo que a maior parte do tempo de processamento do Ambiente o módulo de Navegação fica em execução até que o usuário requisite a finalização do processamento. A figura 33 apresenta a composição deste módulo com os respectivos procedimentos e a seqüência de execução. Os principais procedimentos que constituem este módulo são os seguintes:

- Ativação da Definição Visual default do Objeto do Ambiente corrente;
- Interação com o usuário;
- Ativar a Definição Visual de navegação conforme solicitação do usuário e efetuar o acesso a outro objeto no grafo através do Link de Navegação;
- Ativar a Definição Visual de acesso ao Dicionário de Objetos e executar o acesso à instância requerida;
- Ativar a Definição Visual de acesso ao objeto Versões e efetuar o acesso a instância;
- Ativar a Definição Visual de acesso ao objeto Restrições e efetuar o acesso a respectiva instância;
- Ativar a Definição Visual de acesso às operações internas do Objeto do Ambiente e encaminhar o envio da mensagem correspondente;
- Ativar a Definição Visual de acesso à instância do objeto Link de Processos associada e realizar a

sua execução;
- Ativar a Definição Visual de acesso à instância do objeto Definição Visual associada para permitir a ativação de outras formas de apresentação.

Como foi descrito acima, o módulo de navegação possibilita o acesso aos objetos de gerenciamento do GOLGO permitindo ao usuário manipular cada Objeto do Ambiente conforme descrito no capítulo 2.

3.4.4 **INTERFACEAMENTO DE OPERAÇÕES INTERNAS**

Este módulo foi criado para permitir a execução das operações internas de cada Objeto do Ambiente a partir das instâncias do objeto de gerenciamento Índice de Operações Internas. Este módulo é executado logo após a seleção da operação interna desejada pelo usuário e sua principal função é determinar a mensagem a ser executada e fornecer os parâmetros necessários a sua execução.

3.4.5 **PROCESSOS**

Neste módulo são implementados todos os processos especiais e o protocolo de acesso e execução das rotinas. Desta forma o sistema permite então a execução de uma determinada rotina a partir de uma instância do objeto Link de Processos e assim realizar o processamento solicitado pelo usuário.

3.4.6 **FINALIZAÇÃO**

Este módulo implementa os procedimentos finais para o encerramento do processamento do Ambiente Poesis. Consiste basicamente na realização dos seguintes procedimentos:

- Transferência para a memória persistente (salva) das instâncias dos objetos de gerenciamento: Nó de Classe e Link de Navegação;
- Desalocação da memória real de todas as instâncias de objetos auxiliares ao processamento: Mapa de Navegação, objetos auxiliares de dispositivos, objetos da classe Dispositivos, etc.
- Volta ao Sistema Operacional.

CAPÍTULO 4

OBJETOS DO AMBIENTE POESIS

Os objetos do Ambiente denominados também de Objetos do Sistema tem por objetivo proporcionar aos usuários as ferramentas necessárias para o desenvolvimento integrado e interativo de aplicações para Banco de Dados. Estas ferramentas descritas no capítulo 1 tornam possível a utilização do Ambiente Poesis através das Interfaces existentes para cada Objeto do Ambiente de maneira fácil e objetiva.

O Ambiente POESIS é composto pelos seguintes objetos:

- Usuário;
- Modelo E/D;
- Aplicação Modelada;
- Tutorial;
- Aplicação Relacional.

Em seguida serão especificadas as características das classes dos objetos e suas funcionalidades. Quanto aos aspectos estruturais destes objetos poderão ser encontrados no apêndice III.

4.1 CLASSE: USUÁRIO

Uma das características mais importantes do Ambiente Poesis é a capacidade de administração do uso de suas ferramentas, proporcionando interfaces voltadas ao grau de conhecimento dos diferentes tipos de usuários. Quanto ao grau de conhecimento um determinado usuário poderá ser classificado como:

- Leigo,
- Intermitente,
- Especialista e
- Superusuário.

A classe Usuário considerada como Objeto do Sistema foi criada para controlar as informações que permitem aos usuários do Ambiente manipular os demais objetos, conforme o nível de acesso e grau de conhecimento. Esta classe foi implementada como subclasse da classe Árvore-B Genérica para permitir a recuperação mais rápida das informações armazenadas em suas instâncias. Desta maneira, a classe Usuário pode ser vista como uma àrvore-B cujos nodos correspondem ao registro de informações sobre os diferentes usuários que utilizarão o sistema. Um determinado usuário é identificado pela chave de acesso à árvore denominada Código do Usuário, que deverá ser único no Ambiente. A partir do Código do Usuário é possível recuperar outras informações que serão utilizadas durante a manipulação dos objetos de gerenciamento do GOLGO.

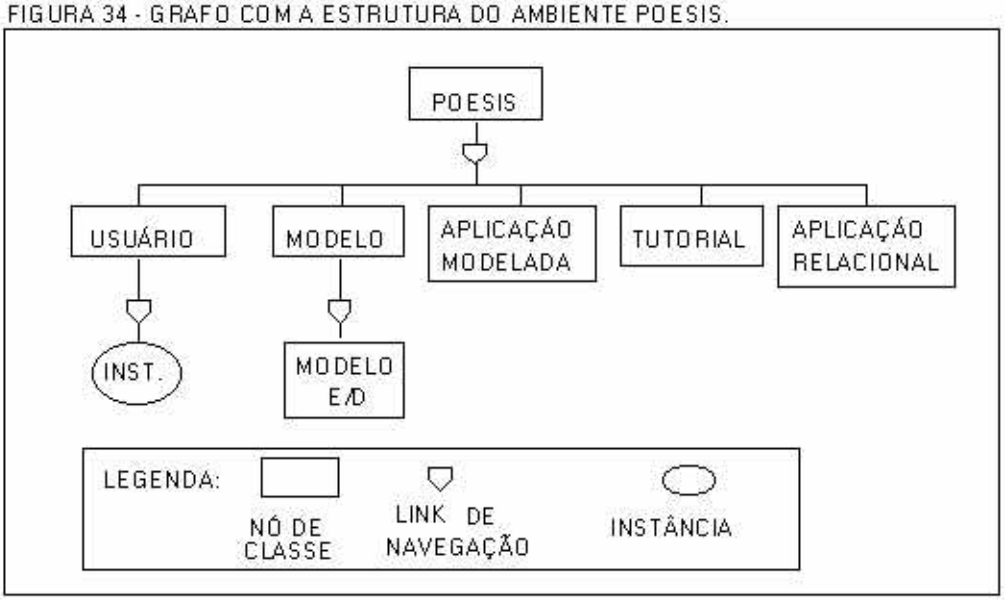
DESCRIÇÃO DO FUNCIONAMENTO DO OBJETO USUÁRIO

O objeto Usuário é utilizado no Ambiente Poesis para administrar as informações relativas aos diferentes tipos de usuários que irão manipula-lo. Esta utilização ocorre geralmente nos seguintes momentos:

(i) na verificação de acesso ao sistema durante a inicialização;
(ii) na manipulação da instância do objeto Usuário durante o processamento no Ambiente;
(iii) no processo de navegação no GOLGO.

Uma das primeiras ações realizadas durante a ativação do Ambiente Poesis, é a verificação do acesso do usuário que está ativando-o, representada pelo item (i). Consiste basicamente, em verificar se o código e a senha de acesso fornecidas pelo usuário estão catalogadas na instância do Objeto Usuário. Isto permitirá restringir o acesso ao sistema por pessoas não autorizadas. Este processo é constituído pelas seguintes ações:

FIGURA 34 - GRAFO COM A ESTRUTURA DO AMBIENTE POESIS.



(i.1) O acesso inicial ao GOLGO é realizado a partir do Nó de Classe denominado POESIS, conforme a figura 34. Este Nó de Classe é o ponto de partida para a navegação através do Ambiente. Este acesso inicial é feito através da execução do método CARGA PARCIAL deste objeto, com o Endereço Relativo Virtual 0 (zero).

(i.2) Após esta carga inicial realiza-se a execução do método OBTER-ERV da variável deste objeto correspondente ao Endereço Relativo Virtual da instância do objeto Link de Navegação associado. Em seguida é feita a carga da instância apontada pelo ERV obtido.

(i.3) Com a instância do Link de Navegação do Nó de Classe POESIS em memória, é executado o método PESQUISA para localização do elemento com informações do Nó de Classe USUÁRIO. Após a localização do elemento, executa-se o método OBTER-ERV que conterá a localização do Nó de Classe desejado.

(i.4) A partir do ERV do Nó de Classe USUÁRIO, é realizada a CARGA PARCIAL desta instância. Com a instância em memória real, efetua-se a execução do método OBTER-ERV do objeto Link de Navegação associado ao Nó de Classe USUÁRIO, para em seguida ser executada a Carga desta Instância de Link de Navegação. O objetivo da carga da instância do Link de Navegação associado ao Nó de Classe USUÁRIO é obter o Endereço Relativo Virtual da instância do Objeto do Sistema Usuário.

(i.5) A partir deste Endereço Relativo Virtual da instância do objeto Usuário, é executado o método PESQUISA PARCIAL deste objeto, cujo objetivo é localizar a posição relativa do nodo correspondente ao usuário que está solicitando o acesso ao Ambiente. Para execução deste método é fornecido o CÓDIGO DO USUÁRIO. Este método realiza uma pesquisa na árvore-B da instância do objeto Usuário sem necessidade de alocá-la totalmente na memória real, ou seja durante o processamento da referida operação ocorre a alocação de memória suficiente para armazenar um nodo.

(i.6) Se o resultado da pesquisa for negativo, significa que o Código do Usuário fornecido não corresponde a nenhum dos nodos pertencentes a árvore. Se for positivo é feita a verificação da senha fornecida com o conteúdo da variável SENHA existente no nodo.

A manipulação direta do objeto Usuário através do Ambiente representada pelo item (ii), somente é realizada pelo Superusuário. Consiste basicamente em incluir, excluir, alterar informações dos usuários. Este processo corresponde a execução de um conjunto de ações constituído pelas ações (i.1) a (i.4) acrescido das ações:

(ii.1) A partir do Endereço Relativo Virtual da instância do objeto Usuário, são executados os métodos respectivos para obtenção dos Endereços Relativos Virtuais das instâncias dos objetos: Dicionário de Objetos, Link de Processos, Índice de Operações Internas, Definição Visual e Restrições, associadas ao Nó de Classe USUÁRIO.

(ii.2) Em seguida ocorrerá a carga de todas estas instâncias para o processamento na memória real. A partir do objeto Definição Visual é mostrada ao Superusuário a Definição Visual "default" do objeto Usuário que permitirá sua interação com o mesmo.

(ii.3) A medida que são ativadas pelo usuário, são exibidas as Definições Visuais que contém as informações sobre as Operações Internas disponíveis dadas pelo objeto Índice de Operações Internas, os processos especiais representados pelo Link de Processos, etc.

(ii.4) Neste momento, o Superusuário poderá executar os métodos disponíveis que irão permitir a inclusão, alteração, consulta e remoção dos usuários. A cada método deste estará associada a correspondente Definição Visual que representa a interface deste objeto.

Durante a navegação no Ambiente através do GOLGO (item (iii)), o nodo correspondente ao usuário que inicializou o sistema é mantido na memória real. Este procedimento garante que as informações referentes ao STATUS estejam sempre disponíveis para manipulação. Desta forma, sempre que for necessário a utilização do STATUS pela Interface do Ambiente para restringir ou permitir o acesso aos recursos disponíveis, será realizada a manipulação desta instância.

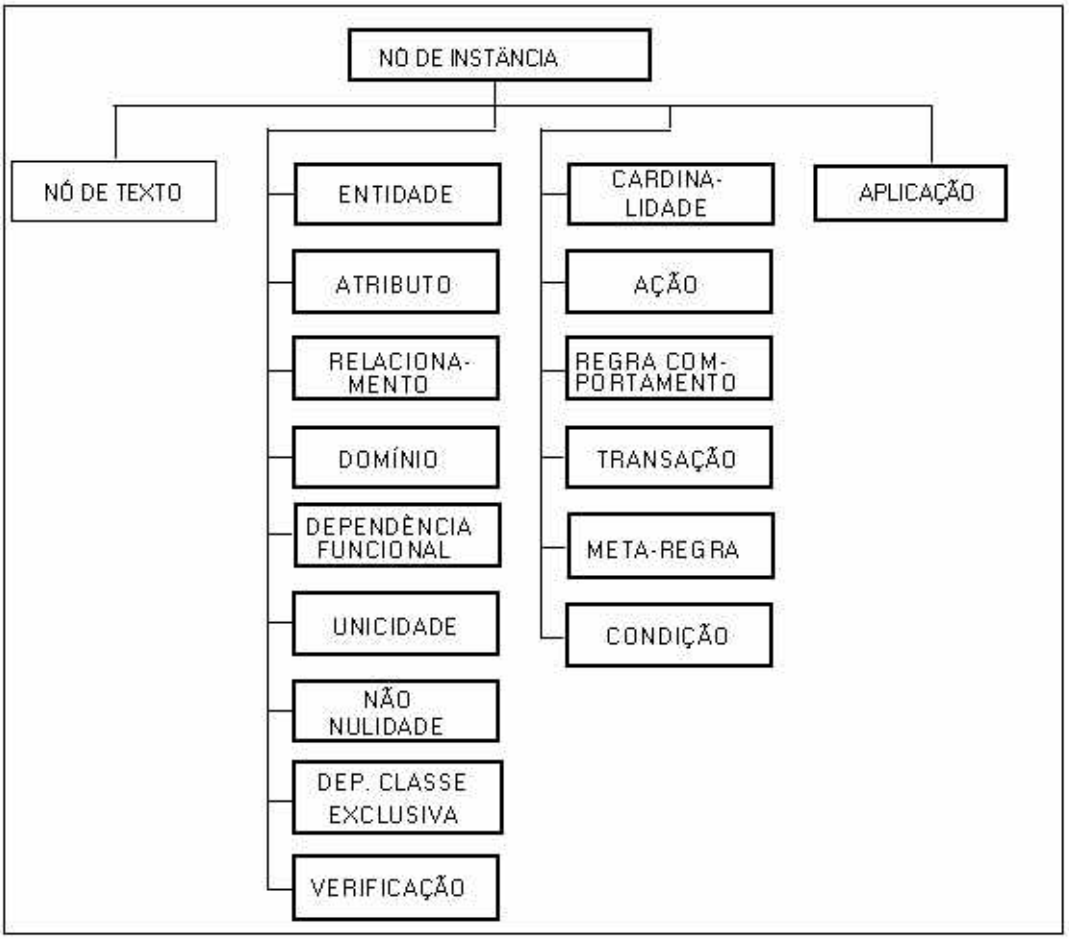
4.2 MODELO ESTÁTICO/DINÂMICO

Uma das ferramentas disponíveis ao usuário que permitirá a concepção de diversas aplicações é o Modelo Estático/Dinâmico [BAND89], desenvolvido no enfoque orientado a objetos. Esta ferramenta para desenvolvimento a nível conceitual dos dados é composta de:

- conjunto de objetos para representação dos conceitos;

- processo de modelagem de dados.

FIGURA 35 - HIERARQUIA DA CLASSE NÓ DE INSTÂNCIA



O conjunto de objetos criado e especificado em [BAND89], correspondem aos objetos utilizados para a representação dos conceitos envolvidos no Modelo E/D, cujas instâncias serão o resultado da captação pelo usuário que está modelando, dos conceitos respectivos de sua aplicação. Estes objetos têm por finalidade então, retratar a realidade da aplicação do usuário, em "instâncias conceituais" que facilitarão uma futura implementação da aplicação modelada. Este conjunto de objetos do Modelo E/D é formado pelas seguintes classes:

- Entidade;
- Atributo;
- Relacionamento;
- Domínio;
- Dependência Funcional;
- Unicidade;
- Não-Nulidade;
- Dependência de Classe Exclusiva;
- Verificação;
- Cardinalidade;
- Ação;
- Condição;
- Regra de Comportamento;
- Transação;
- Meta-Regra.

O processo de modelagem de dados é dado por um conjunto de procedimentos interativos que visam captar as informações relativas à aplicação a ser modelada segundo o Modelo E/D. Estes procedimentos determinam a seqüência de execução das ações, a manipulação dos objetos de gerenciamento e dos objetos do modelo, e a navegação no GOLGO necessária ao processamento no Ambiente Poesis. O processo de modelagem foi implementado no Ambiente Poesis como um Link de Processos associado ao Nó de Classe representativo do Modelo E/D.

Serão especificadas a seguir as classes que compõem o Modelo E/D e o processo de modelagem implementado no Ambiente Poesis. No Apêndice III estão descritas as respectivas estruturas das classes.

4.2.1 CLASSE: ENTIDADE

Esta classe foi criada para implementar o conceito de Entidade especificado em [BAND89]. Implementada como subclasse de Nó de Instância este objeto herda as características estruturais e comportamentais desta super-classe e a hierarquia de classes pode ser visualizada pela figura 35. Suas instâncias são armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância e manipuladas através do processo de ***modelagem de dados*** a ser especificado posteriormente. Estas instâncias representarão as entidades definidas pelo usuário que está modelando a aplicação.

4.2.2 CLASSE: ATRIBUTO

Esta classe foi criada para implementar no Ambiente o conceito de Atributo conforme especificado em [BAND89], sendo implementada como subclasse de Nó de Instância. Desta forma, este objeto herda as características estruturais e comportamentais desta super-classe e a hierarquia de classes pode ser visualizada pela figura 35. As instâncias do objeto Atributo representarão os atributos definidos pelo usuário sendo armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância e manipuladas através do processo de ***modelagem de dados.***

4.2.3 CLASSE: RELACIONAMENTO

A classe Relacionamento foi criada com o objetivo de implementar no Ambiente o conceito de Relacionamento especificado em [BAND89], sendo desenvolvida como uma subclasse de Nó de Instância. Desta maneira, este objeto herdou as características estruturais e comportamentais desta super-classe (ver a hierarquia de classes na figura 35) . As instâncias do objeto Relacionamento representarão os Relacionamentos definidos pelo usuário sendo armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância e manipuladas através do processo de ***modelagem de dados.***

4.2.4 CLASSE: DOMÍNIO

Esta classe foi criada com o objetivo de implementar o conceito de Domínio especificado em [BAND89] no Ambiente Poesis. A classe Domínio foi implementada como uma subclasse de Nó de Instância e desta forma herdou as características estruturais e comportamentais desta super-classe cuja hierarquia de classes pode ser vista pela figura 35) . As instâncias do objeto Domínio representarão os Domínio definidos para os objetos do Modelo E/D que serão utilizados pelo usuário para a concepção das aplicações, sendo armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância e manipuladas através do processo de ***modelagem de dados.***

4.2.5 CLASSE: DEPENDÊNCIA FUNCIONAL

Esta classe foi criada para implementar o conceito de Dependência Funcional especificado em [BAND89]. A maneira de implementa-lo como subclasse de Nó de Instância permitiu que este objeto herdasse as características estruturais e comportamentais desta super-classe. Suas instâncias são armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância e manipuladas através do processo de ***modelagem de dados*** a ser especificado posteriormente. Estas instâncias representarão as

Dependência Funcionais definidas pelo usuário que está modelando a aplicação.

4.2.6 CLASSE: UNICIDADE

Esta classe foi criada para implementar o conceito do tipo de restrição denominado Unicidade criado para o Modelo E/D, especificado em [BAND89]. Esta classe foi implementada como subclasse de Nó de Instância, o que permitiu a este objeto a herança das características estruturais e comportamentais desta super-classe. Portanto, suas instâncias são armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância e manipuladas através do processo de ***modelagem de dados.*** Estas instâncias representarão as restrições de Unicidade definidas pelo usuário que está modelando a aplicação.

4.2.7 CLASSE: NÃO-NULIDADE

Não-Nulidade é um tipo de restrição criada para o Modelo E/D, especificada em [BAND89], para ser utilizada na concepção de Aplicações por usuários do Ambiente Poesis. A classe Não-Nulidade foi também implementada como subclasse de Nó de Instância. Isto permitiu que este objeto herdasse as características estruturais e comportamentais desta super-classe. Logo, suas instâncias são armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância, e representarão as restrições de Não-Nulidade definidas pelo usuário para uma determinada aplicação.

4.2.8 CLASSE: DEPENDÊNCIA DE CLASSE EXCLUSIVA

Esta classe foi criada para implementar o conceito de Dependência de Classe Exclusiva especificado em [BAND89], fazendo parte da hierarquia de classes (ver figura 35) ao ser implementada como subclasse de Nó de Instância. Este objeto do Modelo E/D, herda desta forma, as características estruturais e comportamentais desta super-classe. Suas instâncias são armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância e manipuladas através do processo de ***modelagem de dados*** a ser especificado posteriormente. Estas instâncias representarão as Dependência de Classe Exclusivas definidas pelo usuário que está modelando a aplicação.

4.2.9 CLASSE: VERIFICAÇÃO

Esta classe foi criada para implementar no Ambiente o conceito de Verificação conforme especificado em [BAND89], sendo implementada como subclasse de Nó de Instância. Desta forma, este objeto herda as características estruturais e comportamentais desta super-classe. As instâncias do objeto Verificação representarão as restrições do tipo Verificação definidas pelo usuário para a aplicação, sendo armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância e manipuladas através do processo de ***modelagem de dados.***

4.2.10 CLASSE: CARDINALIDADE

A classe Cardinalidade foi criada para implementar o conceito de Cardinalidade especificado em [BAND89] no Ambiente Poesis, sendo pela hierarquia de classes do GOLGO uma subclasse de Nó de Instância. Desta forma, o objeto Cardinalidade herdou as características estruturais e comportamentais desta super-classe. As instâncias do objeto Cardinalidade representarão os diversos tipos de cardinalidade definidos segundo o Modelo E/D, que serão utilizados pelo usuário para a concepção das aplicações. As instâncias do objeto Cardinalidade são armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância e manipuladas através do processo de ***modelagem de dados.***

4.2.11 CLASSE: AÇÃO

Esta classe foi criada para implementar no Ambiente o conceito de Ação criado para definir a

parte dinâmica de uma aplicação, conforme especificado em [BAND89]. A classe Ação foi implementada segundo a hierarquia de classes do GOLGO (ver figura 35), como subclasse de Nó de Instância. Desta forma, este objeto herda as características estruturais e comportamentais desta super-classe.

As instâncias do objeto Ação serão usadas para representação das Ações definidas pelo usuário para uma determinada aplicação, sendo armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância e manipuladas através do processo de ***modelagem de dados.***

4.2.12 CLASSE: CONDIÇÃO

Esta classe foi criada para implementar o conceito de Condição conforme especificado em [BAND89], para ser utilizado no processo de Modelagem de Dados, visando o desenvolvimento conceitual de aplicações do usuário do Ambiente Poesis. A classe Condição, foi desta maneira implementada segundo a hierarquia do GOLGO, como subclasse de Nó de Instância (ver figura 35). Assim, este objeto herdou as características estruturais e comportamentais de Nó de Instância, e suas instâncias são armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância. As instâncias do objeto Condição serão utilizadas na composição de outros objetos do Modelo E/D a serem definidos pelo usuário para a aplicação, representando expressões condicionais do objeto Regra de Comportamento.

4.2.13 CLASSE: REGRA DE COMPORTAMENTO

A classe Regra de Comportamento foi criada com o objetivo de implementar no Ambiente Poesis, o conceito de Regra de Comportamento especificado em [BAND89] para o Modelo E/D. Esta classe foi desenvolvida como uma subclasse de Nó de Instância. Desta maneira, este objeto herdou as características estruturais e comportamentais desta super-classe (ver a hierarquia de classes na figura 35) . As instâncias do objeto Regra de Comportamento representarão as Regras de Comportamento a serem definidas pelo usuário no momento em que estiver executando a modelagem de dados para a criação de uma aplicação. Estas instâncias serão armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância.

4.2.14 CLASSE: TRANSAÇÃO

Esta classe foi criada para implementar o conceito de Transação especificado em [BAND89]. Esta classe implementada segundo a hierarquia de classes do GOLGO, como uma subclasse de Nó de Instância permitiu que este objeto herdasse as características estruturais e comportamentais desta super-classe. Suas instâncias são armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância e representarão as Transações definidas pelo usuário modelador da aplicação.

4.2.15 CLASSE: META-REGRA

A classe Meta-Regra foi criada para representar no enfoque orientado a objetos, o conceito de Meta-Regra criado para o Modelo E/D, especificado em [BAND89]. Esta classe foi implementada como subclasse de Nó de Instância, o que permitiu a este objeto a herança das características estruturais e comportamentais desta super-classe. Portanto, suas instâncias são armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância, e representarão as Meta-Regras a serem definidas pelo usuário que modelará a aplicação.

4.2.16 DESCRIÇÃO DO PROCESSO DE MODELAGEM DE DADOS

O processo de Modelagem de Dados é uma ferramenta de fundamental importância para o Ambiente Poesis. A partir deste processo o usuário poderá representar o conhecimento do universo de

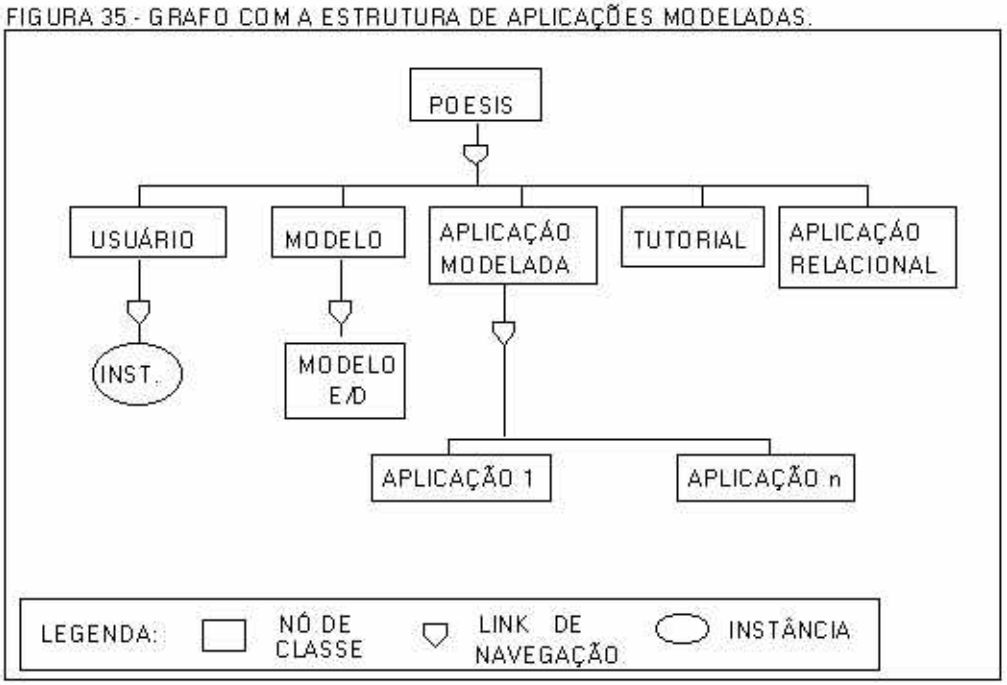
sua aplicação através dos conceitos do Modelo E/D que são mapeados em uma estrutura flexível e capaz de representar a propriedade de composição de objetos.

A modelagem de dados consiste basicamente em organizar de forma sistemática os dados e situações relevantes de uma aplicação a partir de um enfoque que permita representar as características estruturais, relativas ao dado, e as características comportamentais, que correspondem a dinâmica da aplicação.

No Ambiente Poesis, o usuário poderá desenvolver suas aplicações a partir do processo de modelagem usando a metodologia de concepção do Modelo E/D especificada em [BAND89]. O processo de modelagem consiste em fornecer a metodologia de concepção de aplicações do Modelo E/D de forma interativa, onde o usuário é conduzido a fornecer as informações associadas à cada conceito, ou objeto, numa determinada seqüência lógica. O usuário poderá escolher os caminhos de navegação para informar as características de sua aplicação de acordo com suas necessidades e sua maneira de modelar. Por exemplo, o processo de modelagem possui a seguinte seqüência lógica:

- Definição das propriedades estruturais;
- Definição das Restrições de Integridade;
- Definição das propriedades comportamentais;
- Definição do Mecanismo de Herança.

FIGURA 35 - GRAFO COM A ESTRUTURA DE APLICAÇÕES MODELADAS.



Esta seqüência induz o usuário a segui-la. Porém, a forma como o processo de modelagem foi implementado permite a total liberdade de escolha pelo fato de que existe um ***Mecanismo de Composição*** que permite seguir a sintaxe proposta em [BAND89], sem necessariamente forçar o usuário a especificar sua aplicação nesta seqüência.

O processo de Modelagem de Dados conduz o usuário a gerar no Ambiente uma Aplicação Modelada (ver seção 4.3) a partir do processamento efetuado pela interação com os objetos do Modelo E/D especificados na seção 4.2.

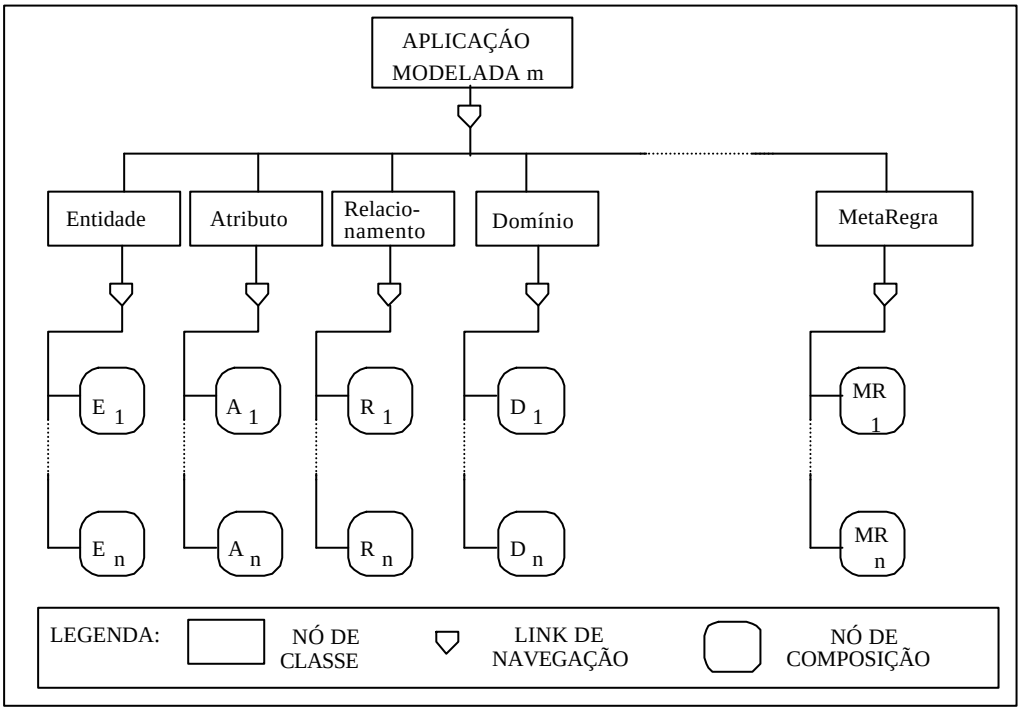
Uma Aplicação Modelada pode ser compreendida como sendo um conjunto de instâncias dos objetos do Modelo E/D criadas a partir do processo de Modelagem. Entretanto, uma Aplicação Modelada é criada usando também uma das características muito importantes deste modelo denominada de MECANISMO DE COMPOSIÇÃO. O Mecanismo de Composição consiste de

representar a composição de objetos conforme a sintaxe descrita em [BAND89] de forma a permitir que o usuário possa através da interface de cada objeto do Modelo E/D, fazer as associações (ou links) entre os objetos de forma interativa.

Para implementar os diversos conceitos e mecanismos envolvidos no processo de Modelagem, o GOLGO utiliza uma forma muito peculiar de representação das Aplicações Modeladas. Uma Aplicação Modelada no GOLGO, corresponde a uma subgrafo implementado a partir dos Objetos de Gerenciamento:

- Nó de Classe;
- Nó de Composição;
- Nó de Instância;
- Link de Navegação;
- Link de Composição;
- Dicionário de Objetos;
- Versões;
- Restrições.

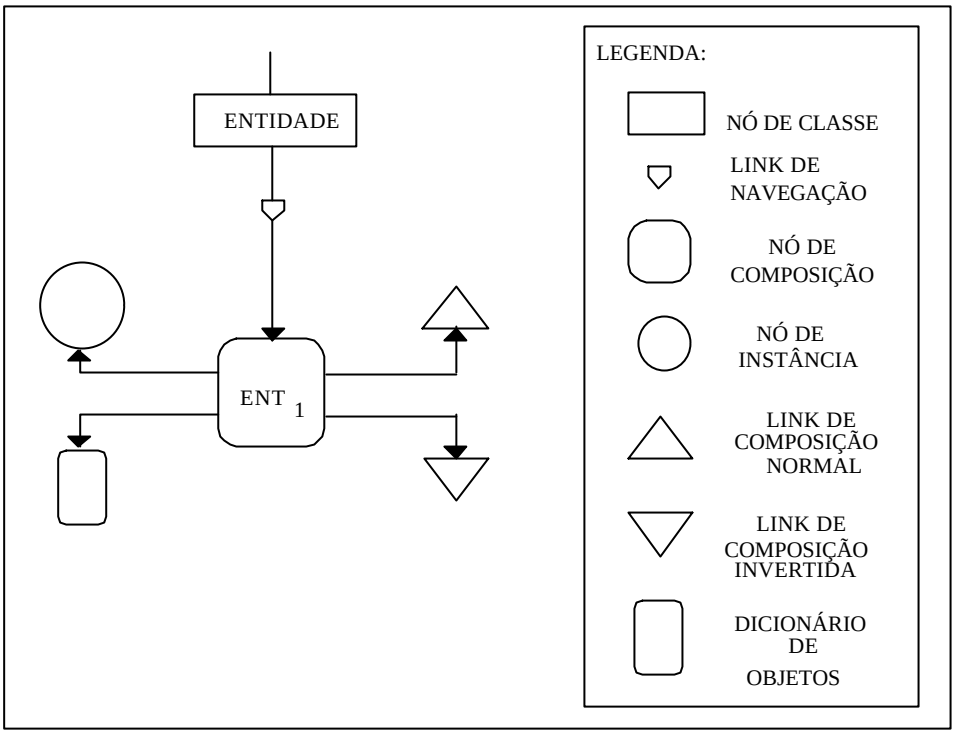
FIGURA 36 - Subgrafo com a estrutura de uma Aplicaçáo Modelada m.



O subgrafo de uma Aplicação Modelada pode ser visualizado pela figura 35. Na figura 36 percebe-se que para cada tipo de objeto do Modelo E/D existe um Nó de Classe representando a respectiva classe. Existe portanto, um Nó de Classe representando as Entidades da Aplicação Modelada, outro Nó de Classe representando os Atributos, etc. A instância propriamente dita de cada objeto do Modelo E/D é representada pelos seguintes objetos:

- Nó de Composição;
- Nó de Instância;
- Link de Composição;
- Dicionário de Objetos.

FIGURA 37 - Diagrama da representação de instâncias dos objetos do Modelo E/D no subgrafo da



A figura 37 apresenta a maneira pela qual estão representadas as instâncias dos objetos do Modelo E/D. Cada Objeto de Gerenciamento tem a sua função na representação de uma instância. O Nó de Composição é usado para recuperar as instâncias dos outros Objetos de Gerenciamento. O Nó de Instância é usado para recuperar e manipular a instância propriamente dita, já que os objetos do Modelo E/D são subclasses deste objeto. O Link de Composição é usado para identificar e acessar as instâncias dos objetos do Modelo E/D que participam do Mecanismo de Composição. O Dicionário de Objetos é usado para identificação das instâncias conforme especificação definida em [BAND89].

A navegação é realizada a partir do objeto Link de Navegação que permite o acesso às instâncias do Nó de Classe e do Nó de Composição que participam do subgrafo da Aplicação Modelada.

DESCRIÇÃO DO MECANISMO DE FUNCIONAMENTO

O processo de Modelagem é ativado pelo usuário a partir da Definição Visual dos Links de Processos associados ao Nó de Classe que representa o Modelo E/D. O usuário ao ativar o Link de Processos denominado MODELAGEM, dá inicio a execução do processo especial definido no Ambiente.

O processo de Modelagem é compostos dos seguintes módulos:

- Inicialização;
- Seleção;
- Criação;
- Associação ou Composição (através de Seleção);
- Alteração;
- Remoção;
- Consulta;
- Impressão;
- Finalização;

O módulo de Inicialização consiste da execução das seguintes ações:

(1) Ativação da Definição Visual que capta o Nome da Aplicação Modelada existente ou uma nova, conforme o caso, efetuando também a validação sobre o Identificador da aplicação.
(2) Carga das instâncias associadas a cada objeto do Modelo E/D representado por instância de Nó de Classe, dos seguintes Objetos de Gerenciamento: Definição Visual, Índice de Operações Internas Restrições. Esta ação irá permitir a utilização das Definições Visuais de cada objeto do Modelo E/D (Janelas, Ícones, Entradas de Dados formatadas, Saídas pré-formatadas, etc.), dos respectivos índices para ativação dos métodos de cada objeto, e das restrições pré-definidas.
(3) Criação das instâncias de Nó de Classe que representam os tipos de objetos.
(4) Criação das instâncias de Link de Navegação associadas às instâncias criadas pela ação (3).

O Módulo de Seleção consiste da ativação das Definições Visuais que permitem ao usuário que está modelando a escolha do objeto do Modelo E/D desejado e a operação interna a ser executada.

O Módulo Criação contém basicamente o procedimento genérico de criação para todos os tipos de objeto do Modelo E/D. Para isto, alguns parâmetros deverão ser enviados para especificar o objeto do Modelo E/D que será criado. Este procedimento genérico consiste da execução das seguintes ações:

(1) Ativação da Definição Visual de Criação do objeto solicitado;
(2) Criação da instância do objeto Nó de Composição que irá representar a instância do objeto do Modelo E/D.
(3) Criação da instância dos objetos Dicionário de Objetos e Índice do Dicionário de Objetos conforme os dados informados através da Definição Visual resultante da ação (1) e validados.
(4) Ativação do método de criação correspondente a partir da instância do objeto Índice de Operações Internas selecionado através de Definição Visual pelo usuário com as informações fornecidas pela ação (1).
(5) Criação da instância do objeto do Modelo E/D referenciado.
(6) Transferência para a memória persistente (SALVA) das instâncias do Dicionário de Objetos, Índice do Dicionário de Objetos e do objeto do Modelo E/D (armazenada como Nó de Instância).
(7) Atualização das informações referentes aos Endereços Relativos Virtuais das instâncias do Dicionário de Objetos e do objeto do Modelo E/D (como Nó de Instância) na instância do Nó de Composição.
(8) Inserção de um elemento na instância do objeto Link de Navegação correspondente ao tipo de objeto do Modelo E/D informado, para registrar o Endereço Relativo Virtual e a Posição Relativa do Nó de Composição criado na ação (2).
(9) Desalocação das instâncias do Dicionário de Objetos e do objeto do Modelo E/D criado.

O Módulo Associação ou Composição corresponde a execução dos procedimentos necessários a manipulação do Mecanismo de Composição de objetos. Para cada tipo de objeto do Modelo E/D existem regras que permitem efetuar a composição. Estas regras estão definidas para cada objeto através da Linguagem de Declarativa dos Objetos (LDO) [FONS87, BAND89]. Portanto existem para cada tipo de objeto submódulos de composição onde estas regras estão implementadas. Em termos gerais, este módulo corresponde a execução das seguintes ações:

(1) Ativação da Definição Visual do objeto do Modelo E/D (Objeto Composto) cuja composição está sendo solicitada, correspondente à execução deste módulo.
(2) Pesquisa na respectiva instância do objeto Link de Navegação referente ao tipo de objeto que se deseja associar (Objeto Componente), para recuperar os identificadores (nomes) das instâncias já criadas para associação.
(3) Ativação da Definição Visual contendo a lista de identificadores para seleção pelo usuário.
(4) Carga das instâncias respectivas do Link de Composição (Normal e Invertido) dos objetos selecionados (Objetos Componentes e Objeto Composto) a partir dos identificadores, pelo usuário.
(5) Inserção de elementos nas instâncias do Link de Composição (Normal) do objeto composto com

informações referentes aos objetos componentes selecionados pelo usuário.
(6) Inserção de elementos nas instâncias do Link de Composição (Invertido) dos objetos componentes com informações referentes ao objeto composto.
(7) Transferência para a memória persistente (SALVA) das instâncias do Link de Composição (Normal e Invertido) dos objeto composto e componentes do Modelo E/D.
(8) Atualização das informações referentes aos Endereços Relativos Virtuais das instâncias do Link de Composição (Normal e Invertido) obtidos na execução da ação (7), nas instâncias do Nó de Composição dos objetos composto e componentes.
(9) Desalocação das instâncias do Link de Composição (Normal e Invertido) dos objetos envolvidos no processamento do módulo (composto e componente).

O Módulo de Alteração tem por objetivo permitir que o usuário modifique as informações que foram introduzidas durante a execução dos procedimentos do Módulo Criação. Consiste basicamente na execução das ações:

(1) Carga da instância do objeto do Modelo E/D referenciado, a partir do Nó de Composição.
(2) Ativação da Definição Visual de Alteração com as informações do objeto solicitado;
(3) Execução dos métodos que alteram as informações da instância do objeto do Modelo E/D.
(4) Transferência para a memória persistente (SALVA) da instância do objeto do Modelo E/D (armazenada como Nó de Instância).
(5) Atualização das informações referentes aos Endereço Relativo Virtual da instância do objeto do Modelo E/D (como Nó de Instância) na instância do Nó de Composição.
(6) Desalocação da instância do objeto do Modelo E/D alterado.

O Módulo Remoção contém também procedimentos genéricos de remoção para todos os tipos de objeto do Modelo E/D. Estes procedimentos genéricos correspondem a execução das seguintes ações:

(1) Ativação da Definição Visual de Remoção do objeto solicitado;
(2) Remoção da instância do objeto Nó de Composição que representava a instância do objeto do Modelo E/D.
(3) Remoção das instância dos objetos Dicionário de Objetos e Índice do Dicionário de Objetos conforme os dados informados através da Definição Visual resultante da ação (1) e validados.
(4) Ativação do método de remoção correspondente a partir da instância do objeto Índice de Operações Internas selecionado através de Definição Visual pelo usuário com as informações fornecidas pela ação (1).
(5) Remoção da instância do objeto do Modelo E/D referenciado.
(6) Remoção do elemento na instância do objeto Link de Navegação correspondente ao tipo de objeto do Modelo E/D informado.
(7) Desalocação das instâncias do Dicionário de Objetos e do objeto do Modelo E/D removido.

O Módulo de Consulta possui duas modalidades: Consulta de Instância (através da navegação pelo subgrafo), e Consulta Gráfica. A Consulta de Instância é realizada pela ativação das Definições Visuais dos objetos com informações obtidas de suas instância. A Consulta Gráfica corresponde a ativação da Definição Visual na forma de ícones interligados para efetuar a representação gráfica da Aplicação Modelada.

O Módulo Impressão consiste da ativação de Definições Visuais que enviam para a impressora acoplada ao equipamento, as informações referentes a uma determinada Aplicação Modelada.

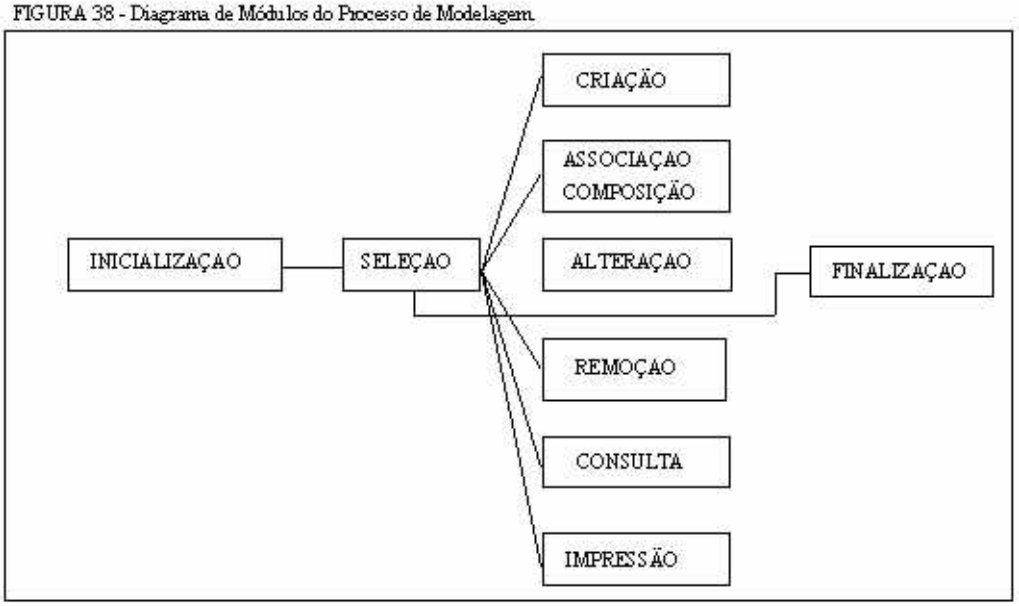
O Módulo Finalização consiste da execução das ações que finalizam o processo de Modelagem de Dados, que são:

(1) Transferência para a memória persistente (SALVA) das instância do objeto Nó de Composição.

(2) Transferência para a memória persistente (SALVA) das instância do objeto Link de Navegação.
(3) Transferência para a memória persistente (SALVA) das instâncias associadas a cada objeto do Modelo E/D representado por instâncias de Nó de Classe.
(4) Atualização das informações referentes aos Endereço Relativo Virtual da instância dos objetos Link de Navegação e Nó de Composição na instância do Nó de Classe que representa a Aplicação Modelada.
(5) Desalocação da Memória Real das instâncias processadas nas ações (1) a (4).

O processamento da Modelagem de Dados implementada para o Modelo E/D segue basicamente a representação dada pelo grafo na figura 38.

FIGURA 38 - Diagrama de Módulos do Processo de Modelagem.



4.3 APLICAÇÃO MODELADA

Uma das principais funções do Ambiente Poesis é o gerenciamento das aplicações modeladas pelos usuários. Esta função consiste em manter as instâncias dos objetos do Modelo E/D, definidos através do processo de modelagem de dados descrito anteriormente, em uma estrutura em forma de grafo, que permita ao processo de ***Modelagem de Dados***, a recuperação e manipulação destas aplicações, de acordo com as regras de composição definidas para o Modelo E/D [BAND89]. O controle das versões também é realizado para as aplicações modeladas conforme especificado no capítulo 2. Este controle será descrito nesta seção.

Uma determinada Aplicação Modelada no GOLGO é representada pelos Objetos de Gerenciamento descritos na seção 3.2.2.2. O Grafo do Ambiente Poesis, apresentado pela figura 39 revela a existência de um nó denominado APLICAÇÃO MODELADA, ponto de partida para a manipulação de todas as Aplicações Modeladas concebidas durante o processamento do Ambiente. Cada Aplicação Modelada presente no grafo do Ambiente corresponde a um subgrafo que poderá ser acessado pelo processo de navegação do GOLGO. O nó inicial, que representa a classe de objetos do sistema Aplicação Modelada, corresponde a uma instância de Nó de Classe. Associada a esta instância, existe uma instância do objeto de gerenciamento Link de Navegação, contendo a lista de todas as aplicações modeladas presentes no sistema. De acordo com as características do objeto Link de Navegação, esta instância possibilitará o acesso à cada Nó de Classe que está representando o nó inicial de cada subgrafo, pelo mecanismo de controle de acesso, ou seja, apenas as aplicações definidas por um determinado usuário poderá ou não estar disponível para os demais usuários.

Um determinado subgrafo, que representa a aplicação modelada é formado por instâncias dos seguintes Objetos de Gerenciamento:

- Nó de Classe;
- Nó de Composição;
- Nó de Instância;
- Link de Navegação;
- Link de Composição;
- Dicionário de Objetos;
- Versões;
- Restrições.

Em geral os subgrafos de aplicação modelada possuem a forma apresentada na figura 36. Aparentemente complexa, esta estrutura permite a representação de todos os conceitos do Modelo E/D de forma otimizada, conforme será visto posteriormente.

No subgrafo representativo da aplicação modelada, o Objeto de Gerenciamento Nó de Classe possui as seguintes funções:

(1) É utilizado para representar o nó inicial do subgrafo, e desta maneira, permitirá :
- que a aplicação modelada seja identificada no Dicionário de Objetos;
- o acesso aos outros objetos de gerenciamento componentes da aplicação, tais como: Link de Navegação, Nó de Composição, Versões, Restrições.
- o acesso ao bloco de instâncias do objeto Nó de Composição.

(2) É usado para representar os conjuntos de instâncias de Nó de Composição referentes a cada tipo de objeto do Modelo E/D, ou seja, para cada objeto do Modelo E/D, existirá um Nó de Classe representando sua classe na aplicação. Por exemplo, para o objeto do Modelo E/D, ENTIDADE, existirá na aplicação modelada CONTROLE ACADÊMICO, um Nó de Classe que permitirá o acesso à todas as instâncias do objeto Entidade definidas para esta aplicação. Neste caso, o Nó de Classe teria a seguintes semântica: "ENTIDADES da aplicação CONTROLE ACADÊMICO".

O objeto de gerenciamento Nó de Composição, no subgrafo da Aplicação Modelada possui a função de representar cada instância de objetos do Modelo E/D de uma determinada aplicação permitindo:

- o acesso ao Dicionário de Objetos, onde esta instância está identificada;
- o acesso à instância propriamente dita, ou seja, o acesso às informações armazenadas por um determinado objeto do Modelo E/D;
- o acesso às outras instâncias de Nó de Composição associadas a um determinado objeto do Modelo E/D, que participam em sua composição. Em outras palavras, o Nó de Composição permite a localização de todos os objetos que compõem um determinado objeto do Modelo E/D, através da instância associada do objeto Link de Composição do tipo normal.
- o acesso aos objetos em cuja composição, um determinado objeto do Modelo participa, através da instância do objeto Link de Composição do tipo invertida.

O objeto Nó de Instância no subgrafo da aplicação modelada, será usado para a recuperação das informações peculiares de cada instância dos objetos do Modelo E/D, que participa da definição da aplicação.

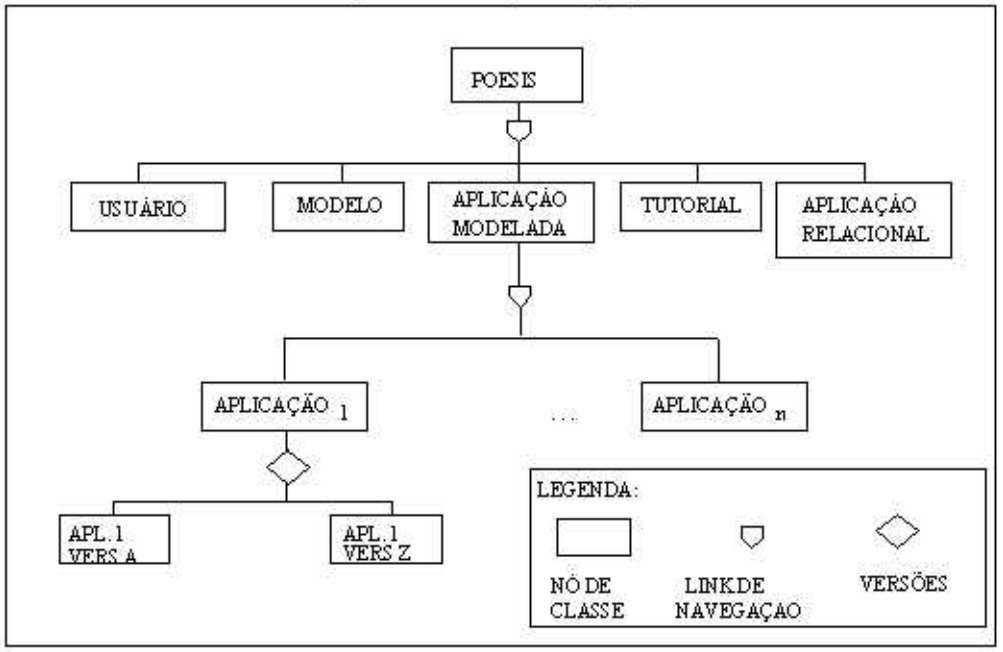
O objeto de gerenciamento Link de Navegação possui as seguintes funções no subgrafo da aplicação modelada:

(1) Permite o acesso às instâncias de Nó de Classe que representam os conjuntos de instâncias de Nó de Composição referentes a cada tipo de objeto do Modelo E/D.
(2) Permite o acesso à cada Nó de Composição que representam as instâncias especiais dos objetos do Modelo E/D. O termo *instâncias especiais* se refere ao fato de que é necessária a representação das propriedades dos objetos do Modelo E/D no que se refere à composição de objetos descrita no capítulo 2.

No subgrafo representativo de uma determinada Aplicação Modelada, um tipo de objeto de gerenciamento é de fundamental importância para permitir a manipulação pelo sistema da composição entre objetos do Modelo E/D. Este objeto é o Link de Composição que possui as seguintes funções:

(1) Permite associar as instâncias de objetos do Modelo E/D denominados ***compostos*** com as respectivas instâncias dos objetos ***componentes***. Isto é feito através de uma instância do objeto Link de Composição (associada ao Nó de Composição que representa o objeto composto) cuja característica é manter uma lista de referências que permitem localizar as instâncias de Nó de Composição representantes dos objetos componentes. Para esta finalidade é usado o tipo de Link de Composição Normal.

FIGURA 39 - Grafo ilustrando o uso do Objeto Versões no subgrafo da Aplicação Modelada.



(2) Permite a localização dos objetos ***compostos*** onde um determinado objeto ***componente*** participa na composição. Para isto é utilizada uma instância do objeto Link de Composição do tipo Invertida (associada ao Nó de Composição que representa o objeto componente) que mantém uma lista contendo a localização das instâncias de Nó de Composição representantes dos objetos ***compostos***.

O Objeto de Gerenciamento Dicionário de Objetos é usado para armazenar as informações que permitem identificar cada instância de objeto do Modelo E/D, além da identificação da própria aplicação modelada.

O objeto de gerenciamento Versões é utilizado para controlar as diversas versões criadas pelo usuário de uma determinada Aplicação Modelada. Para que isto seja possível, o sistema mantém associada à instancia do Nó de Classe que representa o nó inicial da aplicação, uma instância do objeto Versões que permitirá a localização das versões representadas pelos respectivos subgrafos. A figura 39 apresenta um exemplo que ilustra o uso do objeto Versões e os subgrafos de aplicação modelada de cada versão que a aplicação possui. Cada subgrafo é como se fosse uma outra aplicação modelada. Quando um determinado usuário solicita a criação de uma versão para uma determinada aplicação, o sistema efetua uma cópia do respectivo subgrafo da aplicação modelada, duplicando todas as instâncias criadas, pela duplicação do subgrafo. Esta cópia corresponde ao conceito de cópia profunda (*deep copy)* [MEYE88] realizada em modelos orientados a objetos com estruturas de dados complexas. A partir deste novo subgrafo, o usuário poderá efetuar as modificações que julgar necessárias, e estas mudanças farão parte da nova versão da aplicação modelada.

O objeto de gerenciamento Restrições participa da classe de Aplicações Modeladas, a partir do Nó de Classe que representa esta classe. Uma instância de Restrições poderá ser criada pelo Administrador do Ambiente visando restringir o número de elementos usados no processo de modelagem. Esta instância estará associada ao Nó de Classe inicial denominado Aplicação Modelada (ver figura 35) e possuirá o número máximo permitido de criação das instâncias dos objetos: Nó de Composição, Nó de Instância, Link de Navegação, Link de Composição Normal e Invertida, e Versões. Estas restrições poderão ser modificadas de acordo com a conveniência do Administrador do Ambiente Poesis.

4.4 CLASSE: TUTORIAL

O objeto do ambiente denominado TUTORIAL tem por finalidade manter uma documentação interativa visando:

- O aprendizado por parte dos usuários que manipulam os Objetos do Ambiente dos recursos disponíveis;
- A documentação de Aplicações Modeladas pelo usuário;
- A ajuda on-line solicitada pelos usuários do Ambiente.

Sua concepção foi baseada nos conceitos da filosofia de hipertexto [CONK87, MEIR89], para tornar seu uso coerente com a forma de manipulação do Ambiente Poesis, além da flexibilidade exigida no processamento deste tipo de objeto, no que se refere ao armazenamento das informações e navegação. A classe Tutorial consiste de um conjunto de objetos com funções específicas que funcionam de forma integrada. Estes objetos são os seguintes:

- Hiperdocumento;
- Nó_de_Texto;
- Link.

O objeto Hiperdocumento tem por objetivo manipular os diversos temas, ou assuntos presentes no Ambiente. Cada tema descreve de forma objetiva as informações pertinentes a uma determinada função, objeto, processo ou Aplicação Modelada consultadas por usuários que desejam aprender sua utilização. Um determinado tema poderá utilizar as informações de outros temas através do processo de referenciação, conforme será visto posteriormente. Alguns temas estarão disponíveis no Ambiente, tais como:

- O Ambiente Poesis;
- Modelo E/D;
- Como usar o Tutorial do Ambiente;
- etc.

O objeto Nó_de_Texto corresponde a um determinado bloco, ou módulo de informações que descrevem uma parte de um tema. Um tema, ou Hiperdocumento é formado por Nó_de_Textos interligados de forma a permitir o acesso às informações ali contidas. Um Nó_de_Texto poderá conter um parágrafo, uma seção, ou mesmo um capítulo de um tema. Seu conteúdo é definido pelo usuário, porém a eficiência do objeto Tutorial dependerá da modularidade e fragmentação dos textos a serem criados, para permitir o uso da referenciação. A referenciação é um termo usado neste trabalho que compreende a capacidade que os Hiperdocumentos têm de permitir o acesso a Nó_de_Textos pertencentes à outros Hiperdocumentos.

O objeto responsável pela referenciação é o LINK. Consiste basicamente de uma lista de palavras ou referências associadas a um determinado Nó_de_Texto que permitem a recuperação e manipulação de outros Nó_de_Textos. As referências são palavras que aparecem no textos, quando são visualizados pelo usuário, em brilho intensificado, e a medida que o usuário posiciona o cursor sobre as mesmas e pressionam um tecla especial (ou o botão do mouse), o Nó_de_Textos associados

àquelas palavras são exibidos em Definições Visuais pela Interface do Objeto Tutorial.

Serão especificadas a seguir as classes dos objetos que compõem o Objeto do Ambiente Tutorial e a descrição de seu funcionamento. A estrutura das classes e suas operações encontram-se descritas no Apêndice III.

4.4.1 CLASSE: HIPERDOCUMENTO

Esta classe foi criada para implementar o conceito de Hiperdocumento especificado no início desta seção. Implementada como subclasse de Árvore-B Genérica este objeto herda as características estruturais e comportamentais desta super-classe. Suas instâncias são armazenadas na Área de Armazenamento Virtual do Objeto Tutorial e manipuladas através do processo de ***navegação do Objeto Tutorial*** a ser especificado posteriormente. Estas instâncias representarão os Hiperdocumentos existentes no Ambiente e os novos a serem definidos pelo usuário que está modelando a aplicação.

DESCRIÇÃO DO FUNCIONAMENTO DO OBJETO HIPERDOCUMENTO

De uma maneira geral o objeto Hiperdocumento é utilizado nos seguintes momentos:

- No acesso inicial ao Objeto Tutorial, para mostrar ao usuário através da Definição Visual correspondente a Janela de seleção, os Hiperdocumentos disponíveis.
- No processo de referenciação para permitir a criação de Links (referências) em outros hiperdocumentos.

Nos dois momentos descritos acima, a instância de Hiperdocumento (árvore de Hiperdocumentos) é manipulada na modalidade parcial, ou seja, um nodo correspondente ao Hiperdocumento desejado é trazido para a Memória Real para processamento. A pesquisa é realizada também de forma parcial.

4.4.2 CLASSE: NÓ_DE_TEXTO

Esta classe foi criada para implementar no Ambiente o conceito de Hipertexto, sendo implementada como subclasse de Nó de Instância. Desta forma, este objeto herda as características estruturais e comportamentais desta super-classe e a hierarquia de classes pode ser visualizada pela figura 35. As instâncias do objeto Nó_de_Texto representarão os textos existentes para o Objeto Tutorial e aqueles que serão eventualmente definidos pelo usuário sendo armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância e manipuladas através do processo de ***Navegação do Objeto Tutorial.*** Este objeto funciona de forma integrada ao objeto Hiperdocumento.

DESCRIÇÃO DO FUNCIONAMENTO DO OBJETO NÓ DE TEXTO

O objeto Nó_de_Texto é usado durante a manipulação de um Hiperdocumento para: criação, alteração, consulta e remoção dos Nós de textos associados. A criação, alteração e remoção de instâncias de Nó_de_Texto ocorrem nos processos de ***Edição e Referenciação do Objeto Tutorial,*** enquanto que a consulta é realizada no processo de ***Navegação***.

Em cada um destes processos a manipulação de uma instância do objeto Nó_de_Texto corresponde a execução das seguintes ações:

(1) Carga da instância (quando houver) para a Memória Real;
(2) Execução de uma ou mais operações disponíveis na classe: operações internas e operações herdadas;
(3) Salva (ou transferência) da instância da Memória Real para a Área de Armazenamento na memória persistente.

4.4.3 **CLASSE: LINK**

A Classe de Objetos denominada Link foi criada para otimizar o uso do Objeto Tutorial do Ambiente Poesis. Esta otimização consiste basicamente em reduzir o tempo de pesquisa às referências, facilitando sua manutenção. A pesquisa à instância do objeto Link é sempre realizada quando se deseja incluir, alterar, remover, ou eventualmente consultar informações referentes aos Links de um determinado Nó_de_Texto.

O objeto Link funciona em total integração com o objeto Nó_de_Texto. Seu objetivo principal é manter o controle das referências que partem de um determinado Nó_de_Texto para outro visando inclusive garantir a integridade dos Hiperdocumentos. Cada referência possui um identificador representado pela palavra que representa e um número seqüencial, caso esta palavra apareça repetida no texto. A composição destes dois parâmetros caracteriza-se por sua unicidade. Associado ao identificador do Link existem informações destinadas a localização do Nó_de_Texto referenciado.

A classe Link foi implementada como subclasse de Árvore-B Genérica, cuja hierarquia de classe é apresentada na figura 35, herdando suas características relativas a estrutura e funcionamento. Esta maneira de implementar resulta da observação do comportamento deste objeto que necessita de uma estrutura que permita uma busca rápida e manipulação flexível, peculiares à super-classe Árvore-B Genérica. Suas instâncias são armazenadas na Área de Armazenamento Virtual do Objeto Tutorial.

DESCRIÇÃO DO FUNCIONAMENTO DO OBJETO LINK

O objeto Link é usado durante o processo de ***Navegação do Objeto Tutorial*** para acessar as referências existentes nos Nós de Texto de um determinado Hiperdocumento. E também na manipulação de um Hiperdocumento para: criação, alteração e remoção das referências associadas às instâncias de Nó_de_Texto nos processos de ***Edição e Referenciação do Objeto Tutorial.*** O funcionamento deste objeto realiza-se quase sempre na modalidade Parcial que compreende basicamente a realização das seguintes ações:

(1) Transferência da parte da instância referente a um nodo(Carga Parcial) da memória persistente para uma região previamente alocada na memória real;
(2) Manipulação desta fração de instância através do envio de mensagens invocando as operações internas disponíveis (operações da classe e operações herdadas);
(3) Transferência da instância (Salva Parcial) da região alocada na memória real para a memória persistente;

A execução das ações descritas acima, realiza-se sempre que for necessário o acesso a uma determinada instância, ou seja, quando o algum objeto de gerenciamento enviar uma mensagem solicitando o acesso a um determinado nodo.

O funcionamento na modalidade Total compreende a realização das seguintes ações:

(1) Transferência de toda a instância do objeto Link (Carga Total) da memória persistente para uma região previamente alocada na memória real;
(2) Manipulação da instância através do envio de mensagens invocando as operações internas disponíveis (operações da classe e operações herdadas);
(3) Transferência da instância (Salva Total) da região alocada na memória real para a memória persistente;

4.4.4 DESCRIÇÃO DO FUNCIONAMENTO DO OBJETO TUTORIAL

O Objeto do Ambiente denominado Tutorial conforme mencionado anteriormente, pode ser manipulado pelos seguintes processo:
- Navegação;
- Edição;

- Referenciação:

O processo de Navegação consiste em manipular os Hiperdocumentos através dos Nós de textos componentes e referenciados. Compreende a execução das seguintes ações:

(1) A partir do Nó de Classe que representa o Objeto Tutorial, recupera-se a instância do objeto Link de Navegação associada.
(2) Através do Link de Navegação obtém-se o Endereço Relativo Virtual da instância do Objeto Hiperdocumento.
(3) Executa-se então o método para pesquisar os Identificadores dos Hiperdocumentos existentes no Ambiente.
(4) Ativa-se a Definição Visual do Objeto Tutorial que exibe a relação de Hiperdocumentos para escolha pelo usuário.
(5) A partir do Endereço Relativo Virtual existente na instância do Hiperdocumento é recuperada a instância correspondente ao Nó_de_Texto inicial do Hiperdocumento.
(6) Após esta ação, a navegação é realizada pela manipulação das instâncias de Nó_de_Texto e Link, com a ativação de Definições Visuais associadas.
(7) No final da execução do processo de Navegação é realizada a desalocação das instâncias usadas.

O processo de Edição corresponde a manipulação das instâncias dos objetos Hiperdocumento, Nó_de_Texto e Link para criar, alterar e remover Hiperdocumentos.

(1) A partir do Nó de Classe que representa o Objeto Tutorial, é recuperada a instância do objeto Link de Navegação associada.
(2) Através do Link de Navegação obtém-se o Endereço Relativo Virtual da instância do Objeto Hiperdocumento.
(3) Executa-se então o método para pesquisar os Identificadores dos Hiperdocumentos existentes no Ambiente.
(4) Ativa-se a Definição Visual do Objeto Tutorial que exibe a relação de Hiperdocumentos para escolha pelo usuário.
(5) Caso o usuário queira alterar ou remover o Hiperdocumento, a partir do Endereço Relativo Virtual existente na instância do Hiperdocumento é recuperada a instância correspondente ao Nó_de_Texto inicial do Hiperdocumento.
(6) Realiza-se a ativação da Definição Visual correspondente ao Editor de Hipertexto a ser implementado para permitir a modificação (inserção/remoção de texto, Links) ou a remoção do Nó_de_Texto conforme o caso.
(7) Caso seja desejado a criação de um Hiperdocumento, é realizada a inserção deste nodo na instância do Hiperdocumento com as informações obtidas pela ativação da Definição Visual correspondente. Em seguida realiza-se ação (6).
(8) No final da execução do processo de Edição é realizada a desalocação das instâncias usadas.

Os processos descritos acima foram implementados como Links de Processos, sendo que sua ativação é realizada a partir da Definição Visual existente para o Objeto Tutorial.

4.5 APLICAÇÃO RELACIONAL

O Objeto do Ambiente denominado Aplicação Relacional foi criado para permitir ao usuário do Ambiente Poesis a manipulação de sua Aplicação Modelada inicialmente pelo Modelo E/D e transformada posteriormente em uma Aplicação Relacional a ser implantada em Sistemas de Gerenciamento de Banco de Dados convencionais que implementam esta filosofia.

O objeto Aplicação Relacional manipula as aplicações relacionais geradas a partir dos subgrafos de objetos que representam as instâncias de Aplicações Modeladas (ver seção 4.2). Uma Aplicação Relacional é formada por relações, que representam a parte estática da Aplicação Modelada, e por instruções na linguagem SQL, que representam a parte Dinâmica da Aplicação Modelada. Estas tabelas e instruções poderão ser exportadas pelo Ambiente Poesis de acordo com o tipo de SGBD compatível, através de processos especiais (implementados como Links de Processos) denominados FILTROS.

O processo de ***Transformação*** consiste de um conjunto de algoritmos interativos que efetuam a conversão de objetos do Modelo E/D em tabelas e instruções SQL. O processo de Transformação será implementado no Ambiente como um Link de Processos do objeto Aplicação Relacional. Um outro processo que será implementado denomina-se ***Simulação***. Este Link de Processos possibilitará ao usuário a validação de suas Aplicações Modeladas a partir de simulações de execução da Aplicação Relacional transformada.

O objeto Aplicação Relacional é constituído basicamente pelo objeto Aplicação cujo objetivo é manipular as informações (Tabelas e instruções SQL) geradas no processo de Transformação. Suas características serão especificadas a seguir. No Apêndice III encontram-se especificadas a estrutura da classe e suas operações.

4.5.1 CLASSE: APLICAÇÃO

Criado para manipular as informações resultantes do processo de Transformação, o objeto Aplicação permitirá ao usuário do Ambiente Poesis, o acesso às Relações e Instruções SQL geradas. A classe Aplicação foi implementada como uma subclasse de Nó de Instância e desta forma herdou as características estruturais e comportamentais desta super-classe cuja hierarquia de classes pode ser vista pela figura 35. As instâncias do objeto Aplicação serão armazenadas na Área de Armazenamento Virtual do Objeto Nó de Instância e manipuladas através dos métodos implementado nesta classe.

CAPÍTULO 5

ASPECTOS DA IMPLEMENTAÇÃO DO GOLGO

5.1 INTRODUÇÃO

Este capítulo descreve o processo de concepção do GOLGO, descrevendo as características que nortearam o seu desenvolvimento. Desde a escolha do ambiente para implementação, a linguagem de programação. A metodologia adotada para o desenvolvimento de um sistema complexo no enfoque orientado a objetos é descrita também neste capítulo.

5.2 AMBIENTE DE IMPLEMENTAÇÃO

O desenvolvimento de um ambiente é um processo que inicia com a escolha do ambiente de implementação. Compreende-se como ambiente de implementação: o sistema de computação (hardware) e as ferramentas (software) a serem usadas para a construção do protótipo.

O sistema de computação é caracterizado pelo porte do equipamento e Sistema Operacional. Com o avanço da tecnologia de produção de hardware novas arquiteturas surgem com sistemas cada vez mais potentes, tais como a arquitetura RISC implantada nas WorkStations oferecendo recursos gráficos e potencial de processamento que superam os atuais equipamentos da linha PC. Neste quadro de crescente evolução onde a qualidade e sofisticação dos equipamentos existentes no mercado tem melhorado consideravelmente, o desenvolvimento de um ambiente exige um estudo ponderado de equipamentos, visando aspectos como: portabilidade e performance.

Entretanto, esta nova tecnologia ainda não consolidada em termos de capacitação de pessoal, atraso tecnológico do País, faz com que a decisão de produzir o protótipo do Ambiente Poesis viabilize inicialmente para equipamentos da linha PC, onde a capacidade de memória RAM é pequena, além da grande diversidade de placas gráficas oferecidas para estes equipamentos. Apesar das restrições impostas pelo hardware, existe um aspecto atenuante: a primeira versão do Ambiente Poesis tem por objetivo a avaliação prática dos conceitos empregados, a performance, a interação com o usuário, que contribuirá para a efetuação de futuros ajustes.

Um dos aspectos envolvidos na produção deste protótipo inicial é a portabilidade, onde o objetivo básico é implementar o ambiente Poesis em vários tipos de equipamentos disseminando assim sua tecnologia. Assim, a linha de equipamentos PC compatíveis foi adotada. A configuração básica para processar o sistema será definida no decorrer da implementação do ambiente.

A ESCOLHA DA FERRAMENTA DE DESENVOLVIMENTO

Outro aspecto fundamental está na escolha da ferramenta ideal para a produção do software. O enfoque de Orientação para Objetos restringiu a escolha para as consideradas Linguagens de Programação Orientada a Objetos.

A LINGUAGEM DE PROGRAMAÇÃO C++

A linguagem de programação orientada a objetos C++, considerada por alguns pesquisadores como um dialeto da linguagem C, foi desenvolvida no início de 1980 por Bjarne Stroustrup, na

AT&T Bell Laboratories. É uma evolução da linguagem C que fornece:

- suporte para criação e uso de abstração de dados;
- suporte para o Desenvolvimento e programação orientada a objetos;
- novos melhoramentos na linguagem C.

Outros aspectos desta linguagem são:

- definição de "overloading" de operadores;
- possibilidade de escolha entre "early binding" e "dynamic binding".

A linguagem C++ foi escolhida para a implementação do GOLGO e conseqüente implementação do Ambiente Poesis por ser mais "aberta" do ponto de vista da criação de novos objetos, por ser portável para outro tipo de equipamento sem a sobrecarga de alteração.

5.3 **ARQUITETURA FUNCIONAL**

A implementação do GOLGO usando a linguagem C++ foi uma tarefa bastante árdua por esta linguagem não dispor de mecanismos eficientes de gerenciamento de memória e gerenciamento de dispositivos. Assim o projeto do GOLGO teve que incluir o módulo Gerenciador de Memória e o módulo Gerenciador de Dispositivos.

A arquitetura funcional do GOLGO segue basicamente a mesma estrutura descrita no capítulo 3, ou seja, existe a integração dos módulos: gerenciador de objetos, gerenciador de memória e gerenciador de dispositivos. Nas superfícies de contato entre estes módulos ocorre o intercâmbio e a integração entre os objetos que constitui cada módulo gerenciador. A coordenação geral é realizada pelo programa principal denominado POESIS que implementa a Interface Genérica que comanda a navegação pelo ambiente.

As hierarquias de classes existentes foram implementadas usando o conceito de Classes Derivadas. Assim, a hierarquia de Lista Genérica (ver capítulo 3) por ser vista como:

Super-classe:	Lista Genérica
Classes Derivadas:	Lista Fixa Genérica
	Lista Variável Genérica

Super-classe:	Lista Fixa Genérica
Classes Derivadas:	Nó de Classe
	Nó de Composição
	Restrições
	Base de Dados do Dicionário de Objetos

Super-classe:	Lista Variável Genérica
Classes Derivadas:	Link de Navegação
	Link de Processos
	Índice de Operações Internas
	Definição Visual
	Versões
	Link de Composição

A hierarquia de Árvore-B Genérica pode ser vista como:

Super-classe:	Árvore-B Genérica
Classes Derivadas:	Índice do Dicionário de Objetos
	Usuário
	Link
	Hiperdocumento

A hierarquia da classe Nó de Instância pode ser vista como:

Super-classe:	Nó de Instância
Classes Derivadas:	Objetos do Modelo E/D (ver capítulo 3)
	Nó de Texto
	Aplicação

A hierarquia da classe Dispositivos pode ser vista como:

Super-classe:	Dispositivos
Classes Derivadas:	Entrada
	Editor de Texto
	Janela
	Ícone
	Frase
	Saída
	Mouse

O Polimorfismo foi definido para a hierarquia de classes Nó de Instâncias com alguns métodos declarados como Funções Virtuais.

5.4 **ANÁLISE E PROJETO DE IMPLANTAÇÃO**

O processo de análise para o desenvolvimento do GOLGO foi baseado nas idéias sobre Desenvolvimento Orientado a Objetos (DOO) apresentadas em [TAKA88]. A partir deste enfoque uma estratégia para o desenvolvimento que na realidade corresponde a metodologia para desenvolvimento foi idealizada, consistindo das seguintes etapas:

(1) - Definição da abordagem do problema;
(2) - Definição dos requisitos;
(3) - Processo de abstração;
(4) - Especificação;
(5) - Refinamentos;
(6) - Implementação.

(1) - DEFINIÇÃO DA ABORDAGEM DO PROBLEMA

Esta etapa corresponde a escolha da tecnologia de desenvolvimento a ser adotada. Basicamente foi definida quando surgiu o projeto no Departamento de Informática denominado APOIO que definiu as linhas de ação e características dos produtos a serem gerados. Foi adotado o enfoque de Orientação a Objeto visando obter um software com as propriedades de extensibilidade e manutenabilidade. A partir desta definição cada módulo do projeto APOIO passou a ser analisado pelos respectivos pesquisadores responsáveis. O GOLGO foi então idealizado com base na filosofia de hipertexto e no enfoque orientado a objeto.

(2) - DEFINIÇÃO DOS REQUISITOS

A partir da escolha da filosofia de implementação, o projeto do GOLGO encaminhou-se para a etapa de definição dos requisitos, onde foram determinados os objetivos, o universo de atuação e as partes componentes do sistema.

(3) - PROCESSO DE ABSTRAÇÃO

Nesta etapa foi utilizada a técnica de identificação de objetos, também chamada de classificação. Esta técnica visa a construção de classes. Algumas hierarquias de classes foram sendo definidas, tais como: Árvore-B Genérica e Nó de Instância.

(4) - ESPECIFICAÇÃO

Nesta etapa cada objeto foi especificado em termos das variáveis internas e das operações internas.

(5) - REFINAMENTOS

Esta etapa consiste em efetuar uma avaliação das especificações de cada objeto e com a utilização da técnica de Generalização foram determinadas novas Hierarquias de classes: Lista Genérica, Lista Fixa Genérica e Lista Variável Genérica. Além disso foram identificadas as partes herdadas pelos objetos subordinados.

(6) - IMPLEMENTAÇÃO

A implementação de cada objeto foi realizada de forma separada, já que o enfoque de orientação a objetos permite tratá-los modularmente como partes isoladas do sistema. Ao se concluir um conjunto de objetos pertencentes a um Módulo Gerenciador, todo o módulo era testado de forma geral e verificado os problemas de integração.

PROTÓTIPO DO AMBIENTE POESIS

O projeto de implantação do GOLGO inicialmente vislumbra a implementação dos Objetos do Ambiente: Usuário, Modelo E/D e Aplicação Modelada. Com isso estará funcionando o primeiro protótipo do Ambiente Poesis restrito a estes objetos. Posteriormente os outros Objetos do Ambiente: TUTORIAL e APLICAÇÃO RELACIONAL serão acoplados ao ambiente. Como também a medida que o Ambiente Poesis evolua, novos objetos poderão ser acoplados sem necessariamente impactar na modificação do protótipo. Esta condição se deve às características de modularidade e extensibilidade do sistema.

5.4.1 **DIFICULDADES E SOLUÇÕES**

O desenvolvimento de um gerenciador de objetos seria uma tarefa menos árdua se existisse na ferramenta escolhida um sistema de gerenciamento de memória eficiente. Outras ferramentas, tal como o Smalltalk possui um gerenciador de memória mas conforme descrito anteriormente não preenchia alguns requisitos necessários:

- portabilidade;
- O modelo de interface do Smalltalk não se adequa ao pretendido para o ambiente;

Desta forma foi necessário desenvolver um sistema de gerenciamento de memória integrado aos outros módulos do GOLGO que fosse também orientado a objetos controlando o armazenamento e recuperação das instâncias de objetos do Ambiente.

CAPÍTULO 6

CONCLUSÕES E PERSPECTIVAS

6.1 CONTRIBUIÇÕES E CONCLUSÕES

O gerenciador de Objetos descrito nos capítulos anteriores é uma das partes fundamentais do projeto do Ambiente Poesis. Sua concepção foi baseada em conceitos e filosofias de importantes e atuais áreas de pesquisas, tais como: Hipertexto e Programação Orientada a Objetos. A idéia inicial foi capturar as semelhanças de conceitos entre estas áreas e adequá-los a produção de um sistema de gerenciamento de objetos que fornecesse ao ambiente Poesis a capacidade evolutiva tão desejada.

As propriedades dos sistemas de Hipertextos, tais como: flexibilidade, extensibilidade e dinamismo podem ser observadas no projeto do GOLGO. Em um sistema de Hipertexto os documentos são armazenados e recuperados de maneira bastante flexível devido a forma de organização de suas diversas partes, permitindo uma associação (links) entre vários documentos, além das inúmeras possibilidades de *composição de documentos*. O mecanismo de navegação de um sistema de Hipertexto (*browser*) oferece ao usuário a liberdade de escolha do caminho desejado. Da mesma maneira, o GOLGO manipula os Objetos do Ambiente de forma flexível, permitindo o acesso às classes, às instâncias, às operações internas, etc. O mecanismo de Navegação do GOLGO (Interface Genérica) permite também que o usuário percorra a estrutura em busca do objeto desejado.

O GOLGO através de seus objetos gerenciadores permite associar a cada objeto do Ambiente uma ou mais formas de apresentação ou interação com o usuário. É a interface própria do objeto que implementa o conceito de **princípio reativo do objeto** [TAKA88]. Ele permite que o próprio objeto de forma interativa se comunique com o usuário. Analisando este aspecto do GOLGO, podemos perceber inúmeras vantagens, tais como: modularidade na definição da interface do objeto, ou seja, qualquer novo objeto introduzido no Ambiente não irá causar problemas de manutenção, já que o próprio objeto ao ser inserido trás junto sua interface. Outra vantagem deste aspecto é a possibilidade de se manter de forma flexível diversas interfaces que seriam manipuladas em certas circunstâncias, ou seja, suponha que um mesmo objeto necessite interagir de forma diferenciada de acordo com o tipo de usuário. Com esta propriedade do GOLGO isto seria facilmente implementado.

Outra característica muito importante que vale ser ressaltada refere-se a Composição de Objetos. O GOLGO através de seus objetos gerenciadores permite definir formas de composição entre instâncias de objetos diferenciados. Um exemplo prático desta facilidade é o Modelo Estático/Dinâmico implementado no ambiente. Este modelo utiliza a composição de objetos para produzir aplicações modeladas onde os objetos podem participar na composição de outros. O GOLGO administra esta explicitação da composição de objetos.

O gerenciamento de memória do GOLGO é flexível ao ponto de permitir que outros softwares sejam acionados a partir do ambiente, já que a memória principal é alocada a medida que for sendo necessário.

O gerenciamento de dispositivo do GOLGO permite que novos objetos controladores de dispositivos sejam incrementados na configuração de periféricos do equipamento. Esta flexibilidade irá facilitar a utilização de novas formas de apresentação, tais como: imagens, sons, etc. colaborando assim para a manipulação de novos tipos de dados para interfaceamento com o usuário.

6.2 **PERSPECTIVAS PARA TRABALHOS FUTUROS**

O desenvolvimento do GOLGO foi baseado nos conceitos do Paradigma da Orientação para Objetos. Sua arquitetura modular composta por objetos independentes entre si, com funções características bem diferenciadas favorece futuras extensões. Apesar desta facilidade o GOLGO não permite a manipulação dinâmica de classes e métodos tal como o Smalltalk. Compreende-se por manipulação dinâmica a capacidade de criação de objetos, alteração de propriedades estruturais, criação e alteração de métodos, etc. , durante o processamento do Ambiente Poesis. Os objetivos imediatos do projeto de concepção do protótipo do ambiente, impediram que esta facilidade fosse inserida. Isto significa que um novo objeto do ambiente a ser inserido deverá ser codificado na linguagem C++ e compilado para ser utilizado. Desta forma, uma das perspectivas para aprimoramento do GOLGO de médio a longo prazo, consistirá em desenvolver um módulo interpretador de uma linguagem de manipulação de classes, objetos e métodos, denominada de LG (Linguagem do GOLGO) que implemente os conceitos do Paradigma da Orientação a Objetos. Esta linguagem irá permitir que o usuário utilize a estrutura do GOLGO para criar seus próprios objetos de forma interativa. Este projeto possui um outro nível de complexidade que exigirá novas pesquisas e aprimoramentos do GOLGO.

Outra perspectiva para trabalhos futuros mais imediatos, consistirá em implementar as novas ferramentas a serem produzidas pelos trabalhos de pesquisa descritos no capítulo 1 deste trabalho, bem como incrementar os módulo gerenciadores:

- Memória, visando o armazenamento de dados multimídia, e
- Dispositivo, para que o Ambiente outros tipos de periféricos.

REFERÊNCIAS BIBLIOGRÁFICAS

[AKHR89] AKHRAS, F.N., COSTA, M.C.C.: "Meta-Modelos para Ferramentas Genéricas Integráveis ao Ambiente SIPS", 3º Simpósio de Engenharia de Software, Recife, Outubro de 1989.

[ATWO85] ATWOOD, T.M.: "An Object-Oriented DBMS for Design Support Applications", IEEE COMPINT, 1985.

[BANC86] BANCILHON, F.: "A Logic-Programming/Object-Oriented Cocktail", Sigmod Record, Vol. 15, No. 3, September 1986.

[BANC89] BANCILHON, F.: "Query Languages for Object-Oriented Database Systems: Analysis and a Proposal", 4º SBBD - Campinas - SP, Abril de 1989.

[BAND89] BANDEIRA, M.S.: "Modelagem Estática/Dinâmica de Sistema de Informações", UFPe-DI, Tese de Mestrado, 1989.

[BROW86] BROWN P. J: "Interactive Documentation, in Software "- Practice and Experience ,Vol 16, Mar 1986, pp 291-299.

[BROW87] BROWN, P. J.: "Hypertext : The Way Forward", University of Kent, Canterbury, 1987.

[CODD79] CODD, E.F.: "Extending the Relational Model to Capture More Meaning", ACM Transaction on Database Systems, Vol.4, No. 4, Dezembro 1979, pp 397-434.

[COHE81] COHEN, J. : "Garbage Collection of Linked Data Structures", ACM Computing Surveys 13(3), September 1981.

[CONK87] CONKLIN J. : "A Survey of Hypertext", MCC TR, Nr. STP-356-86 Rev1, Austin, TX, Feb 1987.

[COOK87] COOK, S. : "Object-Oriented Programming: An Evolutionary Approach", 1987.

[COX86] COX, B. , HUNT, B.: "Objects, Icons, and Software IC´s,", Byte Magazine, 11(8), August 1986, pp161-176.

[DAMI88] DAMIAN, M.O., FONSECA, D.: "Interface Orientada a Objetos para Concepção de Sistemas de Informações", Relatório Técnico, UFPe-DI, Dezembro 1988.

[DAYA86] DAYAL, U., MANOLA, F.: "PDM: An Object-Oriented Data Model", IEEE, 1986, pp18-25.

[DITT85] DITTRICH, K.R., LORIE, R.A.: "Object-Oriented Database Concepts for Engineering Applications", IEEE COMPINT, 1985.

[DITT86] DITTRICH, K.R.: "Object-Oriented Database System: the Notion and the Issues", IEEE, 1986.

[DODA89] DODANI, M.H., HUGHES, C.E., MOSHELL, J.M.: "Separation of Powers", BYTE Magazine, March 1989, pp 255-262.

[DUFF86] DUFF, C.B.: "Designing an Efficient Language", Byte Magazine, 11(8), August 1986, pp 211-224.

[EGLO90] EGLOWSTEIN, H. : "Reach Out and Touch Your Data", Byte Magazine, July 1990, pp 283-290.

[ELMA83] ELMASRI, R., WIEDERHOLD, G.: "GORDAS: A Formal High-Level Query Language for the Entity-Relationship Model", Entity-Relationship Approach to Information Modeling and Analysis, Elsevier Science Publisher, 1983.

[FISH90] FISHER, S. S., TAZELAAR, J.M. : "Living in a Virtual World", Byte Magazine, July 1990, pp 215-221.

[FONS87] FONSECA, D.: "Un Mecanisme d´Activacion et Controle de Declencheurs Orientee Objets", Universite de Paris VI, These de Doctorat Informatique, 1987.

[GARR86] GARRETT, N., SMITH, K. E., MEYROWITZ, N.: "Intermedia: Issues, Strategies and Tactics in the Design of a Hypermedia Document System, IRIS, Brown University.

[GOLD83] GOLDBERG, A., ROBSON, D.: "Smalltalk-80: The Language and its Implementation", Addison-Wesley Publishing Company, 1983.

[HARA89] HARA, C.S., MAGALHÃES, G.C.: "Uma Experiência em modelar Banco de Dados Orientado a Objetos", 4º SBBD - Campinas - SP, Abril de 1989.

[KAY77] KAY, A. et. al: "Personal dynamic media", Computer, março 1977.

[KELN89] KELNER, J., CAVALCANTI, A.L., PARDO, A.: "LindA: Uma Linguagem de Autoria Automática para Hipertexto", 3º Simpósio de Engenharia de Software, Recife, Outubro de 1989.

[KÜSP87] KÜSPERT, K., DADAM, P., GÜNAUER, J.: "Cooperative Object Buffer in Advanced Information Management Prototype", 13th VLDB Conference, Brighton 1987.

[LIPP89] LIPPMAN, S.B.: "C++ Primer", Addison-Wesley Publishing Company, AT&T Bell Laboratories, 1989.

[MACE89] MACEDO, M.A.C., FONSECA, D., "GOLGO - Gerenciador Orientado a objetos", 4º SBBD - Campinas - SP, Abril de 1989.

[MACE89a] MACEDO, M.A.C., FONSECA, D.: "Especificação de um Gerenciador Orientado a Objetos", 3º Simpósio de Engenharia de Software, Recife, Outubro de 1989.

[MACH89] MACHADO, J.C., PRICE, R.T.: "Modelagem de Dados de um Ambiente de Desenvolvimento de Software", 3º Simpósio de Engenharia de Software, Recife, Outubro de 1989.

[MCEN87] McENTEE, T. J.: "Overview of Garbage Collection in Simbolic Computing", LISP Pointers, 1(3), 1987.

[MEIR89] MEIRA, S.R.L., KELNER, J., ALBUQUERQUE, E., MARTINS, J., MELO, A.C., VASCONCELOS, A.: "Hipertexto: O Projeto do Sistema H", 3º Simpósio de Engenharia de Software, Recife, Outubro de 1989.

[MELO89] MELO, R.N., REZENDE, M.A.M., LANZELOTTE, R.S.G.: "OMS -EITIS: um Sistema de Gerência de Objetos", 4º SBBD, Campinas-SP, Abril de 1989.

[MEYE88] MEYER, B. : "Object-Oriented Software Construction", C.A.R. HOARE Series Editor, Prentice Hall, 1988., pp. 89-91

[SALG88] SALGADO, A.C.A.: "Contribution a un SGBD orienté objet (NICEBD): Traitement des Données et des Interfaces Multimedia", Universite de Nice - CNRS, These de Doctorat Informatique, Octobre 1988.

[SALG89] SALGADO, A.C.A.: "O Nível Físico Orientado a Objetos de um SGBD Multi-Mídia", 4º SBBD - Campinas - SP, Abril de 1989.

[SCHE86] SCHEIFLER R.W.,GETTY J.: "The XWINDOW System". MIT Laboratory for Computer Science - Digital Equipment Corp. MIT Project Athena 1986.

[SCHU86] SCHMUCKER, K. : "MacApp: An Application Framework", Byte Magazine, 11(8), August 1986, pp 189-194.

[SHNE87] SHNEIDERMAN, B., "Designing the User Interface: Strategies for Effective Human-Computer Interaction", Addison-Wesley Publishing Company, 1988.

[TAKA88] TAKAHASHI, T.: "Introdução a Programação Orientada a Objetos", III EBAI , Curitiba, Janeiro de 1988.

[TESL86] TESLER, L.: "Programming Experiences", Byte Magazine, 11(8), August 1986, pp 195-206.

[THOM89] THOMAS, D.: "What´s in an Object?", Byte Magazine, March 1989, pp 231- 240.

[THOM89] THOMPSON, T.: "The Next Step", Byte Magazine, March 1989, pp 265- 269.

[TOMP89] TOMPA, F.W.: "Towards an Electronic Dictionary",4º SBBD, Campinas - SP, Abril de 1989.

[TROT89] TROTTA, C.N.F., MATTOSO, M.L.Q., SOUZA, J.M.: "Extensões do COPPEREL para Aplicações não Convencionais", 4º SBBD, Campinas - SP, Abril de 1989.

[VASC88] VASCONCELOS, A. M. L., MELO, A. C .V., ALBUQUERQUE, E.S., MEIRA, S.R.,"Hipertexto - Conceitos e Características", D.I.- UFPe- Setembro 1988.

[VICT89] VICTORELLI, E.Z., MAGALHÃES, G.C., DRUMMOND, R.: "Mecanismo de Gerenciamento de Versões e Configurações do A_HAND", 3º Simpósio de Engenharia de Software, Recife, Outubro de 1989.

[WEGN89] WEGNER, P.: "Learning the Language", Byte Magazine, March 1989, pp 245- 253.

[WIEN88] WIENER, R.S., PINSON, L.J.: "An Introduction to Object-Oriented Programming and C++", Addison-Wesley Publishing Company, 1988.

[WOEL87] WOELK, D., KIM, W.: "Multimedia Information Management In na Object-Oriented Database System", 13th VLDB Conference, Brighton, 1987.

[ZORT89] ZORTECH, I.: "C++ Compiler Reference", Zortech Incorporated, 1989.

APÊNDICE I

CLASSES DE OBJETOS DE GERENCIAMENTO

CLASSE: NÓ DE CLASSE

ESTRUTURA

A estrutura interna do Nó de Classe pode ser visualizada e compreendida a partir da tupla:

$NC = < E_{DO}, E_{NCP}, E_{LN}, E_{LP}, E_{IO}, E_{LCN}, E_{LCI}, E_{DV}, E_{R}, E_{V}, CR>$

onde:

E_{DO} é o Endereço Relativo Virtual do Dicionário de Objeto;
E_{NCP} é o Endereço Relativo Virtual do Nó de Composição;
E_{LN} é o Endereço Relativo Virtual do Link de Navegação;
E_{LP} é o Endereço Relativo Virtual do Link de Processos;
E_{IO} é o Endereço Relativo Virtual do Índice de Operações Internas;
E_{LCN} é o Endereço Relativo Virtual do Link de Composição Normal;
E_{LCI} é o Endereço Relativo Virtual do Link de Composição Invertida;
E_{DV} é o Endereço Relativo Virtual de Definição Visual;
E_{R} é o Endereço Relativo Virtual de Restrições;
E_{V} é o Endereço Relativo Virtual de Versões;
CR é o Contador de Referências.

As variáveis: E_{DO}, E_{NCP}, E_{LN}, E_{LP}, E_{IO}, E_{LCN}, E_{LCI}, E_{DV}, E_{R} e E_{V} permitem recuperar respectivamente as instâncias dos objetos de gerenciamento: Dicionário de Objeto, Nó de Composição, Link de Navegação, Link de Processos, Índice de Operações Internas, Link de Composição (instâncias Normal e Invertida), Definição Visual, Restrições e Versões. Quando não existe uma determinada instância o conteúdo da variável é modificado para a Constante Negativa (-1). A variável CR tem por finalidade controlar o número de referências feitas a instância no Nó de Classe evitando a sua remoção quando existir algum outro objeto de gerenciamento se referenciando a mesma.

A classe Nó de Classe é subclasse de Lista Fixa Genérica. ou seja, a estrutura desta super-classe descrita na seção 3.2.2.1.2 é herdada compondo uma estrutura final para o objeto. Logo, a implementação do objeto Nó de Classe consiste de uma lista da classe Lista Fixa cujos elementos componentes são instâncias de Nó de Classe.

HERANÇA

A Classe Nó de Classe ao ser implementada como uma subclasse da Classe Lista Fixa Genérica, conforme visto anteriormente, herdou as propriedades estruturais e comportamentais desta super-classe. As propriedades comportamentais correspondem a parte dinâmica do objeto, ou seja suas operações internas descritas na seção 3.2.2.1.2, que podem ser usadas pela classe Nó de Classe para manipulação de suas instâncias, sendo que a utilização destas operações pode também ser feita de duas maneiras: diretamente invocando as operações da Classe Lista Fixa Genérica, ou usando na implementação dos métodos da Classe Nó de Classe.

OPERAÇÕES INTERNAS

A Classe Nó de Classe pode ser manipulada a partir de um conjunto de operações internas, ou métodos, que serão descritos a seguir.
A execução destas operações é realizada a partir do envio da mensagem ao objeto Nó de Classe que a identifica.

CRIAÇÃO - Esta operação interna tem por finalidade efetuar a criação de uma instância do Nó de Classe. Esta criação corresponde a manipulação parcial das instâncias deste tipo de objeto, ou seja, não existe a necessidade de manipular todas as instâncias em Memória Real. Em sua implementação foi utilizado o métodos herdado da classe Lista Fixa Genérica: Inserção-Parcial.
INSERÇÃO - Esta operação possui o mesmo objetivo que Criação, ou seja, a inicialização de uma instância de Nó de Classe. A diferença básica entre eles é que neste método todas as instâncias do Nó de Classe estão sendo manipuladas em Memória Real. Este método corresponde à operação interna Inserção-Real herdado da classe Lista Fixa.
EXTRAÇÃO - Este método efetua a remoção lógica de uma instância do objeto Nó de Classe armazenada na memória persistente a partir do parâmetro que informa seu Endereço Relativo. A remoção somente é realizada quando o conteúdo da variável CR (Contador de Referência) é igual a zero.
REMOÇÃO - Este método efetua a remoção lógica de uma instância do objeto Nó de Classe armazenada na Memória Real a partir do parâmetro que informa seu Endereço Relativo. A remoção somente é realizada quando o conteúdo da variável CR (Contador de Referência) é igual a zero.
OBTER ENDEREÇO RELATIVO VIRTUAL - Este método tem por finalidade
fornecer o conteúdo de uma das variáveis E_{DO} , E_{NCP} , E_{LN} , E_{LP} , E_{IO} , E_{LCN} , E_{LCI} , E_{DV} , E_R e E_V de uma instância de Nó de Classe, cujo Endereço Relativo é informado via parâmetro.
ALTERAR ENDEREÇO RELATIVO VIRTUAL - Este método tem por finalidade alterar conteúdo de uma das variáveis E_{DO} , E_{NCP} , E_{LN} , E_{LP} , E_{IO} , E_{LCN} , E_{LCI} , E_{DV} , E_R e E_V de uma instância de Nó de Classe, cujo Endereço Relativo é informado via parâmetro.
OBTER CONTADOR DE REFERÊNCIA - Esta operação permite obter o conteúdo da variável CR de uma instância de Nó de Classe.
INCREMETAR CONTADOR DE REFERÊNCIA - Esta operação permite incrementar, ou seja, somar o valor 1 ao conteúdo da variável CR de uma instância de Nó de Classe.
DECREMETAR CONTADOR DE REFERÊNCIA - Esta operação permite decrementar, ou seja, subtrair o valor 1 do conteúdo da variável CR de uma instância de Nó de Classe.

CLASSE: NÓ DE COMPOSIÇÃO

ESTRUTURA

A estrutura interna do Nó de Composição pode ser visualizada e compreendida a partir da tupla:

$NCP = < E_{DO}, E_{NI}, E_{LCN}, E_{LCI}, CR>$, onde:

E_{DO} é o Endereço Relativo Virtual do Dicionário de Objeto;
E_{NI} é o Endereço Relativo Virtual do Nó de Instância;
E_{LCN} é o Endereço Relativo Virtual do Link de Composição Normal;
E_{LCI} é o Endereço Relativo Virtual do Link de Composição Invertida;
CR é o Contador de Referências.

As variáveis: $E_{\mathbf{DO}}$, $E_{\mathbf{NS}}$, $E_{\mathbf{LCN}}$ e $E_{\mathbf{LCI}}$ permitem recuperar as respectivamente as instâncias dos objetos de gerenciamento: Dicionário de Objeto, Nó de Instância e Link de Composição (instâncias: Normal e Invertida). Quando não existe uma determinada instância o conteúdo da variável é modificado para a Constante Negativa (-1). A variável CR tem por objetivo manter o controle sobre o número de referências feitas a instância no Nó de Composição evitando sua remoção quando existir algum outro objeto de gerenciamento se referenciando a mesma.

A classe Nó de Composição foi implementada como subclasse de Lista Fixa Genérica. ou seja, a estrutura de Lista Fixa Genérica descrita na seção 3.2.2.1.2 é herdada compondo uma estrutura final para o objeto. Portanto, a implementação do objeto Nó de Composição consiste de uma lista cujas características são dadas pela classe Lista Fixa sendo seus elementos componentes instâncias de Nó de Composição.

HERANÇA

A Classe Nó de Composição implementada como subclasse de Lista Fixa Genérica, conforme visto anteriormente, tornou possível a herança das propriedades estruturais e comportamentais desta super-classe. As propriedades comportamentais correspondentes a parte dinâmica do objeto, ou seja suas operações internas, foram descritas na seção 3.2.2.1.2, podendo assim serem usadas pela classe Nó de Composição para manipulação de suas instâncias. A utilização destas operações herdadas pode ser feita de duas maneiras: diretamente invocando as operações da Classe Lista Fixa Genérica, ou usando na implementação dos métodos da Classe Nó de Composição.

OPERAÇÕES INTERNAS

A Classe Nó de Composição pode ser manipulada a partir de um conjunto de operações internas, ou métodos, cuja execução é realizada a partir do envio da mensagem ao objeto Nó de Composição que a identifica. Estas operações internas serão descritas a seguir.

INSERÇÃO - Esta operação tem por objetivo a inicialização de uma instância de Nó de Composição. Neste método todas as instâncias do Nó de Composição estão sendo manipuladas em Memória Real. Este método corresponde à operação interna Inserção-Real herdado da classe Lista Fixa.
REMOÇÃO - Este método efetua a remoção lógica de uma instância do objeto Nó de Composição armazenada na Memória Real a partir do parâmetro que informa seu Endereço Relativo. A remoção somente é realizada quando o conteúdo da variável CR (Contador de Referência) é igual a zero.
OBTER CONTADOR DE REFERÊNCIA - Esta operação permite obter o conteúdo da variável CR de uma instância de Nó de Composição.
INCREMETAR CONTADOR DE REFERÊNCIA - Esta operação permite incrementar, ou seja, somar o valor 1 ao conteúdo da variável CR de uma instância de Nó de Composição.
DECREMETAR CONTADOR DE REFERÊNCIA - Esta operação permite decrementar, ou seja, subtrair o valor 1 do conteúdo da variável CR de uma instância de Nó de Composição.
OBTER ENDEREÇO RELATIVO VIRTUAL - Este método tem por finalidade
fornecer o conteúdo de uma das variáveis E_{DO} , E_{NS} , E_{LCN} e E_{LCI} de uma instância de Nó de Composição, cujo Endereço Relativo é informado via parâmetro.
ALTERAR ENDEREÇO RELATIVO VIRTUAL - Este método tem por finalidade alterar conteúdo de uma das variáveis E_{DO} , E_{NS} , E_{LCN} e E_{LCI} de uma instância de Nó de Composição, cujo Endereço Relativo é informado via parâmetro.

CLASSE: NÓ DE INSTÂNCIA

ESTRUTURA

A estrutura interna da classe Nó de Instância implementada como uma super-classe no Ambiente por ser genérica é formada por uma única variável: EBNS, ou Endereço Base do Nó de Instância. Esta variável tem por objetivo permitir a localização e manipulação de suas instâncias.

OPERAÇÕES INTERNAS

A Classe Nó de Instância pode ser manipulada a partir de um conjunto de operações internas, ou métodos, que serão descritos a seguir.
A execução destas operações é realizada a partir do envio da mensagem ao objeto Nó de Instância que a identifica.

CRIAÇÃO - Esta operação interna tem por finalidade efetuar a criação de uma instância do Nó de Instância. Esta criação corresponde a alocação de Memória Real cujo tamanho é informado através dos parâmetros que compõem o método.
ALTERAÇÃO - Esta operação tem por objetivo expandir o tamanho da região de Memória Real já alocada para uma instância de Nó de Instância. O valor correspondente ao incremento da região de memória é informada via parâmetro.
REMOÇÃO - Este método efetua a Desalocação Virtual de instância do objeto Nó de Instância armazenada na memória persistente partir do parâmetro que informa seu Endereço Relativo Virtual.
OBTER INFORMAÇAO - Esta operação permite obter o conteúdo da informação armazenada na instância de Nó de Instância.
OBTER TAMANHO - Esta operação interna tem por objetivo fornecer o tamanho de Memória Real alocada pela instância de Nó de Instância.
SALVA - Esta operação interna tem por finalidade efetuar a transferência física de uma instância de Nó de Instância armazenada na Memória Real e identificada pelo parâmetro IDP, para a posição da Área de Armazenamento Virtual correspondente, localizada pelo ERV.
CARGA - Esta operação interna efetua a transferência física de uma instância de Nó de Instância armazenada na posição da Área de Armazenamento Virtual (AAV) localizada pelo ERV, para a Memória Real pré-alocada e identificada pelo parâmetro IDP.

CLASSE: LINK DE NAVEGAÇÃO

ESTRUTURA

A estrutura interna do Link de Navegação corresponde a composição de sua estrutura interna e as estruturas herdadas das super-classes que compõem a hierarquia. A estrutura interna da classe Link de Navegação é composta pelas seguintes sub-estruturas: Header e Componente. A sub-estrutura Header corresponde ao elemento inicial da lista cuja função é manter as informações de controle. O elemento Header pode ser compreendido a partir da tupla:

$H_{LN} = < N_{EL}, ER_{NC} >$, onde:

N_{EL} é o Número de Links Verticais existentes na instância;
ER_{NC} é o Endereço Relativo Virtual do Nó de Classe.

A variável N_{EL} corresponde ao contador de elementos da lista, cuja função é controlar o número de elementos e assim evitar a remoção da lista quando ainda existirem elementos ativos. A variável ER_{NC} contém o Endereço Relativo Virtual da instância do objeto Nó de Classe ao qual o Link de Navegação está associado. Sua função básica é permitir a recuperação a partir do objeto Link de Navegação, da instância do objeto Nó de Classe associada.

O elemento Componente do objeto Link De Navegação é definido a partir da tupla:

$C_{LN} = < ER_{DO}$, STATUS, TIPO, ERV, POS >, onde:

ER_{DO} é o Endereço Relativo Virtual do Dicionário de Objetos;
STATUS indica o Status do Link de Navegação;
TIPO indica o tipo de objeto associado;
ERV é o Endereço Relativo Virtual do objeto associado;
POS é a Posição Relativa do objeto associado.

A variável ER_{DO} tem por função fornecer a localização das informações que permitem a identificação do objeto associado pela instância de Link de Navegação. STATUS informa os parâmetros necessários a manipulação do sistema de segurança permitindo restringir o acesso ao objeto associado. A variável TIPO é usada para identificar o tipo de objeto associado. Esta variável assume os valores 0,

1 e 2 que indicam respectivamente: Nó de Classe, Nó de Composição e Nó de Instância. A variável ERV contém o Endereço Relativo Virtual que permite o acesso a instância do objeto associado possibilitando sua transferência para a Memória Real. POS é a variável que indica, conforme o caso, a posição ocupada pela instância a partir do Endereço Relativo Virtual. Esta variável somente é usada para os objetos Nó de Classe e Nó de Composição.

HERANÇA

A Classe Link de Navegação foi implementada como uma subclasse da Classe Lista Variável Genérica, conforme visto anteriormente, e portanto herdou as propriedades estruturais e comportamentais desta super-classe. No que se refere as propriedades comportamentais, ou seja a parte dinâmica do objeto, constituída por suas operações internas descritas na seção 3.2.2.1.3, a classe Link de Navegação poderá utilizá-las para manipulação de suas instâncias. A utilização destas operações pode ser feita de duas maneiras: diretamente invocando as operações da Classe Lista Variável Genérica, ou usando na implementação dos métodos da Classe Link de Navegação.

OPERAÇÕES INTERNAS

A Classe Link de Navegação possui um conjunto de operações internas, ou métodos, cuja execução é realizada a partir do envio da mensagem correspondente à operação desejada. A mensagem identifica então qual a operação interna a ser processada. Estas operações internas serão descritas a seguir.

CRIAÇÃO - Este método tem por finalidade a inicialização da instância através da criação do Header. A variável ER_{NC} é inicializado a partir do parâmetro informado. Este método utiliza algumas operações herdadas de sua super-classe.
I**NSERÇÃO** - Este método efetua a inclusão de um novo elemento Componente na instância de Link de Navegação. Para a inclusão deste elemento, esta operação utiliza o método de inserção herdado de Lista Variável Genérica.
REMOÇÃO HEADER - Esta operação interna efetua a remoção do elemento Header, quando não houver mais elementos Componentes na instância.
REMOÇÃO COMPONENTE - Esta operação interna efetua a remoção do elemento Componente da instância.
PESQUISA - Esta operação efetua uma busca na lista correspondente a instância de Link de Navegação visando obter a localização do elemento Componente cujo Endereço Relativo do Dicionário de Objetos corresponda ao valor informado via parâmetro.
OBTER N_{EL} e OBTER ER_{NC} - Estes métodos fornecem o conteúdo das variáveis N_{EL} e ER_{NC} respectivamente de uma instância de Link de Navegação armazenada em Memória Real.
OBTER ERDO, OBTER STATUS, OBTER TIPO, OBTER ERV e OBTER POS- Estes métodos têm por finalidade fornecer o conteúdo das variáveis ER_{DO} , STATUS, TIPO, ERV e POS respectivamente de um determinado elemento Componente da instância de Link de Navegação armazenada em Memória Real.
ALTERAR ERNC - Este método possibilita a alteração do conteúdo da variável ER_{NC} de uma instância de Link de Navegação armazenada em Memória Real.
ALTERAR ERDO, ALTERAR STATUS, ALTERAR TIPO, ALTERAR ERV e ALTERAR POS- Estes métodos têm por objetivo alterar o conteúdo das variáveis ER_{DO} , STATUS, TIPO, ERV e POS respectivamente do elemento Componente de uma determinada instância de Link de Navegação armazenada em Memória Real.
ENCONTRAR - Este método permite localizar um determinado elemento Componente a partir de um parâmetro que informa o número de seqüência correspondente.
INCREMENTAR e DECREMENTAR N_{EL} - Estes métodos têm por finalidade efetuar o incremento e o decremento respectivamente, do conteúdo da variável N_{EL}.

CLASSE: LINK DE PROCESSOS

ESTRUTURA

A classe Link de Processos possui um estrutura que resulta da composição de sua estrutura interna com as estruturas herdadas das super-classes que compõem a hierarquia descrita na figura 14. A estrutura interna da classe Link de Processos é composta pelas seguintes sub-estruturas: Header e Componente. A sub-estrutura Header corresponde ao elemento inicial da lista cuja função é manter informações de controle. O elemento Header pode ser compreendido a partir da tupla:

$H_{LP} = < N_{EL}, ER_{NC} >$, onde:

N_{EL} é o Número de procedimentos especiais existentes na instância;
ER_{NC} é o Endereço Relativo Virtual do Nó de Classe de origem.

A variável N_{EL} contém um contador de elementos da lista, cujo objetivo é controlar o número de elementos para evitar uma possível remoção da lista quando ainda existirem elementos ativos. A variável ER_{NC} contém o Endereço Relativo Virtual da instância do objeto Nó de Classe ao qual o Link de Processos está associado. Sua função é permitir a recuperação a partir do objeto Link de Processos, da instância do objeto Nó de Classe que está associada.

O elemento Componente do objeto Link de Processos é definido a partir da tupla: $C_{LP} =$ < IDLP, STATUS, INDP>, onde:

IDLP é o Identificador do Link de Processos;
STATUS indica o Status do Link de Processos;
INDP corresponde ao índice do procedimento associado;

A variável IDLP tem por função identificar um determinado procedimento especial. O conteúdo desta variável consiste de um string de tamanho 8 com o nome que descreve para o usuário o procedimento implementado no Ambiente. O STATUS informa os parâmetros necessários a manipulação do sistema de segurança permitindo restringir o acesso ao objeto associado. A variável INDP contém o Índice que permite ativar a execução do processo. Todos os procedimentos especiais estão implementados em um array cujos elementos são apontadores para as funções que os executa.

A parte da estrutura interna do objeto Link de Processos que se refere as estruturas herdadas estão descritas nas seções 3.2.2.1.1 e 3.2.2.1.2.

HERANÇA

A Classe Link de Processos foi implementada como uma subclasse de Lista Variável Genérica herdando assim as propriedades estruturais e comportamentais desta super-classe. As propriedades comportamentais, ou seja a parte dinâmica do objeto, é constituída pelas operações internas descritas na seção 3.2.2.1.3, e portanto a classe Link de Processos poderá utilizá-las para manipulação de suas instâncias. A utilização destas operações pode ser feita da seguinte forma: em primeiro lugar, usando diretamente através da invocação das operações da Classe Lista Variável Genérica, ou na implementação dos métodos da Classe Link de Processos.

OPERAÇÕES INTERNAS

O conjunto de operações internas, ou métodos, que compõe a classe Link de Processos, cuja execução é realizada a partir do envio da mensagem correspondente à operação desejada, será descrito a seguir.

CRIAÇÃO - Este método utiliza algumas operações herdadas de sua super-classe para efetuar a inicialização da instância com a alocação de memória para a criação do Header usando parâmetro que atualiza a variável ER_{NC}.
INSERÇÃO - Este método tem por objetivo efetuar a inclusão de um novo elemento Componente na

instância de Link de Processos armazenada em Memória Real. Para a inclusão deste elemento, esta operação utiliza o método de inserção herdado de Lista Variável Genérica.

REMOÇÃO HEADER e REMOÇÃO COMPONENTE- Estas operações internas tem por finalidade efetuar respectivamente a remoção do elemento Header e a remoção de um determinado elementos Componente de uma instância de Link de Processos armazenada na Memória Real. A remoção do Header somente é possível quando não houver mais elementos Componentes na instância.

PESQUISA - Esta operação efetua a busca na lista correspondente a instância de Link de Processos para obter a localização do elemento Componente cuja Identificação Link de Processos (IDLP) seja igual ao string informado via parâmetro.

OBTER NEL e OBTER ERNC - Estes métodos fornecem o conteúdo das variáveis NEL e ER_{NC} respectivamente de uma instância de Link de Processos armazenada em Memória Real.

OBTER IDLP, OBTER STATUS e OBTER INDP - Estes métodos têm por finalidade fornecer o conteúdo das variáveis IDLP, STATUS e INDP respectivamente de um determinado elemento Componente da instância de Link de Processos armazenada em Memória Real.

ALTERAR ERNC - Este método possibilita a alteração do conteúdo da variável ER_{NC} de uma instância de Link de Processos armazenada em Memória Real.

ALTERAR IDLP, ALTERAR STATUS e ALTERAR INDP- Estes métodos têm por objetivo alterar o conteúdo das variáveis IDLP , STATUS e INDP respectivamente do elemento Componente de uma determinada instância de Link de Processos armazenada em Memória Real.

ENCONTRAR - Este método permite localizar um determinado elemento Componente a partir de um parâmetro que informa o número de sequência correspondente.

INCREMENTAR e DECREMENTAR NEL - Estes métodos têm por finalidade efetuar o incremento e o decremento respectivamente, do conteúdo da variável NEL.

CLASSE: ÍNDICE DE OPERAÇÕES INTERNAS

ESTRUTURA

A classe Índice de Operações Internas possui um estrutura interna resultante da composição de sua estrutura interna com as estruturas herdadas das super-classes que compõem a hierarquia descrita na figura 14. A estrutura interna da classe Índice de Operações Internas é composta pelas seguintes sub-estruturas: Header e Componente. A sub-estrutura Header corresponde ao elemento inicial da lista cuja função é manter informações de controle. O elemento Header pode ser compreendido a partir da tupla:

$H_{LP} = < N_{EL}, ER_{NC} >$, onde:

N_{EL} é o Número de procedimentos especiais existentes na instância;
ER_{NC} é o Endereço Relativo Virtual do Nó de Classe de origem.

A variável N_{EL} é um contador de elementos da lista, que tem por objetivo controlar o número de elementos existentes visando evitar a remoção da lista quando ainda existirem elementos ativos. A variável ER_{NC} , cujo objetivo é permitir a recuperação a partir do objeto Índice de Operações Internas, da instância do objeto Nó de Classe que está associada, contém o Endereço Relativo Virtual desta instância.

O elemento Componente do objeto Índice de Operações Internas é definido pela tupla:

$C_{LP} = <$ IDOI, STATUS, INDO>, onde:

IDOI é o Identificador do Índice de Operações Internas;
STATUS fornece o Status do Índice de Operações Internas;
INDO é o índice do método ou operação interna;

O conteúdo da variável IDOI consiste de um string de tamanho 8 com o nome que descreve para o usuário a Operação Interna disponível para o objeto do sistema implementado no Ambiente. O STATUS fornece os parâmetros necessários a manipulação do sistema de segurança permitindo restringir o acesso ao método associado. A variável INDO contém o Índice que permite ativar a execução do método. Cada método implementado nos Objetos do Sistema são ativados a partir de um array cujos elementos são apontadores que tornam possível sua execução.

A estrutura da classe Índice de Operações Internas adquire a forma decorrente da herança das estruturas correspondentes as super-classes descritas nas seções 3.2.2.1.1 e 3.2.2.1.2.

HERANÇA

A herança de características comportamentais da super-classe Lista Variável Genérica, que compreende a sua parte dinâmica, pela classe Índice de Operações Internas, tornou possível sua rápida implementação. Esta característica comportamentais constituída pelas operações internas estão descritas na seção 3.2.2.1.3. Portanto a classe Índice de Operações Internas poderá utilizá-las para manipulação de suas instâncias. A utilização destas operações é feita de duas maneiras: usando diretamente através da invocação das operações da Classe Lista Variável Genérica, ou na implementação dos métodos da Classe Índice de Operações Internas.

OPERAÇÕES INTERNAS

O conjunto de operações internas, ou métodos, que compõe a classe Índice de Operações Internas, cuja execução é realizada a partir do envio da mensagem correspondente à operação desejada, será descrito a seguir.

CRIAÇÃO - Este método utiliza algumas operações herdadas de sua super-classe para efetuar a inicialização da instância com a alocação de memória para a criação do Header usando parâmetro que atualiza a variável ER_{NC}.
INSERÇÃO - Este método tem por objetivo efetuar a inclusão de um novo elemento Componente na instância de Índice de Operações Internas armazenada em Memória Real. Para a inclusão deste elemento, esta operação utiliza o método de inserção herdado de Lista Variável Genérica.
REMOÇÃO HEADER e REMOÇÃO COMPONENTE- Estas operações internas tem por finalidade efetuar respectivamente a remoção do elemento Header e a remoção de um determinado elementos Componente de uma instância de Índice de Operações Internas armazenada na Memória Real. A remoção do Header somente é possível quando não houver mais elementos Componentes na instância.
PESQUISA - Esta operação efetua a busca na lista correspondente a instância de Índice de Operações Internas para obter a localização do elemento Componente cuja Identificação Índice de Operações Internas (IDOP) seja igual ao string informado via parâmetro.
OBTER NEL e OBTER ERNC - Estes métodos fornecem o conteúdo das variáveis N_{EL} e ER_{NC} respectivamente de uma instância de Índice de Operações Internas armazenada em Memória Real.
OBTER IDOP, OBTER STATUS e OBTER INDO - Estes métodos têm por finalidade fornecer o conteúdo das variáveis IDOP, STATUS e INDO respectivamente de um determinado elemento Componente da instância de Índice de Operações Internas armazenada em Memória Real.
ALTERAR ERNC - Este método possibilita a alteração do conteúdo da variável ER_{NC} de uma instância de Índice de Operações Internas armazenada em Memória Real.
ALTERAR IDOP, ALTERAR STATUS e ALTERAR INDO- Estes métodos têm por objetivo alterar o conteúdo das variáveis IDOP , STATUS e INDO respectivamente do elemento Componente de uma determinada instância de Índice de Operações Internas armazenada em Memória Real.
ENCONTRAR - Este método permite localizar um determinado elemento Componente a partir de um parâmetro que informa o número de seqüência correspondente.
INCREMENTAR e DECREMENTAR NEL - Estes métodos têm por finalidade efetuar o incremento e o decremento respectivamente, do conteúdo da variável N_{EL}.

CLASSE: LINK DE COMPOSIÇÃO

ESTRUTURA

A estrutura do Link de Composição é formada a partir da composição de suas estruturas interna com as estruturas herdadas das super-classes que compõem a hierarquia de classes do GOLGO. A estrutura interna da classe Link de Composição é composta pelas seguintes sub-estruturas: Header e Componente. A sub-estrutura Header representa o elemento inicial da lista cuja função é manter informações para controle dos elementos Componentes. A estrutura do elemento Header corresponde a tupla:

H_{LC} = < NEL, ILC, TIPO, ERVO >, onde:

NEL é o Número de Links de Composição existentes na instância;
ILC é o Indicador do tipo de Link de Composição;
TIPO indica o tipo de Objeto de Origem;
ERVO é o Endereço Relativo Virtual do Objeto de Origem.

A variável NEL é na realidade um contador de elementos da lista, tendo por função o controle do número de elementos inseridos para evitar a remoção de toda a lista quando ainda existirem elementos ativos. A variável ILC indica o tipo de Link de Composição da instância, podendo assumir os valores: "N" para indicar um Link de Composição Normal, ou "I" que corresponde ao Link de Composição Invertida. A variável TIPO permite identificar o tipo de objeto de origem, já que uma instância de Link de Composição poderá estar associada a um Nó de Classe (TIPO = 0) ou a um Nó de Composição (TIPO = 1). A variável ERVO contém o Endereço Relativo Virtual da instância do objeto Nó de Composição ou Nó de Classe de origem ao qual o Link de Composição está associado. Sua função básica é permitir a recuperação a partir do objeto Link de Composição, da instância do objeto de origem indicado pela variável TIPO.

O elemento Componente do objeto Link de Composição é definido a partir da tupla:

C_{LC} = < ER_{DO} , STATUS, TLC, IOM, TOB, ERV, POS >, onde:

ER_{DO} é o Endereço Relativo Virtual do Dicionário de Objetos;
STATUS indica o Status do Link de Composição;
TLC indica o tipo de Ligação existente;
IOM é o indicador do objeto do Modelo E/D;
TOB é o indicador do tipo de objeto de gerenciamento associado;
ERV é o Endereço Relativo Virtual do objeto associado;
POS é a Posição Relativa do objeto associado.

A variável ER_{DO} tem por função fornecer a localização das informações no Dicionário de Objetos do Ambiente Poesis, que permitem a identificação do objeto associado pela instância de Link de Composição. STATUS informa os parâmetros necessários a manipulação do sistema de segurança permitindo restringir o acesso ao objeto associado. A variável TLC é usada para identificar o tipo de ligação existente representada pelo Link de Composição (ver capítulo 2). Esta variável assume dois valores: 0 que indica uma composição do tipo ***Fraco***, e 1 que indica a composição do tipo ***Forte***. A variável IOM é usada para identificar o objeto do Modelo E/D associado pelo Link de Composição (ver capítulo 4). A variável TOB identifica o tipo de objeto ao qual o Link de Composição está associado, assumindo os valores: 0 para indicar Nó de Classe e 1 para indicar Nó de Composição. A variável ERV contém o Endereço Relativo Virtual que permite o acesso a instância do objeto associado possibilitando sua transferência para a Memória Real. POS é a variável que indica, conforme o caso, a posição ocupada pela instância a partir do Endereço Relativo Virtual. Esta variável somente é usada para o objeto Nó de Composição.

HERANÇA

A Classe Link de Composição foi implementada como uma subclasse de Lista Variável Genérica herdando assim as propriedades estruturais e comportamentais desta super-classe. As propriedades comportamentais, ou seja a parte dinâmica do objeto, é constituída pelas operações internas descritas na seção 3.2.2.1.3, e portanto a classe Link de Composição poderá utilizá-las para manipulação de suas instâncias. A utilização destas operações pode ser feita da seguinte forma: em primeiro lugar, usando diretamente através da invocação das operações da Classe Lista Variável Genérica, ou na implementação dos métodos da Classe Link de Composição.

OPERAÇÕES INTERNAS

A Classe Link de Composição possui um conjunto de operações internas, ou métodos, cuja execução é realizada a partir do envio da mensagem correspondente à operação desejada. A mensagem identifica então qual a operação interna a ser processada. Estas operações internas serão descritas a seguir.

CRIAÇÃO - Este método tem por finalidade a inicialização da instância através da criação do Header. As variáveis ILC, TIPO e ERVO são inicializadas a partir dos parâmetros informados durante o envio da mensagem correspondente. Este método utiliza algumas operações herdadas de sua super-classe.
INSERÇÃO - Este método efetua a inclusão de um novo elemento Componente na instância de Link de Composição. Para a inclusão deste elemento, esta operação utiliza o método de inserção herdado de Lista Variável Genérica. As variáveis ER_{DO} , STATUS, TLC, IOM, TOB, ERV e POS são inicializadas neste método a partir dos parâmetros informados durante o envio da mensagem correspondente.
REMOÇÃO HEADER - Esta operação interna efetua a remoção do elemento Header, quando não houver mais elementos Componentes na instância.
REMOÇÃO COMPONENTE - Esta operação interna efetua a remoção do elemento Componente da instância.
PESQUISA - Esta operação efetua uma busca na lista correspondente a instância de Link de Composição visando obter a localização do elemento Componente cujo Endereço Relativo do Dicionário de Objetos corresponda ao valor informado via parâmetro.
OBTER NEL, OBTER ILC, OBTER TIPO e OBTER ERVO - Estes métodos fornecem o conteúdo das variáveis NEL, ILC, TIPO e ERVO respectivamente de uma instância de Link de Composição armazenada em Memória Real.
OBTER ERDO, OBTER STATUS, OBTER TLC, OBTER IOM, OBTER TOB, OBTER ERV e OBTER POS- Estes métodos têm por finalidade fornecer o conteúdo das variáveis ER_{DO} , STATUS, TLC, IOM, TOB, ERV e POS respectivamente de um determinado elemento Componente da instância de Link de Composição armazenada em Memória Real.
ALTERAR ILC, ALTERAR TIPO e ALTERAR ERVO - Estes métodos possibilitam a alteração do conteúdo das respectivas variáveis ILC, TIPO e ERVO de uma instância de Link de Composição armazenada em Memória Real.
ALTERAR ERDO, ALTERAR STATUS, ALTERAR TLC, ALTERAR IOM, ALTERAR TOB, ALTERAR ERV e ALTERAR POS- Estes métodos têm por objetivo alterar o conteúdo das variáveis ER_{DO} , STATUS, TLC, IOM, TOB, ERV e POS respectivamente do elemento Componente de uma determinada instância de Link de Composição armazenada em Memória Real.
ENCONTRAR - Este método permite localizar um determinado elemento Componente a partir de um parâmetro que informa o número de seqüência correspondente.
INCREMENTAR e DECREMENTAR NEL - Estes métodos têm por finalidade efetuar o incremento e o decremento respectivamente, do conteúdo da variável NEL.

CLASSE: ÍNDICE DO DICIONÁRIO DE OBJETOS

ESTRUTURA

A estrutura interna do objeto de gerenciamento Índice do Dicionário de Objetos pode ser visualizada e compreendida a partir da tupla:

ID = < IDOB, ER_{DO} >, onde:

IDOB é o identificador do Objeto;
ER_{DO} é o Endereço Relativo Virtual do Dicionário de Objeto;

A variável IDOB corresponde a chave de acesso da Árvore-B que permite recuperar as informações relativas a um determinado Objeto do Sistema. A variável ER_{DO} contém o Endereço Relativo Virtual que permite localizar as informações de um determinado objeto do sistema na instância do Dicionário de Objeto.

A classe Índice do Dicionário de Objetos é subclasse de Árvore-B Genérica. ou seja, a composição da estrutura final que corresponde ao nodo da Árvore-B, deste objeto resultou da herança da estrutura desta super-classe descrita na seção 3.2.2.1.4.

HERANÇA

A Classe Índice do Dicionário de Objetos pelo mecanismo de herança obtido de sua implementação como uma subclasse da Classe Árvore-B Genérica, conforme visto anteriormente, herdou as propriedades estruturais e comportamentais desta super-classe. As propriedades comportamentais que correspondem a parte dinâmica do objeto, ou seja suas operações internas estão descritas na seção 3.2.2.1.2, e podem ser usadas pela classe Índice do Dicionário de Objetos para manipulação de suas instâncias. A utilização destas operações internas, ou métodos da classe, pode ser realizada de duas formas: através do uso direto pela invocação das operações da Classe Árvore-B Genérica, ou através do uso na implementação dos métodos da Classe Índice do Dicionário de Objetos.

OPERAÇÕES INTERNAS

A Classe Índice do Dicionário de Objetos é manipulada a partir do um conjunto de operações internas, ou métodos, que serão descritos a seguir. A execução destas operações é realizada a partir do envio da mensagem ao objeto Índice do Dicionário de Objetos que as identifica.

CRIAÇÃO - Este método efetua a criação de nodos numa instância do Índice do Dicionário de Objetos parcialmente alocada na Memória Real. Esta criação corresponde a alocação e manipulação parcial das instâncias deste tipo de objeto, ou seja, não existe a necessidade de manipular toda a Árvore-B em Memória Real. Em sua implementação foi utilizado o métodos herdado da classe Árvore-B Genérica: Inserção-Parcial.
INSERÇÃO - Este método efetua a inserção de nodos numa instância de Índice de Dicionário de Objetos totalmente armazenada em Memória Real. Este método corresponde à operação interna Inserção herdada da classe Árvore-B Genérica.
OBTER ERDO - Este método tem por finalidade fornecer o conteúdo da variável ER_{DO} de um nodo da instância do Índice do Dicionário de Objetos armazenada em Memória Real.
ALTERAR ERDO - Este método permite alterar o conteúdo da variável ER_{DO} de um nodo da instância do Índice do Dicionário de Objetos armazenada em Memória Real.
OBTER IDENTIFICADOR DO OBJETO - Este método tem por finalidade fornecer o conteúdo da variável IDOB de um determinado nodo localizado na instância do Índice do Dicionário de Objetos.

CLASSE: DICIONÁRIO DE OBJETOS

ESTRUTURA

A classe Dicionário de Objetos é o resultado da composição de estruturas herdadas com a estrutura própria da classe. A classe Dicionário de Objetos é subclasse de Lista Fixa Genérica. Isto significa que todas as estruturas que formam esta super-classe, descrita na seção 3.2.2.1.2 , serão herdadas para compor uma estrutura final para o objeto. Logo, a implementação do objeto Dicionário

de Objetos consiste de uma lista da classe Lista Fixa cujos elementos componentes são instâncias de Dicionário de Objetos.

A estrutura própria da classe Dicionário de Objetos consiste de um conjunto de variáveis que pode ser compreendido a partir da tupla:

DO = < IDOB, NOME, DESCRITOR, TOB, ERV. POS >, onde:

IDOB é o Identificador do objeto (chave de identificação);
NOME é nome externo usado para representar o objeto;
DESCRITOR é a cadeia de caracteres que descreve o objeto;
TOB indica o tipo de objeto associado à identificação;
ERV é o Endereço Relativo Virtual do Objeto associado;
POS é a Posição Relativa de localização do objeto associado.

A variável IDOB corresponde a chave de acesso ao Dicionário de Objetos que permite recuperar as informações relativas a um determinado Objeto do Sistema. NOME é a variável cujo conteúdo representa a denominação do Objeto sendo usada na interface com o usuário. A variável DESCRITOR é usada para descrever o objeto. A variável TOB contém o tipo de objeto de gerenciamento associado e assume os valores: 0 para indicar o objeto Nó de Classe, e 1 para indicar o Nó de Composição. A variável ERV contém o Endereço Relativo Virtual que permite localizar a instância do objeto de gerenciamento associada. E finalmente, a variável POS contém a posição relativa ocupada pela instância do objeto de gerenciamento associado.

HERANÇA

A Classe Dicionário de Objetos conforme visto anteriormente herdou as propriedades estruturais da Classe Lista Fixa Genérica. As propriedades comportamentais desta super-classe que correspondem a parte dinâmica do objeto, ou seja suas operações internas e descritas na seção 3.2.2.1.2, são também herdadas e podem ser usadas pela classe Dicionário de Objetos para manipulação de suas instâncias de duas maneiras: diretamente invocando as operações da Classe Lista Fixa Genérica, ou usando na implementação dos métodos da Classe Dicionário de Objetos.

OPERAÇÕES INTERNAS

A Classe Dicionário de Objetos somente poderá ser manipulada a partir de um conjunto de operações internas, ou métodos, cuja execução é realizada a partir do envio da mensagem ao objeto Dicionário de Objetos que a identifica. Estes métodos serão descritos a seguir.

CRIAÇÃO - Esta operação interna efetua a criação de uma instância do Dicionário de Objetos em Memória Real de forma parcial, não existe a necessidade de manipular todas as instâncias nesta memória. Em sua implementação foi utilizado o métodos herdado da classe Lista Fixa Genérica: Inserção-Parcial.
INSERÇÃO - Esta operação tem por objetivo efetuar a inicialização de uma instância de Dicionário de Objetos, porém com todas as instâncias do Dicionário de Objetos estão sendo manipuladas em Memória Real. Este método corresponde à operação interna Inserção-Real herdado da classe Lista Fixa.
EXTRAÇÃO - Este método efetua a remoção lógica de uma instância do objeto Dicionário de Objetos armazenada na memória persistente a partir do parâmetro que informa seu Endereço Relativo Virtual.
REMOÇÃO - Este método efetua a remoção lógica de uma instância do Dicionário de Objetos armazenada na Memória Real, a partir do fornecimento de um parâmetro que irá informar o Endereço Relativo Virtual de localização da instância.
OBTER IDOB, OBTER NOME, OBTER DESCRITOR, OBTER TOB, OBTER ERV e OBTER POS - Estes métodos têm por finalidade fornecer respectivamente o conteúdo das variáveis: IDOB,

NOME, DESCRITOR, TOB, ERV, POS de uma instância do Dicionário de Objetos.
ALTERAR NOME, ALTERAR DESCRITOR, ALTERAR TOB, ALTERAR ERV e ALTERAR POS - Estes métodos têm por objetivo efetuar a alteração do conteúdo das respectivas variáveis: NOME, DESCRITOR, TOB, ERV, POS de uma instância do Dicionário de Objetos.

CLASSE: VERSÕES

ESTRUTURA

A estrutura do objeto Versões é formada pela composição de suas estruturas interna com as estruturas herdadas das super-classes que compõem a hierarquia de classes do GOLGO apresentada pela figura 14. A estrutura interna da classe Versões é composta pelas seguintes sub-estruturas: Header e Componente. A sub-estrutura Header representa o elemento inicial da lista cuja função é manter informações para controle dos elementos Componentes. A estrutura do elemento Header corresponde a tupla:

$H_V = < NEL,\ ERV_{NC} >,$ onde:

NEL indica o Número de elementos existentes na instância;
ERV_{NC} é o Endereço Relativo Virtual do Nó de Classe associado.

A variável NEL é um contador de elementos da lista, tendo por função o controle do número de elementos inseridos para evitar a remoção de toda a lista quando ainda existirem elementos ativos. A variável ERV_{NC} contém o Endereço Relativo Virtual da instância do objeto Nó de Classe.

O elemento Componente do objeto Versões é definido a partir da tupla:

$C_V = < ER_{DO}\ ,\ STATUS,\ DATA,\ ER_{NC}\ >,$ onde:

ER_{DO} é o Endereço Relativo Virtual do Dicionário de Objetos;
STATUS indica o Status do Versões;
DATA indica a data da versão;
ER_{NC} é o Endereço Relativo Virtual do Nó de Classe da versão.

A variável ER_{DO} visa fornecer a localização das informações no Dicionário de Objetos do Ambiente Poesis, que permitem a identificação do objeto associado pela instância de Versões. A variável STATUS informa os parâmetros necessários a manipulação do sistema de segurança permitindo restringir o acesso à versão. A variável DATA é usada para identificar a data de criação da versão. A variável ERV_{NC} contém o Endereço Relativo Virtual que permite o acesso a instância do objeto Nó de Classe que representa a versão associada, possibilitando sua transferência para a Memória Real.

As estruturas herdadas correspondem as estruturas das super-classes Lista Genérica e Lista Variável Genérica descritas nas seções 3.2.2.1.1 e 3.2.2.1.2.

HERANÇA

As propriedades comportamentais, ou seja a parte dinâmica do objeto, herdadas pela classe Versões correspondem às operações internas descritas na seção 3.2.2.1.3. Portanto a classe Versões poderá utilizá-las para manipulação de suas instâncias. A utilização destas operações pode ser feita da seguinte forma: em primeiro lugar, usando diretamente através da invocação das operações da Classe Lista Variável Genérica, ou na implementação dos métodos da Classe Versões.

OPERAÇÕES INTERNAS

A Classe Versões possui um conjunto de operações internas, ou métodos, cuja execução é realizada a partir do envio da mensagem correspondente à operação desejada. A mensagem identifica

então qual a operação interna a ser processada. Estas operações internas serão descritas a seguir.

CRIAÇÃO - Este método tem por finalidade a inicialização da instância através da criação do Header. Este método utiliza algumas operações herdadas da super-classe Lista Variável Genérica.
INSERÇÃO - Este método efetua a inclusão de um novo elemento Componente na instância de Versões. Para a inclusão deste elemento, esta operação utiliza o método de inserção herdado de Lista Variável Genérica. As variáveis ER_{DO} , STATUS, DATA e ER_{NC} são inicializadas neste método a partir dos parâmetros informados durante o envio da mensagem correspondente.
REMOÇÃO HEADER - Esta operação interna efetua a remoção do elemento Header, quando não houver mais elementos Componentes na instância.
REMOÇÃO COMPONENTE - Esta operação interna efetua a remoção do elemento Componente da instância.
PESQUISA - Esta operação efetua uma busca na lista correspondente a instância de Versões visando obter a localização do elemento Componente cujo Endereço Relativo do Dicionário de Objetos corresponda ao valor informado via parâmetro.
OBTER NEL e OBTER ERV_{NC} - Estes métodos fornecem o conteúdo das variáveis NEL, e ERV_{NC} respectivamente de uma instância de Versões armazenada em Memória Real.
OBTER ERDO, OBTER STATUS, OBTER DATA e OBTER ER_{NC} - Estes métodos têm por finalidade fornecer o conteúdo das variáveis ER_{DO} , STATUS, DATA e ER_{NC} respectivamente de um determinado elemento Componente da instância de Versões armazenada em Memória Real.
ALTERAR ERDO, ALTERAR STATUS, ALTERAR DATA e ALTERAR ER_{NC} - Estes métodos têm por finalidade fornecer o conteúdo das variáveis ER_{DO} , STATUS, DATA e ER_{NC} respectivamente de um determinado elemento Componente da instância de Versões armazenada em Memória Real.
ENCONTRAR - Este método permite localizar um determinado elemento Componente a partir de um parâmetro que informa o número de seqüência correspondente.
INCREMENTAR e DECREMENTAR NEL - Estes métodos têm por finalidade efetuar o incremento e o decremento respectivamente, do conteúdo da variável NEL.

CLASSE: RESTRIÇÕES

ESTRUTURA

A estrutura interna da classe é o resultado da composição decorrente de sua implementação como subclasse de Lista Fixa Genérica. Logo, a estrutura final do Objeto de Gerenciamento Restrições é composta de:

- Estrutura própria da classe.
- Estruturas Herdadas da super-classe;

A estrutura própria da classe Restrições pode ser visualizada e compreendida a partir da tupla:

$R = < L_{NCP}, L_{NS}, L_{LCN}, L_{LCI}, L_V >$, onde:

L_{NCP} é o número máximo de instâncias de Nó de Composição;
L_{NI} é o número máximo de instâncias de Nó de Instância;
L_{LN} é o número máximo de instâncias de Link de Navegação;
L_{LCN} é o número máximo de instâncias de Links de Composição Normal;
L_{LCI} é o número máximo de instâncias de Link de Composição Invertida;
L_V é o número máximo de instâncias de Versões

As variáveis: L_{NCP} , L_{NI} , L_{LN} , L_{LCN} , L_{LCI} e L_V são utilizadas para manipular as restrições sobre o número de instâncias dos respectivos objetos de gerenciamento: Nó de Composição, Nó de Instância, Link de Navegação, Link de Composição (instâncias: Normal e Invertida) e Versões. O valor nulo (no caso, o zero) indica que um determinado Objeto do Sistema, representado pelo Nó de Classe não poderá ter associado a instância controlada por um dos parâmetros descritos acima.

As estruturas herdadas da super-classe Lista Fixa Genérica estão descritas na seção 3.2.2.1.2. A estrutura final para o objeto Restrições pode ser visualizada 3222l. Portanto, a implementação do objeto Restrições consiste de uma lista cujas características são dadas pela classe Lista Fixa sendo seus elementos componentes instâncias de Restrições.

HERANÇA

Conforme visto anteriormente, a classe Restrições foi implementada como subclasse de Lista Fixa Genérica e por isso desta forma herdou suas propriedades estruturais e comportamentais. As propriedades comportamentais, ou operações internas, descritas na seção 3.2.2.1.2, são utilizadas pela classe Restrições, para manipulação de suas instâncias, de duas maneiras: através do uso direto, invocando as operações da Classe Lista Fixa Genérica, ou usando algumas destas operações internas na implementação dos métodos da Classe Restrições.

OPERAÇÕES INTERNAS

As operações internas da classe Restrições são executadas a partir do envio da mensagem ao objeto Restrições que as identifica. Estas operações internas serão descritas a seguir.

CRIAÇÃO - Este método realiza a inicialização de uma instância na Memória Real na modalidade de processamento parcial, ou seja, apenas a instância que está sendo inicializada é alocada na Memória Real. Este método utiliza em sua implementação a operação Inserção Parcial herdada da classe Lista Fixa Genérica.
INSERÇÃO - Esta operação inicializa a instância de Restrições. Neste método todas as instâncias do Restrições estão sendo manipuladas em Memória Real. Este método corresponde à operação interna Inserção-Real herdado da classe Lista Fixa.
OBTER LIMITE - Este método tem por finalidade fornecer o conteúdo de uma das variáveis L_{NCP} , L_{NI} , L_{LN} , L_{LCN} , L_{LCI} e L_V, a partir de um parâmetro que indica qual das variáveis deve ser recuperada, de uma instância de Restrições, cujo Endereço Relativo é informado via parâmetro.
ALTERAR LIMITE - Este método tem por finalidade fornecer o conteúdo de uma das variáveis L_{NCP} , L_{NI} , L_{LN} , L_{LCN} , L_{LCI} e L_V, a partir de um parâmetro que indica qual das variáveis deve ser recuperada, de uma instância de Restrições, cujo Endereço Relativo é informado via parâmetro.

CLASSE: DEFINIÇÃO VISUAL

ESTRUTURA

A implementação do objeto Definição Visual como subclasse de Lista Variável Genérica permitiu que a estrutura deste Objeto de Gerenciamento herdasse as estruturas que compõe a super-classe. A estrutura da classe Definição Visual é formada então, pela composição de suas estruturas interna com as estruturas herdadas das super-classes que compõem a hierarquia de classes do GOLGO apresentada pela figura 14. A estrutura interna da classe Definição Visual é formada pelas seguintes sub-estruturas: Header e Componente. A sub-estrutura Header representa o elemento inicial da lista cujo objetivo é manter informações que controlam os elementos Componentes. A estrutura do elemento Header corresponde a tupla:

H_{DV} = < NDV, ERV_{NC} >, onde:

NDV indica o Número de Definições Visuais existentes na instância;
ERV_{NC} é o Endereço Relativo Virtual do Nó de Classe associado.

A variável NDV é um contador de elementos ou Definições Visuais da lista, tendo por função o controle do número de elementos inseridos. A variável ERV_{NC} contém o Endereço Relativo Virtual da instância do objeto Nó de Classe, para permitir assim o acesso ao Objeto do Sistema ao qual a instância de Definição Visual está associada.

O elemento Componente do objeto Definição Visual é definido a partir da tupla:

C_{DV} = < ID_{DV} , STATUS, T_F, T_{MT} , T_{CS} , FORM, MT, CS >, onde:

ID_{DV}	é o Identificador da Definição Visual;
STATUS	indica o Status do Definição Visual;
T_F	é o tamanho do Formato;
T_{MT}	é o tamanho do Mapa de Teclas;
T_{CS}	é o tamanho do Contexto Sensitivo;
FORM	contém o código para geração da Definição Visual;
MT	contém o Mapa de Teclas da Definição Visual;
CS	contém o Contexto Sensitivo da Definição Visual.

A variável ID_{DV} é usada para identificar no Ambiente a Definição Visual de um determinado Objeto do Sistema, como também localiza as informações que permitem a ativação da Definição Visual do objeto associado. A variável STATUS informa os parâmetros necessários a manipulação do sistema de segurança permitindo restringir a ativação de determinadas Definições Visuais. As variáveis T_F , T_{MT}, T_{CS} são usadas para controlar os tamanhos respectivos do Formato, Mapa de Teclas e Contexto Sensitivo. São também utilizadas para localização da posição de início de cada parâmetro, bem como determinar de existe para aquela Definição Visual os parâmetros Mapa de Teclas e Contexto Sensitivo. O parâmetro FORM contém o código que será fornecido ao Objeto da classe Dispositivos para ativação da Definição Visual. O parâmetro MT é opcional e contém o conjunto de teclas ativas. O parâmetro CS, também opcional, contém o conjunto das regiões sensíveis ao click do Mouse da Definição Visual associada.

As estruturas herdadas correspondem as estruturas das super-classes Lista Genérica e Lista Variável Genérica descritas nas seções 3.2.2.1.1 e 3.2.2.1.2.

HERANÇA

As propriedades comportamentais que correspondem às operações internas descritas na seção 3.2.2.1.3, são herdadas pela classe Definição Visual pelo mecanismo de herança da hierarquia de classes. Portanto, a classe Definição Visual poderá utilizá-las para manipulação de suas instâncias da seguinte forma: em primeiro lugar, usando-as diretamente através da invocação das operações da Classe Lista Variável Genérica, ou na implementação dos métodos da Classe Definição Visual.

OPERAÇÕES INTERNAS

A Classe Definição Visual possui um conjunto de operações internas, ou métodos, próprios implementados a partir de métodos herdados das super-classes que compõem a hierarquia de classes do GOLGO. A execução destas operações internas é realizada pelo envio da mensagem que as identifica sendo assim processada. Estas operações internas serão descritas a seguir.

CRIAÇÃO - Este método tem por finalidade a inicialização da instância através da criação do Header. Este método utiliza algumas operações herdadas da super-classe Lista Variável Genérica.
INSERÇÃO - Este método efetua a inclusão de um novo elemento Componente na instância de Definição Visual. Para a inclusão deste elemento, esta operação utiliza o método de inserção herdado de Lista Variável Genérica. As variáveis que compõem a estrutura Componente são fornecidas a partir dos parâmetros informados durante o envio da mensagem correspondente.
REMOÇÃO HEADER - Esta operação interna efetua a remoção do elemento Header, quando não houver mais elementos Componentes, ou Definições Visuais, na instância.
REMOÇÃO COMPONENTE - Esta operação interna efetua a remoção do elemento Componente da instância.
PESQUISA - Esta operação efetua uma busca na lista correspondente a instância de Definição Visual visando obter a localização do elemento Componente cuja Identificação da Definição Visual (ID_{DV}) corresponda ao valor informado via parâmetro.

OBTER NDV e OBTER ERV_{NC} - Estes métodos fornecem o conteúdo das variáveis NDV, e ERV_{NC} respectivamente de uma instância de Definição Visual armazenada em Memória Real.

OBTER IDDV, OBTER_STATUS, OBTER TAM-FORMATO, OBTER TAM-MTECLAS, OBTER TAM-CSENSITIVO, OBTER FORMATO, OBTER MTECLAS, OBTER CSENSITIVO - Estes métodos têm por finalidade fornecer o conteúdo das variáveis ID_{DV} , STATUS, T_F, T_{MT}, T_{CS}, FORM, MT, CS respectivamente de um determinado elemento Componente da instância de Definição Visual armazenada em Memória Real.

ALTERAR IDDV, ALTERAR STATUS, ALTERAR TAM-FORMATO, ALTERAR TAM-MTECLAS, ALTERAR TAM-CSENSITIVO, ALTERAR FORMATO, ALTERAR MTECLAS, ALTERAR CSENSITIVO - Estes métodos têm por objetivo efetuar a alteração do conteúdo das variáveis ID_{DV} , STATUS, T_F, T_{MT}, T_{CS}, FORM, MT, CS respectivamente de um determinado elemento Componente da instância de Definição Visual armazenada em Memória Real.

ENCONTRAR - Este método permite localizar um determinado elemento Componente a partir de um parâmetro que informa o número de seqüência correspondente.

INCREMENTAR e DECREMENTAR NDV - Estes métodos têm por finalidade efetuar o incremento e o decremento respectivamente, do conteúdo da variável NDV.

APÊNDICE II

CLASSES DO GERENCIAMENTO DE DISPOSITIVOS

CLASSE: DISPOSITIVOS

ESTRUTURA

A classe Dispositivos por ser uma super-classe possui uma estrutura interna com variáveis genéricas herdadas por todas as subclasses subordinadas. Estas variáveis também denominadas de *variáveis de classe* podem ser compreendidas pela tupla:

$< ID_D , STAT >$, onde:

ID_D é o Identificador do Dispositivo;
STAT indica o Status (estado) de um determinado Dispositivo.

A variável ID_D permite identificar uma instância de Dispositivo que está ativa no sistema sendo que os valores assumidos por ela pertence ao conjunto { 0, 1, 2, 3, 4, 5, 6, 7, 8 }, onde:

0 indica objeto ENTRADA;
1 indica objeto EDITOR DE TEXTO;
2 indica objeto JANELA;
3 indica objeto ÍCONE;
4 indica objeto FRASE;
5 indica objeto SAÍDA;
6 indica objeto MOUSE;

A variável STAT é usada para armazenar o estado em que se encontra um determinado dispositivo além de informar quando ocorrer, o tipo de erro produzido durante o processamento deste. Esta variável assume os seguintes valores:

0 indica que o dispositivo está ativo, sem problemas;
1 indica falha na inicialização;
2 indica que o dispositivo não corresponde às características previstas;
3 indica que o dispositivo está em pausa.

OPERAÇÕES INTERNAS

As operações internas da classe Dispositivo serão compartilhadas com as subclasses que compõem a hierarquia de dispositivo apresentada anteriormente. As operações internas, ou métodos que manipulam as instâncias da classe Dispositivos serão descritas a seguir.

INICIALIZAÇÃO - Este método tem por finalidade básica a inicialização de um determinado dispositivo acoplado ao sistema. Em caso de falha, é informado ao Ambiente o tipo de erro ocorrido.
OBTER IDENTIFICADOR DO DISPOSITIVO - Este método fornece o conteúdo da variável ID_D de uma determinada instância de dispositivo que foi ativado.
OBTER STATUS - Este método tem por objetivo fornecer o conteúdo da variável STAT de uma determinada instância de dispositivo ativo.
FINALIZAÇÃO - Este método permite a desativação de um determinado dispositivo.

Estes métodos serão herdados por todas as subclasses que compõem a hierarquia de dispositivos apresentada na figura 27.

CLASSE: ENTRADA

ESTRUTURA

A classe Entrada possui uma estrutura composta por variáveis herdadas da super-classe: Dispositivo, dada pela tupla:

$< ID_D , STAT >$,

além das variáveis da que constituem a estrutura própria desta classe dadas pela tupla:

< COORD, TC, TAM, IPO, IE, IAV , IDP > , onde:

COORD indica as ***coordenadas*** X e Y do vídeo onde serão ecoados ou não o campo que será processado. Estas coordenadas estão limitadas à fronteira imposta pelo modalidade de processamento do vídeo sendo definidas em função de outro tipo de Definição Visual (ver objeto Janela) ao qual está associada.

A variável TC indica o ***tipo de campo*** a ser processado podendo assumir os valores pertencentes ao conjunto { I, L, C, S, F, D, H }, onde:

I indica que o campo é do tipo Inteiro;
L indica que o campo é do tipo Inteiro Longo;
C indica que o campo é do tipo Caracter;
S indica que o campo é do tipo String;
F indica que o campo é do tipo Real;
D indica que o campo é do tipo Data (dd-mm-aaaa);
H indica que o campo é do tipo Hora (hh-mm-ss).

A variável TAM indica o ***tamanho do campo*** a ser ecoado ou não no dispositivo Vídeo. Para alguns tipos de campos o tamanho já é pré-definido, como no caso dos tipos C, D e H. A variável IPO é o ***Indicador de Preenchimento Obrigatório*** usado para indicar se um determinado campo poderá ser aceito com (valor 1) ou sem (valor 0) seu preenchimento quando for ativado. A variável IE é o ***Indicador de Eco do Teclado*** usado para determinar se os caracteres que estão sendo digitados para um determinado campo serão ou não ecoados no vídeo. Esta variável assume o valor 0 para indicar que o campo não será ecoado e 1 caso contrário. A variável IAV é o ***Indicador de Atributo de Vídeo*** usado para determinar a forma pela qual o campo aparecerá visualmente para o usuário. Esta variável assume os seguintes valores que poderão ser combinados para propiciar uma forma agradável de recepção do campo:

0 indica a modalidade Normal;
1 indica a modalidade de campo em Vídeo Reverso;
2 indica a modalidade de campo Piscante;
3 indica a modalidade de campo em Brilho Intensificado;
4 indica a modalidade de campo Sublinhado.

A variável IDP indica a região de memória usada para armazenar o valor do campo informado quando da ativação deste tipo de objeto durante o processamento do ambiente.

OPERAÇÕES INTERNAS

A classe Entrada pela hierarquia apresentada nas seções anteriores herda todas as operações definidas para a super-classe Dispositivo, sendo usadas para manipulação de suas instâncias. Esta classe porém, possui algumas operações internas próprias, criadas usando inclusive as operações herdadas de sua super-classe. Estas operações internas, ou métodos, serão descritas a seguir.

INICIALIZAÇÃO - Este método tem por finalidade básica a inicialização do objeto Entrada verificando os dispositivos teclado e vídeo acoplados ao sistema. Em caso de falha, é informado ao Ambiente o tipo de erro ocorrido.
ATIVAÇÃO - Este método permite que a classe Entrada execute o processamento da Definição Visual do tipo Entrada informada via parâmetro.
OBTER-COORDENADAS, OBTER-TC, OBTER-TAM, OBTER-IPO, OBTER-IE, OBTER-IAV , OBTER-IDP - Estes métodos fornecem o conteúdo das variáveis COORD, TC, TAM, IPO, IE, IAV e IDP respectivamente, de uma determinada instância do processador genérico de Definição Visual que foi ativada.
ALTERAR-COORDENADAS, ALTERAR-TC, ALTERAR-TAM, ALTERAR-IPO, ALTERAR-IE, ALTERAR-IAV , ALTERAR-IDP - Estes métodos permitem a alteração do conteúdo das variáveis COORD, TC, TAM, IPO, IE, IAV e IDP respectivamente, de uma determinada instância do processador genérico de Definição Visual que foi ativada.
FINALIZAÇÃO - Este método permite a desativação do dispositivo Entrada e utiliza a operação interna com o mesmo nome da super-classe Dispositivo.

CLASSE: EDITOR DE TEXTO

ESTRUTURA

A classe Editor de Texto possui uma estrutura composta por variáveis herdadas da super-classe Dispositivo, dada pela tupla:

$< ID_D , STAT >$,

acrescida das variáveis que constituem sua estrutura própria dadas pela tupla:

< FRONT, COR, TAM, MODO, IDP > , onde:

FRONT indica a ***fronteira*** que delimita a região da tela onde será exibido e manipulado o texto a ser editado. Este parâmetro é composto pelas variáveis: X1 , Y1 e X2, Y2, sendo que o par (X1, Y1) determina as coordenadas do canto inferior esquerdo e o par (X2, Y2) as coordenadas do canto superior direito da fronteira. A variável COR indica a ***cor*** na qual será exibido o texto no dispositivo Vídeo. O conjunto de cores é dado em função do tipo de placa gráfica (CGA-EGA-VGA) que constitui a configuração do equipamento onde será implantado o Ambiente Poesis. A variável TAM indica o ***tamanho do texto*** a ser editado. A variável MODO indica a modalidade de exibição do texto a ser editado que poderá ser 0 indicando o modo texto (25x80) ou 1 em modo gráfico (alta resolução) variável conforme a placa gráfica instalada. A variável IDP é o Identificador do Processo de Alocação que indica a região de memória usada para armazenar o texto a ser editado durante o processamento do ambiente.

OPERAÇÕES INTERNAS

A classe Editor de Texto herda as operações definidas para a super-classe Dispositivo, sendo usadas para manipulação de suas instâncias. Esta classe possui algumas operações internas próprias que serão descritas a seguir.

INICIALIZAÇÃO - Este método tem por finalidade básica a inicialização do objeto Editor de Texto

verificando os dispositivos teclado e vídeo acoplados ao sistema. Em caso de falha, é informado ao Ambiente o tipo de erro ocorrido.

ATIVAÇÃO - Este método permite que a classe Editor de Texto execute o processamento da Definição Visual do tipo Editor de Texto informada via parâmetro.

OBTER-FRONTEIRAS, OBTER-COR , OBTER-TAM, OBTER-MODO e OBTER-IDP - Estes métodos fornecem o conteúdo das variáveis FRONT, COR, TAM, MODO e IDP respectivamente, de uma determinada instância do processador genérico de Definição Visual que foi ativada.

ALTERAR-FRONTEIRAS, ALTERAR-COR, ALTERAR-TAM, ALTERAR-MODO e ALTERAR-IDP - Estes métodos permitem a alteração do conteúdo das variáveis FRONT, COR, TAM, MODO e IDP respectivamente, de uma determinada instância do processador genérico de Definição Visual que foi ativada.

FINALIZAÇÃO - Este método permite a desativação do dispositivo Editor de Texto e utiliza a operação interna com o mesmo nome da super-classe Dispositivo.

CLASSE: JANELA

ESTRUTURA

A classe Janela possui uma estrutura composta por variáveis herdadas da super-classe: Dispositivo, dada pela tupla:

$< ID_D , STAT >$,

além das variáveis da que constituem a estrutura própria desta classe dadas pela tupla:

< FRONT , TB , CI , PP , Ne, Ae >, onde:

FRONT indica a ***fronteira*** da janela composta pelas variáveis: X1 , Y1 e X2, Y2 que delimitam a região do vídeo onde será montada a Definição Visual, sendo que o par (X1, Y1) determina as coordenadas do canto inferior esquerdo e o par (X2, Y2) as coordenadas do canto superior direito da janela. A variável TB indica o ***tipo de moldura ou borda*** da Definição Visual Janela a ser processada. A variável CI indica a ***cor interna*** na qual será exibida a janela no dispositivo Vídeo. O conjunto de cores é dado em função do tipo de placa gráfica (CGA-EGA-VGA) que constitui a configuração do equipamento onde será implantado o Ambiente Poesis. A variável PP indica o ***Padrão de Preenchimento de fundo*** usado para gerar janelas que permitam uma melhor apresentação ao usuário, através do preenchimento da região de fundo da janela com padrões pré-definidos. A variável Ne indica o ***número de elementos*** ou Definições Visuais que compõem a janela. A variável Ae é um ***array de elementos ou Definições Visuais*** que compõem a janela e que serão ativadas no momento da ativação da referida Definição Visual. Esta estrutura permite inclusive que outra janela seja ativada fornecendo a este tipo de Definição Visual uma flexibilidade para composição de elementos. A variável ***Ae*** possui a quantidade de ocorrências determinada pela variável ***Ne,*** e sua estrutura corresponde a tupla:

$< C_{DV} , IA >$, onde:

C_{DV} indica o código da Definição Visual componente da janela;
IA é o Indicador de Ativação da Definição Visual.

A variável C_{DV} permite a identificação da Definição Visual que será ativada ou não quando a janela estiver sendo construída. A variável IA é o indicador que permite determinar se a Definição Visual componente será ativada no momento da ativação da janela (IA = 1), ou não (IA = 0).

OPERAÇÕES INTERNAS

A classe Janela herda todas as operações definidas para a super-classe Dispositivo, sendo usadas para manipulação de suas instâncias. Esta classe porém, possui algumas operações internas próprias, criadas usando inclusive as operações herdadas de sua super-classe. Estas operações internas, ou métodos, serão descritas a seguir.

INICIALIZAÇÃO - Este método tem por finalidade básica a inicialização do objeto Janela verificando o dispositivo vídeo acoplado ao sistema. Em caso de falha, é informado ao Ambiente o tipo de erro ocorrido.
ATIVAÇÃO - Este método permite que a classe Janela execute o processamento da Definição Visual do tipo Janela informada via parâmetro.
OBTER-FRONTEIRAS, OBTER-TB, OBTER-CI, OBTER-PP, OBTER-Ne - Estes métodos fornecem o conteúdo das variáveis FRONT, TB, CI, PP e Ne respectivamente, de uma determinada instância do processador genérico de Definição Visual que foi ativada.
OBTER-ELEMENTO - Este método permite obter as informações referentes às Definições Visuais que compõem a Janela, definidas através das variáveis C_{DV} e IA.
ALTERAR-FRONTEIRAS, ALTERAR-TB, ALTERAR-CI, ALTERAR-PP, ALTERAR-Ne - Estes métodos possibilitam a alteração do conteúdo das variáveis FRONT, TB, CI, PP e Ne respectivamente, de uma determinada instância do processador genérico de Definição Visual que foi ativada.
ALTERAR-ELEMENTO - Este método permite efetuar alterações das informações referentes às Definições Visuais que compõem a Janela, definidas através das variáveis C_{DV} e IA.
FINALIZAÇÃO - Este método permite a desativação do dispositivo Janela e utiliza a operação interna com o mesmo nome da super-classe Dispositivo.

CLASSE: ÍCONE

ESTRUTURA

A classe Ícone possui uma estrutura composta por variáveis herdadas da super-classe Dispositivo, dada pela tupla:

$< ID_D , STAT >$,

além das variáveis da que constituem a estrutura própria desta classe dadas pela tupla:

< FRONT, CD, CFR, CFU, TM, MASC >, onde:

FRONT indica a ***fronteira*** do Ícone composta pelas variáveis: X1 , Y1 e X2, Y2 que delimitam a região correspondente ao tamanho do ícone, sendo que o par (X1, Y1) determina as coordenadas do canto inferior esquerdo e o par (X2, Y2) as coordenadas do canto superior direito do Ícone. A variável CD indica ***as coordenadas default*** (X, Y) onde a Definição Visual Ícone será exibida. As variáveis CFR e CFU indicam respectivamente a ***cor de frente*** e a ***cor de fundo*** nas quais será exibido o Ícone no dispositivo Vídeo. O conjunto de cores é dado em função do tipo de placa gráfica (CGA-EGA-VGA) que constitui a configuração do equipamento onde será implantado o Ambiente Poesis. A variável TM indica o ***Tamanho da Máscara de Edição*** e a variável MASC contém a ***Máscara de Edição*** que correspondem aos códigos que definem o ícone a ser gerado.

OPERAÇÕES INTERNAS

A classe Ícone herda todas as operações definidas para a super-classe Dispositivo, sendo usadas para manipulação de suas instâncias. Esta classe porém, possui algumas operações internas próprias, criadas usando inclusive as operações herdadas de sua super-classe. Estas operações internas, ou métodos, serão descritas a seguir.

INICIALIZAÇÃO - Este método tem por finalidade básica a inicialização do objeto Ícone

verificando o dispositivo vídeo acoplado ao sistema. Em caso de falha, é informado ao Ambiente o tipo de erro ocorrido.

ATIVAÇÃO - Este método permite que a classe Ícone execute o processamento da Definição Visual do tipo Ícone informada via parâmetro.

OBTER-FRONTEIRAS, OBTER-CD, OBTER-CFR, OBTER-CFU, OBTER-TM e OBTER-MASC - Estes métodos fornecem o conteúdo das variáveis FRONT, CD, CFR, CFU, TM e MASC respectivamente, de uma determinada instância do processador genérico de Definição Visual que foi ativada.

A**LTERAR-FRONTEIRAS, ALTERAR-CD, ALTERAR-CFR, ALTERAR-CFU, ALTERAR-TM e ALTERAR-MASC** - Estes métodos possibilitam a alteração do conteúdo das variáveis FRONT, CD, CFR, CFU, TM e MASC respectivamente, de uma determinada instância do processador genérico de Definição Visual que foi ativada.

FINALIZAÇÃO - Este método permite a desativação do dispositivo Ícone e utiliza a operação interna com o mesmo nome da super-classe Dispositivo.

CLASSE: FRASE

ESTRUTURA

A classe Frase possui uma estrutura composta por variáveis herdadas da super-classe: Dispositivo, dada pela tupla:

$< ID_D , STAT >$,

e pelas variáveis constituem a estrutura própria desta classe dadas pela tupla:

< FRONT, CD, COR, STRING >, onde:

FRONT indica a ***fronteira*** da Frase composta pelas variáveis: X1 , Y1 e X2, Y2 que delimitam a região correspondente ao tamanho do Frase, sendo que o par (X1, Y1) determina as coordenadas do canto inferior esquerdo e o par (X2, Y2) as coordenadas do canto superior direito do Frase. A variável CD indica ***as coordenadas default*** (X, Y) onde a Definição Visual Frase será exibida. A variável COR indica a ***cor de frente*** na qual será exibido a Frase no dispositivo Vídeo. O conjunto de cores é dado em função do tipo de placa gráfica (CGA-EGA-VGA) que constitui a configuração do equipamento onde será implantado o Ambiente Poesis. A variável STRING corresponde à cadeia de caracteres que compõem a Definição Visual Frase a ser gerada.

OPERAÇÕES INTERNAS

A classe Frase pela hierarquia de classes definida, herda todas as operações definidas para a super-classe Dispositivo, sendo usadas para manipulação de suas instâncias. Esta classe porém, possui algumas operações internas próprias, criadas usando inclusive as operações herdadas de sua super-classe. Estas operações internas, ou métodos, serão descritas a seguir.

ATIVAÇÃO - Este método permite que a classe Frase execute o processamento da Definição Visual do tipo Frase informada via parâmetro.

OBTER-FRONTEIRAS, OBTER-CD, OBTER-COR, OBTER-STRING - Estes métodos fornecem o conteúdo das variáveis FRONT, CD, COR e STRING respectivamente, de uma determinada instância do processador genérico de Definição Visual Frase que foi ativada.

ALTERAR-FRONTEIRAS, ALTERAR-CD, ALTERAR-COR, ALTERAR-STRING - Estes métodos possibilitam a alteração do conteúdo das variáveis FRONT, CD, COR e STRING respectivamente, de uma determinada instância do processador genérico de Definição Visual Frase que foi ativada.

FINALIZAÇÃO - Este método permite a desativação da Definição Visual Frase.

CLASSE: SAÍDA

ESTRUTURA

A classe Saída possui uma estrutura composta por variáveis herdadas da super-classe: Dispositivo, dada pela tupla:

< ID_D , STAT > ,

além das variáveis da que constituem a estrutura própria desta classe dadas pela tupla:

< COORD, COR, TC, TAM, MASC, IAV , IDP > , onde:

COORD indica as ***coordenadas*** X e Y do vídeo onde será exibido o campo. Estas coordenadas estão limitadas à fronteira imposta pelo modalidade de processamento do vídeo sendo definidas em função de outro tipo de Definição Visual (ver objeto Janela) ao qual está associada. A variável COR indica a ***cor*** na qual será exibido o campo. O conjunto de cores é dado em função do tipo de placa gráfica (CGA-EGA-VGA) que constitui a configuração do equipamento onde será implantado o Ambiente Poesis. A variável TC indica o ***tipo de campo*** a ser processado podendo assumir os valores pertencentes ao conjunto { I, L, C, S, F, D, H }, onde:

I indica que o campo é do tipo Inteiro;
L indica que o campo é do tipo Inteiro Longo;
C indica que o campo é do tipo Caracter;
S indica que o campo é do tipo String;
F indica que o campo é do tipo Real;
D indica que o campo é do tipo Data (dd-mm-aaaa);
H indica que o campo é do tipo Hora (hh-mm-ss).

A variável TAM indica o ***tamanho do campo*** a ser ecoado ou não no dispositivo Vídeo. Para alguns tipos de campos o tamanho já é pré-definido, como no caso dos tipos C, D e H.
A variável MASC contém a máscara de edição a ser usada para exibir o campo.

O formato da máscara obedece a seguinte sintaxe:

(m) onde:

m é um string que representa a máscara de edição podendo ser usado os caracteres pertencentes ao conjunto { x , . , , , $, / , - , _ , *, +. B, : } para compor o formato de saída. Exemplos:

Inteiro:	1233	máscara: (+x.xxx)	saída: 1.233
Inteiro Longo:	-12254	máscara: (*.***.xxx-)	saída: ***12.254-
Caracter:	W	máscara: (x)	saída: W
String:	"isto"	máscara: (xxx)	saída: ist
Real:	12,232	máscara: (xx$xx)	saída: 12$23
Data:	29081990	máscara: (xx/xx/xxxx)	saída: 29/08/1990
Hora:	114000	máscara: (xxB:BxxB:Bxx)	saída: 11 : 40 : 00

A variável IAV é o ***Indicador de Atributo de Vídeo*** usado para determinar a forma pela qual o campo aparecerá visualmente para o usuário e assume os seguintes valores:

0 indica a modalidade Normal,
1 indica a modalidade de campo em Vídeo Reverso,
2 indica a modalidade de campo Piscante,
3 indica a modalidade de campo em Brilho Intensificado,
4 indica a modalidade de campo Sublinhado,

que poderão ser combinados para propiciar uma forma agradável de exibição do campo.

A variável IDP é o Identificador do Processo de Alocação que indica a região de memória usada para armazenar o valor do campo a ser exibido quando da ativação deste tipo de objeto durante o processamento do ambiente.

OPERAÇÕES INTERNAS

A classe Saída herda todas as operações definidas para a super-classe Dispositivo, sendo usadas para manipulação de suas instâncias. Esta classe porém, possui algumas operações internas próprias, criadas usando inclusive as operações herdadas de sua super-classe. Estas operações internas, ou métodos, serão descritas a seguir.

ATIVAÇÃO - Este método permite que a classe Saída execute o processamento da Definição Visual do tipo Saída informada via parâmetro.
OBTER-COORDENADAS , OBTER-COR , OBTER-TC , OBTER-TAM, OBTER-MASC, OBTER-IAV , OBTER-IDP - Estes métodos fornecem o conteúdo das variáveis COORD, TC, TAM, MASC, IAV e IDP respectivamente, de uma determinada instância do objeto Saída.
ALTERAR-COORDENADAS, ALTERAR-COR, ALTERAR-TC, ALTERAR-TAM, ALTERAR-MASC, ALTERAR-IAV , ALTERAR-IDP - Estes métodos permitem efetuar alteração do conteúdo das variáveis: COORD, TC, TAM, MASC, IAV e IDP respectivamente, de uma determinada instância do objeto Saída.
FINALIZAÇÃO - Este método permite a desativação do dispositivo Saída e utiliza a operação interna com o mesmo nome da super-classe Dispositivo.

CLASSE: MOUSE

ESTRUTURA

A classe Mouse possui uma estrutura composta por variáveis herdadas da super-classe: Dispositivo, dada pela tupla:

$< ID_D , STAT >$,

além das variáveis da própria estrutura dadas pela tupla:

< FRONT, CD, IDP, IDV >, onde:

FRONT indica a ***fronteira*** de atuação do Mouse composta pelas variáveis: X1 , Y1 e X2, Y2 que delimitam a região onde o Mouse tem efeito, sendo que o par (X1, Y1) determina as coordenadas do canto inferior esquerdo e o par (X2, Y2) as coordenadas do canto superior direito do Mouse.

A variável CD indica ***as coordenadas default*** (X, Y) onde a Definição Visual correspondente ao ícone do cursor do Mouse será exibida.

As variáveis IDP e IDV indicam respectivamente o ***Identificador do Processo de Alocação*** e o ***Identificador da Definição Visual*** referentes ao ícone que representa o cursor do Mouse.

OPERAÇÕES INTERNAS

A classe Mouse herda as operações definidas para a super-classe Dispositivo, sendo usadas para manipulação de suas instâncias. Esta classe possui algumas operações internas próprias, ou métodos, serão descritas a seguir.

INICIALIZAÇÃO - Este método tem por finalidade básica a inicialização do objeto Mouse verificando também o dispositivo vídeo acoplado ao sistema. Em caso de falha, é informado ao

Ambiente o tipo de erro ocorrido.

ATIVAÇÃO - Este método permite que a classe Mouse execute o processamento da Definição Visual do tipo Mouse informada via parâmetro.
OBTER-FRONTEIRAS, OBTER-CD, OBTER-IDP e OBTER-IDV - Estes métodos fornecem o conteúdo das variáveis FRONT, CD, IDP e IDV respectivamente, de uma determinada instância do processador genérico de Definição Visual que foi ativada.
ALTERAR-FRONTEIRAS, ALTERAR-CD, ALTERAR-IDP e ALTERAR-IDV - Estes métodos possibilitam a alteração do conteúdo das variáveis FRONT, CD, IDP e IDV respectivamente, de uma determinada instância do processador genérico de Definição Visual que foi ativada.
FINALIZAÇÃO - Este método permite a desativação do dispositivo Mouse e utiliza a operação interna com o mesmo nome da super-classe Dispositivo.

CLASSE: MAPA-TECLAS

ESTRUTURA

A classe Mapa-Teclas possui uma estrutura composta por variáveis definidas através da tupla:

< IDP, IDV, Nt , At >, onde:

IDP é o Identificador do Processo de Alocação da Definição Visual correspondente;
IDV é o Identificador da Definição Visual correspondente;
Nt indica o número de teclas ativas para a Definição Visual;
At é o Array de Teclas cujo tamanho é dado por Nt.

O parâmetro At compreende um conjunto de variáveis dada pela tupla:

< CT, TA , CA >, onde:

CT é o código da tecla;
TA é o Indicador do Tipo de Ação a ser executada assumindo os valores: 0 para transferir o controle ao GOLGO, e 1 para ativar uma nova Definição Visual;
CA é o Código da Ação a ser executada podendo assumir os seguintes valores:

- Indicador de Ação;
- Identificador de Definição Visual a ser ativada;

O Indicador de Ação corresponde a um parâmetro a ser utilizado pelo gerenciador na manipulação do Ambiente. O Identificador de Definição Visual permitirá que a Interface efetue a ativação de uma nova Definição Visual pré-definida.

OPERAÇÕES INTERNAS

Esta classe possui algumas operações internas próprias, ou métodos, que serão descritas a seguir.

INICIALIZAÇÃO - Este método tem por finalidade básica a inicialização do objeto Mapa-Teclas.
VERIFICA - Este método permite que a classe Mapa-Teclas verifique se o conjunto de teclas pressionadas pertence àquelas que foram definidas para uma determinada Definição Visual.
OBTER-IDP, OBTER-IDV, OBTER-CT, OBTER-TA, OBTER-CA - Estes métodos fornecem o conteúdo das variáveis IDP, IDV, CT, TA, CA respectivamente, de uma determinada instância do objeto Mapa-Teclas.
ALTERAR-IDP, ALTERAR-IDV, ALTERAR-CT, ALTERAR-TA, ALTERAR-CA - Estes métodos permitem alterar o conteúdo das variáveis IDP, IDV, CT, TA, CA respectivamente, de uma

determinada instância do objeto Mapa-Teclas.

CLASSE: CONTEXTO-SENSITIVO

ESTRUTURA

A classe Contexto-Sensitivo possui uma estrutura composta por variáveis dadas pela tupla:

< IDP, IDV, Nr , Ar >, onde:

IDP é o Identificador do Processo de Alocação da Definição Visual correspondente;
IDV é o Identificador da Definição Visual correspondente;
Nr indica o número de Regiões Sensitivas ativas para a Definição Visual;
Ar é o Array de Regiões Sensitivas cujo tamanho é dado por Nr.

O parâmetro Ar compreende um conjunto de variáveis dada pela tupla:

< FRONT, TA , CA >, onde:

FRONT indica a ***fronteira*** da Região Sensitiva composta pelas variáveis: X1 , Y1 e X2, Y2 que delimitam a região sensível ao botão de seleção do Mouse, sendo que o par (X1, Y1) determina as coordenadas do canto inferior esquerdo e o par (X2, Y2) as coordenadas do canto superior direito da Região.

TA é o Indicador do Tipo de Ação a ser executada assumindo os valores: 0 para transferir o controle ao GOLGO, e 1 para ativar uma nova Definição Visual;
CA é o Código da Ação a ser executada podendo assumir os seguintes valores:

- Indicador de Ação;
- Identificador de Definição Visual a ser ativada;

O Indicador de Ação corresponde a um parâmetro a ser utilizado pelo gerenciador na manipulação do Ambiente. O Identificador de Definição Visual permitirá que a Interface efetue a ativação de uma nova Definição Visual pré-definida.

OPERAÇÕES INTERNAS

Esta classe possui algumas operações internas próprias, ou métodos, que serão descritas a seguir.

INICIALIZAÇÃO - Este método tem por finalidade básica a inicialização do objeto Contexto-Sensitivo.
VERIFICA - Este método permite que a classe Contexto-Sensitivo verifique se o conjunto de teclas pressionadas pertence àquelas que foram definidas para uma determinada Definição Visual.
OBTER-IDP, OBTER-IDV, OBTER-FRONT, OBTER-TA, OBTER-CA - Estes métodos fornecem o conteúdo das variáveis IDP, IDV, FRONT, TA, CA respectivamente, de uma determinada instância do objeto Contexto-Sensitivo.

ALTERAR-IDP, ALTERAR-IDV, ALTERAR-FRONT, ALTERAR-TA, ALTERAR-CA - Estes métodos permitem alterar o conteúdo das variáveis IDP, IDV, FRONT, TA, CA respectivamente, de uma determinada instância do objeto Mapa-Teclas.

CLASSE: PILHA-MAPA-TECLAS

ESTRUTURA

A classe Pilha-Mapa-Teclas possui uma estrutura em forma de pilha sendo que cada elemento da pilha é composto por variáveis dadas pela tupla:

< IDP, IDV, Pmt >, onde:

IDP é o Identificador do Processo de Alocação da Definição Visual correspondente;
IDV é o Identificador da Definição Visual correspondente;
Pmt é o apontador para o início do segmento de memória onde se encontra armazenado o Mapa de Teclas.

Esta classe possui ainda a variável TOPO que tem por objetivo controlar o topo da Pilha-Mapa-Teclas.

OPERAÇÕES INTERNAS

Esta classe possui algumas operações internas próprias, ou métodos, que serão descritas a seguir.

INICIALIZAÇÃO - Este método tem por finalidade básica a inicialização do objeto Pilha-Mapa-Teclas.
INSERÇÃO - Este método tem por finalidade inserir um novo elemento na Pilha-Mapa-Teclas.
REMOÇÃO - Este método efetua a remoção do elemento correspondente ao topo da Pilha-Mapa-Teclas.
OBTER-IDP, OBTER-IDV e OBTER-PMT - Estes métodos fornecem o conteúdo das variáveis IDP, IDV, Pmt respectivamente, do elemento correspondente ao topo da Pilha-Mapa-Teclas.

CLASSE: PILHA-CONTEXTO-SENSITIVO

ESTRUTURA

A classe Pilha-Contexto-Sensitivo possui uma estrutura em forma de pilha sendo que cada elemento da pilha é composto por variáveis dadas pela tupla:

< IDP, IDV, Pmt >, onde:

IDP é o Identificador do Processo de Alocação da Definição Visual correspondente;
IDV é o Identificador da Definição Visual correspondente;
Pmt é o apontador para o início do segmento de memória onde se encontra armazenado o Contexto-Sensitivo.

Esta classe possui ainda a variável TOPO que tem por objetivo controlar o topo da Pilha-Contexto-Sensitivo.

OPERAÇÕES INTERNAS

Esta classe possui algumas operações internas próprias, ou métodos, que serão descritas a seguir.

INICIALIZAÇÃO - Este método tem por finalidade básica a inicialização do objeto Pilha-Contexto-Sensitivo.
INSERÇÃO - Este método tem por finalidade inserir um novo elemento na Pilha-Contexto-Sensitivo.
REMOÇÃO - Este método efetua a remoção do elemento correspondente ao topo da Pilha-Contexto-Sensitivo.
OBTER-IDP, OBTER-IDV e OBTER-PMT - Estes métodos fornecem o conteúdo das variáveis

IDP, IDV, Pmt respectivamente, do elemento correspondente ao topo da Pilha-Contexto-Sensitivo.

APÊNDICE III

CLASSES DE OBJETOS DO AMBIENTE

CLASSE: USUÁRIO

ESTRUTURA

A classe Usuário possui uma estrutura composta da seguinte forma:

- Estrutura própria da classe;
- Estrutura herdada;

A estrutura própria da classe Usuário compreende duas sub-estruturas usadas para manipulação das instâncias. Estas sub-estruturas são: Nodo-Raiz e o Nodo-Usuário. O Nodo-Raiz tem por objetivo controlar as informações armazenadas na instância sendo composto pelas seguintes variáveis descritas pela tupla:

< COD, ALFC, N, SA >, onde:

COD corresponde ao código de acesso à instância;
ALFC é o alfabeto criptográfico;
N é o número de usuários cadastrados;
SA é a senha de acesso à instância da classe Usuário.

A variável COD é a chave de acesso à instância do tipo alfanumérica de tamanho 6. Esta chave permite que a instância do Objeto Usuário seja manipulada durante o processamento do Ambiente. A variável ALFC é o alfabeto com 30 caracteres usados para criptografar as senhas de acesso da instância. O parâmetro N indica o número de usuários inseridos na instância do objeto Usuário, e permite o controle da árvore. SA é a senha que permite o acesso à instância.

O Nodo-Usuário é a sub-estrutura usada para manipular as informações de cada usuário que pode ser compreendida pela tupla:

< CODU, NOME, CODI, TU, TA, SENHA > , onde:

CODU é o Código do usuário;
NOME é a variável usada para identificar o usuário;
CODI é o Código Interno do Usuário;
TU indica o tipo de usuário;
TA indica o tipo de acesso;
SENHA corresponde a senha do usuário.

A variável CODU é usada para identificação do Usuário, sendo portanto a chave de acesso. A variável NOME corresponde ao nome do usuário que será utilizada para fins de documentação. A variável CODI é o código interno usado para construção do STATUS do usuário conforme descrito no capítulo 2. A variável TU é usada para indicar o Tipo de Usuário podendo assumir os seguintes valores: Leigo (001), Especialista (010), Intermitente (011) e Superusuário (000). A variável TA é utilizada para determinar a restrição de acesso do usuário e assume dois valores: Parcial (1) e Total (0). Estas duas variáveis são também usadas para compor o STATUS do usuário. A variável SENHA é usada pelo mecanismo de segurança do sistema para verificar a autorização para acesso ao Ambiente.

A estrutura herdada pela classe Usuário corresponde a estrutura da super-classe Árvore-B Genérica especificada na seção 3.2.2.2.4. As estruturas das instâncias da classe Usuário resultam da composição das estruturas acima especificadas.

HERANÇA

A implementação da classe Usuário como subclasse de Árvore-B Genérica determinou a herança das propriedades estruturais e comportamentais. Das propriedades comportamentais, definidas pelas operações internas, a classe usuário permite a utilização direta, como se estas fossem implementadas na classe, ou indiretamente, usando-as na implementação dos métodos próprios da classe. Portanto, de uma forma ou de outra, a classe Usuário utiliza todas as operações disponíveis para a super-classe Árvore-B Genérica, descritas na seção 3.2.2.2.4.

OPERAÇÕES INTERNAS

A classe Usuário utiliza para manipulação de suas instâncias, um conjunto de operações internas, ou métodos, que serão descritos a seguir.

CRIAÇÃO - Esta operação interna tem por objetivo criar uma instância do Objeto do Sistema Usuário, através da alocação de memória e posterior inicialização da estrutura Nodo-Raiz com informações para atualização das variáveis que compõem esta estrutura. Neste método são utilizadas as operações internas herdadas da super-classe Árvore-B Genérica.

INSERÇÃO - Esta operação interna é usada para inserção de um nodo correspondente a estrutura Nodo-Usuário, em uma instância de usuário que está sendo manipulada na memória real.

INSERÇÃO PARCIAL - Seu objetivo consiste basicamente em inserir um nodo correspondente a um novo usuário numa instância do Objeto Usuário manipulado parcialmente durante o processamento do Ambiente.

OBTER CÓDIGO, OBTER NOME, OBTER CODI, OBTER TIPO DE USUÁRIO, OBTER TIPO DE ACESSO e OBTER SENHA - Estes métodos visam fornecer o conteúdo das respectivas variáveis CODU, NOME, CODI, TU, TA e SENHA, de um determinado nodo correspondente a um usuário inserido na instância.

OBTER CÓDIGO-MESTRE, OBTER QUANTIDADE-USUÁRIOS, OBTER SENHA-MESTRE e OBTER ALFABETO - Estes métodos têm por objetivo fornecer respectivamente o conteúdo das variáveis COD, N, SA e ALFC, de uma determinada instância do objeto Usuário.

ALTERAR QUANTIDADE-USUÁRIOS, ALTERAR SENHA-MESTRE e ALTERAR ALFABETO - Estes métodos são utilizados para efetuar a alteração do conteúdo das respectivas variáveis N, SA e ALFC, de uma determinada instância do objeto Usuário.

ALTERAR NOME, ALTERAR TIPO DE USUÁRIO, ALTERAR TIPO DE ACESSO e ALTERAR SENHA - Estes métodos visam fornecer o conteúdo das respectivas variáveis NOME, TU, TA e SENHA, de um determinado nodo correspondente a um usuário inserido na instância.

CLASSE: ENTIDADE

ESTRUTURA

A classe Entidade possui uma estrutura interna implementada através do vetor ESCOPO de dimensão igual a 6. Este vetor determina a partir de cada elemento a participação da instância do objeto Entidade na composição dos objetos do Modelo E/D. Portanto, cada posição do vetor possui uma semântica associada que torna possível determinar o escopo dos outros objetos do modelo. Cada elemento poderá conter os seguintes valores:

0 indica que não existe a composição para este escopo;
N indica que existe a composição e pode ser obtida a partir do Link de Composição Normal;
I indica que a composição existe e pode ser obtida a partir do Link de Composição Invertida.

Cada elemento determina que tipo de objeto pode efetivamente participar da composição do objeto Entidade, cuja semântica é a seguinte:

Elemento 0 - Atributos que compõem a entidade;
Elemento 1 - Restrições associadas (Dependência Funcional e Verificação);
Elemento 2 - Ações associadas a entidade;
Elemento 3 - Regras de Comportamento da entidade;
Elemento 4 - Especializações da entidade;
Elemento 5 - Generalizações da entidade;

HERANÇA

A classe Entidade ao ser implementada como subclasse da classe Nó de Instância herda as operações internas criadas para esta super-classe e definidas na seção 3.2.2.2.3.

OPERAÇÕES INTERNAS

A classe Entidade possui algumas operações internas que são usadas para manipular suas instâncias. Esta operações internas, ou métodos serão descritos a seguir.

CRIAÇÃO - Esta operação interna tem por finalidade inicializar uma instância do objeto Entidade. Consiste basicamente em invocar o método Criação da classe Nó de Instância para alocar a memória suficiente para manipulação e atribuir à variável ESCOPO o valor 0 para cada elemento.
OBTER ESCOPO - Esta operação visa fornecer o conteúdo de um determinado elemento do vetor ESCOPO de uma instância do objeto Entidade.
ALTERAR ESCOPO - Esta operação interna tem por objetivo alterar o conteúdo de um determinado elemento do vetor ESCOPO de uma instância do objeto Entidade.

CLASSE: ATRIBUTO

ESTRUTURA

A classe Atributo possui uma estrutura interna que pode ser compreendida através da tupla:

< ESCOPO, TIPO, MV, SIN, MED >, onde:

ESCOPO indica o escopo associado ao atributo;
TIPO indica se o atributo é composto ou não;
MV indica se o atributo é multivalorado;
SIN é o Sinônimo do atributo;

MED é a máscara de edição do atributo;

A variável ESCOPO é um vetor de dimensão igual a 4 (semelhante ao especificado em 4.2.1).

Cada elemento determina que tipo de objeto pode efetivamente participar da composição do objeto Atributo, cuja semântica é a seguinte:

Elemento 0 - Atributos que compõem esta instância de Atributo;
Elemento 1 - Domínio associado ao Atributo;
Elemento 2 - Restrições associadas (Unicidade, Não-nulidade, e Verificação);
Elemento 3 - Indica onde o Atributo é Determinante.

A variável TIPO assume dois valores: "S" para indicar que o atributo é composto, e "N" para indicar que não é composto. A variável MV indica se o atributo é multivalorado (valor igual a "S") ou não (valor igual a "N"). A variável SIN é usada para armazenar um string de tamanho 30 que conterá o sinônimo do atributo. A variável MED irá conter a máscara de edição conforme especificado em [BAND89].

HERANÇA

A classe Atributo ao ser implementada como subclasse da classe Nó de Instância herda as operações internas criadas para esta super-classe e definidas na seção 3.2.2.2.3.

OPERAÇÕES INTERNAS

A classe Atributo possui algumas operações internas que são usadas para manipular suas instâncias. Esta operações internas, ou métodos serão descritos a seguir.

CRIAÇÃO - Esta operação interna tem por finalidade inicializar uma instância do objeto Atributo, através da alocação de memória obtida pela execução do método CRIAÇÃO da super-classe Nó de Instância.
OBTER ESCOPO, OBTER TIPO, OBTER MV, OBTER SIN e OBTER MED - Estes métodos têm por objetivo fornecer o conteúdo das respectivas variáveis: ESCOPO, TIPO, MV, SIN e MED de instância do objeto Atributo.
ALTERAR ESCOPO, ALTERAR TIPO, ALTERAR MV, ALTERAR SIN e ALTERAR MED - Estes métodos têm por finalidade alterar o conteúdo das respectivas variáveis: ESCOPO, TIPO, MV, SIN e MED de instância do objeto Atributo.

CLASSE: RELACIONAMENTO

ESTRUTURA

A classe Relacionamento possui uma estrutura interna que pode ser compreendida através da tupla:

< ESCOPO, TIPO >, onde:

ESCOPO indica o escopo associado ao Relacionamento;
TIPO indica o tipo de Relacionamento representado;

A variável ESCOPO é um vetor de dimensão igual a 5 (semelhante ao especificado em 4.2.1).

Cada elemento determina que tipo de objeto pode efetivamente participar da composição do objeto Relacionamento, cuja semântica é a seguinte:

Elemento 0 - Objetos participantes do relacionamento (Entidades e respectivos Atributos);

Elemento 1 - Atributos do Relacionamento;
Elemento 2 - Restrições associadas (Dependência Funcional, Verificação e Cardinalidade);
Elemento 3 - Indica as Ações associadas ao Relacionamento;
Elemento 4 - Regras de Comportamento associadas.

A variável TIPO assume dois valores: "T" para indicar que o Relacionamento é do tipo TOTAL, e "P" para indicar que não é do tipo PARCIAL.

HERANÇA

As operações herdadas pela classe Relacionamento compreendem as operações internas da classe Nó de Instância que foram herdadas, sendo definidas na seção 3.2.2.2.3.

OPERAÇÕES INTERNAS

A classe Relacionamento possui um conjunto de operações internas usado para manipular suas instâncias. Esta operações internas, ou métodos serão descritos a seguir.

CRIAÇÃO - Esta operação interna tem por finalidade inicializar uma instância do objeto Relacionamento, através da alocação de memória obtida pela execução do método CRIAÇÃO da super-classe Nó de Instância.
OBTER ESCOPO e OBTER TIPO - Estes métodos têm por objetivo fornecer o conteúdo das respectivas variáveis: ESCOPO e TIPO de uma instância do objeto Relacionamento.
ALTERAR ESCOPO e ALTERAR TIPO - Estes métodos têm por finalidade alterar o conteúdo das respectivas variáveis: ESCOPO e TIPO de instância do objeto Relacionamento.

CLASSE: DOMÍNIO

ESTRUTURA

A classe Domínio possui uma estrutura interna constituída pela variável TIPO usada para definir o tipo de Domínio conforme especificado em [BAND89].

HERANÇA

As operações herdadas pela classe Domínio correspondem às operações internas da classe Nó de Instância herdadas, definidas na seção 3.2.2.2.3.

OPERAÇÕES INTERNAS

A classe Domínio possui um conjunto de operações internas usado para manipular suas instâncias. Esta operações internas, ou métodos serão descritos a seguir.

CRIAÇÃO - Esta operação interna tem por finalidade inicializar uma instância do objeto Domínio, através da alocação de memória obtida pela execução do método CRIAÇÃO da super-classe Nó de Instância.
OBTER TIPO - Estes métodos têm por objetivo fornecer o conteúdo da variável TIPO de uma instância do objeto Domínio.
ALTERAR TIPO - Estes métodos têm por finalidade alterar o conteúdo da variável TIPO de instância do objeto Domínio.

CLASSE: DEPENDÊNCIA FUNCIONAL

ESTRUTURA

A classe Dependência Funcional possui uma estrutura interna constituída pelo vetor ESCOPO de dimensão igual a 3 (semelhante ao especificado em 4.2.1).

Cada elemento determina que tipo de objeto pode efetivamente participar da composição do objeto Dependência Funcional, cuja semântica é a seguinte:

Elemento 0 - Objetos onde a Dependência Funcional está definida (Entidade, Relacionamento);
Elemento 1 - Atributos determinantes;
Elemento 2 - Atributos dependentes;

HERANÇA

A classe Dependência Funcional ao ser implementada como subclasse da classe Nó de Instância herda as operações internas criadas para esta super-classe e definidas na seção 3.2.2.2.3.

OPERAÇÕES INTERNAS

A classe Dependência Funcional possui algumas operações internas que são usadas para manipular suas instâncias. Esta operações internas, ou métodos serão descritos a seguir.

CRIAÇÃO - Esta operação interna tem por finalidade inicializar uma instância do objeto Dependência Funcional.
OBTER ESCOPO - Esta operação visa fornecer o conteúdo de um determinado elemento do vetor ESCOPO de uma instância do objeto Dependência Funcional.
ALTERAR ESCOPO - Esta operação interna tem por objetivo alterar o conteúdo de um determinado elemento do vetor ESCOPO de uma instância do objeto Dependência Funcional.

CLASSE: UNICIDADE

ESTRUTURA

A classe Unicidade possui uma estrutura interna constituída pelo vetor ESCOPO de dimensão igual a 2 (semelhante ao especificado em 4.2.1).

Cada elemento determina que tipo de objeto pode efetivamente participar da composição do objeto Unicidade, cuja semântica é a seguinte:

Elemento 0 - Objetos onde a Unicidade está definida (Entidade, Relacionamento);
Elemento 1 - Atributos que possuem este tipo de restrição;

HERANÇA

A classe Unicidade herdou as operações internas criadas para a super-classe Nó de Instância, ao ser implementada como subclasse. Estas operações internas estão definidas na seção 3.2.2.2.3.

OPERAÇÕES INTERNAS

A classe Unicidade possui algumas operações internas que são usadas para manipular suas instâncias. Esta operações internas, ou métodos serão descritos a seguir.

CRIAÇÃO - Esta operação interna efetua a inicialização de uma instância do objeto Unicidade.

OBTER ESCOPO - Esta operação fornece o conteúdo de um determinado elemento do vetor ESCOPO de uma instância do objeto Unicidade.
ALTERAR ESCOPO - Esta operação interna tem por finalidade alterar o conteúdo de um determinado elemento do vetor ESCOPO de uma instância do objeto Unicidade.

CLASSE: NÃO-NULIDADE

ESTRUTURA

A classe Não-Nulidade possui em sua estrutura interna apenas uma variável do tipo array, denominada de vetor ESCOPO com dimensão igual a 2 (semelhante ao especificado em 4.2.1).

Um elemento do vetor ESCOPO determina o tipo de objeto que está efetivamente participando da composição do objeto Não-Nulidade, cuja semântica é a seguinte:

Elemento 0 - Objetos onde a Não-Nulidade está definida (Entidade, Relacionamento);
Elemento 1 - Atributos que possuem este tipo de restrição;

HERANÇA

A classe Não-Nulidade herdou as operações internas criadas para a super-classe Nó de Instância, ao ser implementada como subclasse. Estas operações internas estão definidas na seção 3.2.2.2.3.

OPERAÇÕES INTERNAS

A classe Não-Nulidade possui algumas operações internas que são usadas para manipular suas instâncias. Esta operações internas, ou métodos serão descritos a seguir.

CRIAÇÃO - Esta operação interna efetua a inicialização de uma instância do objeto Não-Nulidade.
OBTER ESCOPO - Esta operação fornece o conteúdo de um determinado elemento do vetor ESCOPO, informado via parâmetro, de uma instância do objeto Não-Nulidade.
ALTERAR ESCOPO - Esta operação interna tem por finalidade alterar o conteúdo de um determinado elemento do vetor ESCOPO, informado via parâmetro, de uma instância do objeto Não-Nulidade.

CLASSE: DEPENDÊNCIA DE CLASSE EXCLUSIVA

ESTRUTURA

A classe Dependência de Classe Exclusiva possui uma estrutura interna composta pela variável ESCOPO (semelhante ao especificado em 4.2.1).

A variável ESCOPO define do ponto de vista semântico e sintático, conforme o Modelo E/D, a associação com instâncias do objeto Entidade definido pelo usuário.

HERANÇA

A classe Dependência de Classe Exclusiva ao ser implementada como subclasse da classe Nó de Instância herda as operações internas criadas para esta super-classe e definidas na seção 3.2.2.2.3.

OPERAÇÕES INTERNAS

A classe Dependência de Classe Exclusiva possui algumas operações internas que são usadas para manipular suas instâncias. Esta operações internas, ou métodos serão descritos a seguir.

CRIAÇÃO - Esta operação interna tem por finalidade inicializar uma instância do objeto Dependência de Classe Exclusiva, através do método Criação, herdado da classe Nó de Instância, para alocar a memória suficiente a manipulação da variável ESCOPO deste objeto.
OBTER ESCOPO - Esta operação visa fornecer o conteúdo da variável ESCOPO de uma instância do objeto Dependência de Classe Exclusiva.
ALTERAR ESCOPO - Esta operação interna tem por objetivo alterar o conteúdo da variável ESCOPO de uma instância do objeto Dependência de Classe Exclusiva.

CLASSE: VERIFICAÇÃO

ESTRUTURA

A classe Verificação possui uma estrutura interna que pode ser compreendida através da tupla:

< ESCOPO, TCOM, COM >, onde:

ESCOPO	indica o escopo associado ao objeto Verificação;
TCOM	é o tamanho do comentário;
COM	é o comentário informado pelo usuário;

A variável ESCOPO é usada para determinar a participação da instância do objeto Verificação na composição dos objetos do Modelo E/D (semelhante ao especificado em 4.2.1).

A variável TCOM contém o tamanho do string COM usado para descrição do comentário a ser informado pelo usuário conforme especificado em [BAND89].

HERANÇA

A classe Verificação ao ser implementada como subclasse da classe Nó de Instância herda as operações internas criadas para esta super-classe e definidas na seção 3.2.2.2.3.

OPERAÇÕES INTERNAS

A classe Verificação possui algumas operações internas que são usadas para manipular suas instâncias. Esta operações internas, ou métodos serão descritos a seguir.

CRIAÇÃO - Esta operação interna tem por finalidade inicializar uma instância do objeto Verificação, através da alocação de memória obtida pela execução do método CRIAÇÃO da super-classe Nó de Instância.
OBTER ESCOPO, OBTER TCOM e OBTER COM - Estes métodos têm por objetivo fornecer o conteúdo das respectivas variáveis: ESCOPO, TCOM e COM de uma instância do objeto Verificação.
ALTERAR ESCOPO e ALTERAR COM - Estes métodos têm por finalidade alterar o conteúdo das respectivas variáveis: ESCOPO e COM de uma instância do objeto Verificação.

CLASSE: CARDINALIDADE

ESTRUTURA

A classe Cardinalidade possui uma estrutura interna constituída pela variável TIPO usada para definir o tipo de cardinalidade conforme especificado em [BAND89].

HERANÇA

As operações herdadas pela classe Cardinalidade correspondem às operações internas da classe Nó de Instância herdadas, definidas na seção 3.2.2.2.3.

OPERAÇÕES INTERNAS

A classe Cardinalidade possui um conjunto de operações internas, ou métodos, usado para manipular suas instâncias que será descrito a seguir.

CRIAÇÃO - Esta operação interna efetua a inicialização de uma instância do objeto Cardinalidade, através da alocação de memória obtida pela execução do método CRIAÇÃO da super-classe Nó de Instância.
OBTER TIPO - Este método têm por finalidade fornecer o conteúdo da variável TIPO de uma instância do objeto Cardinalidade.
ALTERAR TIPO - Este método permite a alteração do conteúdo da variável TIPO de instância do objeto Cardinalidade.

CLASSE: AÇÃO

ESTRUTURA

A classe Ação possui uma estrutura interna que pode ser compreendida através da tupla:

< ESCOPO, OPER, TVAL, VALOR >, onde:

ESCOPO	indica o escopo associado à Ação;
OPER	indica o tipo de operação realizada pela Ação;
TVAL	indica o tamanho da expressão VALOR;
VALOR	é a expressão que caracteriza a Ação.

A variável ESCOPO é um vetor de dimensão igual a 2 (semelhante ao especificado em 4.2.1). Este vetor determina a partir de cada elemento a participação da instância do objeto Ação na composição dos objetos do Modelo E/D. Cada posição do vetor possui uma semântica associada que torna possível determinar o escopo dos outros objetos do modelo, ou seja, cada elemento determina que tipo de objeto pode efetivamente participar da composição do objeto Ação, cuja semântica é a seguinte:

Elemento 0	- indica sobre que objetos ocorre a Ação (Entidade ou Relacionamento);
Elemento 1	- indica a composição de outra Ação.

A variável OPER é usada para identificar o tipo de operação padrão definida para o Modelo E/D, conforme especificado na sintaxe deste objeto em [BAND89]. TVAL tem por finalidade controlar o tamanho em bytes ocupado pela expressão usada para definir o objeto Ação, representada pela variável VALOR.

HERANÇA

A classe Ação ao ser implementada como subclasse da classe Nó de Instância tornou possível, pelo mecanismo de herança, a utilização das operações internas criadas para esta super-classe e definidas na seção 3.2.2.2.3.

OPERAÇÕES INTERNAS

A classe Ação possui algumas operações internas que são usadas para manipular suas instâncias. Esta operações internas, ou métodos serão descritos a seguir.

CRIAÇÃO - Esta operação interna tem por finalidade inicializar uma instância do objeto Ação.
OBTER ESCOPO, OBTER OPER, OBTER TVAL e OBTER VALOR - Estes métodos têm por objetivo fornecer o conteúdo das respectivas variáveis: ESCOPO, OPER, TVAL e VALOR de instância do objeto Ação.
ALTERAR ESCOPO, ALTERAR OPER e ALTERAR VALOR - Estes métodos têm por

finalidade alterar o conteúdo das respectivas variáveis: ESCOPO, OPER e VALOR de instância do objeto Ação.

CLASSE: CONDIÇÃO

ESTRUTURA

A classe Condição possui uma estrutura interna dada pela tupla:

< TEXP, EXP >, onde:

TEXP é o tamanho da expressão condicional;
EXP é a expressão que representa a condição;

A variável TEXP contém o tamanho ocupado pela expressão condicional armazenada pela variável EXP conforme sintaxe especificada em [BAND89].

HERANÇA

A classe Condição ao ser implementada como subclasse da classe Nó de Instância herda as operações internas criadas para esta super-classe e definidas na seção 3.2.2.2.3.

OPERAÇÕES INTERNAS

A classe Condição possui algumas operações internas que são usadas para manipular suas instâncias. Esta operações internas, ou métodos serão descritos a seguir.

CRIAÇÃO - Esta operação interna tem por finalidade inicializar uma instância do objeto Condição.
OBTER TEXP e OBTER EXP - Estes métodos têm por objetivo fornecer o conteúdo das respectivas variáveis: TEXP e EXP de uma instância do objeto Condição.
ALTERAR EXP - Estes métodos tem por finalidade alterar o conteúdo da variável EXP de uma instância do objeto Condição.

CLASSE: REGRA DE COMPORTAMENTO

ESTRUTURA

A classe Regra de Comportamento possui uma estrutura interna que pode ser compreendida através da tupla:

< ESCOPO, OPERAÇÃO >, onde:

ESCOPO indica o escopo associado à Regra de Comportamento;
OPERAÇÃO indica o tipo operação a ser executada pela Regra de Comportamento;

A variável ESCOPO é um vetor de dimensão igual a 3 (semelhante ao especificado em 4.2.1).

Cada elemento determina que tipo de objeto pode efetivamente participar da composição do objeto Regra de Comportamento, cuja semântica é a seguinte:

Elemento 0 - Objetos sobre os quais é aplicada a Regras de Comportamento (Entidades ou Relacionamento);
Elemento 1 - Indica a Condição associada à Regra de Comportamento;
Elemento 2 - indica a(s) Ação(ões) associada(s) à Regra de Comportamento;

A variável OPERAÇÃO contém o código correspondente à operação a ser definida pelo usuário conforme a sintaxe especificada em [BAND89].

HERANÇA

As operações herdadas pela classe Regra de Comportamento correspondem às operações internas da classe Nó de Instância, que foram herdadas, sendo definidas na seção 3.2.2.2.3.

OPERAÇÕES INTERNAS

A classe Regra de Comportamento possui as seguintes operações internas.

CRIAÇÃO - Esta operação efetua a inicialização de uma instância do objeto Regra de Comportamento. Este método utiliza a operação herdada CRIAÇÃO da super-classe Nó de Instância.
OBTER ESCOPO e OBTER OPERAÇÃO - Estes métodos têm por objetivo fornecer o conteúdo das respectivas variáveis: ESCOPO e OPERAÇÃO de uma instância do objeto Regra de Comportamento.
ALTERAR ESCOPO e ALTERAR OPERAÇÃO - Estes métodos têm por finalidade alterar o conteúdo das respectivas variáveis: ESCOPO e OPERAÇÃO de instância do objeto Regra de Comportamento.

CLASSE: TRANSAÇÃO

ESTRUTURA

A estrutura interna da classe Transação é constituída pela variável ESCOPO, um array de dimensão igual a 3 (semelhante ao especificado em 4.2.1).

Cada elemento do vetor ESCOPO determina o tipo de objeto que pode participar da composição do objeto Transação, cuja semântica é a seguinte:

Elemento 0 - indica as Entidades envolvidas;
Elemento 1 - indica os Relacionamentos envolvidos;
Elemento 2 - indica o processamento da transação através dos objetos Regra de Comportamento e Ação;

HERANÇA

A classe Transação poderá utilizar as operações internas de Nó de Instância, definidas na seção 3.2.2.2.3, pelo mecanismo de herança.

OPERAÇÕES INTERNAS

A classe Transação possui um conjunto de operações internas que são usado para manipular suas instâncias. Esta operações internas, ou métodos serão descritos a seguir.

CRIAÇÃO - Esta operação interna tem por finalidade inicializar uma instância do objeto Transação.
OBTER ESCOPO - Esta operação fornece o conteúdo de um determinado elemento do vetor ESCOPO de uma instância do objeto Transação.
ALTERAR ESCOPO - Esta operação interna tem por objetivo alterar o conteúdo de um determinado elemento do vetor ESCOPO de uma instância do objeto Transação.

CLASSE: META-REGRA

ESTRUTURA

A classe Meta-Regra possui uma estrutura interna que pode ser compreendida pela tupla:

< ESCOPO, COMB>, onde:

ESCOPO indica o escopo do objeto Meta-Regra;
COMB indica a combinação usada na Meta-Regra;

A variável ESCOPO é um array de dimensão igual a 3 (semelhante ao especificado em 4.2.1).

Cada elemento determina que tipo de objeto pode efetivamente participar da composição do objeto Meta-Regra, cuja semântica é a seguinte:

Elemento 0 - indica as Regras de Comportamento com Prioridade de execução;
Elemento 1 - indica as Regras de Comportamento mutuamente exclusivas;
Elemento 2 - indica as Regras de Comportamento dependentes.

HERANÇA

A classe Meta-Regra herdou as operações internas criadas para a super-classe Nó de Instância, definidas na seção 3.2.2.2.3.

OPERAÇÕES INTERNAS

A classe Meta-Regra possui algumas operações internas que são usadas para manipular suas instâncias. Esta operações internas, ou métodos serão descritos a seguir.

CRIAÇÃO - Esta operação interna efetua a inicialização de uma instância do objeto Meta-Regra.
OBTER ESCOPO e OBTER COMB - Estas operações fornecem respectivamente, o conteúdo de um determinado elemento do vetor ESCOPO, e o conteúdo da variável COMB de uma instância do objeto Meta-Regra.
ALTERAR ESCOPO e ALTERAR COMB- Estas operações internas têm por finalidade alterar o conteúdo das respectivas variáveis: ESCOPO e COMB de uma instância do objeto Meta-Regra.

CLASSE: HIPERDOCUMENTO

ESTRUTURA

A estrutura da classe Hiperdocumento é o resultado da composição de sua estrutura interna e a estrutura herdada da super-classe Árvore-B Genérica. A classe Hiperdocumento possui uma estrutura interna que é mostrada através da tupla:

< ID_{HD}, VERS, ERV_{NT} , DATA, STAT, N_{NT} >, onde:

ID_{HD} é o identificador do Hiperdocumento;
VERS indica a versão do Hiperdocumento;
ERV_{NT} é o Endereço Relativo Virtual de Nó_de_Texto inicial;
DATA é a Data de criação do Hiperdocumento;
STAT é o Status do Hiperdocumento;
N_{NT} é o número de Nó_de_Textos do Hiperdocumento.

A variável ID_{HD} é usada para identificar um Hiperdocumento através de uma cadeia de caracteres de tamanho 15. A variável VERS contém o número da versão do Hiperdocumento. Estas duas variáveis são chave de pesquisa na árvore-B que implementa esta classe. ERV_{NT} contém o Endereço Relativo Virtual que permitirá localizar o Nó_de_Texto inicial de um Hiperdocumento. DATA contém a data (dia, mês e ano) da criação do Hiperdocumento. STAT contém o STATUS do Hiperdocumento usado

no controle de acesso conforme descrito no capítulo 2. A variável N_{NT} contém o número de instâncias do objeto Nó_de_Texto associadas ao Hiperdocumento.

A estrutura herdada pela classe Hiperdocumento corresponde a estrutura da super-classe Árvore-B Genérica especificada na seção 3.2.2.2.4. As estruturas das instâncias da classe Hiperdocumento resultam da composição das estruturas acima especificadas.

HERANÇA

A implementação da classe Hiperdocumento como subclasse de Árvore-B Genérica determinou a herança das propriedades estruturais e comportamentais. As propriedades comportamentais, definidas pelas operações internas desta super-classe descritas na seção 3.2.2.2.4, são utilizadas pela classe Hiperdocumento de forma direta, como se estas fossem implementadas na classe, ou indiretamente, usando-as na implementação dos métodos próprios da classe. Logo, de uma forma ou de outra, a classe Hiperdocumento utiliza todas as operações disponíveis na super-classe Árvore-B Genérica.

OPERAÇÕES INTERNAS

A classe Hiperdocumento utiliza para a manipulação de suas instâncias, um conjunto de operações internas, ou métodos, que serão descritos a seguir.

CRIAÇÃO - Esta operação interna tem por objetivo criar uma instância do Objeto do Sistema Hiperdocumento, através da alocação de memória para atualização das variáveis que compõem sua estrutura. Neste método são utilizadas as operações internas herdadas da super-classe Árvore-B Genérica.
INSERÇÃO - Esta operação interna é usada para inserção de um nodo correspondente na instância de Hiperdocumento que está sendo manipulada na memória real.
INSERÇÃO PARCIAL - Seu objetivo consiste basicamente em inserir um nodo correspondente a um novo Hiperdocumento numa instância do Objeto Hiperdocumento manipulado parcialmente durante o processamento do Ambiente.
OBTER ID-HIPERDOC, OBTER VERSÃO, OBTER DATA, OBTER STATUS, OBTER ERV-NÓ-TEXTO e OBTER N-NÓ-TEXTO - Estes métodos visam fornecer o conteúdo das respectivas variáveis ID_{HD}, VERS, DATA, STAT, ERV_{NT} e N_{NT}, de um determinado nodo correspondente a um Hiperdocumento inserido na instância.
ALTERAR ID-HIPERDOC, ALTERAR VERSÃO, ALTERAR DATA, ALTERAR STATUS, ALTERAR ERV-NÓ-TEXTO e ALTERAR N-NÓ-TEXTO - Estes métodos permitem alterar o conteúdo das respectivas variáveis ID_{HD}, VERS, DATA, STAT, ERV_{NT} e N_{NT}, de um determinado nodo correspondente a um Hiperdocumento inserido na instância.

CLASSE: NÓ DE TEXTO

ESTRUTURA

A classe Nó_de_Texto possui uma estrutura interna que pode ser compreendida através da tupla:

$< T_{HT}, ID_{HD}, VERS, ERV_L, N_L, CREF, TEXTO >,$ onde:

T_{HT}	é o tamanho do Nó_de_Texto;
ID_{HD}	é o identificador do Hiperdocumento;
VERS	indica a Versão do Hiperdocumento;
ERV_L	é o Endereço Relativo Virtual do Objeto Link associado;
N_L	é o Número de Links (referências) existentes no Nó_de_Texto;
CREF	é o Contador de Referências;
TEXTO	corresponde ao texto contido no Nó_de_Texto.

A variável T_{NT} contém o tamanho (em bytes) do texto armazenado na variável TEXTO da instância. ID_{HD} e VERS são usados para recuperação das informações do Hiperdocumento associado. ERV_L contém o Endereço Relativo Virtual que permite localizar e recuperar a instância do objeto Link associada ao Nó_de_Texto. A variável N_L contém o número de Links (referências) existentes no texto. CREF é utilizada para registrar o número de vezes que um determinado Nó_de_Texto foi referenciado visando evitar a sua remoção "acidental". A variável TEXTO é uma cadeia de caracteres de tamanho T_{NT} usada para armazenar o texto correspondente.

HERANÇA

A classe Nó_de_Texto ao ser implementada como subclasse da classe Nó de Instância herda as operações internas criadas para esta super-classe e definidas na seção 3.2.2.2.3.

OPERAÇÕES INTERNAS

A classe Nó_de_Texto possui algumas operações internas que são usadas para manipular suas instâncias. Esta operações internas, ou métodos serão descritos a seguir.

CRIAÇÃO - Esta operação interna tem por finalidade inicializar uma instância do objeto Nó_de_Texto, através da alocação de memória obtida pela execução do método CRIAÇÃO da super-classe Nó de Instância.
OBTER TAMANHO, OBTER ID-HIPERDOC, OBTER VERS, OBTER ERV-LINK, OBTER NUM-LINK e OBTER CREF - Estes métodos têm por objetivo fornecer o conteúdo das respectivas variáveis: T_{NT}, ID_{HD}, VERS, ERV_L, N_L e CREF de instância do objeto Nó_de_Texto.
ALTERAR ID-HIPERDOC, ALTERAR VERS, ALTERAR ERV-LINK e ALTERAR NUM-LINK - Estes métodos têm por objetivo alterar o conteúdo das respectivas variáveis: ID_{HD}, VERS, ERV_L, N_L de uma instância do objeto Nó_de_Texto.
INCREMENTAR CREF - Este método efetua o incremento unitário da variável CREF.
DECREMENTAR CREF - Este método efetua o decremento unitário da variável CREF.

CLASSE: LINK

ESTRUTURA

A estrutura interna do objeto Link pode ser visualizada e compreendida a partir da tupla:

< TC, REF, SEQ, ERV_{NT} >, onde:
TC é o Tamanho da Chave de Acesso do Link;
REF é a palavra que identifica o Link;
SEQ é o seqüencial da referência;
ERV_{NT} é o Endereço Relativo Virtual do Nó_de_Texto associado.

A variável TC contém o tamanho correspondente à chave de acesso do Link, cujo valor é igual a tamanho da variável REF somado ao tamanho da variável SEQ. A variável REF corresponde a palavra identificada no texto que representará o Link. A variável SEQ indica o número de seqüência da palavra no Nó_de_Texto, caso exista mais de uma. Estas duas variáveis REF e SEQ compõem a chave de acesso da Árvore-B que permite recuperar as informações relativas a um determinado Link. A variável ERV_{NT} contém o Endereço Relativo Virtual que permite localizar instância do Nó_de_Texto referenciado.

A classe Link é subclasse de Árvore-B Genérica. ou seja, a composição da estrutura final que corresponde ao nodo da Árvore-B, deste objeto resultou da herança da estrutura desta super-classe descrita na seção 3.2.2.1.4.

HERANÇA

A Classe Link , conforme visto anteriormente, herdou as propriedades estruturais e comportamentais da super-classe Árvore-B Genérica. As propriedades comportamentais que correspondem a parte dinâmica do objeto, ou seja suas operações internas estão descritas na seção 3.2.2.1.2, e podem ser usadas pela classe Link para manipulação de suas instâncias. A utilização destas operações internas, ou métodos da classe, pode ser realizada de duas formas: através do uso direto pela invocação das operações da Classe Árvore-B Genérica, ou através do uso na implementação dos métodos da Classe Link.

OPERAÇÕES INTERNAS

A Classe Link é manipulada a partir do conjunto de operações internas, ou métodos, que serão descritos a seguir.

CRIAÇÃO - Este método efetua a criação de nodos numa instância do objeto Link parcialmente alocada na memória real. Esta criação corresponde a alocação e manipulação parcial das instâncias deste tipo de objeto, ou seja, não existe a necessidade de manipular toda a Árvore-B em memória real.
INSERÇÃO - Este método efetua a inserção de nodos numa instância de Link totalmente armazenada em memória real. Este método corresponde à operação interna Inserção herdada da classe Árvore-B Genérica.
OBTER TC, OBTER REF, OBTER SEQ, OBTER ERV-NÓ-TEXTO - Estes métodos têm por finalidade fornecer o conteúdo das respectivas variáveis: TC, REF, SEQ e ERV_{HT} de um nodo da instância do objeto Link armazenado em memória real.
ALTERAR ERV-NÓ-TEXTO - Este método permite alterar o conteúdo da variável ERV_{HT} de um nodo da instância do objeto Link armazenado em memória real.

CLASSE: APLICAÇÃO

ESTRUTURA

A classe Aplicação possui uma estrutura interna constituída pela seguinte tupla:
< TA, TEXTO>, onde:
TA indica o tamanho da instância do objeto Aplicação, sendo usada para armazenar o tamanho da instância armazenada em TEXTO; TEXTO é o conteúdo da aplicação.

HERANÇA

As operações herdadas pela classe Aplicação correspondem às operações internas da classe Nó de Instância herdadas, definidas na seção 3.2.2.2.3.

OPERAÇÕES INTERNAS

A classe Aplicação possui um conjunto de operações internas usado para manipular suas instâncias. Esta operações internas, ou métodos serão descritos a seguir.

CRIAÇÃO - Esta operação interna tem por finalidade inicializar uma instância do objeto Aplicação, através da alocação de memória obtida pela execução do método CRIAÇÃO da super-classe Nó de Instância.
OBTER TAMANHO - Estes métodos têm por objetivo fornecer o conteúdo da variável TA de uma instância do objeto Aplicação.
ALTERAR TAMANHO - Estes métodos têm por finalidade alterar o conteúdo da variável TA de instância do objeto Aplicação.

Subject: **aprovacao de dissertacao**
Date: Thu, 4 Dec 2003 09:59:17 -0200 (BRST)
From: Ana Carolina Salgado <acs@cin.ufpe.br>
To: Ivoneide da Silva Ribeiro <isr@cin.ufpe.br>

```
Apos analise, autorizo a publicacao da
dissertacao de mestrado de Marcos Aurelio Macedo.

Ana Carolina Salgado
```