UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE SOFTWARE PARA AUXILIAR NA APRENDIZAGEM DOS FUNDAMENTOS DAS ESTRUTURAS DE DADOS

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

MIKE BRUNNER

BLUMENAU, NOVEMBRO/2000

2000/2-41

PROTÓTIPO DE SOFTWARE PARA AUXILIAR NA APRENDIZAGEM DOS FUNDAMENTOS DAS ESTRUTURAS DE DADOS

MIKE BRUNNER

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Roberto Heinzle — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Roberto Heinzle

_____

Prof. Carlos E. Negrão Bizzotto

_____

Prof. Dalton Solano dos Reis

AGRADECIMENTOS

Aos meus pais, Gerd Brunner e Jeane Monique Brunner, por sempre terem dado condições para que estudasse e concluísse todos os níveis do ensino.

À minha namorada Fernanda Andrade, pelo apoio durante todo o tempo de realização deste trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

RESUMO

Este trabalho de conclusão de curso aborda a implementação de um protótipo de software para o auxílio no processo de aprendizagem das estruturas de dados básicas utilizadas na computação, enquadrando-se na tipologia de software educacional conhecida como Simulação e Modelagem.

ABSTRACT

This work present an implementation of a software prototype to assist the apprenticement process from the basic data structures used in computation. It will be classified as a Simulation and Modeling educational software.

1 INTRODUÇÃO

É cada vez mais freqüente a utilização de softwares educacionais nos processos de aprendizado. Os sistemas educacionais são usados principalmente na ajuda da compreensão de conceitos ([CAM1994]).

Baseado neste método de ensino, esta monografia propõe o estudo e implementação de um protótipo de software para o ensino das estruturas de dados básicas.

Segundo [AND2000], as modalidades mais empregadas de software educacional são as seguintes:

a) Exercício e Prática – é a forma mais tradicional empregadas nos computadores, onde o software pode ser desenvolvido rapidamente. Visa a aquisição de uma habilidade ou aplicação de um conteúdo já conhecido pelo aluno, inteiramente dominado;
b) Tutorial – os programas tutoriais podem introduzir conceitos novos, apresentar habilidades, pretender a aquisição de conceitos, princípios e/ou generalizações;
c) Jogos – os jogos devem ser fonte de recreação com vista a aquisição de um determinado tipo de aprendizagem;
d) Simulação e Modelagem – é a representação ou modelagem de um objeto real, de um sistema ou evento, por meio de um modelo simbólico ou representativo da realidade;
e) Tutores Inteligentes – o objetivo dos tutores inteligentes é trazer mais flexibilidade e interatividade no domínio da tutoria, sobretudo em matemática, programação e medicina. Estes sistemas podem ser definidos como uma integração da IA (Inteligência Artificial) e uma teoria da psicologia de aquisição de conhecimento dentro de um plano de ensino;

Dentre os tipos acima citados, o protótipo se baseará nos conceitos do tipo Simulação e Modelagem, viabilizando o entendimento do processo de construção das estruturas de dados básicas.

Em relação às estruturas de dados, serão abordas as estruturas do tipo lista: pilha, fila e lista duplamente encadeada; e as estruturas do tipo árvore: árvore binária e árvore ternária.

Esta proposta visa, portanto, o desenvolvimento de um protótipo de software para auxiliar no entendimento dos processos internos básicos de um computador, através de uma interface que fará a simulação destes processos. Neste caso será abordado o processo de implementação destas estruturas através da alocação dinâmica de memória na construção de filas, pilhas, listas duplamente encadeadas, árvores binárias e árvores ternárias. O protótipo será enquadrado nos moldes do tipo de software educacional Simulação e Modelagem. A implementação será feita no ambiente de programação Delphi e especificada com base no padrão para orientação a objetos Unified Modeling Language (U.M.L.).

1.1 MOTIVAÇÃO

Para qualquer estudante da área de computação é essencial o domínio das estruturas de dados básicas, que são o alicerce para estruturas mais complexas, utilizadas no desenvolvimento de softwares. Em função disso, é importante que esses fundamentos sejam bem assimilados pelos alunos, tornando-se útil e providencial o desenvolvimento de uma ferramenta que auxilie esse processo.

1.2 OBJETIVOS

Implementação de um protótipo de software para o auxílio ao ensino/aprendizado das estruturas de dados básicas utilizadas na computação, destinando-se a estudantes universitários dessa área.

1.3 ORGANIZAÇÃO DO TEXTO

Este trabalho de conclusão de curso está organizado de forma a permitir maior entendimento sobre o assunto proposto, da seguinte maneira:

Capítulo 2 – Software Educacional – apresenta uma breve classificação dos softwares educacionais e aborda com mais detalhes a técnica de Simulação;

Capítulo 3 – Estruturas de Dados – descreve os conceitos e as características de cada uma das estruturas de dados presentes no protótipo;

Capítulo 4 – Orientação a Objetos – traz uma explanação sobre a técnica de orientação a objetos, seus conceitos fundamentais, características e vantagens. Traz ainda uma

comparação com o enfoque tradicional orientado a funções e algumas características do padrão U.M.L. de modelagem;

Capítulo 5 – Desenvolvimento do Protótipo – descreve a especificação e implementação do protótipo, ou seja, a modelagem da definição e alguns códigos fonte;

Capítulo 6 – Conclusão – mostra o resultado do trabalho, suas limitações e possíveis melhoramentos.

2 SOFTWARE EDUCACIONAL

Seguem abaixo algumas colocações feitas por [ABR1998] a respeito da ligação entre o software educacional e o ensino.

Sabe-se que o “paradigma do ensino” sustentou a escola até hoje, cuja função era a de difundir o conhecimento elaborado, no qual o professor, dono do saber, deveria passá-lo aos alunos. Hoje, o desenvolvimento científico-tecnológico revela novas formas de aquisição de conhecimento, tanto no que diz respeito ao humano (como o homem aprende), quanto às questões da técnica (sobre a difusão do conhecimento).

Nesse contexto, o paradigma do ensino não cabe mais e é preciso trocá-lo pelo “paradigma da aprendizagem”. O mundo pós-moderno está exigindo, cada vez mais, competências cognitivas de nações inteiras. A escola que se faz necessária, nesse momento, deverá dar condições para que o indivíduo desenvolva sua capacidade de pensar, de resolver problemas, de tomar decisões, de adaptar-se ao novo e de desenvolver sua autonomia, senso crítico, criatividade e sensibilidade.

O software educacional, para ter um caráter verdadeiramente educacional compatível com o momento, deverá, portanto, superar o paradigma do ensino e caminhar para o paradigma da aprendizagem.

2.1 TIPOLOGIA

Os softwares educacionais são utilizados como recursos alternativos para o processo de ensino/aprendizagem nos vários aspectos de aquisição de habilidades. Estas habilidades podem ser adquiridas ou estimuladas dependendo do tipo de exercício ([CAM1994]).

A seguir, serão apresentados cada um dos tipos de software educacional, segundo alguns autores.

2.1.1 EXERCÍCIO E PRÁTICA

É a forma tradicional na qual os computadores têm sido utilizados em educação. É o tipo de software mais fácil de ser desenvolvido e utilizado. Visa a aquisição de uma habilidade ou a aplicação de um conteúdo já conhecido pelo aluno, mas não inteiramente

dominado. Pode suplementar o ensino em sala de aula, aumentar e/ou automatizar habilidades básicas. Em geral, utiliza *feedback* positivo e não julga as respostas erradas ([LAT1983]).

Os alunos trabalham com uma seleção aleatória de problemas, repetindo o exercício quantas vezes forem necessárias para atingirem os objetivos determinados pelo programa. As respostas erradas são rapidamente detectadas, o que reduz a possibilidade de reforço em procedimentos errôneos ([LAT1983]).

2.1.2 TUTORIAL

Os programas tutoriais podem introduzir conceitos novos, apresentar habilidades, pretender a aquisição de conceitos, princípios e/ou generalizações através da transmissão de determinado conteúdo ou da proposição de atividades que verifiquem a assimilação deste conteúdo. Servem como apoio ou reforço para aulas, para preparação ou revisão de atividades, entre outros aspectos ([CAM1994]).

2.1.3 SIMULAÇÃO E MODELAGEM

É a representação ou modelagem de um objeto real, de um sistema ou evento. É um modelo simbólico e representativo da realidade que deve ser utilizado a partir da caracterização dos aspectos essenciais do fenômeno. Isto significa que a simulação deve ser utilizada após a aprendizagem de conceitos e princípios básicos do tema em questão ([CAM1994]).

Inúmeros fenômenos podem ser representados neste modelo simbólico e representativo da realidade. As características de realismo e imaginação são básicas à simulação; dependem do fenômeno trabalhado e de seu objetivo. Os componentes principais e invariáveis de uma simulação são: o sistema de apresentação; o aluno; o sistema de controle; e o sistema de gerenciamento. Na apresentação do sistema o conteúdo pode ser trabalhado sob diversas formas, desde que estimule o aluno-usuário. O usuário é o elemento participante da simulação que controla o sistema, isto é, as variáveis e os parâmetros. O sistema gerenciador ajusta a simulação baseado na manipulação dos sistemas de controle pelos alunos/usuários ([MER1986]).

Segundo [OGB1992], a modelagem computacional é um sistema dinâmico e cada passo do cálculo pode ser definido de maneira elementar. Os modelos são construídos passo a passo, modificando-se os modelos anteriores. No caso das simulações, os modelos não são visíveis, não podendo ser analisados. A vantagem da modelagem reside em verificar como atuam as variáveis de um sistema.

2.1.4 JOGOS

[MER1986] afirma que os jogos devem ser fonte de recreação com vista a aquisição de um determinado tipo de aprendizagem. Geralmente envolvem elementos de desafio ou competição. Muitos jogos são confundidos com a simulação, pois utilizam algum tipo de habilidade. Ele acredita que os jogos deveriam ser mais adequadamente chamados de jogos de simulação.

[COB1982] identificou os diversos atributos básicos de jogos educacionais por computador. Estes atributos sugerem critérios de seleção para o uso. [MAR1986] classificou os atributos motivacionais em duas categorias: individual e impessoal. São consideradas motivações individuais o desafio, a curiosidade, o controle e a fantasia. São consideradas motivações interpessoais a cooperação, a competição e o reconhecimento.

2.1.5 TUTORES INTELIGENTES

O objetivo dos tutores inteligentes (ITS) é trazer maior flexibilidade e interatividade no domínio da tutoria, sobretudo em matemática, programação e medicina ([FIS1990]). [WER1993] define estes sistemas como de conhecimento dentro de um plano de ensino.

O objetivo do ITS deve ser muito exato pois, o sistema deve fornecer capacidade de raciocínio e resolução de problemas no domínio de aplicação. O ITS deve possuir um conhecimento do perfil do aluno a fim de ser sensível ao comportamento do estudante. Evidentemente, o ITS deve possuir uma interface com linguagem natural e a capacidade de diálogo deve ser desenvolvida ([FIS1990]).

2.2 SIMULAÇÃO

Segundo [BAR1973], a simulação é simplesmente a execução ou manipulação dinâmica de um modelo de um sistema-objeto com um objetivo qualquer. Sistema-objeto é o "objeto" ou tema de uma investigação ou experiência de aprendizado. Pode-se estudar o sistema-objeto diretamente, não precisando de um sistema simulado para aprender ou utilizar em experiências. O sistema-objeto é, às vezes, chamado de o mundo real.

Se a simulação adquire grande influência nas decisões que controlam o sistema-objeto, uma nova simulação pode ser executada para um sitema-objeto maior que inclua o sistema-objeto inicial e mais seu simulador. Isto não é raro no campo da computação, onde é comum que um computador simule um outro. Se o segundo computador estava simulando algum sistema-objeto, o primeiro estaria simulando o simulador. Esta prática poderia culminar numa progressão infinita de simulações. Enquanto questão prática, para que o trabalho de simulação prossiga, o usuário deve concentrar-se em um sistema-objeto particular que permaneça restrito a certos limites conceituais intencionais durante o período de estudo ([BAR1973]).

Na tentativa de explicação de um determinado fenômeno, o cientista muitas vezes recorre a modelos que são uma representação da realidade e que mostram os seus elementos constitutivos e relacionamento entre eles. Existem diversos tipos de modelos: gráficos, matemáticos, físicos, sociais, etc.. A simulação é um modelo que se diferencia de todos os outros por ser um modelo dinâmico ([RON1980]).

Um exemplo pode ilustrar este ponto: imaginando, de uma parte, um aluno do colegial que tente reproduzir com o lápis, num pedaço de papel, o modelo do sistema solar. De outra parte, observa-se o modelo encontrado num planetário. Este último mostra não apenas o número e tamanho dos planetas mas também suas velocidades, o padrão de relações que cria os eclipses, etc. Trata-se, sem dúvida, de um modelo simulado muito mais dinâmico, pois demostra não apenas o estado do sistema num dado momento mas também a forma como o sistema muda ([RON1980]).

2.2.1 INDICAÇÃO E VANTAGENS

[STR1984] traz algumas situações para as quais é indicado o uso da simulação:

a) não há uma formulação matemática completa para o problema;

b) não há método analítico para a resolução do modelo matemático;
c) a obtenção de resultados com o modelo é mais fácil de ser realizada por simulação que por método analítico;
d) não existe habilidade pessoal para a resolução do modelo matemático por técnica analítica ou numérica;
e) é necessário observar o desenvolvimento do processo desde o início até os resultados finais, e são necessários detalhes específicos;
f) não é possível ou é muito difícil e custosa a experimentação no sistema real;
g) é desejado estudar longos períodos de tempo ou são necessárias alternativas que os modelos físicos dificilmente fornecem.

Além disso, uma vantagem adicional é a capacidade de treinamento e instrução com custos e riscos reduzidos.

A simulação permite, também, a realização de estudos com bastante detalhe, ou com alto nível de agregação, dependendo do objetivo, já que pode ser maior ou menor o detalhamento dos procedimentos. Pode-se dizer que o grau de realismo incorporado no modelo de simulação reflete diretamente no nível de detalhe incorporado. O grau de realismo, por sua vez, está diretamente vinculado ao custo.

2.2.2 OBJETIVOS

A seguir, [BAR1973] determina os possíveis objetivos de uma simulação. Dos objetivos específicos do construtor e usuário da simulação depende qual será o sistema-objeto a ser simulado. Os objetivos podem ser diversos como a atividade humana. De qualquer forma, os objetivos podem ser classificados, de uma maneira geral, da seguinte forma:

a) simulações para ajudar nossa compreensão do funcionamento de sistemas-objetos;
b) simulações para ajudar os que tomam decisões que controlam alguns aspectos de sistemas-objetos;
c) simulações para treinar pessoas no conhecimento existente de sistemas-objetos.

[STR1984], entretanto, faz uma classificação diferente. Embora o uso da simulação seja múltiplo, os objetivos fundamentais da simulação podem ser categorizados em três grupos:

a) projeção absoluta - refere aos estudos de projeções e comparações absolutas de desempenho para situações previstas ou propostas. A característica fundamental é a necessidade de avaliar a função objetivo em termos absolutos, com grau de precisão menor que 10%. Neste caso, a simulação abrange todas as partes do sistema;

b) análise de sensibilidade - destina-se às comparações de similaridade entre áreas ou partes do modelo em estudo. As decisões são tomadas na base de mudanças de valores, enquanto outras variáveis são mantidas constantes, sendo úteis ao estudo de alternativas e realização de mudanças;

c) investigação de diagnóstico - visa a compreensão detalhada do comportamento do sistema simulado. Examina as inter-relações e investiga o processo das transações ou produtos através do sistema. A ênfase normalmente é dada às partes restritas do sistema, refletindo o interesse do estudo.

2.2.3 TÉCNICAS

[BAR1973] e [RON1980] trazem quatro técnicas para estudar os sistemas-objeto:

a) Análise – utiliza-se no corpo de dados surgidos durante simulações. Na análise de dados, tenta-se transformar o complexo de itens de dados em classes fundamentais ou significativas e descobrir relações entre estas classes. Faz-se aqui a distinção entre analisar um sistema-objeto e simulá-lo ([BAR1973]);

b) Simulação Homem-Modelo - sua característica distintiva é a de que algumas partes humanas do sistema-objeto são representadas por participantes reais enquanto que quaisquer outras entidades humanas e as características relevantes não-humanas do sistema-objeto são representados por um modelo. A propriedade essencial da simulação é a interação entre entidades reais e o modelo ([RON1980]);

c) Simulação Homem-Máquina - é especialmente recomendada quando o treinamento na situação real é prematuro, muito caro ou mesmo impossível. Ou então, quando na situação real, a vida do aprendiz estará em perigo. Este tipo de simulação pode também ser muito útil no treinamento de futuros professores. Quando a máquina é um computador eletrônico digital, referimo-nos à experiência como uma Simulação Homem-Computador ([RON1980]);

d) Simulação Computadorizada - os controles da experiência são fornecidos através do programa de instruções. O programa também fornece representações do comportamento que quer-se estudar, passo a passo. O experimentador pode refazer sua experiência, mudando os controles mas usando o mesmo comportamento simulado, ou mudando o comportamento simulado mas mantendo os controles, ou mudando ambos ([BAR1973]).

2.2.4 FASES

[RON1980] apresenta as três fases pelas quais uma simulação deve passar: preparação, aplicação e avaliação.

Dentro da preparação há uma sequência de passos para o projeto e a aplicação de jogos e simulação que é muito semelhante à sequência usada para aprendizagem por solução de problemas. Esta sequência descreve-se da seguinte forma:

a) fixação de objetivos - no caso de jogos e simulação, como nas demais técnicas de ensino, é fundamental importância que se tenha muito claro o fim a atingir, ou seja, o comportamento que se pretende mudar, já que técnicas de ensino podem contribuir para a modificações de comportamentos. Uma das grandes vantagens de uma determinação clara de objetivos a serem atingidos, é permitir ao professor saber se a técnica empregada está em função das necessidades dos participantes e também dar uma segurança maior no momento da avaliação a qual terá como parâmetros os objetivos que deveriam ser atingidos. Outra vantagem é a possibilidade que um objetivo cria de, definindo claramente a natureza do comportamento a ser modificado, permitir ao professor dimensionar as variáveis que poderão interferir no processo de aprendizagem, sejam eles de natureza pessoal situacional ou outras, trabalhá-las e controlá-las adequada e eficientemente;

b) determinação do contexto - uma vez que o objetivo esteja definido, há necessidade da identificação de um contexto particular no qual o exercício terá lugar. Em se tratando de uma simulação será importante determinar os papéis que cada um dos participantes deverá desempenhar. Como as simulações exigem que os participantes assumam determinada identidade e reajam de acordo com esta identidade é indispensável que os protagonistas tenham interiorizado os papéis a

representar. Ainda para a simulação ser eficiente algumas considerações sobre o campo de ação devem ser feitas;

c) identificação dos recursos - em primeiro lugar é indispensável que o professor se conscientize que ele é um recurso auxiliar muito importante para o bom desempenho dessas técnicas. Para tanto será necessário que ele se familiarize o suficiente com elas para dirigi-las bem. Para isso o professor poderá: participar de jogos e simulações dirigidos por outra pessoa para obter a perspectiva do participante e das técnicas que não podem ser obtidas com a simples leitura do material, dirigir a técnica com um grupo de amigos ou um pequeno grupo de estudantes, ir ao material e procurar ler as regras e procedimentos na perspectiva de um operador e ter certeza que o que os participantes estarão fazendo a cada momento está perfeitamente claro, compor um esquema de tempo para a execução da técnica. Em segundo lugar, deve-se ter o cuidado de preparar todos os recursos didáticos antes que os participantes cheguem, e;

d) determinação da sequência de interações - para quem está programando um jogo ou uma simulação é muito importante tentar determinar dinâmica de possíveis interações.

Com a simulação ou jogo planejados e preparados é possível considerar a operação da técnica em si. Para isso é importante que tanto a simulação como o jogo sejam sempre considerados como uma parte de uma sequência de aprendizagem e não como uma atividade isolada que se construirá num evento ao acaso.

Desta forma para a apresentação do jogo ou simulação aos participantes é necessário a preparação dos mesmos. Por exemplo, em muitos casos, dependendo do assunto, é necessário e conveniente transmitir aos alunos antes da simulação ou do jogo uma série de informações sobre o conteúdo que está sendo estudado. Um outro tipo de preparação também necessária consiste na introdução dos alunos a estas técnicas como modelos instrucionais. Algumas palavras sobre sua eficácia, suas vantagens e exigências ajudam muito o desenvolvimento dos que irão participar.

Principalmente a simulação exige que o professor ou instrutor empregue certas estratégicas preliminares tentando aumentar, gradativamente, o envolvimento dos

participantes. As estratégias podem ser: distribuição de papéis, discussão sobre os objetivos, apresentação dos participantes, etc..

A fim de ajudar os alunos a melhor entenderem os mecanismos iniciais do jogo ou da simulação, pode ser necessário expô-los a uma experiência em câmara lenta a respeito dos procedimentos a serem adotados. Segundo [TAY1972], estas ocasiões são os únicos momentos em que o professor pode ser visto assumindo o tradicional papel de expositor. Ele pode ensinar a sequência dos eventos e responder a questões sobre problemas essenciais, tempo para o exercício, problemas operacionais, etc..

Segundo [DUK1975], são três as etapas de discussão da avaliação:

- Dar oportunidade aos participantes para desabafarem tudo que perceberam durante a(s) atividade(s). Alguns participantes costumam ter um alto nível de envolvimento emocional. Alguns sentimentos são claramente manifestos a todos os participantes, outros não. Porém o importante não é apenas o relato mas tentar uma análise do por que as coisas se passaram desta ou daquela forma.
- Permitir o exame sistemático do modelo apresentado pelo jogo ou simulação a partir da perspectiva dos vários papéis. Tal etapa tem por objetivo principal dar oportunidade a todos de verem o que aconteceu segundo o olhar dos outros.
- Os participantes devem focalizar a realidade que foi representada ao invés do jogo ou simulação em si ([RON1980]).

2.2.5 PROBLEMAS

[STR1984] identifica alguns problemas a serem enfrentados com a simulação:

a) recursos humanos, materiais e de equipamentos;
b) mudanças, sob o aspecto da apreciação adequada das necessidades de modificações do modelo, tendo em vista alterações de objetivos antes ou durante a implementação;
c) definição dos limites do ambiente ou sistema a ser simulado;
d) custos, de acordo com o nível de detalhamento;
e) projeto e determinação das experiências a serem realizadas;
f) nível de detalhe, desde alta agregação e simplificação até grande detalhamento total ou parcial;

g) grau de precisão requerido para a obtenção dos resultados que satisfaçam os objetivos;

h) validação dos modelos e dos resultados.

3 ESTRUTURAS DE DADOS

[HOR1984] afirma que a ciência da computação é o estudo sobre os dados, suas representações e transformações por um computador digital. Estes dados podem ser:

- máquinas que manejam dados;
- linguagens que descrevem a manipulação dos dados;
- fundamentos que descrevem quais os dados refinados que podem ser produzidos dos dados crus;
- estruturas para a representação dos dados.

Sendo assim, será abordado neste capítulo o último tópico citado acima: as estruturas de dados.

De acordo com [VEL1986], uma estrutura de dados retrata as relações lógicas existentes entre os dados, de modo análogo ao uso de um modelo matemático para espelhar alguns aspectos de uma realidade física.

A definição de [HOR1984] diz que a estrutura de dados é o conjunto de domínios D, um domínio designado de D, um conjunto de funções F e o conjunto de axiomas A. O triplo (D,F,A) caracteriza a estrutura de dados.

A seguir, serão apresentadas as estruturas de dados presentes no protótipo, sendo divididas em estruturas do tipo Lista e estruturas do tipo Árvore.

3.1 LISTAS LINEARES

Uma lista linear é uma sequência de zero ou mais itens $x_1,x_2,....,x_n$, onde x_i é de um determinado tipo e n representa o tamanho da lista linear. Sua principal propriedade estrutural envolve as posições relativas dos itens em uma dimensão. Assumindo $n >= 1$, x_1 é o primeiro item da lista e x_n é o último item da lista. Em geral x_i precede x_{i+1} para $i = 1,2,...,n-1$, e x_i sucede x_{i-1} para $i = 2,3,...,n$. Em outras palavras, o elemento x_i é dito estar na i-ésima posição da lista ([ZIV1993]).

Uma das formas mais simples de interligar os elementos de um conjunto é através de uma lista. Lista é uma estrutura onde as operações inserir, retirar e localizar são definidas.

Listas são estruturas muito flexíveis porque podem crescer ou diminuir de tamanho durante a execução de um programa, de acordo com a demanda. Itens podem ser acessados, inseridos ou retirados de uma lista. Duas listas podem ser concatenadas para formar uma lista única, assim como uma lista pode ser partida em duas ou mais listas ([ZIV1993]).

Para determinadas aplicações é imposto um critério que restringe a inserção e retirada dos elementos que compõem um conjunto de dados. O critério escolhido impõem uma ordem ao conjunto, ordem esta que em geral não depende da ordem natural dos valores dos dados. Os dois critérios mais usuais são o LIFO (*Last In First Out*) e o FIFO (*First In First Out*) ([VEL1986]).

No primeiro, dentre os elementos que ainda permanecem no conjunto, o primeiro elemento a ser retirado é o último que tiver sido inserido. No segundo, dentre os elementos que ainda permanecem no conjunto, o primeiro a ser retirado é o primeiro que tiver sido inserido. Estruturas lineares com essas disciplinas de acesso são denominadas pilhas (critério LIFO) e filas (critério FIFO) ([VEL1986]). Pilhas e filas são dois dos mais comuns objetos de dados, encontrados nos algoritmos de computação ([HOR1984]).

[VEL1986] cita ainda um exemplo real: “Estas estruturas disciplinadas não ocorrem somente em computação. O critério FIFO é o que rege o funcionamento de uma fila de pessoas em um guichê: as pessoas são atendidas na ordem em que chegaram. Já o critério LIFO corresponde a uma pilha de livros, por exemplo: um novo livro é colocado no topo da pilha e só pode-se retirar o volume do topo; de outro modo a pilha desabaria. Para se ter acesso a um livro que está abaixo do topo, precisa-se, primeiro, retirar os que estão por cima dele”.

3.1.1 PILHAS

Segundo [ZIV1993], existem aplicações para listas lineares nas quais inserções, retiradas e acessos a itens ocorrem sempre em um dos extremos da lista. Uma pilha é uma lista linear em que todas as inserções, retiradas e geralmente todos os acessos são feitos em apenas um extremo da lista.

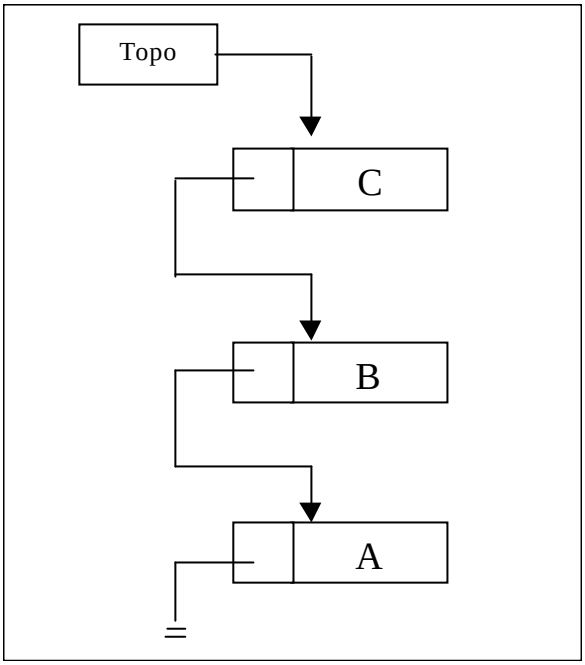
As pilhas possuem a seguinte propriedade: o último item inserido é o primeiro item que pode ser retirado da lista. Por esta razão as pilhas são chamadas de listas LIFO, termo

formado a partir de *Last-In, First-Out*. Existe uma ordem linear para pilhas, que é a ordem do "mais recente para o menos recente". Esta propriedade torna a pilha uma ferramenta ideal para processamento de estruturas aninhadas de profundidade imprevisível, situação em que é necessário garantir que subestruturas mais internas sejam processadas antes da estrutura que as contenham. A qualquer instante uma pilha contém uma sequência de obrigações adiadas, cuja ordem de remoção da pilha garante que as estruturas mais internas serão processadas antes de estruturas mais externas ([ZIV1993]).

No caso da pilha, a operação de inserir um novo átomo é chamada de empilhamento, enquanto sua remoção é denominada desempilhamento. Geralmente não têm-se outros operadores, pois processos de visitação ou busca não são comuns em situações em que pilhas são indicadas. Nada impede, naturalmente que eles sejam, desenvolvidos ([SWA1991]).

Quando lida-se as pilhas, torna-se útil pensar na pilha de papel que um burocrata tem sobre sua mesa de trabalho. Novos documentos que ele recebe para carimbar são armazenados em cima daqueles já existentes. Quando ele se sente movido a carimbar outro documento, ele vai buscá-lo no topo da pilha. Ele nunca retira um documento do meio ou fim da pilha. É dessa analogia que o ponteiro ao início da lista recebe o nome de topo ([SWA1991]). A representação de uma pilha pode ser vista na fig. 1.

Figura 1 - Representação de uma pilha



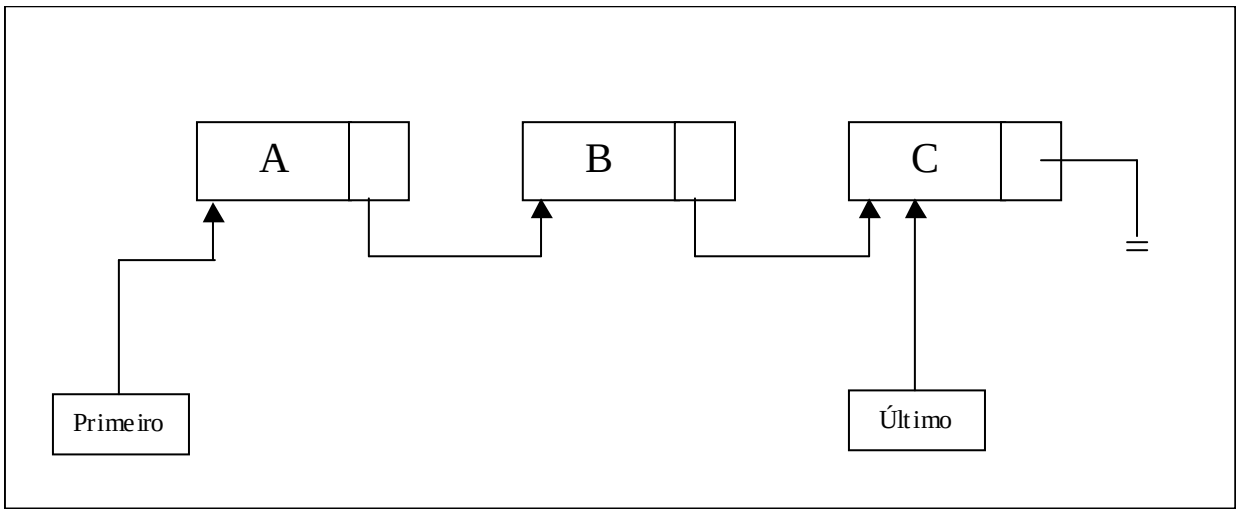
Estruturas aninhadas ocorrem freqüentemente na prática. Um exemplo simples é a situação em que é necessário caminhar em um conjunto de dados e guardar uma lista de coisas a fazer posteriormente. O controle de sequências de chamadas de subprogramas e sintaxe de expressões aritméticas são exemplos de estruturas aninhadas. As pilhas ocorrem também em conexão com algoritmos recursivos e estruturas de natureza recursiva, tais como as árvores ([ZIV1993]).

3.1.2 FILAS

Segundo [ZIV1993], uma fila é uma lista em que todas as interseções são realizadas em um extremo da lista, e todas as retiradas e geralmente os acessos são realizados no outro extremo da lista.

O modelo intuitivo de uma fila é o de uma fila de espera em que as pessoas no início da fila são servidas primeiro e as pessoas que chegam entram no fim da fila. Por esta razão as filas são chamadas de listas FIFO, termo formado a partir de *First-In, First-Out*. Existe uma ordem linear para filas que é a “ordem de chegada”. Sistemas operacionais utilizam filas para regular a ordem na qual tarefas devem receber processamento e recursos devem ser alocados a processos ([ZIV1993]). A representação de uma fila pode ser vista na fig. 2.

Figura 2 - Representação de uma fila



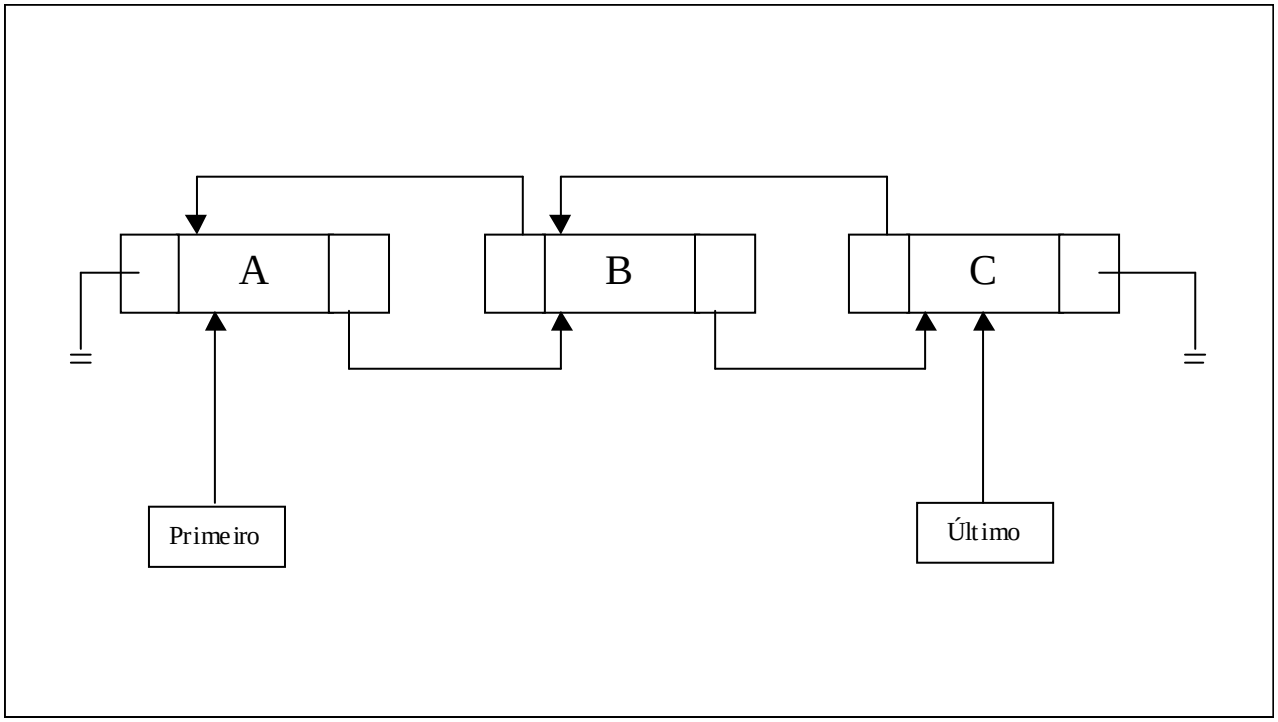
Essa estrutura de dados é idêntica à fila encontrada no cinema, no banco, no posto de gasolina, entre outros. É exatamente na modelagem desse tipo de fenômeno que está uma das grandes utilidades da fila. Uma fila tem dois ponteiros, um na frente e outro na traseira da estrutura ([SWA1993]).

3.1.3 LISTAS DUPLAMENTE ENCADEADAS

As pilhas e filas são listas lineares com encadeamento único, que para alguns problemas seria um tanto restritivo. Uma dificuldade com essas listas é que se um nódulo específico é apontado, pode-se apenas mover com facilidade na direção das ligações. A única maneira de encontrar um nódulo precedente é voltar ao início da lista ([HOR1984]).

O mesmo problema surge quando se quer suprimir um nódulo arbitrário de uma lista com encadeamento único. Nestes casos, deve-se conhecer o nódulo precedente, a fim de suprimir com facilidade um nódulo arbitrário. Se houver um problema onde haja freqüentemente a necessidade de mover em ambas as direções, então é útil ter listas encadeadas duplamente ([HOR1984]). A representação de uma pilha pode ser vista na fig. 3.

Figura 3 - Representação de uma lista duplamente encadeada



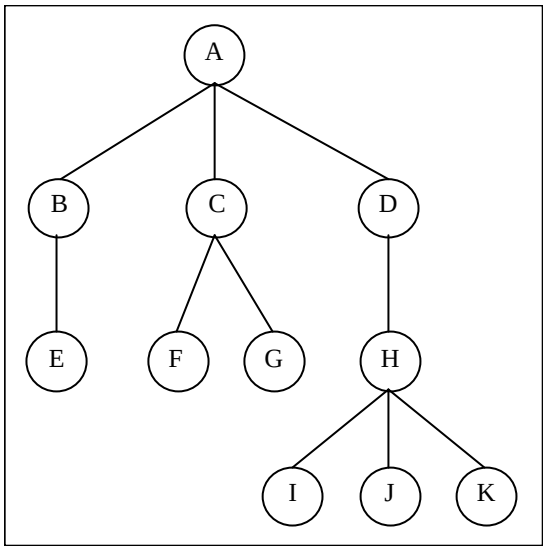
Sendo assim, cada nódulo terá dois campos de ligação, um apontando para o nódulo seguinte e outro para o anterior ([HOR1984]). O gasto de memória imposto por um novo campo de ponteiro pode ser justificado pela economia em não reprocessar a lista toda ([LUI2000]).

3.2 ÁRVORES

Da mesma forma que as listas lineares, árvores são estruturas de dados que caracterizam uma relação entre os dados que a compõem. No caso de árvores, a relação existente entre os dados (denominados nós) é uma relação de hierarquia ou de composição, onde um conjunto de dados é hierarquicamente subordinado a outro ([VEL1986]).

Na definição de [HOR1984], árvore é um conjunto finito de um ou mais nódulos de tal natureza que: (1) existe um nódulo especialmente designado denominado raiz; (2) os nódulos restantes estão desdobrados em $n >= 0$ conjuntos separados T_1, ..., T_n, em que cada um dos referidos conjuntos se constitui numa árvore. T_1, ..., T_n são denominadas as subárvores da raiz. Pela definição de árvore, cada nó da árvore é a raíz de uma subárvore. O número de subárvores de um nó é o grau daquele nó. Um nó de grau igual a zero é denominado folha ou nó terminal. Uma árvore pode ser representada como mostra a fig. 4.

Figura 4 - Representação de uma árvore.

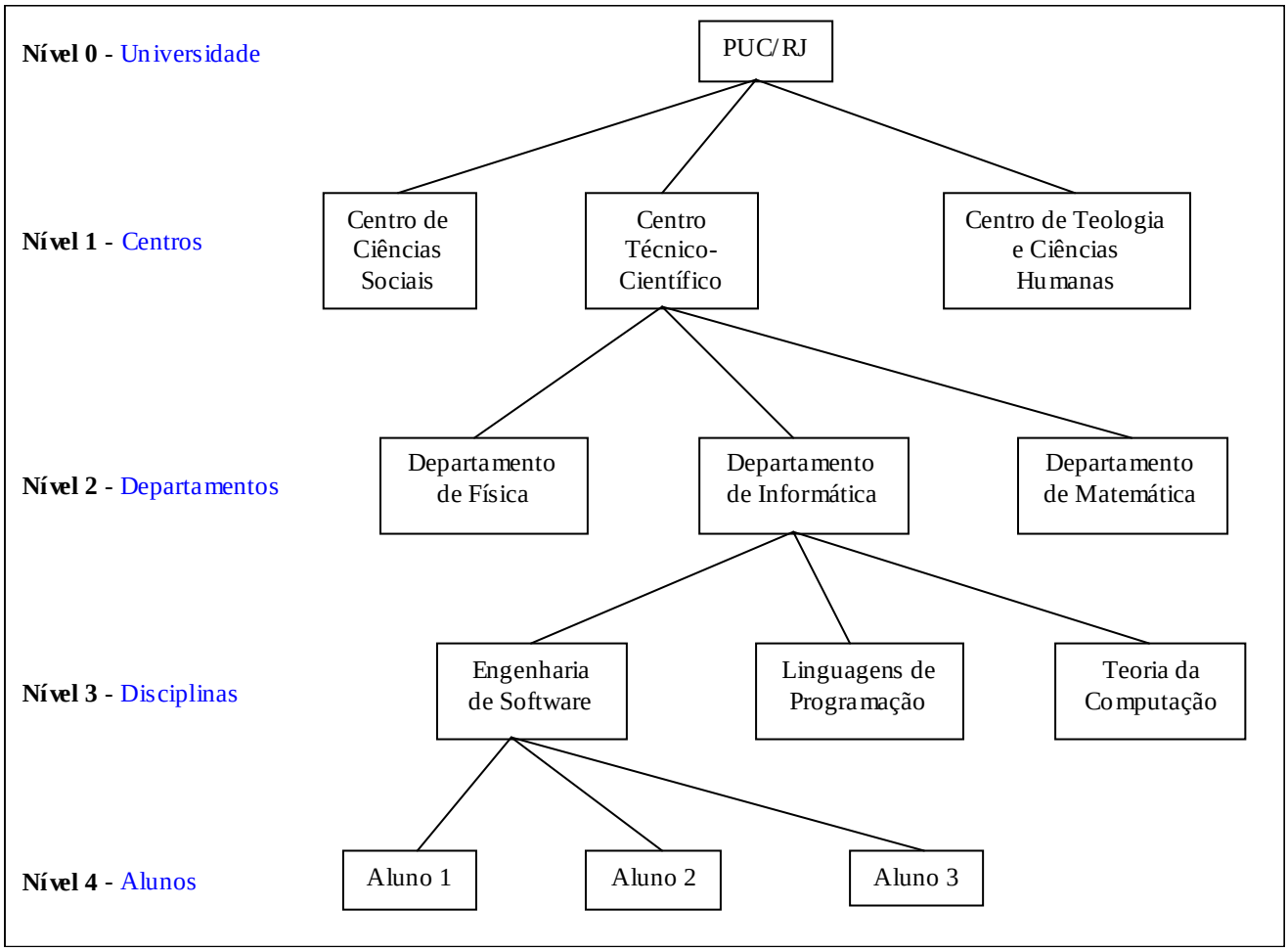


Os seguintes conceitos complementares podem ser encontrados em [HEI2000]: A raíz da árvore tem nível zero, enquanto o nível dos demais nós é igual ao número de “linhas” que

o liga à raíz. A altura de uma árvore é definida como sendo o nível mais alto da árvore. Denomina-se de floresta um conjunto de zero ou mais árvores disjuntas. Assim, se for eliminado o nó raíz de uma árvore, o que dela restar forma uma floresta. Quando a ordem das subárvores é significativa, diz-se que a árvore é ordenada. Quando a ordem não é significativa, diz-se que a árvore é orientada, uma vez que apenas a orientação é importante.

A estrutura de árvore é utilizada em casos onde os dados ou objetos a serem representados possuem relações hierárquicas entre si. Um exemplo é a estrutura de uma "universidade" composta de "centros". Cada "centro" composto por um certo número de "departamentos". Cada "departamento" oferece um conjunto de "disciplinas", nas quais estão matriculados os "alunos" ([VEL1986]). A exemplificação do esquema desta hierarquia pode ser vista na fig. 5.

Figura 5 – Exemplificação de hierarquia em árvores.



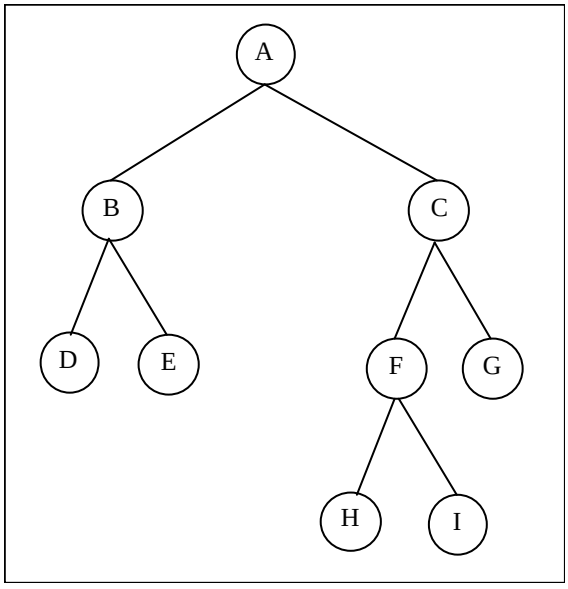
3.2.1 ÁRVORES BINÁRIAS

Segundo [VEL1986], árvores binárias são estruturas do tipo árvore, onde o grau de cada nó é menor ou igual a dois. No caso de árvores binárias, distinguem-se as subárvores de um nó entre subárvores da esquerda e subárvores da direita.

Na definição de [HOR1984], a árvore binária é um conjunto finito de nódulos, que se apresenta vazia ou que consiste de uma raiz e de duas árvores binárias separadas, denominadas a subárvore esquerda e a subárvore direita.

A árvore binária é um importante tipo de estrutura de árvore, que ocorre com muita freqüência. É caracterizada pelo fato de que qualquer dos nódulos pode possuir dois ramos, isto é, não existe nenhum nódulo com grau superior a dois. No caso das árvores binárias, diferencia-se entre a subárvore do lado esquerdo e a do lado direito, ao passo que nas árvores propriamente ditas, é indiferente a sequência das subárvores ([HOR1984]). A representação de uma árvore binária é mostrada na fig. 6.

Figura 6 - Representação de uma árvore binária

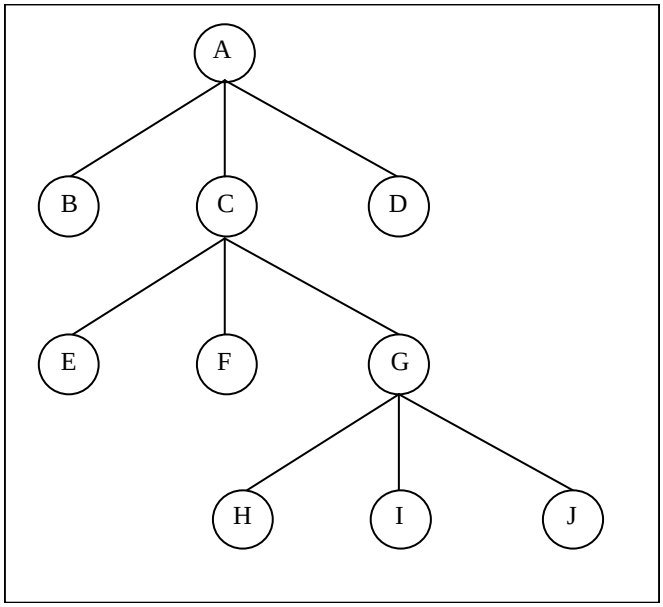


3.2.2 ÁRVORES TERNÁRIAS

Árvores cujos nós têm grau menor ou igual a três são chamadas de árvores ternárias, cujos conceitos são similares aos de árvore binárias. No caso de árvores ternárias, distinguem-

se as subárvores de um nó entre subárvores da esquerda, subárvores do meio e subárvores da direita [VEL1986]. A representação de uma árvore ternária é mostrada na fig. 7.

Figura 7 - Representação de árvore ternária



4 ORIENTAÇÃO A OBJETOS

A tecnologia baseada em objetos foi o maior avanço em software dos anos 90. Seu destino é o de mudar não só a maneira de se construir software, mas também a forma como eles se comunicam pelas redes híbridas no mundo inteiro. A modelagem baseada em objetos mudará a maneira de se projetar processos organizacionais e a maneira de se ver as empresas ([MAR1994]).

As técnicas orientadas a objetos mudam a visão que os analistas de sistemas de informação (I.S.) têm do mundo. Em vez de pensarem em processos e na sua decomposição, eles pensam em objetos e no comportamento deles. Os objetos podem ser internamente complexos, como uma máquina eletrônica, mas os analistas não precisam entender essa complexidade (a menos que a projetem). Saber como os objetos se comportam e como usá-los é o bastante ([MAR1995]).

4.1 CONCEITOS FUNDAMENTAIS

A seguir serão apresentados alguns conceitos básicos fundamentais para a compreensão da orientação a objetos, segundo diversos autores:

a) objeto – é qualquer coisa, real ou abstrata, a respeito da qual armazenam-se dados e as operações que os manipulam ([MAR1995]). Um objeto corresponde a uma concepção, abstração ou coisa que pode ser identificada distintamente. Durante a análise, objetos têm atributos e podem ser envolvidos em relacionamentos com outros objetos. Durante o projeto, a noção de objeto é estendida pela introdução de métodos e atributos de objetos. Na fase de implementação a noção de objeto é determinada pela linguagem de programação ([COL1996]);

b) atributo – é um puro valor de dado guardado pelos objetos de uma classe, portanto cada atributo possui um valor para cada instância de objeto. Cada nome de atributo é único dentro de uma classe (não considerando todas as classes de uma hierarquia de classes) ([RUM1994]). Atributos definem o estado interno de um objeto, suas características. Estes atributos podem ser modificados ao longo da vida do objeto. Aos atributos de um objeto apenas o próprio objeto deve ter acesso ([JUN2000]);

c) métodos – são códigos para implementação em uma classe ou operação interna, ou seja, o processo de desenvolvimento ([COL1996]). Os métodos são executados

toda vez que um objeto recebe uma mensagem, pois eles são a implementação da operação que foi solicitada pela mensagem. Estes métodos somente poderão manipular a estrutura de dados do próprio objeto ([RUM1994]);

d) classe – descreve um grupo de objetos com propriedades (atributos) semelhantes, o mesmo comportamento (operações), os mesmos relacionamentos com outros objetos e a mesma semântica ([RUM1994]). A classe consiste de um texto que descreve quais são os atributos e os métodos de todos os objetos pertencentes a esta classe. Chama-se o objeto de instância de uma determinada classe, o que significa que ele possui o comportamento e as características definidos pela classe para suas instâncias. Este processo é chamado de instanciação ([JUN2000]);

e) mensagem (ou solicitação) – é a forma de comunicação entre os objetos. Se um objeto necessitar de algum serviço ou informação de outro objeto, ele enviará uma mensagem a este último, uma vez que os dados dele só são acessíveis por ele mesmo ([LOR1993]). Para fazer com que um objeto faça alguma coisa, é necessário enviar a ele uma solicitação. Essa solicitação faz com que uma operação seja ativada. A operação executa o método adequado e, opcionalmente, devolve uma resposta ([MAR1994]). Quando um objeto envia uma mensagem para outro objeto, quem envia está solicitando que o recebedor da mensagem execute a operação chamada e (possivelmente) retorne alguma informação ([WIR1990]);

f) polimorfismo – é um conceito teórico onde uma operação com um determinado nome, quando executada por objetos diferentes, pode apresentar comportamento completamente diferente. Isso ocorre porque as linguagens orientadas a objetos permitem a criação de uma hierarquia de objetos com nomes de métodos comuns para operações conceitualmente similares, porém implementados diferentemente para cada classe na hierarquia ([WIN1993]);

g) herança – permite que o analista especifique operações e atributos comuns uma só vez, assim como especificar e estender estes atributos e operações em casos específicos ([COA1991]). Ao criar uma nova classe, o programador precisa incluir apenas as diferenças existentes em relação à classe da qual ela foi herdada, sem a necessidade de duplicar o código fonte ([WIN1993]);

h) encapsulamento – é o termo que descreve a junção de métodos e dados dentro de um objeto de forma que o acesso aos dados seja permitido somente por meio dos

próprios métodos do objeto ([WIN1993]). Um dos aspectos positivos do encapsulamento é a ocultação de informações. Pois, por exemplo, caso seja necessário alterar a estrutura de dados do objeto, o impacto da alteração será bem menor, uma vez que ela somente está sendo acessada pelo próprio objeto ([COA1991]);

i) abstração – é a capacidade de olhar apenas uma parte de um todo, ou seja, retirar da realidade apenas as entidades e fenômenos considerados essenciais, excluindo-se todos os aspectos irrelevantes ou secundários. Através do mecanismo de abstração, o ser humano pode entender formas complexas, ao dividi-las em partes e analisar cada parte separadamente ([JUN2000]). A abstração é uma descrição simplificada, ou especificação, de um sistema que enfatiza alguns dos detalhes do sistema ou propriedades enquanto suprime outros. Uma boa abstração é aquela que enfatiza detalhes que são significantes para o leitor ou usuário e suprime detalhes que são, ao menos para o momento, irrelevantes ([WIR1990]).

4.2 CARACTERÍSTICAS

[ERI1998], traz as características da orientação a objetos que Coad, Yordon, Pressman, entre outros, abordaram, discutiram e definiram em suas publicações:

a) A orientação a objetos é uma tecnologia para a produção de modelos que especifiquem o domínio do problema de um sistema;
b) Quando construídos corretamente, sistemas orientados a objetos são flexíveis a mudanças, possuem estruturas bem conhecidas e provêm a oportunidade de criar e implementar componentes totalmente reutilizáveis;
c) Modelos orientado a objetos são implementados convenientemente utilizando uma linguagem de programação orientada a objetos. A engenharia de software orientada a objetos é muito mais que utilizar mecanismos de sua linguagem de programação, é saber utilizar da melhor forma possível todas as técnicas orientada a objetos;
d) A orientação a objetos não é só teoria, mas uma tecnologia de eficiência e qualidade comprovadas usada em inúmeros projetos e para construção de diferentes tipos de sistemas.

4.3 VANTAGENS

[COL1996] relata as principais vantagens do desenvolvimento orientado a objetos:

a) abstração de dados – os detalhes referentes às representações das classes serão visíveis apenas a seus métodos; implementações diferentes de uma classe poderão ser utilizadas sem alterações no código que utilize a classe em questão. Abstração de dados não é uma característica exclusiva da tecnologia de objetos, mas surge naturalmente em seu contexto;
b) compatibilidade – as heurísticas para a construção das classes e suas interfaces levam a componentes de software que são mais fáceis de serem combinados;
c) flexibilidade – as classes, ou coleções de classes fortemente relacionadas, delimitam unidades naturais para a alocação de tarefas no desenvolvimento de software;
d) reutilização – o encapsulamento de métodos e representações de dados para a construção de classes facilita o desenvolvimento de software “reutilizável”;
e) extensibilidade – o software construído com o uso de técnicas orientadas a objetos tendem a ser mais facilmente estendido. Há duas razões para isso: a herança permite que novas classes sejam construídas a partir de outras já existentes, sem deixar de participar de todos os relacionamentos originais; além disso, as classes formam uma estrutura fracamente acoplada, o que facilita as alterações;
f) manutenção – a modularidade natural da estrutura de classe facilita o controle dos efeitos gerados pelas alterações e o uso da herança diminui a quantidade de conceitos díspares necessários ao entendimento do código.

4.4 ORIENTAÇÃO A FUNÇÕES X ORIENTAÇÃO A OBJETOS

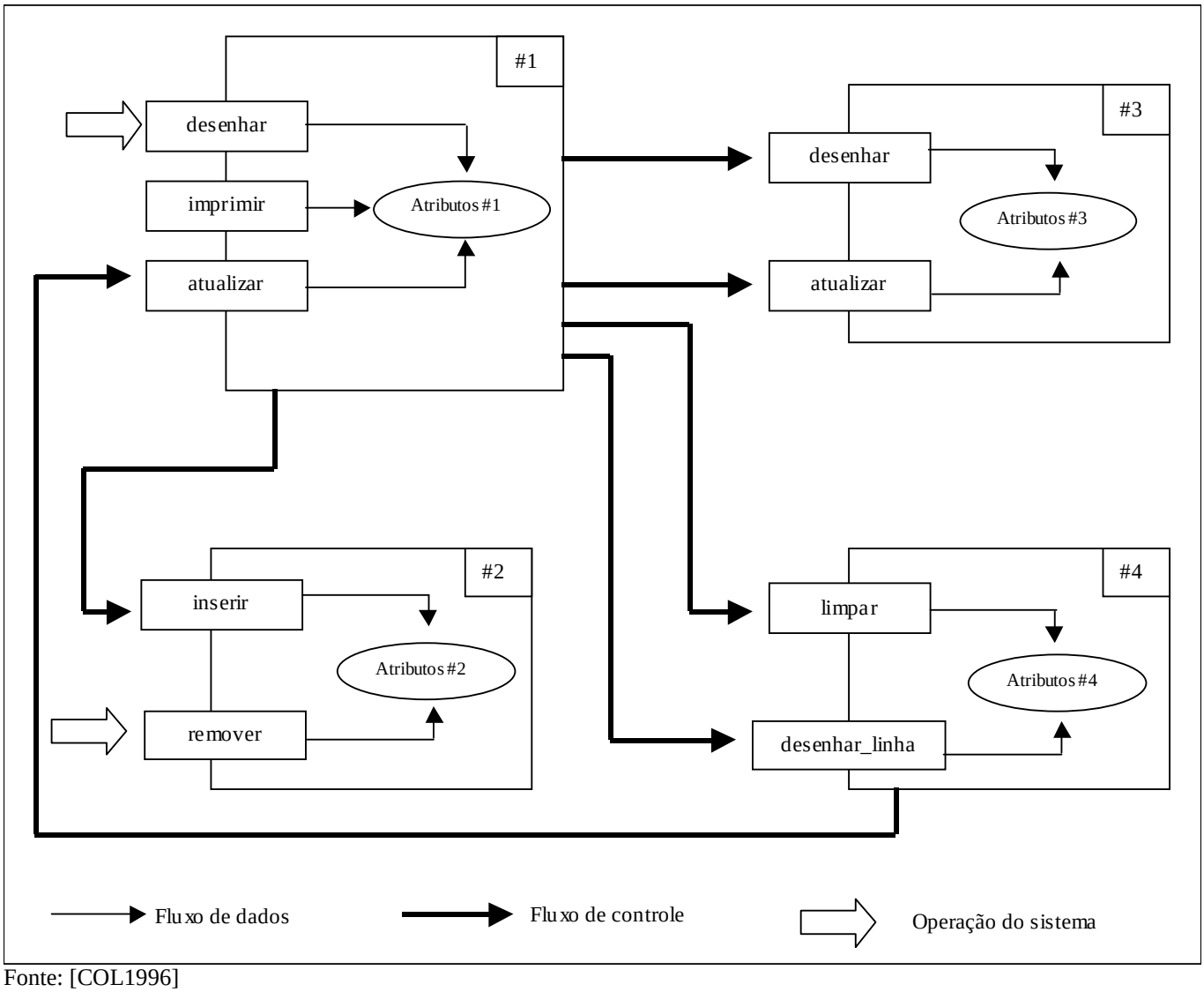
[COL1996] faz uma comparação entre o enfoque tradicional da orientação a funções e o enfoque orientado a objetos.

No enfoque orientado a objetos, os átomos do processo de computação são os objetos, que trocam mensagens entre si. Estas mensagens resultam na ativação de métodos, os quais realizam as ações necessárias. O emissor da mensagem não precisa saber como o objeto

receptor organiza o seu estado interno, mas apenas que este objeto responde a certas mensagens de maneira bem definida.

Este modelo está ilustrado em detalhes na fig. 8. Os retângulos maiores representam os objetos, com as setas entre eles representando as mensagens. Os estados dos objetos, representados pelas elipses no diagrama, são internos a eles. A única forma de se obter ou alterar o estado de um objeto é pelo envio de mensagens, e cada objeto "assume a responsabilidade" sobre como deve ser respondida uma determinada mensagem.

Figura 8 - Modelo computacional orientado a objetos



Fonte: [COL1996]

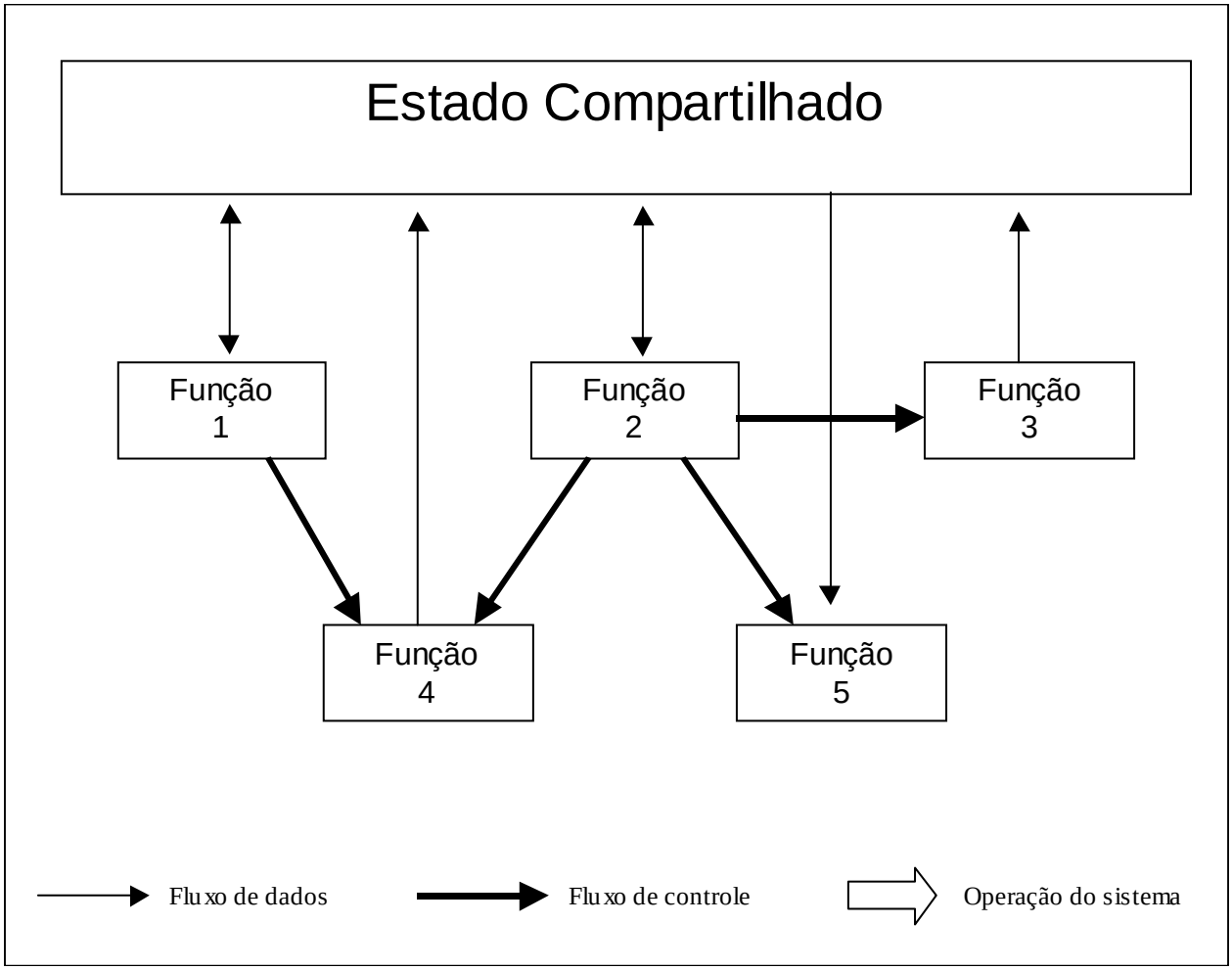
Os objetos, quando compartilharem uma única interface, são agrupados em classes, ou seja, respondem as mesmas mensagens da mesma maneira. Isso permite que vários objetos

sejam descritos por apenas algumas classes. As classes são os blocos de construção da maior parte das linguagens orientadas a objetos.

Durante a execução do sistema, os objetos podem ser construídos, executar ações, ser destruídos ou tornar-se inacessíveis. O modelo computacional é essencialmente dinâmico.

Na fig. 9 pode-se comparar este modelo com o enfoque mais tradicional orientado a funções. Nesse segundo caso, os átomos da computação são as funções, as quais atuam sobre um único estado compartilhado, possivelmente apresentando um alto grau de estruturação.

Figura 9 - Modelo computacional orientado a funções



Fonte: [COL1996]

Normalmente, o estado é estático, ou seja, a sua estrutura não se altera com o passar do tempo, apesar de, certamente, existirem alterações nos valores armazenados na estrutura. Qualquer função pode atuar sobre qualquer parte do estado. Mesmo que o sistema possa ser

dividido em módulos independentes, cada um deles tendo o controle sobre uma parte do estado, este enfoque não possui um modelo subjacente que diga como essa divisão seria realizada.

4.5 U.M.L – UNIFIED MODELING LANGUAGE

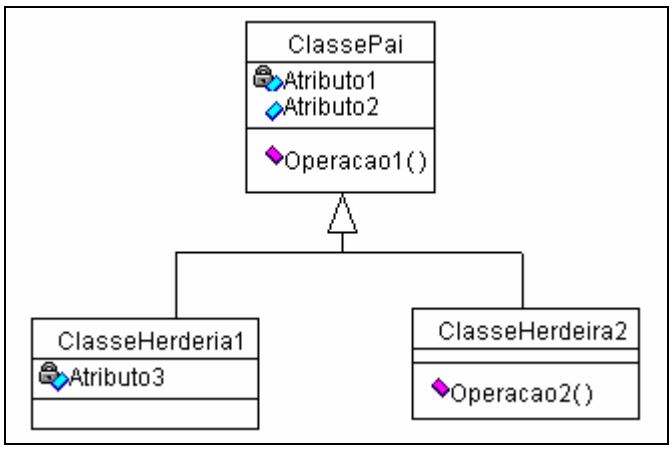
Na definição de [RAT2000], a U.M.L. é uma linguagem para especificação, visualização, construção e documentação de artefatos que compõem um sistema, bem como para modelagem de negócios e outras notações.

Segundo [LAR2000], esta é uma notação (principalmente diagramática) para modelagem de sistemas, usando conceitos orientados a objetos. Além disso, este padrão está emergindo como a notação padrão para modelagem orientada a objetos.

A seguir são apresentados, com base em [LAR2000], um exemplo de diagrama de classe e de diagrama de sequência da U.M.L., uma vez que estes serão utilizados no capítulo seguinte, na especificação do protótipo.

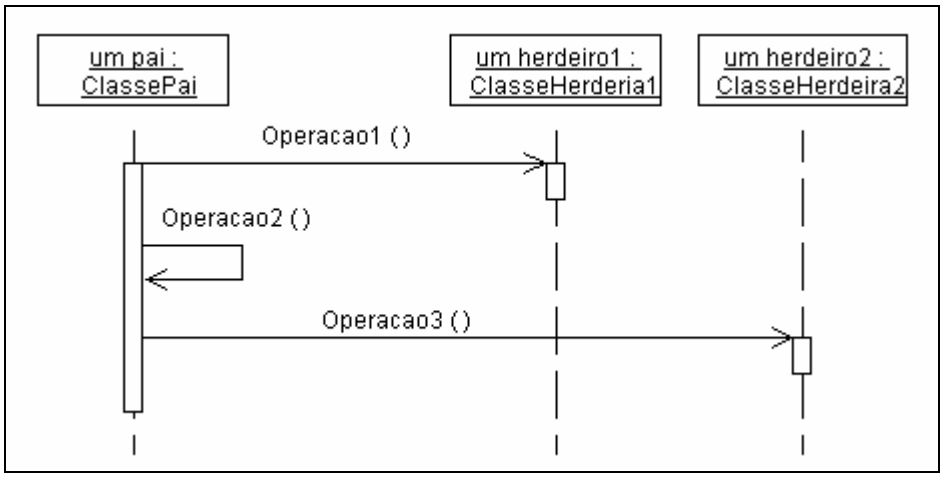
A fig.10 traz um exemplo de diagrama de classe da U.M.L. Este diagrama mostra os atributos e métodos referentes às classes, assim como o relacionamento entre estas.

Figura 10 - Diagrama de classe da U.M.L.



A fig.11 traz um exemplo de diagrama de sequência da U.M.L. Este diagrama mostra onde está o controle do programa a cada momento da chamada de uma função.

Figura 11 - Diagrama de sequência da U.M.L.



5 DESENVOLVIMENTO DO PROTÓTIPO

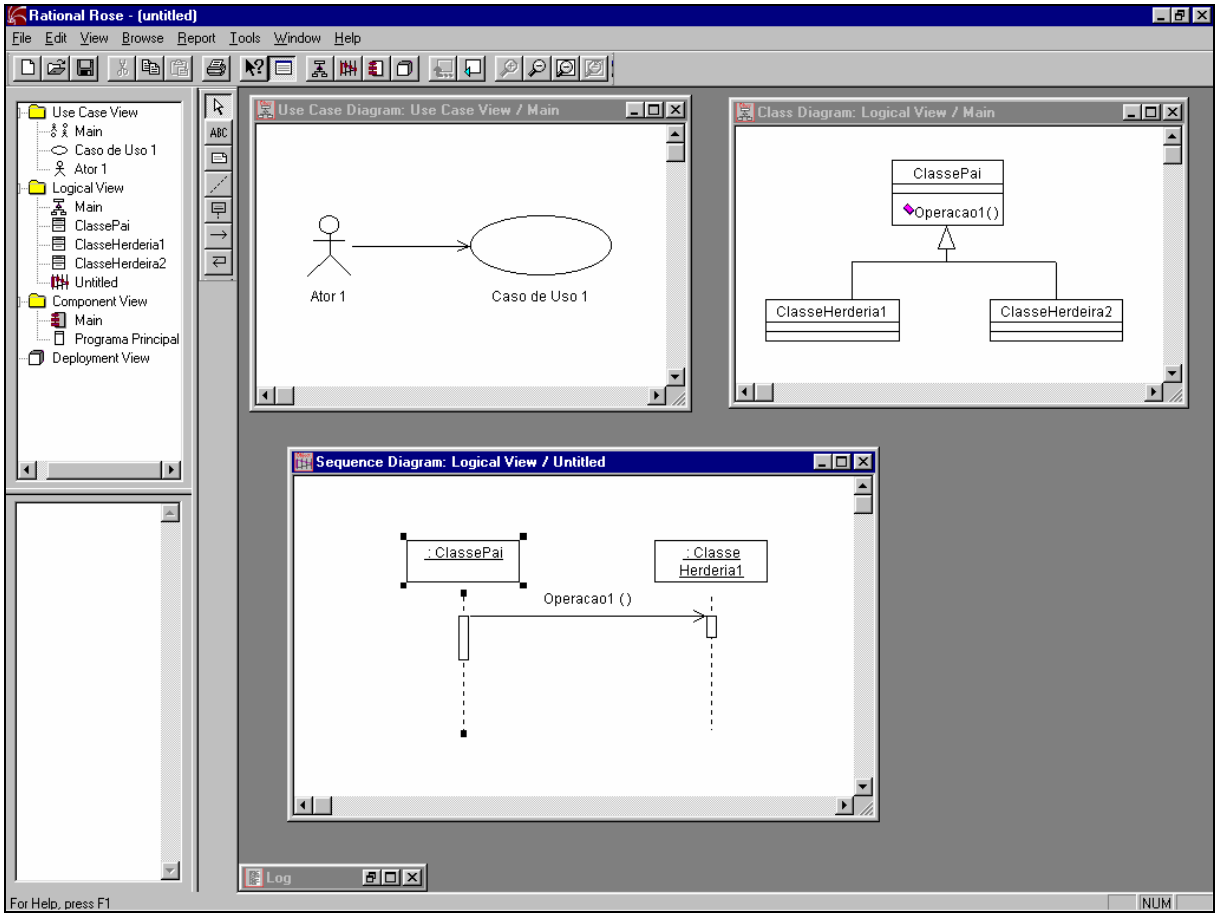
Com base nos conceitos apresentados nos capítulos anteriores, tornou-se possível o desenvolvimento de um protótipo que permite auxiliar o processo de aprendizagem de estruturas de dados. Neste capítulo, serão abordados a especificação e o funcionamento deste protótipo.

Para o desenvolvimento foram utilizadas as ferramentas Rational Rose versão 4.0.3, para fazer a modelagem e Delphi versão 5.0, para a implementação. A seguir, serão apresentadas algumas características de cada uma delas.

Através da ferramenta de modelagem Rational Rose é possível demonstrar o diagrama de classe e o diagrama de sequência para melhor compreensão da formação das classes e as sequências das chamadas de seus métodos.

Segundo [RAT2000], o Rational Rose é uma ferramenta de modelagem visual para projetar e criar aplicações de software utilizando o padrão U.M.L. Aplicações de software contemporâneas estão crescendo em complexidade, o que exige uma real disciplina de engenharia. Na fig.12 pode-se observar a tela principal da ferramenta.

Figura 12 - Tela principal do Rational Rose.



O ambiente de desenvolvimento Delphi permite a programação orientada a objetos através de algumas palavras reservadas. Com elas é possível realizar uma implementação totalmente orientada o objetos [(CAN1996]). A seguir estão algumas dessas palavras reservadas:

a) `class` - é usada para definir um novo tipo de dados, uma nova classe. Ela suporta métodos e campos `private`, `protected` e `public`, assim como herança, ou seja, indica um novo modelo de objeto, o qual pode-se descrever como um modelo de referência;

b) `private` - denota campos e métodos de uma classe que não são acessíveis fora do arquivo do código fonte (*unit*) declarando a classe. Em outras palavras, apenas a própria classe na qual estão declarados podem acessá-los;

c) `protected` - é usada para indicar métodos e campos parcialmente protegidos. Elementos protegidos podem ser acessados pela classe atual e por todas as suas classes descendentes, mas não pelos usuários desta classe;

d) `public` - denota métodos e campos que são livremente acessíveis a partir de outra parte do código de um programa, bem como na *unit* em que são definidos;

e) `virtual` – é usada para determinar que um método é virtual, ou seja, ele poderá ser redeclarado e reimplementado nas classes herdeiras, permitindo o polimorfismo entre as classes;

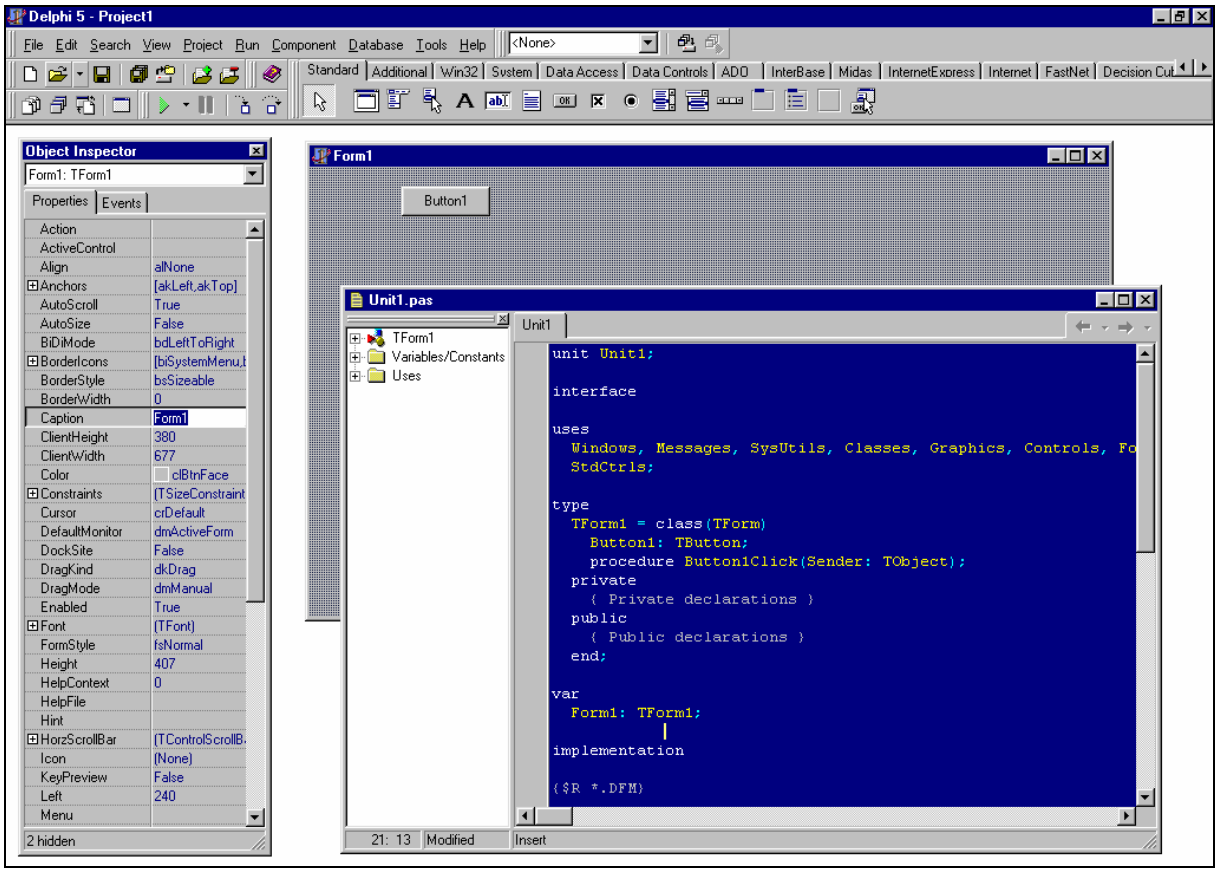
f) `abstract` - é usada para declarar métodos que estarão efetivamente implementados somente em subclasses da classe atual;

g) `override` – serve para anular um método virtual de uma classe ancestral, permitindo um novo tratamento (uma nova implementação) para o mesmo;

h) `inherited` - é usada para chamar um método da classe ancestral;

Na fig.13 é apresentada a tela principal do ambiente delphi.

Figura 13 - Tela principal do Delphi 5.

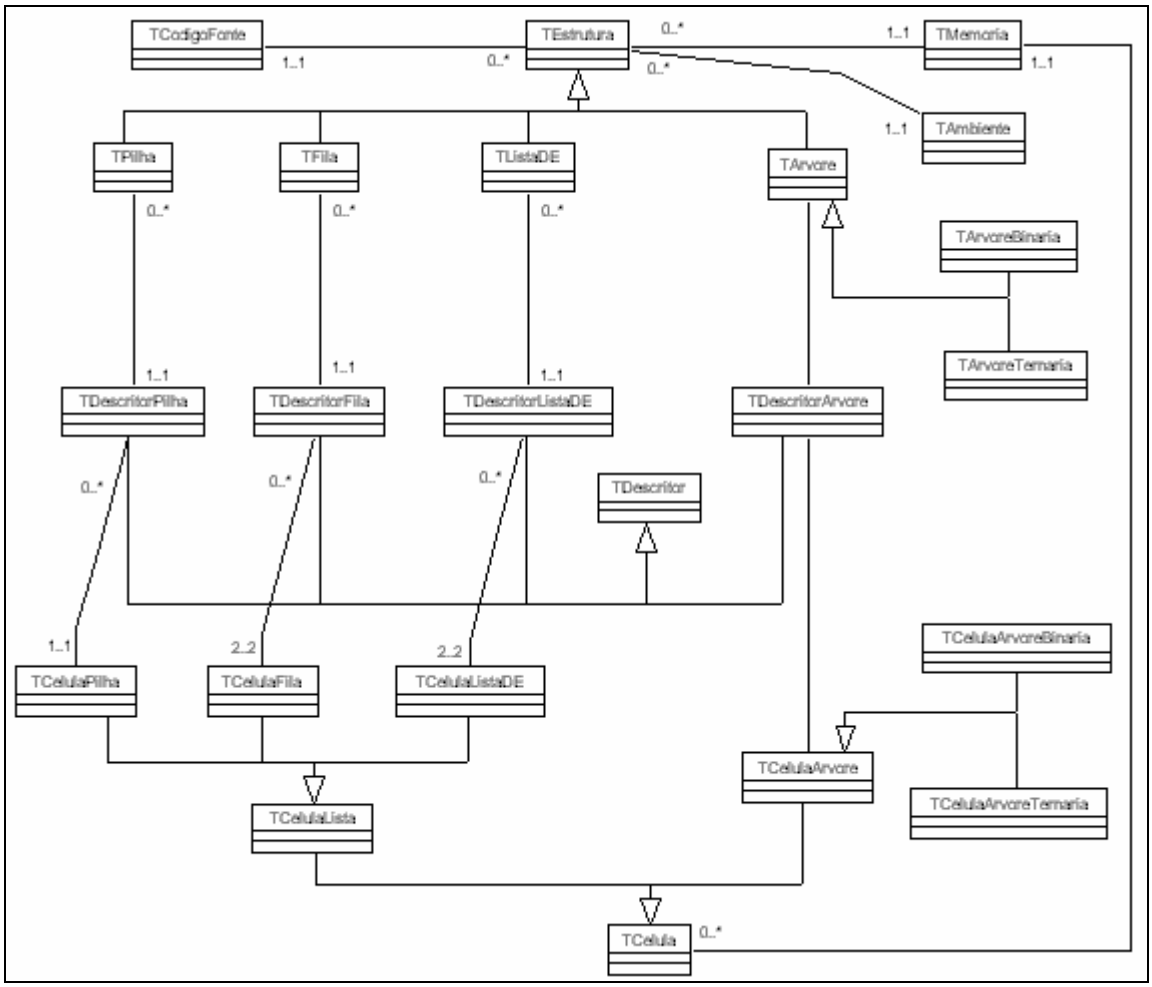


5.1 ESPECIFICAÇÃO DO PROTÓTIPO

Neste tópico serão apresentados os diagramas de classe, de sequência e hierárquico funcional. Estes diagramas possibilitam uma visão geral da especificação do protótipo, utilizando a representação gráfica.

Na fig.14 é mostrado o diagrama de classe simplificado de todo o protótipo, omitindo-se os atributos e métodos de cada classe.

Figura 14 - Diagrama de classe do protótipo.

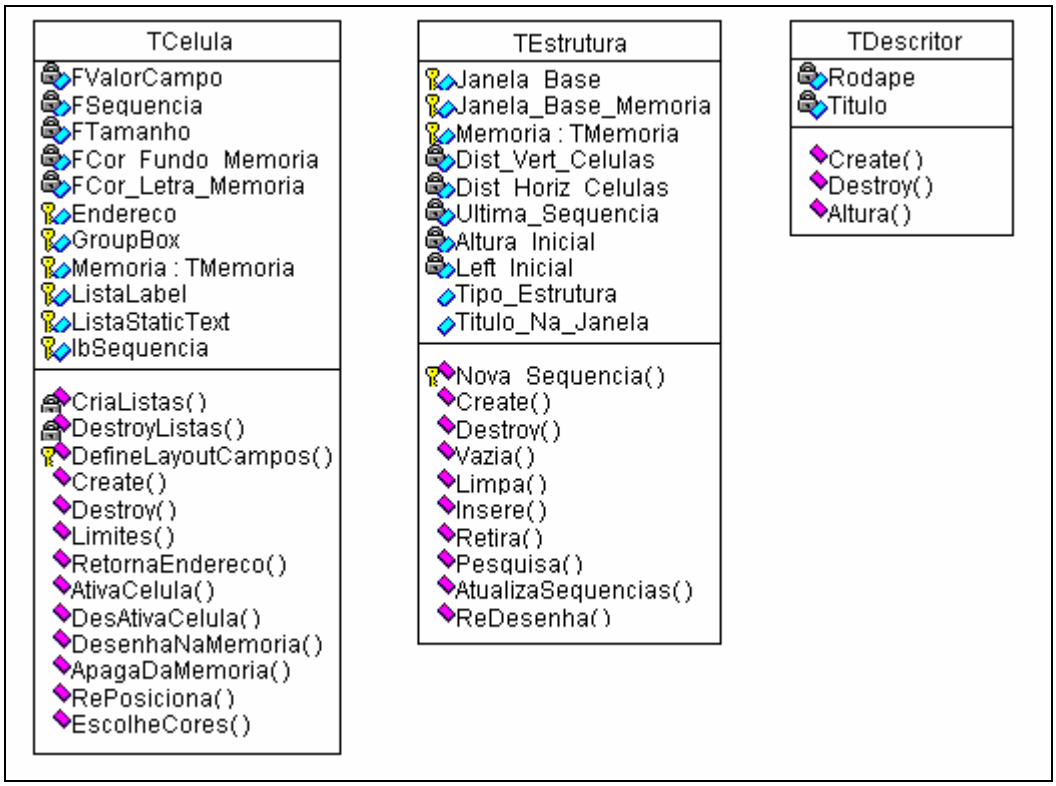


Pode-se observar neste diagrama que existem três classes principais (`TEstrutura`, `TCelula` e `TDescritor`) que são ancestrais de todas as demais classes, exceto de `TCodigoFonte` e `TMemoria`, que representam, respectivamente, o código fonte da estrutura e a área de memória na qual as estruturas e células serão exibidas.

As classes descendentes de `TEstrutura` representam cada uma das estruturas de dados abordadas no capítulo 3. As classes descendentes de `TCelula` representam as unidades fundamentais de cada uma dessas estruturas. Um objeto da classe `TDescritorPilha`, por exemplo, fará referência a um objeto da classe `TCelulaPilha`, que é o topo da pilha.

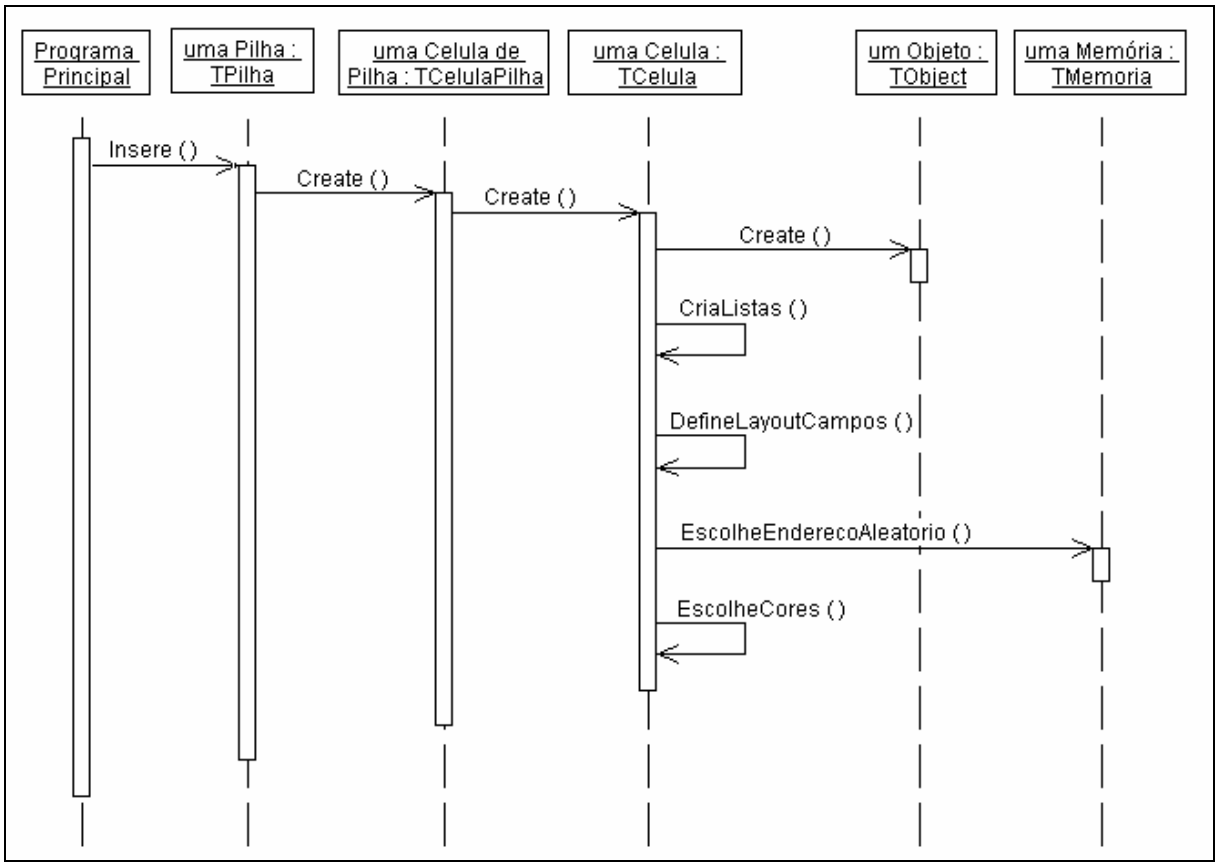
Abaixo, na fig.15, são detalhadas (com atributos e métodos) as três classes mais importantes do modelo.

Figura 15 – Detalhamento das classes mais importantes.



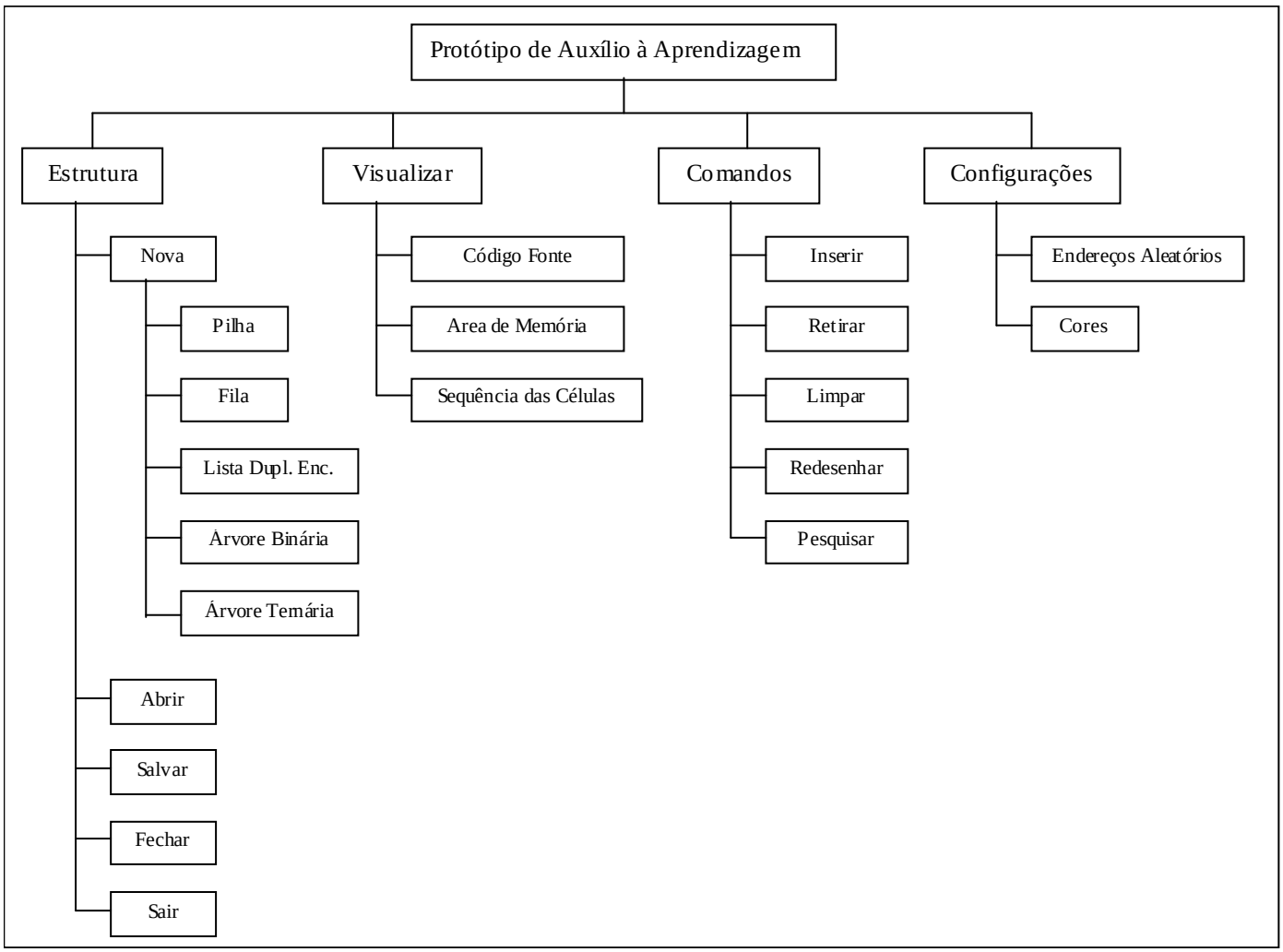
Na fig.16 é possível acompanhar, através do diagrama de sequência, todo o processo de inserção de uma célula numa estrutura de pilha.

Figura 16 - Diagrama de sequência do protótipo.



Na fig.17 está representado o diagrama hierárquico funcional, que identifica e localiza as funções disponíveis para o usuário.

Figura 17 - Diagrama hierárquico funcional



5.2 IMPLEMENTAÇÃO DO PROTÓTIPO

Apesar da linguagem e do ambiente de programação permitirem o desenvolvimento de aplicações em modo estruturado, este protótipo foi implementado utilizando exclusivamente os conceitos de orientação a objetos.

O conceito de herança foi amplamente utilizado na definição das classes. Um exemplo pode ser visto nas declarações das classes `TDescritor`, `TDescritorArvore`, e `TDescritorArvoreBinaria`, mostradas no quadro 1.

Quadro 1 - Declaração das classes `TDescritor` e descendentes.

```
Type

  TDescritor = class
    Rodape : TPanel;
    Titulo : TLabel;
  public
    Constructor Create(Janela_Base:TWinControl); virtual;
    Destructor Destroy; override;
    Function Altura : Integer;
  end;

-----------------------------------------------------------------------
  TDescritorArvore = class(TDescritor)
  protected
    Msg_Tit_Raiz,
    Msg_Raiz     : TLabel;
  public
    Ultimo_Nivel : Integer;
    Constructor Create(Janela_Base:TWinControl); override;
    Destructor Destroy; override;
  end;
-----------------------------------------------------------------------
  TDescritorArvoreBinaria = class(TDescritorArvore)
  private
    FRaiz : TCelulaArvoreBinaria;
    Procedure SetRaiz(Value:TCelulaArvoreBinaria);
    Function  GetRaiz: TCelulaArvoreBinaria;
  public
    Property Raiz : TCelulaArvoreBinaria read GetRaiz write SetRaiz;
    Constructor Create(Janela_Base:TWinControl); override;
  end;
```

Como pode ser visto, `TDescritorArvoreBinaria` herda `TDescritorArvore`, que herda `TDescritor`, que herda por sua vez `TObject`, que é a classe primária do Delphi, da qual todos os objetos derivam. Dessa forma, não é necessário redeclarar e reimplementar atributos e métodos já declarados e implementados na classe base, com exceção daqueles métodos que se queira alterar o código. É o caso do método `Create`, que nos casos acima precisa ser redefinido, para que possa ser efetuada a criação e inicialização dos novos atributos.

O conceito de polimorfismo foi outro usado com bastante freqüência. Um exemplo está na declaração das classes `TEstrutura` e suas classes descendentes `TFila`, `TPilha`, `TListaDE`, `TArvoreBinaria` e `TArvoreTernaria`, mostradas nos quadros 2, 3, 4, 5, 6 e

7, respectivamente. Os quadros trazem apenas a parte do código relevante para a visualização do polimorfismo.

Quadro 2 - Declaração da classe `TEstrutura`.

```
type
  TEstrutura = class
  (...)
  public
    (...)
    Function  Vazia : Boolean; virtual; abstract;
    Procedure Insere(Valor_Campo:TArrayValorCampo); virtual; abstract;
    Procedure Retira; virtual; abstract;
    Procedure Pesquisa; virtual; abstract;
    Procedure AtualizaSequencias; virtual; abstract;
    Procedure ReDesenha; virtual; abstract;
  end;
```

Quadro 3 - Declaração da classe `TFila`.

```
type
  TFila = class(TEstrutura)
  (...)
  public
    (...)
    Function  Vazia : Boolean; override;
    Procedure Insere(Valor_Campo:TArrayValorCampo); override;
    Procedure Retira; override;
    Procedure Pesquisa; override;
    Procedure AtualizaSequencias; override;
    Procedure ReDesenha; override;
  end;
```

Quadro 4 - Declaração da classe `TPilha`.

```
type
  TPilha = class(TEstrutura)
  (...)
 public
    (...)
    Function  Vazia : Boolean; override;
    Procedure Insere(Valor_Campo:TArrayValorCampo); override;
    Procedure Retira; override;
    Procedure Pesquisa; override;
    Procedure AtualizaSequencias; override;
    Procedure ReDesenha; override;
  end;
```

Quadro 5 - Declaração da classe `TListaDE`.

```
type
  TListaDE = class(TEstrutura)
  (...)
  public
    (...)
    Function  Vazia : Boolean; override;
    Procedure Insere(Valor_Campo:TArrayValorCampo); override;
    Procedure Retira; override;
    Procedure Pesquisa; override;
    Procedure AtualizaSequencias; override;
    Procedure ReDesenha; override;
  end;
```

Quadro 6 - Declaração da classe `TArvoreBinaria`.

```
type
  TArvoreBinaria = class(TEstrutura)
  (...)
  public
    (...)
    Function  Vazia : Boolean; override;
    Procedure Insere(Valor_Campo:TArrayValorCampo); override;
    Procedure Retira; override;
    Procedure Pesquisa; override;
    Procedure AtualizaSequencias; override;
    Procedure ReDesenha; override;
  end;
```

Quadro 7 - Declaração da classe `TArvoreTernaria`.

```
type
  TArvoreTernaria = class(TEstrutura)
  (...)
  public
    (...)
    Function  Vazia : Boolean; override;
    Procedure Insere(Valor_Campo:TArrayValorCampo); override;
    Procedure Retira; override;
    Procedure Pesquisa; override;
    Procedure AtualizaSequencias; override;
    Procedure ReDesenha; override;
  end;
```

Se forem comparados os quadros acima (de 2 a 7), é nítida a similaridade entre os códigos. Isso se deve ao fato de na classe base `TEstrutura` existirem métodos declarados como virtuais, permitindo que cada classe descendente faça seu próprio tratamento para estes métodos. Dessa forma, os métodos declarados como virtuais são redeclarados em cada uma

das classes descendentes, trocando-se a palavras reservadas `virtual` por `override`, e reimplementados com os novos códigos.

Na declaração da classe base pode-se observar ainda a palavra reservada `abstract` depois de cada método. Isso significa que estas rotinas não serão implementadas na classe base, mas apenas nas descendentes. Caso seja feita uma chamada de um destes métodos para a classe base, ocorrerá um erro do tipo `RunTime Abstract Error`, uma vez que o compilador não tem como acusar o erro durante a compilação. Normalmente este recurso é utilizado quando a classe toda é abstrata, ou seja, não serão instanciados objetos dessa classe.

Para facilitar a leitura e possíveis alterações, foi criada uma *unit* para cada classe e uma para o programa principal. Assim sendo, uma classe específica estará definida na *unit* com o seu nome e o programa principal na *unit* `uPrincipal.pas`.

5.3 FUNCIONAMENTO DO PROTÓTIPO

Este tópico explica as principais funções encontradas no protótipo para criação, manipulação e salvamento das estruturas de dados.

Ao executar o protótipo, aparece a tela principal com seu menu e barra de atalhos. O menu está dividido em quatro partes: Estrutura, Visualizar, Comandos e Configurações.

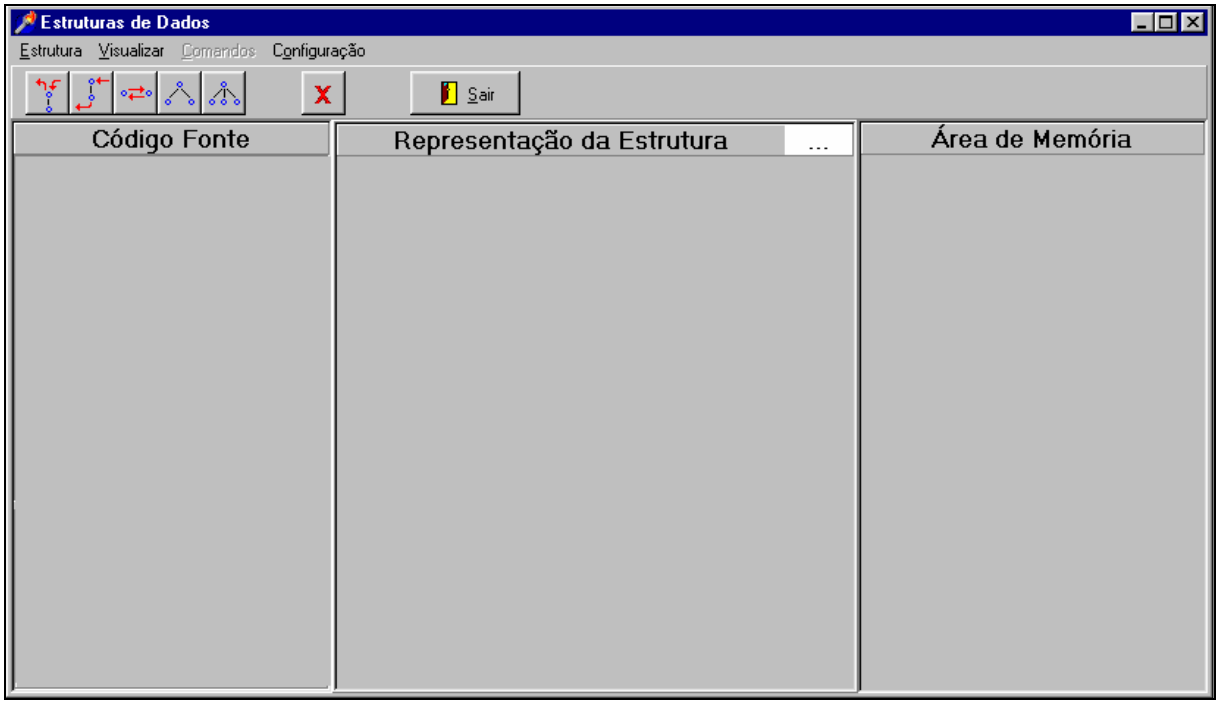
O menu Estrutura refere-se às operações a serem efetuadas com os arquivos que guardam as estruturas criadas pelo protótipo, cuja extensão é “.est”. Neste menu é possível abrir, salvar, fechar ou ainda criar uma estrutura nova. Mantendo um padrão já conhecido dos usuários de aplicativos, foi incluída também a opção de sair do protótipo.

O menu Visualizar permite que sejam visualizadas ou ocultadas as janelas que mostram o código fonte e a área de memória. A opção Sequência das Células permite que apareça uma indicação com números sequenciais ao lado de cada célula, facilitando o entendimento da ordem estabelecida entre as células.

O menu Comandos contém todos os comandos possíveis para a manipulação da estrutura aberta num determinado instante. Pode-se inserir ou retirar células, limpar toda a estrutura (excluindo todas as células), redesenhá-la, ou ainda pesquisar valores dentro dela.

O último menu é o de Configuração, que tem duas opções. A primeira permite ao usuário determinar se a alocação da área de memória, feita no momento da inclusão de uma célula, obedecerá uma ordem sequencial ou aleatória. A Segunda permite a alteração de algumas cores no protótipo, como a cor que representará os endereços de memória, por exemplo. A tela inicial pode ser vista na fig.18.

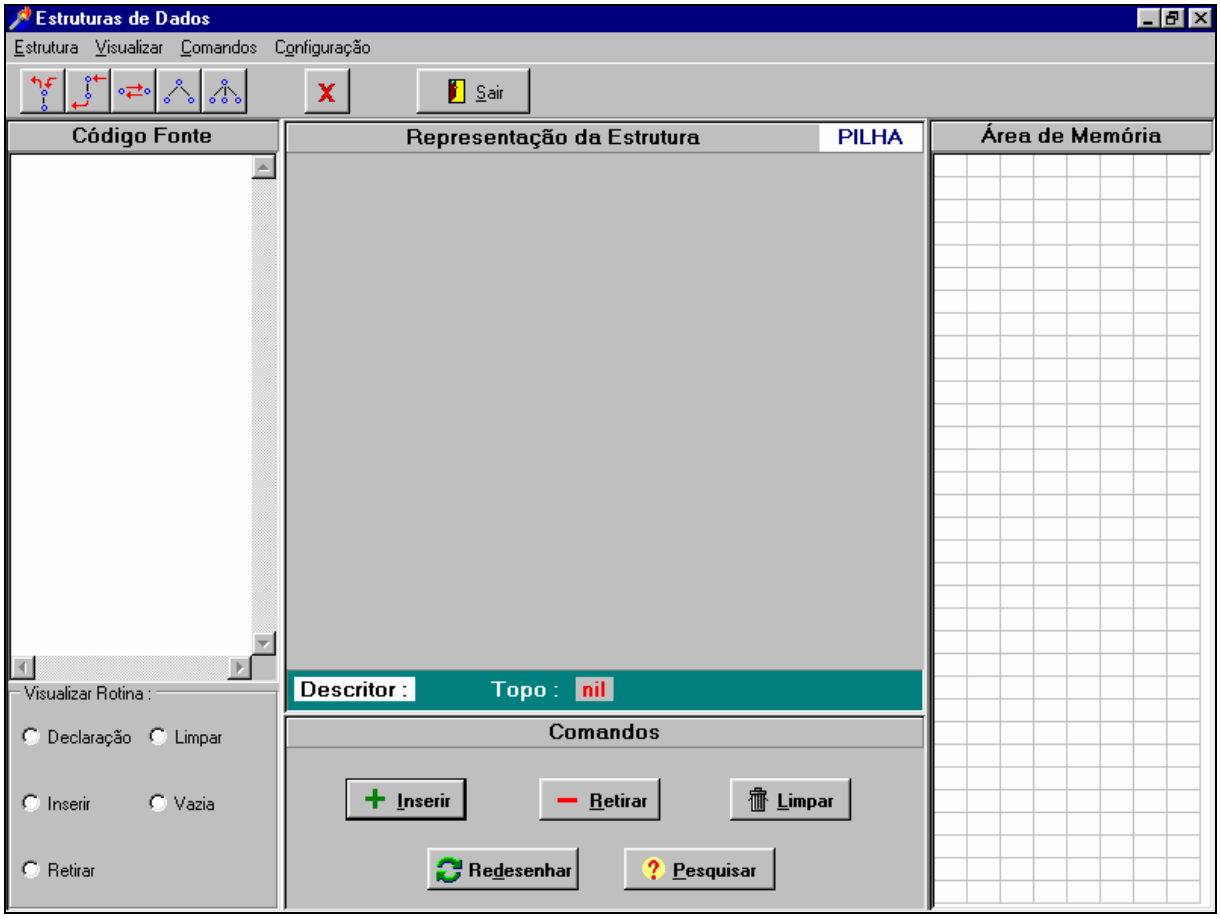
Figura 18 - Tela principal



Neste momento, o menu Comando está desabilitado, pois nenhuma estrutura foi aberta (criada). O usuário pode escolher uma das estruturas apresentadas selecionando-a através dos botões da barra de atalhos ou ainda abrindo o menu Estrutura/Nova.

Ao selecionar uma estrutura, o protótipo fará a inicialização da mesma, habilitando as três grandes áreas que compõem a tela. Este passo pode ser visualizado na fig.19.

Figura 19 - Tela depois da estrutura criada.

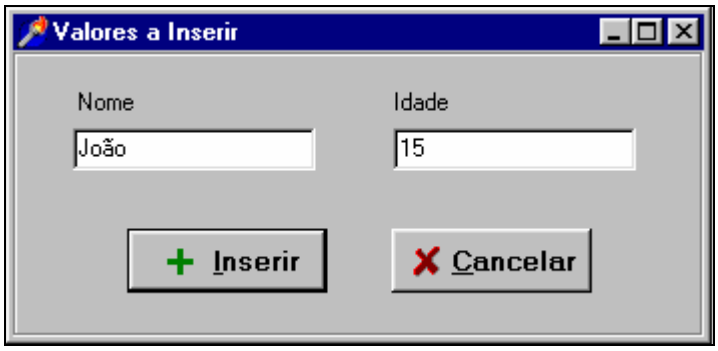


Aqui aparecem as três janelas com informações a respeito da estrutura selecionada, que neste caso é uma Pilha. Na janela da esquerda aparecem as opções de visualização do código fonte, onde o usuário pode ver o código das funções da estrutura ou ainda da declaração do tipo. Na janela da direita aparece a área de memória, na qual serão representadas em detalhes todas as células da estrutura em questão. Na parte superior da janela central é colocado um título identificando a estrutura escolhida pelo usuário e na parte inferior aparecem as informações referentes ao descritor da estrutura.

Depois da estrutura criada, é possível manipulá-la através dos botões de comando na parte central inferior, ou ainda pela opção de menu “Comandos”. É possível ainda fechar (destruir) a estrutura, ou abrir imediatamente uma nova. Neste último caso, o protótipo se encarregará de fechar a estrutura que por ventura esteja aberta, para depois abrir a que foi selecionada.

Para inserir uma célula na estrutura, o usuário deve clicar no botão Inserir. Uma pequena caixa de diálogo será exibida, pedindo sejam digitadas as informações para a nova célula, sendo que já aparecem alguns valores iniciais como sugestão. Esta caixa de diálogo é mostrada na fig.20.

Figura 20 - Caixa de diálogo para inserir uma célula.



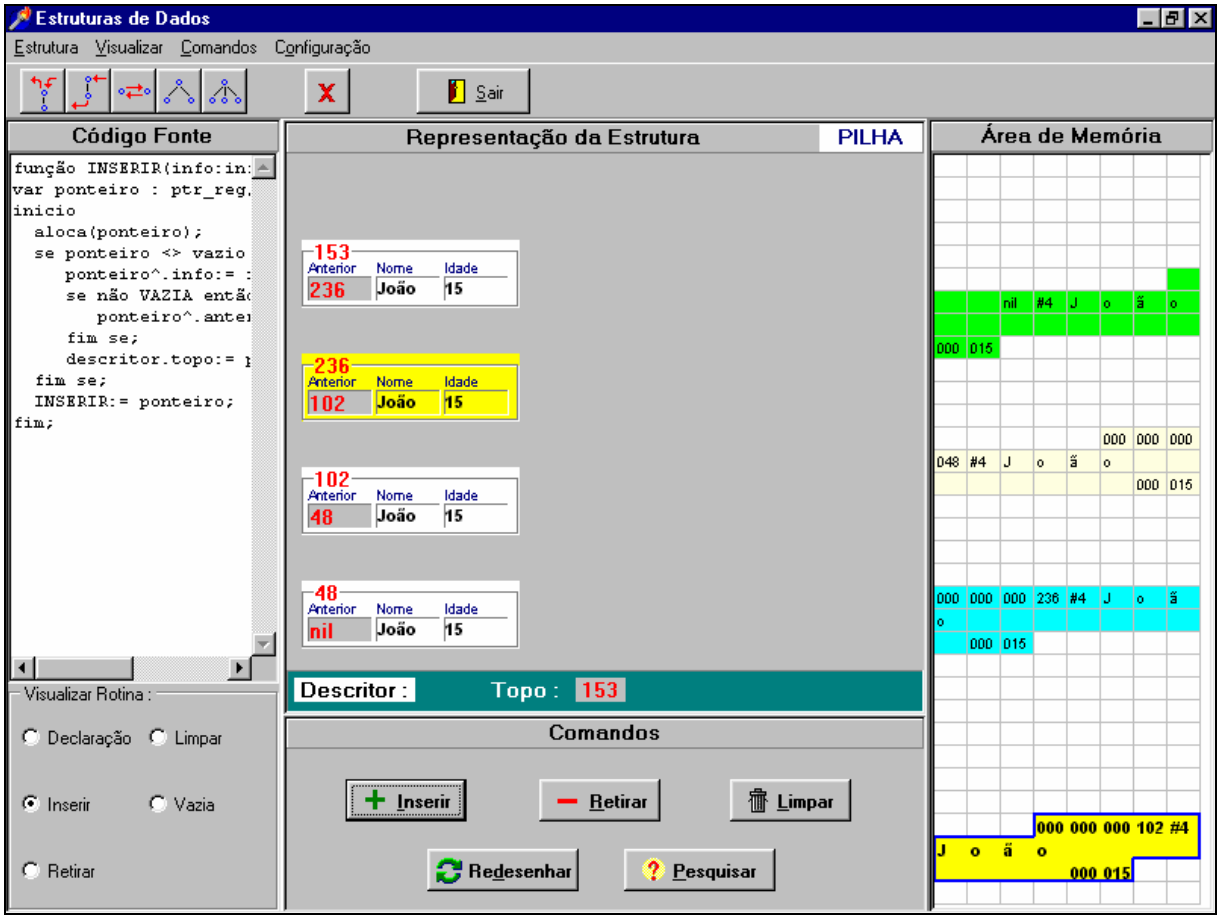
Após informar os valores que deseja inserir, o usuário deve clicar no botão Inserir para efetivar a inserção. Este procedimento resultará na inclusão de uma célula semelhante a da fig.21.

Figura 21 - Uma célula.

Esta é uma célula de uma pilha. Ela é composta por um endereço (102), dois campos contendo dados (aqueles que o usuário inseriu) e um campo do tipo ponteiro, que aponta para uma outra célula, neste caso a célula de endereço 48.

Feito isso, a estrutura agora terá um novo elemento, uma nova célula. Depois de algumas inserções a tela ficará como mostra a fig.22.

Figura 22 - Uma estrutura com várias células.



Pode-se observar na figura acima as três áreas da tela preenchidas com as informações específicas de cada uma.

A área de memória mostra a representação interna das células que aparecem na tela central. Cada pequeno retângulo dentro da região pintada corresponde a um byte ocupado pela célula para guardar as suas informações. No caso acima, cada célula do tipo pilha tem nos quatro primeiros bytes (retângulos) a indicação do endereço de memória (000 000 000 102), que faz a ligação da mesma com outra célula. Os treze bytes seguintes correspondem à informação do campo “Nome”, que é uma cadeia de até doze caracteres representativos (João), mais um valor indicador da quantidade destes caracteres (#4). Por último estão os dois bytes referentes ao campo “Idade”, que é um valor inteiro (000, 015).

O código fonte oferece a possibilidade de visualização dos procedimentos e funções mais comuns de cada estrutura, bem como suas respectivas declarações de tipos. O código encontra-se escrito em portugol e em formato estruturado, para que não haja vínculo com nenhuma linguagem específica e para que seja facilmente compreendido pelo usuário.

O sistema permite o salvamento e a recuperação de estruturas montadas. Para tanto, basta que seja selecionado no menu “Estrutura” a opção “Salvar”. O sistema irá pedir um nome para o arquivo e realizará o salvamento. Para recuperar uma estrutura salva anteriormente, a opção selecionada deverá ser “Abrir”. No quadro 8, é mostrado um exemplo de arquivo, com extensão “.est”, gerado pelo sistema.

Quadro 8 – Exemplo de arquivo de extensão “.est” gerado pelo sistema.

```
[identificacao]
tipo estrutura=1

[celula1]
campo2=Nome&João&13
campo3=Idade&15&2

[celula2]
campo2=Nome&João&13
campo3=Idade&15&2

[celula3]
campo2=Nome&João&13
campo3=Idade&15&2

[celula4]
campo2=Nome&João&13
campo3=Idade&15&2
```

Os campos referentes aos endereços de memória não são armazenados, uma vez que a alocação é feita novamente no momento da inserção de cada célula.

6 CONCLUSÕES

Apesar do empenho ao longo da pesquisa bibliográfica e do desenvolvimento do protótipo, não é possível afirmar ainda que o protótipo tenha alcançado seu objetivo didático e pedagógico. Isso porque, até o presente momento, o mesmo ainda não passou por uma avaliação dos futuros usuários, que são os estudantes universitários da área de computação. Dessa forma, espera-se que esta avaliação seja feita o mais breve possível, para poder validar, ou não, o produto final deste trabalho.

Em relação às ferramentas utilizadas para a especificação e o desenvolvimento, ambas mostraram-se eficientes. A utilização das técnicas de orientação a objetos facilitou bastante a implementação do protótipo, principalmente pela simplicidade para realizar alterações na especificação e no código.

De qualquer forma, graças ao estudo realizado e à experiência adquirida, ficou muito mais fácil o desenvolvimento de softwares educacionais, uma vez que foi adquirido todo um conhecimento técnico e pedagógico a respeito dos mesmos. Pode ser uma excelente porta aberta para futuros projetos de software nessa área.

6.1 LIMITAÇÕES

A maior dificuldade enfrentada foi a falta de consenso entre os autores da área pedagógica, o que causou uma confusão inicial em relação aos conceitos e à relevância de cada tópico ou assunto estudado.

Não foram apresentadas todas as estruturas de dados conhecidas, em função da abrangência do assunto e do limite de tempo para realização deste trabalho.

6.2 EXTENSÕES

Sugere-se como extensão para trabalhos futuros nesta área, a abordagem de outras estruturas de dados, utilizando os mesmos fundamentos aqui apresentados.

Outra sugestão é a utilização de agentes inteligentes que dêem uma outra dinâmica ao processo de manipulação das estruturas.

REFERÊNCIAS BIBLIOGRÁFICAS

[ABR1998] ABREU, Rosane de Albuquerque dos Santos. **Software educacional ou o caráter educacional do software?** Revista Tecnologia Educacional, Rio de Janeiro : EBT, n. 142, p. 26, Set. 1998.

[AND2000] ANDRES, Daniele Pinto; CYBIS, Walter de Abreu. Um estudo sobre as técnicas de avaliação de software educacional. In: Seminário de Computação, 9, 2000, Blumenau. **Anais**... Blumenau: FURB, 2000. P.95-104.

[BAR1973] BARTON, Richard F.. **Manual de Simulação e Jogo.** Trad. de Roberto Adler. Petrópolis : Vozes, 1973.

[CAM1994] CAMPOS, Gilda Helena Bernardino de. **Metodologia para avaliação da qualidade de software educacional**: Diretrizes para desenvolvedores e usuários. Rio de Janeiro, 1994. Tese de Doutorado (Ciências em Engenharia de Produção) Coordenação de Programas de Pós-Graduação em Engenharia, Universidade Federal do Rio de Janeiro.

[CAN1996] CANTÙ, Marco. **Dominando o Delphi 2.** Trad. de Edmilson K. Miyasaki. São Paulo : Makron Books, 1996.

[COA1991] COAD, Peter; YOURDON, Ed. **Análise baseada em objetos.** Rio de Janeiro : Campus, 1991.

[COB1982] COBURN, Peter et al. **Pratical guide to computers in education**. New York : Addison-Wesley, 1982.

[COL1996] COLEMAN, Derek et al. **Desenvolvimento orientado a objetos**: o método fusion. Trad. de Geraldo Costa Filho. Rio de Janeiro : Campus, 1996.

[DUK1975] DUKE, R. D.; GREENBLAT, Cathy S.. **Gaming Simulation: Rationale, Design and Applications.** New York : Halsted Press Division, 1975.

[ERI1998] ERIKSSON, Hans; PENKER, Magnus. **UML Toolkit.** Nova York : Ed. Wiley, 1998.

[FIS1990] FISCHETTI, E.; GISOLFI, A.. **From computer-aided instruction to intelligent tutoring systems**. New Jersey : Educational Technology, 1990.

[HEI2000] HEINZLE, Roberto. **Material didático** 2000. Endereço Eletrônico: http://www.inf.furb.rct-sc.br/~heinzle/disciplinas.htm. Data da consulta: 08/10/2000.

[HOR1984] HOROWITZ, Ellis; SAHNI, Sartaj. **Fundamentos de estruturas de dados.** Trad. de Thomasz R. Rawicki. Rio de Janeiro : Campus, 1984.

[JUN2000] JÚNIOR, Silvino Schlickmann. **Linguagens orientadas a objetos**, 1998. Endereço Eletrônico: http://www.cfh.ufsc.br/~junior/linguagens.htm. Data da consulta: 09/10/2000.

[KAH1991] KAHN, Brian. **Os computadores no ensino da ciência**. Lisboa : Publicações Dom Quixote, 1991.

[LAR2000] LARMAN, Craig. **Utilizando U.M.L. e padrões**: uma introdução à análise e ao projeto orientados a objetos. Trad. de Luiz A. Meirelles Salgado. Porto Alegre : Bookman, 2000.

[LAT1983] LATHROP, Ann; GOODSON, Bobby. **Courseware in the classroom: Selecting, organinzing and using educational software**. Califórnia : Addison-Wesley, 1983.

[LOR1993] LORENZ, Mark. **Object-oriented software development**: a practical guide. New Jersey : Prehtice-Hall, 1993.

[LUI2000] LUIZ, José. **Material didático** 2000. Endereço Eletrônico: http://.inf.ulbrajp.com.br/~joseluiz/index.htm. Data da consulta: 05/11/2000.

[MAR1992] MARTIN, James. **Hiperdocumentos e como criá-los.** Rio de Janeiro : Campus, 1992.

[MAR1994] MARTIN, James. **Princípios de análise e projeto baseados em objetos.** Rio de Janeiro : Campus, 1994.

[MAR1995] MARTIN, James; ODELL, James J. **Análise e projeto orientado a objeto.** São Paulo : Makron Books do Brasil, 1995.

[MEN1993] MENDONÇA, Luciana F.; ROCHA, A.R.C.. **Avaliação de Hipertextos**. Relatório Técnico do Programa de Engenharia de Sistemas e Computação n. 280 arquivado na Universidade Federal do Rio de Janeiro, abr. 1993.

[MER1986] MERRILL, Paul F. et al. **Computers in Education**. New Jersey : Prentice-Hall, 1986.

[OGB1992] OGBORN, John; TEODORO, V.; FREITAS, J. . **Modelação com o computador:** possibilidades e perspectivas. Desenvolvimento de sistemas educativos. São Paulo : GEP, 1992.

[RAT2000] RATIONAL, Software Corporation. **Unified Modeling Language**, version 1.3 2000. Endereço Eletrônico: http://www.rational.com/uml/index.jsp. Data da consulta: 18/11/2000.

[RON1980] RONCA, Antônio C. C.; ESCOBAR, Virgínia F.. **Técnicas Pedagógicas: domesticação ou desafio à participação?**. Petrópolis : Vozes, 1980.

[RUM1994] RUMBAUGH, James et al. **Modelagem e projetos baseados em objetos.** Rio de Janeiro : Campus, 1994.

[STR1984] STRACK, Jair. **GPSS modelagem e simulação de sistemas.** Rio de Janeiro : Livros Técnicos e Científicos Editora S.A., 1984.

[SWA1991] SWAIT JÚNIOR, Joffre Dan. **Fundamentos computacionais, algoritmos e estrutura de dados**. São Paulo : Makron, McGraw-Hill, 1991.

[TAY1972] TAYLOR, J. R.; WALFORD, R.. **Simulation in the classroom**. Middlesex : Peguin Books, 1972.

[VEL1986] VELOSO, Paulo et al. **Estruturas de dados**. Rio de Janeiro : Campus, 1986.

[WER1993] WERNECK, V. M. B.; CARVALHO, Luis A.. **Sistemas Baseados em Conhecimento**. Relatório Técnico do Programa de Engenharia de Sistemas e Computação n. 286 arquivado na Universidade Federal do Rio de Janeiro, nov. 1993.

[WIN1993] WINBLAD, Ann L.; EDWARDS, Samuel D.; KING, David R. **Software orientado ao objeto.** São Paulo : Makron Books do Brasil, 1993.

[WIR1990] WIRFS-BROCK, Rebecca; WILKERSON, Brian; WIENER, Lauren. **Designing object-oriented software.** New Jersey : Prehtice-Hall, 1990.

[ZIV1993] ZIVIANI, Nivio. **Projeto de algoritmos**: com implementações em Pascal e C. São Paulo : Pioneira, 1993.