ELEANDRO MASCHIO KRYNSKI

UMA ABORDAGEM METACOGNITIVA ATRAVÉS DE MÚLTIPLAS REPRESENTAÇÕES EXTERNAS PARA O ENSINO DE PROGRAMAÇÃO DE COMPUTADORES

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Alexandre Ibrahim Direne

CURITIBA

2007

ELEANDRO MASCHIO KRYNSKI

UMA ABORDAGEM METACOGNITIVA ATRAVÉS DE MÚLTIPLAS REPRESENTAÇÕES EXTERNAS PARA O ENSINO DE PROGRAMAÇÃO DE COMPUTADORES

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Alexandre Ibrahim Direne

CURITIBA

2007

ELEANDRO MASCHIO KRYNSKI

UMA ABORDAGEM METACOGNITIVA ATRAVÉS DE MÚLTIPLAS REPRESENTAÇÕES EXTERNAS PARA O ENSINO DE PROGRAMAÇÃO DE COMPUTADORES

Dissertação aprovada como requisito parcial à obtenção do grau de Mestre no Programa de Pós-Graduação em Informática da Universidade Federal do Paraná, pela Comissão formada pelos professores:

Orientador: Prof. Dr. Alexandre Ibrahim Direne
Departamento de Informática, UFPR

Prof. Dr. Sergio Crespo Coelho da Silva Pinto
Programa Interdisciplinar de Pós-Graduação em Computação Aplicada, UNISINOS

Prof. Dr. Renato José da Silva Carmo
Departamento de Informática, UFPR

Prof. Dr. André Luiz Pires Guedes
Departamento de Informática, UFPR

Curitiba, 18 de abril de 2007

"Bom mesmo é ir à luta com determinação,
abraçar a vida e viver com paixão,
perder com classe e vencer com ousadia,
porque o mundo pertence a quem se atreve
e a vida é muito para ser insignificante."
Charles Spencer Chaplin

AGRADECIMENTOS

"Se eu vi mais longe, foi por estar
de pé sobre ombros de gigantes."
Isaac Newton

À vida, pelas lições e oportunidades, por sempre me dar mais motivos para agradecer do que para lamentar e por haver me privilegiado com a amizade das pessoas valiosas que aqui demonstro verdadeira gratidão.

Aos meus pais, Cristian e Joceline, pelo amparo, desvelos e saudades.

À minha irmã, Christielli, sempre amiga nas difíceis jornadas.

À Ana Carolina, minha namorada, pelo amor, compreensão e carinho.

À minha família como um todo, pela receptividade e confiança transmitida.

Aos amigos Carolina e Richard, por terem possibilitado muito desta caminhada.

Às amizades de uma vida inteira, por compartilharmos alegres momentos e auxiliarmo-nos naqueles mais custosos.

Ao professor Alexandre Direne, orientador e grande amigo, pela assiduidade, motivação e confiança com que conduziu este trabalho, pelo extraordinário exemplo humano que transformou a maneira com que leciono.

Ao Departamento de Informática da Universidade Federal do Paraná, pelo profissionalismo e comprometimento com suas atividades. Agradecimentos especiais aos professores Renato e Antônio Urban, pela cordialidade e prudência com que contribuíram para este trabalho, e à Jucélia pelo ânimo e prontidão com que nos auxilia.

Aos amigos de curso que compartilharam desta bela experiência. Sucesso a vocês!

Aos professores, alunos e funcionários da Universidade Estadual do Centro-Oeste, pelo companheirismo, inspiração e incentivo.

SUMÁRIO

LISTA DE SIGLAS

ADG	Ambiente de Descoberta-Guiada
AIA	Ambiente Interativo de Aprendizagem
AMBAP	Ambiente de Aprendizado de Programação
DeFT	*Design, Functions, Tasks*; Projeto, Funções, Tarefas
IA	Inteligência Artificial
IAC	Instrução Assistida por Computador
IHC	Interação Humano-Computador
IIAC	Instrução Inteligente Assistida por Computador
ILA	Interpretador de Linguagem Algoritmica
JVM	*Java Virtual Machine*; Máquina Virtual Java
MREs	Múltiplas Representações Externas
RE	Representação Externa
RGB	*Red, Green, Blue*; Vermelho, Verde, Azul
RI	Representação Interna
STI	Sistema Tutor Inteligente
UFPR	Universidade Federal do Paraná
UNICENTRO	Universidade Estadual do Centro-Oeste
VP	*Visual Programming*; Programação Visual
XML	*Extensible Markup Language*; Linguagem de Marcação Extensível

LISTA DE FIGURAS

RESUMO

Este trabalho aborda linguagens e ferramentas que ajudem na cobertura da aquisição de conhecimento na transição entre o aprendizado de princípios e o desenvolvimento de perícias de programação de computadores. Para isto, propõe uma abordagem metacognitiva suportada por múltiplas representações externas, substanciada na arquitetura de um ambiente interativo de aprendizagem correlato. Esta pesquisa foi, em parte, validada através da implementação de um protótipo que possibilita a construção de algoritmos em notação de fluxograma e a geração automática do código equivalente em linguagem de alto nível, evidenciando aspectos de correspondência e sincronização entre estas representações. Apresenta-se, ao final, o prospecto de alguns trabalhos futuros que podem se embasar no estado atual de desenvolvimento desta dissertação.

Palavras-chave: ensino de programação de computadores, ambiente interativo de aprendizagem, múltiplas representações externas.

ABSTRACT

This dissertation reports on languages and tools to support the learning process that covers the phase between the acquisition of programming principles and skills. A metacognitive approach is proposed here to accomplish a solution based on the concepts of multiple external representations and interactive learning environments (ILE). The present research is partially validated through the implementation of a prototype ILE that enables an algorithm construction in flowchart notation, as well as the automatic generation of its equivalent code in a hight-level language, also enhancing correspondence and synchronization aspects between both representations. Lastly, future work is presented as a logical consequence of the current achievements.

Keywords: teaching of computer programming, interactive learning environment, multiple external representations.

CAPÍTULO 1

INTRODUÇÃO

1.1 Problema Central

Um curso de Ciência da Computação fundamenta-se, essencialmente, em programação de computadores e solução de problemas [23]. Distribui-se o ensino de programação dentre várias disciplinas ao longo do curso, cabendo à mais básica delas - geralmente "Introdução à Lógica de Programação" - abordar, segundo [53], os princípios de lógica de programação e desenvolvê-los até consolidarem perícias. Somados, os princípios e as perícias, alicerçam a análise e resolução de problemas através de algoritmos, atividade de indubitável importância na Ciência da Computação.

Distinguem-se, os princípios das perícias [26, 53], porquanto os primeiros tangem à compreensão de instruções primitivas isoladas, atendo-se à rigidez léxica, sintática e semântica determinada pela linguagem que se usa. As perícias, por sua vez, abrangem o encadeamento e o aninhamento de duas ou mais destas instruções a fim de compor um algoritmo.

Contudo, conforme revisão feita sobre as pesquisas da área, pouco se fez acerca da aquisição de conhecimento no entremeio de princípios e perícias. Assim, do que diz respeito aos últimos estágios do aprendizado de princípios até a transição evolutiva destes a perícias mais básicas, nota-se uma carência de sistemas voltados à educação. Convém ressaltar que, ainda nesta fronteira, reside parte da dificuldade do aprendiz em trabalhar com linguagens formais.

Frente a tudo isso, os sistemas educacionais neste nicho tendem, até então, a ambientes de micromundo que tratam o ensino de programação por meio de linguagens formais e permitem ao aprendiz desenvolver seus experimentos. Porém, falta atenção voltada ao monitoramento destes experimentos por meio de (múltiplas) representações externas adequadas. Trata-se de uma abordagem proveitosa visto haver representações que podem

compor um conjunto importante de ser mantido em correspondência e em sincronia.

O benefício estende-se também à experiência de alteração, pelo aprendiz, de soluções (algoritmos) previamente modelados nestes ambientes. Desta forma, mesmo que tais modificações impliquem no não-atendimento do enunciado anterior, há vantagens porque o aprendiz pode monitorar amplamente o impacto das mudanças realizadas visualizando as representações externas.

A partir disso, observa-se certa intersecção com os difundidos depuradores. Estes, todavia, inserem-se em outros contextos que não o educacional. Por conseqüência, restringem-se à visualização de células de memória em dados instantes. Assim, o foco não se mantém em representações de mais alto nível e muito menos na correspondência entre representações.

De maneira subseqüente, a utilização de um conjunto multirrepresentacional correlato às atividades de programação resultaria numa abordagem metacognitiva por parte do instrutor e do aprendiz. Esta abordagem, amplamente estudada pela área de educação [15, 16, 31, 33, 56], vem sendo apontada como próspera por delegar a quem aprende maior responsabilidade sobre o próprio aprendizado e, mediante esta autonomia, conseguir influenciar positivamente o desempenho naquilo que se ensina.

1.2 Objetivos Gerais e Contribuição

Diante do exposto, objetivou-se tratar do problema apresentado explorando linguagens e ferramentas que ajudem na cobertura da aquisição de conhecimento na transição entre o aprendizado de princípios e o desenvolvimento de perícias de programação de computadores. Para isso, propõe-se uma abordagem metacognitiva suportada por múltiplas representações externas que foi substanciada em uma arquitetura correlata.

Conforme revisão feita acerca do estado da arte nesta área, observou-se uma notória carência de pesquisas relacionadas ao foco desta proposta. Embora se possa considerar isto como um indício de originalidade da idéia aqui exposta, não compete ao presente texto tal julgamento.

Desta forma, figuraram como objetivos específicos:

- Consolidar o arcabouço conceitual de uma arquitetura para um ambiente interativo de aprendizagem através de múltiplas representações externas destinado ao ensino de programação de computadores;

- Definir um conjunto de múltiplas representações externas que atue como mediador à abordagem metacognitiva no ensino de programação de computadores. Para este conjunto foi necessário categorizar as representações componentes e precisar sobre dimensões como número, informação, forma, seqüência e tradução destas representações [2, 3];

- Implementar um protótipo correspondente à arquitetura e ao conjunto multirrepresentacional propostos. Através dele pretendeu-se validar e aperfeiçoar grande parte desta pesquisa;

- Disseminar o conhecimento alcançado em publicações da área. Desejou-se legar um sólido arcabouço para que outras pesquisas também venham a contribuir com o estado da arte do domínio em questão.

1.3 Estrutura da Dissertação

Depois desta introdução, no capítulo (2) destinado aos trabalhos correlatos, discorre-se sobre (múltiplas) representações externas e ambientes interativos de aprendizagem. Cabe ao Capítulo 3 a formalização do arcabouço conceitual adotado na solução do problema abordado, apresentando o embasamento que se teve nas concepções ligadas a micromundos e representações externas. Descreve-se no Capítulo 4 os módulos componentes da arquitetura que sustenta a abordagem proposta por este trabalho, constando também detalhes de implementação do protótipo desenvolvido.

No Capítulo 5, discute-se as repercussões esperadas do trabalho desenvolvido. Apresenta-se, no Capítulo 6, as conclusões alcançadas, bem como sugestões para trabalhos futuros. Os anexos encerram este documento.

CAPÍTULO 2

TRABALHOS CORRELATOS

2.1 Representação Externa

O termo Representação Externa[1] (RE) [21, 72], no escopo do presente documento, refere-se ao uso de técnicas para representar, organizar e apresentar conhecimento. Remete, portanto, a uma ampla variedade de representações que compreende desde modelos proposicionais/sentenciais até modelos gráficos/diagramáticos. Preenchendo a lacuna entre esses extremos, há representações intermediárias que associam elementos gráficos e textuais (e.g. tabelas, que denotam matrizes graficamente). Logo, são exemplos de RE: sentenças em linguagem natural, sentenças em linguagens formais (como lógica de primeira-ordem), tabelas, listas, grafos, mapas, projetos, diagramas, animações e até mesmo as realidades aumentada e virtual.

Segundo definição [51] apud [1], qualquer representação deve ser descrita em termos:

1. do mundo representado;
2. do mundo representante;
3. de quais aspectos do mundo representado estão sendo expressos;
4. de quais aspectos do mundo representante compõem a modelagem; e
5. da correspondência entre os dois mundos.

Desta forma, pode-se encarar RE como um fenômeno cotidiano, presente quando se faz uma lista de compras para ir ao supermercado, rascunha-se um mapa para especificar direção, informa-se através das placas de trânsito e embalagens de produtos, vê-se as horas num relógio ou disputa-se uma partida de xadrez (e mesmo jogo-da-velha). Convém mencionar que todos esses são exemplos da utilização de RE na solução de problemas.

[1] Do inglês *External Representation* (*ER*).

Como conseqüência do bom emprego de REs nas tarefas do dia-a-dia, estas tornaram-se um suporte valioso à resolução de uma diversidade de problemas de caráter mais formal, bem como um poderoso auxílio cognitivo, a saber:

- Arquitetura e design: esboços (ou croquis) a mão livre têm valor incomensurável neste domínio [36] apud [21], não somente pelo poder representativo e esquematizador, mas também em razão do estímulo criativo que são capazes de angariar. Ao encontro disso, alguns autores [64, 27, 49] sustentam que o próprio ato de esboçar garante acesso às várias imagens mentais, figurais ou conceituais que potencialmente resultarão em alternativas para o problema em que se trabalha. Outrossim, tais pesquisadores evocam a serventia do rascunhar em outros campos do conhecimento. Não consideram exagero, portanto, a difundida premissa de que sem o desenho não se pode conceber o projeto;

- Raciocínio analógico (ou por analogia): segundo [35], a organização do sistema perceptivo humano concentra-se em objetos e em relações entre objetos. Dessa forma, REs que enfatizem estruturas relacionais, tais como diagramas e afins, vêm a constituir um conveniente recurso em se tratando de raciocínio analógico [9] apud [21], [35, 52]. Há pesquisas específicas em que se manifesta o termo "analogia visual" [37, 24], dizendo respeito à importância do raciocínio visual mesmo quando se trata de outros tipos de analogia, pois situações aparentemente dissimilares podem se revelar parecidas quando re-representadas visualmente [11, 25];

- Classificação de informações hierárquicas: destacam-se REs como árvores de classificação, árvores e tabelas de decisão, regras e também sentenças *if-then* [19]. Ademais, realizou-se experimentos com crianças [38] apud [21] incentivando-as a criarem suas próprias REs para informações que, por natureza, eram hierárquicas. Conferiu-se que a qualidade das representações criadas relacionava-se com a capacidade daquelas crianças de responderem perguntas sobre o assunto;

- Álgebra: o uso de diferentes REs demonstrou-se útil na interpretação de enunciados algébricos [61, 14]. Em estimativa prévia, apenas 17% de um grupo de colegiais

solucionariam um determinado problema de baixa dificuldade. Após estudos dirigidos houve acréscimo de 10% a 16% na proporção de respostas certas, bem como uma melhora de 50% na formulação de equações que levavam a alguma solução;

- Raciocínio lógico e analítico: a RE constitui uma ferramenta primordial ao raciocínio lógico e analítico [22]. Isto pode ser verificado pela necessidade intuitiva de se representar determinadas propriedades do problema que se tenta resolver. Experimentos realizados com o passatempo clássico "Torre de Hanoi" [50, 68] detectaram que oscilações no tempo de resolução associam-se não somente ao nível intelectual de quem resolve, mas também à RE empregada;

- Física: a resolução de problemas [8], principalmente aqueles que não se restringem à mera parametrização das equações aplicadas, demandam REs que contenham, além de entidades concretas, entidades abstratas como: velocidade, aceleração, força, movimento, calor, correntes, ondas. Embora estas variem conforme o segmento (e.g. estática, termodinâmica, eletrodinâmica, acústica), a necessidade de RE permanece inerente;

- Matemática: assim como na física, a resolução de problemas matemáticos recai em REs. Além disso, ultimamente, aproveita-se destas na educação especial [40], atendendo a estudantes excepcionais, com distúrbios de aprendizado ou ainda com aversão à disciplina de matemática;

- Química: sem o suporte de REs, a rotina e o aprendizado [18, 66] nesta área seriam um tanto conturbados (inexeqüíveis sob alguns aspectos). A tabela periódica, o modelo atômico de Bohr, a fórmula estrutural de compostos, entre outros, constituem ricos exemplos de RE. Recentemente, animações [65] e ambientes de ensino [71] conquistam espaço;

- Música: sem os elementos de escrituração musical não se legaria as obras de virtuosos do passado. Seria um feito irrealizável formalizar e comunicar tal conhecimento. Através da partitura transcreve-se informações como altura, intensidade e duração

das notas musicais, assim como a organização disso tudo em um determinado ritmo. Salienta-se que essa notação deve constantemente evoluir para que possa compreender técnicas que a música passa a incorporar (distorções de guitarra, por exemplo). Outra característica marcante reside no fato de ser universalmente utilizada, permitindo que músicos de diferentes idiomas e culturas se reúnam para executar uma mesma composição. Além da partitura, há outras representações como tablaturas, cifras, diagramas de escalas de instrumentos, diagramas de posicionamento das mãos, notações para dedilhados e batidas, entre inúmeras.

Isto posto, muitos crêem que o único benefício do uso de REs seja facilitar o processo de memorização, reduzindo a carga cognitiva necessária para se realizar determinada tarefa. Embora esta seja a característica mais notável, observa-se ainda as seguintes propriedades [72] permeando REs:

1. Prover informações possíveis de serem diretamente percebidas e utilizadas sem nenhuma interpretação ou formulação explícita;

2. Fixar e estruturar comportamento cognitivo, porquanto a estrutura física das REs restringe a gama de comportamentos (alguns são permitidos e outros proibidos);

3. Modificar a natureza das tarefas em questão, tornando-as potencialmente mais fáceis serem realizadas.

Atualmente, o compartilhamento de REs construídas por aprendizes e REs dinâmicas são duas linhas de pesquisa que têm sido bastante proeminentes. A primeira [39] defende que há grandes vantagens quando os aprendizes constroem (ou mesmo criam), compartilham e avaliam as representações uns dos outros. Há, portanto, influência positiva na forma com que os aprendizes se comunicam, analisam e compreendem conteúdos difíceis, visto que a simples exposição de REs por parte do instrutor com freqüência incorre em interpretação e aplicação errôneas do que se transmite. As REs dinâmicas [5, 55], por sua vez, vêm sendo antecipadas como benéficas na educação, dada a facilidade de apresentar explicitamente a atividade de um sistema ou processo. Ainda assim, pouco se conhece sobre benefícios (ou desvantagens) cognitivos específicos do uso de REs dinâmicas.

Embora uma visão geral de RE até aqui tenha sido apresentada, o âmbito deste trabalho restringe-se àquelas destinadas a ambientes de ensino/aprendizagem. Desta forma, são consideradas de natureza tipicamente metacognitiva.

2.1.1 Múltiplas Representações Externas

Além do supracitado, há evidências abundantes [2] expondo vantagens da utilização de REs como suporte tanto ao ensino quanto à resolução de problemas. Estudos ressaltam que se eleva o desempenho do aprendiz ao proporcionar-lhe interação com REs apropriadas. Neste sentido, ultimamente as pesquisas concentram-se na combinação de diferentes REs para intermediar o aprendizado. Múltiplas Representações Externas[2] (MREs) [1, 2], denominação dada ao conjunto, tem se mostrado um nicho próspero no atendimento às necessidades cognitivas de aprendizes diversos.

Os estudos precursores restringiam-se a investigar acerca da influência que a apresentação de imagens ao longo da informação escrita poderia exercer sobre a compreensão de textos [44, 69] apud [4]. Posteriormente, a discussão foi estendida de modo a abranger uma ampla variedade de representações que incluía até mesmo som, vídeo, animação e simulação dinâmica. Nesse entremeio, pesquisadores punham-se a indagar se representações específicas ofereciam vantagens equivalentes ao ensino e à resolução de problemas. A exemplo disso, observou-se que diagramas [43] promovem a percepção agrupando informações relevantes e facilitando tarefas como busca e reconhecimento. Da mesma forma, tabelas tendem a explicitar informação, enfatizando espaços não explorados (células vazias) e diminuindo o esforço de leitura. Representações ilustrativas, sendo de natureza icônica, mostram-se úteis no fornecimento de informações concretas a respeito de instanciações particulares do objeto em questão. As representações descritivas, em contraste, tratam de manifestar informações gerais do objeto - são de natureza simbólica [59] apud [4].

Houve outras pesquisas abordando exclusivamente a potencialidade e os possíveis problemas associados à aprendizagem com, simultaneamente, mais de uma RE [62] apud [4]. Estas sustentam que MREs relacionadas resultam num aprendizado mais flexível e

[2]Do inglês *Multiple External Representations* (*MERs*).

intuitivo, dando sustentação a diferentes idéias e estratégias. Somando a isso, sucederam considerações como a de que, para o aprendiz, existe benefício tanto ao se construir quanto ao se observar MREs [21]; ou de que a habilidade de construir e transpor entre diferentes representações (múltiplas perspectivas) de um domínio tem cunho fundamental num aprendizado bem-sucedido, visto que conduz a uma melhor compreensão do conteúdo [63] apud [4], [41] apud [7]. Contudo, também se constatou que aprendizes podem não ter êxito em se beneficiarem das vantagens anteditas, pois estas nem sempre são facilmente atingidas [70] apud [4], [17, 58] apud [5].

Não obstante, generalizações concernentes à efetividade de MREs no aprendizado são um tanto difíceis de serem alcançadas [4], porque para isso se faz necessária a predição das condições sobre as quais MREs realmente são vantajosas. Ainda não são evidentes mesmo questões de natureza mais primária, como as circunstâncias sob as quais a performance do raciocínio assistido por MREs apresenta melhora [21]. Indaga-se, por exemplo, se a troca ou transposição entre representações durante o raciocínio é cognitivamente admissível.

2.1.1.1 Taxonomia Funcionalista

São três as funções principais [2] que MREs exercem em situações de aprendizado (conforme figura 2.1): complementar, restringir e construir.

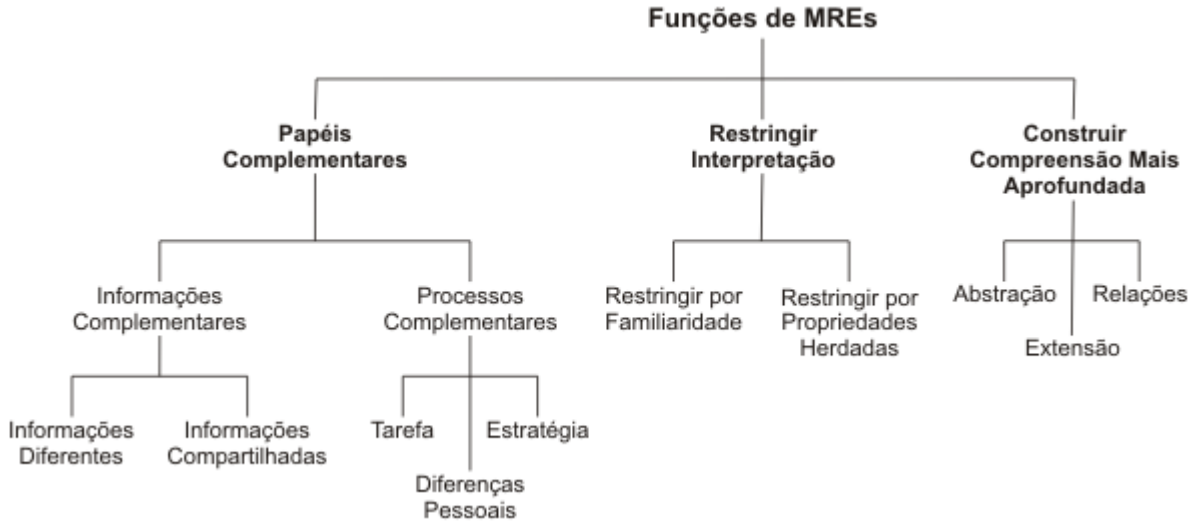


Figura 2.1: Taxonomia funcionalista de MREs (adaptado de [2])

1. Uso em papéis complementares: duas ou mais MREs têm funções complementares quando diferem na informação que cada uma expressa ou no processo que individualmente suportam. Através da combinação de MREs com essas propriedades, espera-se que o aprendiz usufrua da soma das vantagens envolvidas no conjunto de representações.

   (a) Suporte a informações complementares: explora-se as diferenças na informação expressa por cada RE em particular. Ocorre quando uma única RE mostra-se insuficiente para tratar de toda informação abrangida ou nos casos em que aglomerar a informação relevante sob uma RE seria por demais complicado. Divide-se em duas subclasses:

      i. Informações diferentes: ambas as RE modelam aspectos únicos do domínio e, por conseqüência, apresentam informações distintas;

      ii. Informações compartilhadas: há, não exclusivamente, uma intersecção de informações que as REs utilizadas compartilham.

   (b) Suporte a processos complementares: esta função assume que se admite processos complementares pela disposição de MREs. Embora estas eventualmente contenham informações equivalentes, podem ainda diferir nas inferências e processos que venham a sustentar.

      i. Diferenças pessoais: possibilita-se ao aprendiz escolher, dentre um grupo de REs, aquelas mais adequadas às suas necessidades;

      ii. Compatibilidade com a tarefa: promove-se o desempenho quando há compatibilidade entre a estrutura da informação requerida pelo problema e notação da RE escolhida. Através do fornecimento de MREs, permite-se aos aprendizes optarem pela melhor RE para o problema em foco;

      iii. Interação com estratégias: REs e estratégias de resolução de problemas interagem, sendo cada uma das primeiras associada a diferentes da última. Portanto, MREs relacionam-se com o uso de estratégias mais (ou menos) efetivas para resolver-se determinado problema. Além disso, estimulam a

tentativa de resolução por mais de uma estratégia.

2. Uso para restringir interpretação: concerne o uso de uma RE com a finalidade de restringir a interpretação de uma segunda. Pode-se obter restringindo-se por familiaridade ou por propriedades herdadas.

   (a) Restrição por familiaridade: trata-se de empregar uma RE com a qual se tem familiaridade para auxiliar na interpretação de outra menos familiar;

   (b) Restrição por propriedades herdadas: são os casos em que a interpretação de uma RE (ambígua) pode ser restringida por uma segunda RE (específica). Isto independe de questões como familiaridade ou experiência, pois uma RE age no sentido de forçar a interpretação de outra.

3. Uso para construir compreensão mais aprofundada: refere-se a quando aprendizes reúnem informação de MREs para atingir uma compreensão que seria bastante onerosa de obter com uma só RE. A compreensão mais aprofundada através de MREs dá-se por:

   (a) Abstração: compreende o processo através do qual se cria entidades mentais que embasem conceitos e procedimentos num nível mais alto de organização. Aprendizes podem construir referências através de MREs e então desvelar a estrutura que permeia o domínio tratado;

   (b) Extensão: considera-se o ato de estender o conhecimento prévio do aprendiz a novas situações sem, fundamentalmente, reconhecer a natureza deste conhecimento;

   (c) Relações: a compreensão relacional refere-se ao processo pelo qual duas REs são associadas sem haver reorganização de conhecimento. O ensino de relações entre REs pode ser uma atividade-fim (e.g. construção de um gráfico dada uma equação), ou ainda servir de base para abstração em alguns casos.

2.1.2 Abordagem Metacognitiva

Embora não se defina metacognição de maneira unívoca, na intenção deste trabalho refere-se ao conhecimento que o indivíduo tem sobre o próprio conhecimento [32] apud [31], bem como sobre os próprios processos cognitivos, as formas de operá-los e a capacidade de controlá-los [15, 31]. Nuclearmente, diz respeito à "cognição da cognição", sendo portanto "um termo novo para uma idéia antiga, isto é, aprender a aprender" [15].

O conceito foi originalmente difundido na década de setenta pois, antes disso, as pesquisas no âmbito da aprendizagem centravam-se nas capacidades cognitivas e nos fatores motivacionais convenientes [56]. Percebeu-se então uma terceira categoria de variáveis a serem estudadas, a dos processos metacognitivos que coordenam as aptidões cognitivas envolvidas na memória, leitura, compreensão de textos, resolução de problemas, entre outros. Notou-se que, além do emprego de estratégias, vem a ser importante o conhecimento sobre quando e como empregá-las, e também sobre a respectiva utilidade, eficácia e oportunidade [54] apud [56].

Alguns autores [16] enfatizam que todos os aprendizes são naturalmente metacognitivos, cabendo então aos procedimentos pedagógicos aprimorarem essas habilidades. O desenvolvimento metacognitivo, como denominado, pode ser descrito como o progresso das habilidades metacognitivas do indivíduo, conduzindo a um maior conhecimento, consciência e controle sobre o próprio aprendizado. Para isto, propõem uma visão integrativa em que se reconhece este desenvolvimento como elemento fortemente ligado a questões de contexto e de conteúdo (planos curriculares, em muitos casos). Em uma abordagem anterior [56], ensinava-se habilidades metacognitivas de maneira isolada (os chamados programas de "habilidades de estudo") e os resultados, se não fracassados, foram duvidosos.

Em contrapartida, obteve-se sucesso nas tentativas de ensinar tais habilidades entremeando um contexto integrado a conteúdos com os quais havia envolvimento e estudo prévio por parte dos aprendizes. Dessa abordagem metacognitiva originaram-se estratégias eficientes na potencialização da aprendizagem, atuando no desenvolvimento (pelo aprendiz) de métodos para lidar com a informação provinda do meio e com os próprios

processos cognitivos [56]. Compartilhando do mesmo raciocínio, propôs-se a idéia da "intenção e das capacidades"[3] [10] apud [16] para descrever metacognição em termos de duas perguntas-chave que deveriam ser auto-questionadas pelos aprendizes:

1. O que pretendo com isso? (Quais são os meus motivos?)

2. Como proponho chegar até lá? (Quais são as minhas estratégias?)

Em outras palavras, sugeriu-se que as abordagens de ensino fossem consideradas como grupos de motivos e estratégias congruentes, sendo cada intenção relacionada a uma determinada estratégia.

Uma abordagem metacognitiva, por tudo isso, ajuda o estudante a identificar e utilizar abordagens individuais de aprendizado [16]. Estas podem ser as mais diversas, como realçar texto, fazer anotações ao longo do conteúdo, sintetizar em esquemas gráficos, elaborar resumos, procurar por palavras-chave, encontrar exemplos externos, explicar em voz alta, realizar experimentos, entre outros. Através disto, delega-se ao aluno responsabilidade e controle sobre o próprio aprendizado, criando um vínculo de envolvimento muito mais intenso e ativo do que o proposto por abordagens tradicionais.

2.1.2.1 Resolução de Problemas e Metacognição

No enfoque de resolução de problemas, a metacognição pode prover um suporte também proveitoso [33]. Todo problema constitui-se de três importantes características: os dados, um objetivo e obstáculos. Os dados são os elementos, respectivas relações e condições que compõem o estado inicial da situação-problema. Os obstáculos são as características (tanto da situação quanto do indivíduo que resolve) que dificultam a transformação do estado inicial no estado final do problema. O objetivo, por fim, é a solução ou o resultado desejado do problema.

Portanto, a resolução consiste no processo de se transformar o estado inicial do problema no estado desejado. Assim, de uma forma geral, a metacognição assiste a quem

[3] Do inglês, *"the will and the skill"*.

resolve em: reconhecer que há um problema a ser resolvido, compreender exatamente de que o problema consiste e entender como alcançar a solução.

Durante a resolução de problemas, o aprendiz incorpora comportamentos tanto cognitivos quanto metacognitivos. Percebe-se isso nas três fases conceituais (e cíclicas) que compõem o processo de resolução:

1. Preparação para resolver o problema: também denominada fase de familiarização. O aprendiz empenha-se em entender a natureza do problema, os dados e o objetivo. Trata-se de uma etapa crucial no processo de resolução, porque uma solução correta depende de um entendimento adequado dessas informações;

2. Resolução do problema: ou fase de produção, corresponde ao estágio em que se produzem as rotas de solução que definem o espaço de solução do problema; e

3. Verificação do problema resolvido: fase de julgamento ou avaliação. O aprendiz resolve o problema avalia as rotas de solução a fim de selecionar a melhor delas. No caso de insucesso, retorna-se à fase 1.

Levantou-se, portanto, que o sucesso nessa atividade não se deve à solução do problema por si. Ao invés disso, deve-se ao processo metacognitivo que resulta em esforço para elaborar uma estratégia de resolução no intuito de alcançar a solução desejada [6, 50].

2.1.2.2 Tarefas (Meta)Cognitivas Envolvidas no Aprendizado com RE

Comparando-se com o domínio de Representação Interna[4] (RI), relativamente pouco se pesquisou acerca da natureza das Representações Externas na (meta)cognição. Supõe-se que isto se deva à crendice de que, com tais estudos, pouco conhecimento seria obtido sobre os modelos mentais estruturais. Outras conjecturas são a idéia pejorativa de que REs não passam de entradas ou estímulos à mente interna e a simples falta de metodologia adequada para se proceder com esses estudos [72].

[4]Do inglês *Internal Representation* (*IR*).

Embora bastante tenha tardado para que se delegasse esforços a tais estudos, recentemente percebeu-se que muito se pode aprender sobre a mente interna através de REs. Verificou-se que grande parte da estrutura da mente interna vem a ser, primordialmente, reflexão da estrutura do ambiente externo. Dessa forma, passou-se a observar REs como algo intrínseco às várias tarefas (meta)cognitivas que aquelas guiam, restringem ou determinam o comportamento cognitivo.

A seguir, descreve-se as tarefas (meta)cognitivas envolvidas no aprendizado com RE e as dificuldades com as quais os aprendizes podem se deparar enquanto não proficientes nestas tarefas [3]. Embora sejam listadas em seqüência, a abordagem de compreensão de uma nova RE pode seguir outra ordem que não esta. Ademais, salienta-se que muito dos aspectos (meta)cognitivos tangentes a MREs remete à taxonomia funcionalista anteriormente descrita (seção 2.1.1.1).

1. Aprendizes devem entender a forma da RE: isto é, saber como uma RE transcreve e apresenta informações. Outrossim, faz-se necessário conhecer os respectivos operadores da RE em questão;

2. Aprendizes devem entender a relação entre a RE e o domínio: a interpretação de REs é uma atividade inerentemente contextualizada, visto a necessidade de se compreender a relação da RE e o domínio representado. Todavia, há uma característica distinta para tarefas de aprendizado que, em oposição à prática profissional e à resolução de problemas, têm essa compreensão formada a partir de um conhecimento superficial (ou incompleto) do domínio;

3. Aprendizes provavelmente precisam entender como escolher uma RE apropriada: situações específicas exigem que o aprendiz opte pela RE que julgue mais apropriada e, para isto, leve em conta aspectos da situação, da tarefa e da RE, bem como suas preferências pessoais. Porém, aprendizes sem uma boa percepção do problema tendem a simplesmente arriscar escolhendo REs que não os aproximam da solução. Invariavelmente, esta perícia vem a ser mais difícil para iniciantes do que para expertos, porquanto aos primeiros falta uma maior imersão nas tarefas que tentam

solucionar. Em contraste, é uma característica de proficiência saber a correlação entre tarefas e REs apropriadas;

4. Aprendizes provavelmente precisam entender a construção de uma RE apropriada: a situação pode exigir que se construa uma RE ao invés de se interpretar uma RE pronta. Freqüentemente, aprendizes constroem REs imprecisas mas, em contrapartida, podem tirar conclusões corretas a partir delas [21]. Há também evidências de que a criação de REs conduz a um melhor entendimento da situação(-problema). Assim, se os aprendizes estiverem livres para construir REs, o processo de interpretação torna-se mais fácil. Presumidamente, os aprendizes estarão mais familiarizados com a forma da RE e a relação desta com o domínio.

2.1.3 Ambientes Interativos de Aprendizagem (AIAs)

Estudos específicos [21] relataram o emprego de REs em AIAs destinados à resolução de problemas. Constataram que sistemas como **Bridge**, **Geometry** e **Gil** valiam-se de grafos de prova como recurso metacognitivo, tendo-os como meio para tornar o processo de planejamento mais evidente. **Bridge** [12] apud [67] abordava o detalhamento de planos em alusão simplificada a algoritmos; **Geometry** [13] apud [67] remetia-se a provas geométricas no ensino colegial; e **Gil** [45] apud [21] tangia ao ensino da linguagem LISP. Foram também relacionados pelos mesmos estudos ambientes em que o raciocínio com REs era essencial à atividade de ensino suportada. **Hyperproof** e **Algebra (Word Problem Tutor)** são exemplos disso.

Hyperproof [22] destinava-se ao ensino de lógica de primeira-ordem, incorporando o uso simultâneo de REs sentenciais e gráficas (o que na época era recomendado e também descrito pelos seus autores como "raciocínio heterogêneo"). Inovou ao usar um tabuleiro xadrez e blocos de diferentes formatos, tamanhos e posições, junto a alguns mecanismos, para considerar representações indeterminadas (abstrações). Os axiomas e o objetivo a serem provados eram todos moldados em termos dos blocos e a provisão era feita movendo-se informações entre o sistema de provas (sentencial) e o mundo de blocos (gráfico). O

primeiro, uma janela de texto, estabelecia-se na parte inferior da tela e constava de uma lista de predicados. A representação do mundo de blocos, por vez, era feita em três dimensões e ocupava a região acima à anterior. As provas podiam envolver formato, tamanho, posição, identidade ou descrições sentenciais dos objetos, consistindo em determinar uma dada propriedade ou demonstrar que esta não podia ser inferida através das informações disponíveis. Dispunha-se de um conjunto de regras para a construção de provas, sendo algumas delas sentenciais e outras gráficas (no sentido de consultar ou alterar uma situação ilustrada no tabuleiro). Havia também regras verificando propriedades de uma prova em desenvolvimento. Como defendido pelos seus autores, Hyperproof deveria ser visto como um ambiente de verificação de provas, desenvolvido para assistir à prova de teoremas utilizando informações heterogêneas.

Por seu turno, o ambiente Algebra (Word Problem Tutor) [61] tinha por objeto ensinar habilidades básicas de manipulação algébrica. Provia aos aprendizes ferramentas de modelagem matemática, ajuda e *feedback* à medida que eles resolviam problemas sobre uma superfície plana. Diferia muito do Hyperproof no nível em que os aprendizes participavam da construção da RE. Enquanto o Hyperproof permitia interação com o mundo de blocos (e.g. adicionando e movendo objetos, modificando tamanhos, alterando rótulos), o Algebra apresentava um grupo de diagramas pré-elaborados para que o aprendiz pudesse escolher entre eles.

O ambiente **SimForest** [47], que simula o crescimento de árvores em florestas, inovou ao introduzir em AIAs os conceitos de caixa-preta e caixa-de-vidro[5]. Estes, foram implementados em versões distintas do programa. **SimForest-B**, a versão caixa-preta, permitia ao aprendiz alterar os parâmetros utilizados pelo modelo interno de simulação e acompanhar os resultados destas mudanças. Direcionava-se a principiantes que poderiam, por exemplo, aumentar a taxa de crescimento de uma determinada espécie de árvore e observar a simulação dela dominando a floresta. Aos aprendizes experientes dedicou-se a versão caixa-de-vidro **SimForest-G**. Nesta, concede-se acesso aos detalhes internos do modelo usado para a simulação, permitindo tanto inspecionar quanto modificar as

[5]Respectivamente, *black box* e *glass box* em inglês. O termo *glass* aqui denota transparência.

equações que o constituem, bem como propor novas dependências e parâmetros. Ambas as versões têm as funcionalidades assistidas por REs, assim contemplando aprendizes iniciantes e experientes. Além disso ressalta-se que, havendo essas duas versões, evita-se que o programa seja gradualmente abandonado conforme o progresso da aptidão do aprendiz.

O ambiente **SwitchER** [21] voltava-se à resolução de problemas encorajando o uso e a construção de MREs. Apesar de relativamente modesto e pouco repercutido[6], alicerçou um estudo detalhado do uso de MREs na resolução de problemas. A maioria das pesquisas realizadas na época respaldava-se em fundamentos ou dados por ele coletados. Derivam destas, por exemplo, a constatação de que a escolha de uma certa RE tem por implicação o exame minucioso da tarefa pelo aprendiz.

Sucedendo o trabalho, **SwitchERII** [20] foi desenvolvido a partir do antecessor sobredito aliado à observação de REs rascunhadas durante a resolução de problemas. Dispõe de ambientes para construção de REs (textuais, matriciais e gráficas) para que o aprendiz possa escolher e alternar entre elas. Quando se trabalha particularmente com diagramas de Euler, o SwitchERII pode analisar dinamicamente a representação construída e fornecer *feedback* ao aprendiz; pois o ambiente incorpora a representação da semântica de tipo de diagrama. Portanto, poderia gerar *feedback* detalhado diagnosticando precisamente a natureza do erro. Todavia, optou-se por exibir informações menos aprofundadas de modo que induzissem processos metacognitivos, tais como reflexão, do aprendiz. Diante dessa ferramenta, dirigiu-se estudos comparando as performances de interpretação e construção de diagramas. Foram apresentados a dezesseis aprendizes problemas que poderiam ser resolvidos com diagramas de Euler. Seis dos indivíduos cometeram erros na construção da RE, contudo interpretaram corretamente as REs prontas. Em contraste, quatro aprendizes erraram na interpretação das REs, porém não cometeram erros na construção. Inferiu-se que, se os aprendizes fossem livres para construirem REs que julgassem mais apropriadas (ou até preferidas), o processo de interpretar REs construídas seria facilitado. Desta forma, presumidamente, os aprendizes estariam mais familiarizados com a forma da representação e o relacionamento com o domínio.

[6]Se comparado com seu sucessor, SwitchERII.

Calques3D [42], um ambiente micromundo concebido para o ensino de geometria dinâmica, apóia-se em MREs para apresentar uma mesma informação sob perspectivas diferentes. Uma figura geométrica criada pode ser exibida através de REs como: desenho em duas ou três dimensões, listagem das figuras componentes, descrição declarativa (textual), descrição algébrica de elementos componentes e grafo de dependência estrutural. A abordagem adotada para a criação de Calques3D defende a priorização de questões cognitivas em detrimento das pedagógicas

Considerando as evidências supracitadas, constata-se que desenvolver AIAs multirrepresentacionais eficientes não é tarefa simples, pois os aprendizes devem se beneficiar das vantagens de MREs sem serem sobrecarregados com os custos correlatos. Para tanto, um grupo pesquisadores propôs um arcabouço conceitual para a construção destes sistemas. **DeFT** (*Design*, *Functions*, *Tasks*)[7] [3] trata-se de um *framework* para ensino com MREs, produto de pesquisas em aprendizado, ciência cognitiva das representações e teorias construtivistas de educação. Basicamente, determina que a efetividade de MREs pode ser melhor compreendida quando considerados três aspectos fundamentais do aprendizado:

1. As funções que MREs desempenham no suporte ao aprendizado (exposto anteriormente na seção 2.1.1.1);

2. As tarefas metacognitivas que precisam ser assumidas (e cumpridas) pelo aprendiz que interage com MREs (abordado na seção 2.1.2.2); e

3. Os parâmetros de projeto que são exclusivos do aprendizado com MREs.

Como os dois primeiros itens foram oportunamente descritos, a seguir cabe somente a explanação do último.

2.1.3.1 Parâmetros de Projeto

Ambientes diferem pelo conteúdo explorado, pelos aprendizes-alvo e mesmo pelas técnicas de comunicar o primeiro aos segundos. Portanto, geralmente há razões específicas fundamentando a escolha de uma determinada RE. Para isso, deve-se levar em conta a

[7] Projeto, Funções, Tarefas.

efetividade dessa RE, considerando tanto as informações por ela fornecidas (mundo representado) quanto a forma com a qual se apresenta isto (mundo representante). Não obstante, ressalta-se que optar por (ou projetar) REs eficazes demanda um esforço substancial no desenvolvimento de AIAs.

Em vista disso, há um conjunto de dimensões de projeto unicamente aplicáveis a ambientes multirrepresentacionais, a saber:

1. Número: ambientes são multirrepresentacionais por empregarem, ao menos, duas representações. Comumente disponibilizam um conjunto maior, para as REs serem utilizadas simultaneamente ou em situações predefinidas de interação. Em contrapartida, um número e variação muito grandes de REs raramente beneficia o aprendizado;

2. Informação: ambientes multirrepresentacionais podem permitir flexibilidade quanto à maneira com que as informações são distribuídas entre as representações. Isto implica em duas outras características contrastantes: na complexidade das informações em cada RE e na redundância de informações entre REs. No primeiro extremo, há exclusividade no conteúdo apresentado por cada RE e inexiste redundância no conjunto destas (pois referem-se a diferentes mundos representados). Simplifica-se individualmente as representações ao custo de requerer-se outras adicionais. Cabe aos aprendizes, integrar informações de múltiplas fontes. Os sistemas então podem ser parcialmente redundantes, permitindo que partes das informações sejam constantes ao longo de algumas REs, até que se atinja o extremo oposto. Neste, cada RE expressa as mesmas informações de modo que a diferença resida somente nos respectivos mundos representantes. Conseqüentemente, o sistema torna-se repleto de redundância e, em geral, constituído de REs mais complexas;

3. Forma: considerar cada RE de um ambiente de maneira isolada demonstra-se insuficiente, visto que se despreza a interação que há entre as MREs e que constitui um sistema representacional. Logo percebe-se que a forma pode se referir a muitos diferentes aspectos de representação, por exemplo: sistemas heterogêneos, que com-

binam textos e imagens; multissensoriais, que acrescem narração verbal a conteúdos não-sonoros; estáticos e dinâmicos; 2D, 3D e híbridos, entre outros. Portanto, ainda que se desconheça como a forma de um sistema representacional influencia o aprendizado, tem-se um vasto espaço de pesquisa a ser explorado;

4. A seqüência de representações: trata das questões que vêm à tona quando não se exibe as REs simultaneamente. Primeiro, deve-se predeterminar a seqüência em que as REs serão apresentadas ou construídas. Depois, o ambiente (ou mesmo o aprendiz) precisa decidir em que ponto adicionar uma nova RE ou alternar entre as existentes;

5. Suporte de tradução entre representações: os ambientes dispõem de uma ampla variedade de maneiras para indicar aos aprendizes a relação entre REs. Nisto residem duas questões substanciais: o suporte ativo desempenhado pelo ambiente em auxiliar o aprendiz; e se esta assistência é provida em nível sintático ou semântico (também respectivamente ditos superficial e aprofundado).

2.1.4 Domínio de Programação de Computadores

Muitos sistemas de ensino foram implementados para o domínio de programação de computadores [29]. Tratava-se de uma direção bastante lógica a se seguir, haja vista que a maioria dos pesquisadores tinha por especialidade a programação.

Bridge [12] apud [67], anteriormente citado, propunha-se ensinar principiantes de programação a decomporem planos maiores em passos intermediários detalhados. Baseava-se na presunção de que iniciantes raciocinam primeiramente em linguagem natural, devendo então desenvolver novos modelos mentais que vão gradativamente ao encontro do formalismo de uma linguagem de programação. As atividades que propunha consistiam em chegar a fases menores no desenvolvimento de planos e depois articular cada uma destas como feito em uma linguagem de programação. Abordava a linguagem natural através de um conjunto de "linguagens de programação informais" que associava a cada fase um número de expressões típicas relacionadas. Cabia ao aprendiz compor sentenças mediante

a seleção de expressões (como “adicione”, “conte”, “continue os passos” e “assim por diante”) em um *menu*. No estágio final de resolução, o aprendiz articulava os estágios menores (em RE, encaixados como peças de quebra-cabeça) de modo a comporem o plano objetivado. Até onde se soube, Bridge necessitava de uma avaliação mais prolongada para verificar acerca do caráter limitativo, podendo forçar a um processo de planejamento que não fosse natural aos usuários.

Outro sistema de notoriedade foi **LAURA** [67], que teve destaque por utilizar-se de soluções de referência (algoritmos) para diagnosticar os algoritmos do aprendiz. Basicamente, LAURA transformava ambas as soluções em grafos de fluxo de controle e tentava casar (mediante transformações) o grafo do aluno com o de referência. Contudo, o processo de casamento de padrões era bastante complexo e sofisticado, uma vez que concedia ao sistema a capacidade de aceitar algoritmos corretos embora não idênticos à solução de referência. Cita-se isso como vantagem, porquanto não se induz o aprendiz a restringir sua criatividade em prol do caminho especificado pela solução prevista. No entanto, salienta-se que LAURA tem pouca capacidade de resposta, sendo apenas capaz de diagnosticar erros de baixo nível de abstração. Além disso, os grafos eram internamente empregados, inexistindo abordagem por outra RE que não o algoritmo.

Muitos sistemas consideráveis [29, 67] foram desenvolvidos, entretanto, assim como LAURA, tinham outros enfoques que não RE: **Mycroft**, sistema para detectar e reparar erros em programas projetados para traçar desenhos em Logo; **Aurac**, depurador de programas codificados na linguagem Solo; **Pudsy** e **Proust**, sistemas de diagnóstico para algoritmos em Pascal; **Malt**, para o ensino de linguagem-de-máquina; **Greaterp**, tutor para a linguagem LISP; e **Spade**, para Logo.

Mesmo com esse progresso, houve um período em que o campo de sistemas de ensino de programação permaneceu estagnado. Uma hipótese cabível era que, inicialmente, tinha-se bastante facilidade por se dispor de especialistas no domínio ensinado entre os pesquisadores. Com a evolução dos estudos, percebeu-se que muito dos sistemas se devia aos especialistas em ciências cognitivas sendo, portanto, o domínio de programação tão difícil de ser ensinado quanto qualquer outro.

Nos últimos anos, estudos no âmbito de programação de computadores têm emergido, fruto de esforços de pesquisadores bastante renomados. Porém, dada a demanda, destinam-se majoritariamente ao paradigma Orientado a Objetos. Vem também ocorrendo a inserção de REs como recurso de depuração em ambientes não-educacionais. Evidência disso são os termos "visualização de algoritmos", "visualização de programas" e "visualização de *software*", todos análogos[8] e referentes à re-representação gráfica de aspectos do código textual. Há também a chamada "programação visual"[9] que diz respeito à implementação de programas (não somente da interface) em duas (ou mais) formas dimensionais [48].

No que corresponde à programação visual utilizando variações de fluxograma, as linguagens **Pictorius Prograph** e **National Instruments LabVIEW**, conquistaram notoriedade. Ambas são proprietárias, empregam REs patenteadas, não se concentram no paradigma Imperativista[10] e não têm enfoque educacional.

Em estudo recente [57], foram avaliadas as representações externas nos ambientes de programação mais populares. Abrangeu-se os ambientes de desenvolvimento: **Visual J++** (Microsoft), **JBuilder** (Borland), **Visual Age** e **Code Warrior** (ambos da IBM); as ferramentas para visualização de código: **JaViz**, **Jinsight** (ambas da IBM), **VisiComp** e **CodeVizor**; os ambientes de aprendizagem **BlueJ** e **Cocoa**; e a linguagem visual de programação **Prograph**. Levantou-se como importante problema potencial a dificuldade de coordenar REs adicionais ao código. Também frisou-se a incerteza de quando as REs adicionais deveriam ser redundantes (evidenciando informações comuns às REs principais) e quando deveriam ser complementares (salientando diferentes informações).

No escopo educacional, com âmbito pouco mais próximo àquele tratado por esta dissertação, encontrou-se os ambientes **AMBAP** e **devFlowCharter**. O AMBAP (Ambiente de Aprendizado de Programação)[11], desenvolvido pela Universidade Federal de Alagoas, consiste basicamente de um editor textual de algoritmos associado a opções de execução e monitoramento. Os algoritmos são desenvolvidos baseados na sintaxe do ILA (Interpreta-

[8] *Algorithm/Program/Software Visualization.*

[9] *Visual Programming* (*VP*).

[10] Por questões de fidelidade de tradução, o termo "Imperativo", utilizado e difundido para o designar o respectivo paradigma, foi substituído pelo termo "Imperativista".

[11] Disponível em http://www.ufal.br/tci/ambap/.

dor de Linguagem Algoritmica) [30], uma forma de português estruturado caracterizada pela simplicidade. Permite-se a execução direta ou em modo passo-a-passo, sendo possível, nesta última, a visualização tanto dos valores correntes das variáveis declaradas quanto da chamada de subprogramas (aqui, funções).

O devFlowCharter[12], trata-se de um ambiente para a criação de algoritmos em fluxograma. Embora admita a geração de código (Pascal), faz isto de forma alheia ao fluxograma, não demarcando correspondências entre as REs. Ademais, possui interface pouco intuitiva e só recentemente começou a permitir que elementos fossem excluídos.

Por fim, diante da resenha neste tópico apresentada, salienta-se a ausência de pesquisas que contemplem o paradigma de programação Imperativista através de MREs numa abordagem metacognitiva. Isto, portanto, sinaliza parte da contribuição do trabalho aqui apresentado.

2.2 Sistemas Tutores Inteligentes e Ambientes Interativos de Aprendizagem

Os tradicionais sistemas de **Instrução Assistida por Computador**[13] (IAC) [67] eram recipientes organizados de forma estática, estruturados para incorporar o conhecimento tanto do domínio quanto pedagógico de professores especialistas. Seguindo a linha evolutiva, esses sistemas passaram a agregar técnicas da Inteligência Artificial[14] (IA) e a adotar outras abordagens para o ensino. Por algum tempo, essas pesquisas foram conduzidas sob a denominação de **Instrução Inteligente Assistida por Computador**[15] (IIAC), coexistindo com a então designação de **Sistemas Tutores Inteligentes**[16] (STIs). Contudo, esta última foi ganhando a preferência dos pesquisadores em detrimento da primeira, pois os STIs representam, sob diversos aspectos, uma mudança de metodologia muito maior do que a simples adição do "I" em IAC.

[12] Disponível em sourceforge.net/projects/devflowcharter.
[13] *Computer-Aided Instruction* (*CAI*).
[14] *Artificial Intelligence* (*AI*).
[15] *Intelligent Computer-Aided Instruction* (*ICAI*).
[16] *Intelligent Tutoring Systems* (*ITS*).

Portanto, STIs são programas de computador projetados para incorporar técnicas de IA de modo a prover tutores que saibam como ensinar, o que ensinar e a quem ensinar. Para isso, há intersecção das áreas de ciência da computação, psicologia cognitiva e pesquisa educacional; sendo este novo campo normalmente denominado Ciência Cognitiva.

Embora não haja arquitetura única para construção de STIs, uma abordagem bastante clássica compõe-se dos seguintes módulos:

- **Modelo do Domínio (ou do Especialista):** representação por meio de objetos, regras ou procedimentos, que codifica explicitamente o conhecimento do especialista;
- **Modelo do Aprendiz:** representa a proficiência do aprendiz no conteúdo tutorado (deduzido com base no Modelo do Domínio) e, adicionalmente, as preferências do aprendiz relativas ao processo de aquisição do conhecimento;
- **Modelo (Didático-)Pedagógico:** abrange as estratégias de ensino que, interagindo com os modelos anteriores, permite que a apresentação do conteúdo seja adaptada às preferências e ao progresso do aprendiz;
- **Módulo Interface:** basicamente, provê a dinâmica de troca de informações (interação) entre o aprendiz e o sistema.

Outras arquiteturas são dirigidas por diferentes enfoques, podendo incluir elementos ou mesmo decompor e/ou suprimir os módulos descritos. Apenas como exemplo, há arquiteturas que contém catálogo de erros prováveis do aprendiz para que, quando este cometer um erro durante a resolução de problemas, o STI possa avaliar o desvio do curso de aprendizado implicado.

Um **Ambiente Interativo de Aprendizagem**[17] (AIA) típico distingue-se de um STI por ser de caráter mais passivo, não interferindo/interagindo (pró-)ativamente com o aprendiz, embora seja semanticamente rico. Parte de atividades como exploração, investigação e descoberta para que o aprendiz construa individualmente seu conhecimento.

[17] *Interactive Learning Environment* (*ILE*).

Como caso bastante característico de AIAs, merecem destaque os chamados micromundos[18] ou microcosmos. São ambientes com riqueza semântica que atuam como instrumento de abstração para representar situações reais. Os elementos disponibilizados por um micromundo agem, portanto, como um conjunto metafórico dentro das situações representadas.

Apesar do exposto, salienta-se que a fronteira entre AIAs e STIs, além de tênue, é pouco definida. Evidência disso são os Ambientes de Descoberta-Guiada[19] (ADGs).

Outrossim, pode-se encarar o atributo "inteligência" (até então próprio dos STIs) como decorrente de um alto grau de interatividade provido por determinado ambiente. Neste panorama, um AIA seria inteligente à devida proporção que dispusesse de uma escala variada de oportunidades de interação (ou exploração) ao aprendiz. Tal ambiente, no dado contexto, tornaria ainda mais sutil e intricada a demarcação entre STIs e AIAs.

Diante do que se apresentou, sobretudo, ressalta-se a inexistência de *software* educacional, seja STI ou AIA, no âmbito de programação de computadores que contemple a perspectiva nesta dissertação apresentada. Por conseguinte, demarca-se parte da contribuição requerida para o corrente trabalho. Frisa-se, no entanto, que o foco da pesquisa e desenvolvimento desta dissertação não foi o de construir um STI, mas sim um AIA (mais especificamente, um micromundo).

[18] Do inglês *microworlds*.
[19] *Guided-Discovery Environments* (*GDEs*).

CAPÍTULO 3

FORMALISMOS ADOTADOS NA SOLUÇÃO DO PROBLEMA

A este capítulo compete a formalização do arcabouço conceitual adotado no presente projeto. Levou-se em consideração diversos aspectos, positivos e negativos, referentes aos trabalhos correlatos. Dentre tais aspectos, houve relevante embasamento nas concepções ligadas a micromundos e múltiplas representações externas, ambas apresentadas na seqüência.

Os exemplos citados no decorrer deste capítulo são apresentados na íntegra ao final da dissertação (anexos A, B e C).

3.1 O Potencial de Micromundos

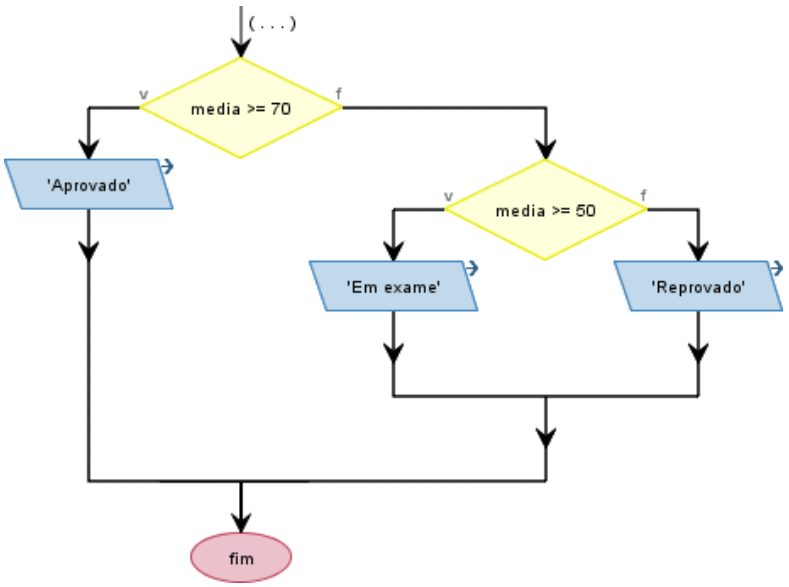
Conforme explicitado, um micromundo consiste num ambiente semanticamente rico porquanto oferece uma escala variada de recursos de interação. Estes, constituem ferramentas de auto-estudo que assistem à construção de conhecimento por parte do aprendiz. Nesse sentido, o aprendiz deve ser visto como um agente ativo na interação, ao passo que o micromundo, um agente passivo.

Convém ressaltar que um micromundo voltado à aprendizagem de programação difere de um depurador convencional. Embora este último possa ter utilidade em cursos introdutórios, um micromundo merece destaque por focar representações de mais alto nível, bem como a possível correspondência e sincronização entre tais representações, cujo carater é exclusivamente educacional.

3.1.1 A Finalidade de Fluxogramas

Baseou-se a escolha da notação de fluxograma, como correspondente ao algoritmo textual, na notável necessidade do aprendiz desenvolver conceitos de mais baixo nível que precedem a noção de escopos aninhados das linguagens de alto nível de abstração. Entre estes conceitos, sobressai a compreensão do funcionamento de desvios progressivos e regressivos, mais explícitos no fluxograma.

O trecho de algoritmo apresentado pela figura 3.1 expõe um caso de desvio progressivo. A instrução de saída que será executada depende exclusivamente do valor assumido pela variável *media*. Supondo, então, que lhe seja atribuído *100*, no fluxograma fica evidente o passo dado até a impressão da mensagem e, posteriormente, a progressão ao fim do algoritmo. Porém, num código em linguagem de alto nível, este desvio pode não se mostrar transparente ao aprendiz.

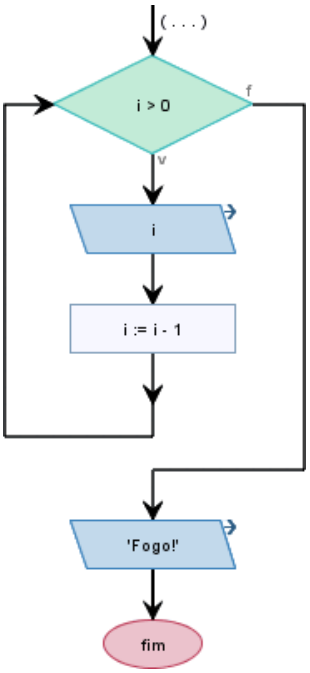



```
1  //...
2    if (media >= 70) then
3       writeln('Aprovado')
4    else
5       if (media >= 50) then
6          writeln('Em exame')
7       else
8          writeln('Reprovado');
9  end.
```

Figura 3.1: Desvio progressivo em um trecho de algoritmo

De forma análoga, o trecho de algoritmo expresso pela figura 3.2 demonstra uma ocorrência de desvio regressivo. Independentemente de qualquer outra circunstância, se o fluxo de controle está na atribuição $i := i + 1$, haverá em seguida um desvio regressivo para a avaliação do condicional da estrutura *enquanto-faça*. No fluxograma, este desvio sucede de forma bastante natural. Numa linguagem de alto nível, contudo, isto não se

manifesta, exigindo conhecimento prévio.




```
1  //...
2    while (i > 0) do
3       begin
4          writeln(i);
5          i := i - 1;
6       end;
7    writeln('Fogo!');
8  end.
```

Figura 3.2: Desvio regressivo em um trecho de algoritmo

Diante disso, acredita-se que o entendimento do fluxo de controle em instruções isoladas se faz muito necessário para consolidar a aquisição de princípios de programação de computadores. Ademais, tal conhecimento não apenas orientará o aprendiz para a aquisição de perícia no referido campo, como também implicará positivamente no decorrer desta atividade.

3.1.2 A Finalidade de Algoritmos em Pascal

A intenção de correlacionar o fluxograma com o algoritmo em Pascal adveio, primeiramente, da necessidade de se remeter o aprendizado a uma situação concreta de programação de computadores. Elegeu-se Pascal por consistir em uma linguagem de alto nível que tem se mostrado adequada ao ambiente acadêmico[1]. Dentre outras características que impactaram na escolha desta linguagem, cita-se a correspondência ao paradigma tratado (Imperativista), alta difusão de códigos escritos e variedade de compiladores[2].

[1]Sendo utilizada em ambas as instituições que o proponente tem contato direto: Universidade Federal do Paraná (UFPR) e Universidade Estadual do Centro-Oeste (UNICENTRO).

[2]Havendo compiladores em versões livres para várias plataformas.

Além disso, Pascal particulariza a idéia de programação estruturada ao redor do conceito de embutimento de escopos (ou contextos). Neste, um bloco de código contido em uma instrução (e.g. estrutura condicional *if*) corresponde a um subcontexto daquele mais geral que abriga a instrução recipiente (contexto).

Desta forma, sobressai o fato de que a seqüência de processamento das instruções de um bloco ocorre sem saltos. Isso não diz respeito às instruções alocadas em subcontextos, que por sua vez podem ser desviadas por eventuais comandos condicionais. Assim, o final lógico da execução coincide com o final sintático do texto.

O trecho de algoritmo exibido na figura 3.3 exemplifica o embutimento de contextos. Neste código, as instruções *writeln(i)* e *i := i - 1* estão contidas em um subcontexto delimitado pelas palavras reservadas *begin* e *end* que correspondem à estrutura *while-do*. Esta, por sua vez, encontra-se em um contexto maior, o do algoritmo, delimitado por *begin* e *end.* (este, com ponto final).

```
1  program contagem_regressiva;
2  var
3    i : integer;
4  begin
5    i := 10;
6    writeln('Iniciando contagem regressiva...');
7    while (i > 0) do
8       begin
9          writeln(i);
10         i := i - 1;
11      end;
12   writeln('Fogo!');
13 end.
```

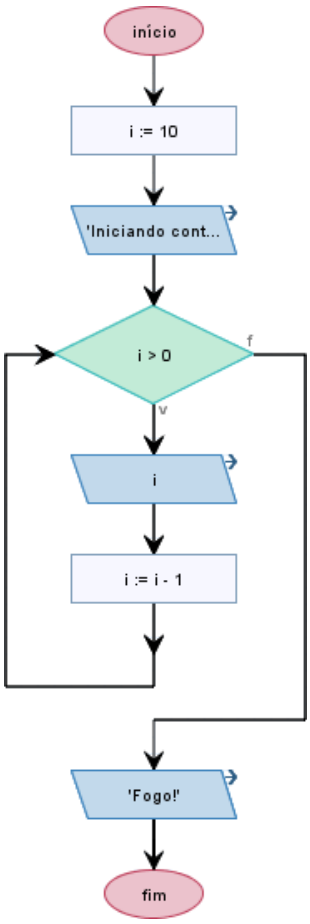


Figura 3.3: Subcontexto em um trecho de algoritmo

Convém fazer aqui uma digressão para relatar que o próprio conceito de programação estruturada proveio, em parte, como reação à má legibilidade causada pelas instruções de controle [60]. O cenário precedente destacava-se pelo uso indiscriminado de instruções *go to*, que permitiam ao fluxo de controle saltar de uma instrução a outra não-adjacente, implicando em redução crítica na legibilidade do código, como também na perturbação do fluxo de leitura do algoritmo como um todo. Como conseqüências típicas, havia a diminuição da capacidade de escrita (visto que fazê-lo exige uma releitura freqüente do que foi redigido) e o aumento do custo de manutenção dos sistemas (relacionado com a legibilidade).

Isto posto, em favor da prática de programação estruturada e como característica evolutiva, muitas das linguagens atuais sequer implementam instrução equivalente ao *go to*. Portanto, vem a ser de grande importância que o aprendiz tenha pleno domínio do conceito de embutimento de escopo.

Por fim, diante do apresentado, verifica-se a necessidade de melhor detalhar o conjunto de MREs definido. Faz-se isso nas seções seguintes, contando também com a criação de controles de correspondência entre as funções didático-pedagógicas das representações envolvidas.

3.2 Conjunto Multirrepresentacional Envolvido

Cabe a esta seção introduzir o conjunto multirrepresentacional definido como parte da solução neste trabalho proposta. Além disso, será apresentada a correspondência entre alguns dos recursos semânticos de relacionamento entre as REs e as funções propostas na taxonomia funcionalista de Ainsworth [2] (descrita anteriormente na seção 2.1.1.1).

3.2.1 Categorização de REs

Foram estudadas e definidas (ou mesmo criadas), como parte das soluções da presente dissertação, várias categorias de RE. Houve cautela no desempenho de tal atividade porquanto esta remete às prescrições delineadas pelo trabalho de Ainsworth [2, 3], cujas

referências foram essenciais na corrente pesquisa. Diante disso, como parâmetro de projeto, para que se disponha de um conjunto de MREs bem-definido, tem-se como tarefa indispensável precisar dimensões como número, informação, forma, seqüência e tradução das REs componentes.

Assim, relaciona-se nesta seção o elenco das categorias de REs empregadas no protótipo, associadas ao devido valor metacognitivo. Embora se tenha consolidado estas categorias, leva-se em consideração que, com o recolhimento de resultados, estas sejam aperfeiçoadas em alguns aspectos, assim como outras mais possam vir a compor o conjunto. Frisa-se que as representações de fluxograma e o código em Pascal foram abordados na seção anterior.

3.2.1.1 Cliques de Correspondência

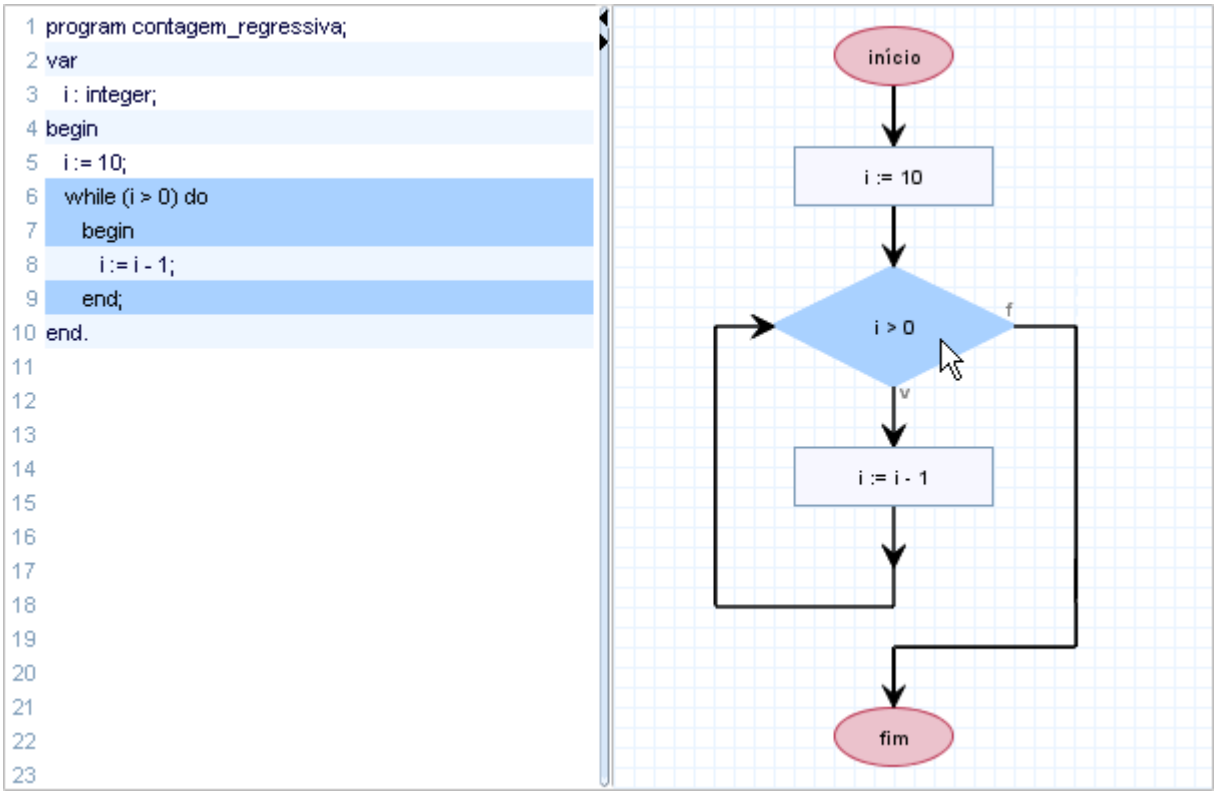


Figura 3.4: Clique de correspondência

Mediante um clique em determinado elemento do fluxograma (figura 3.4), destaca-se automaticamente a linha de código Pascal equivalente. A situação inversa também ocorre.

Esta RE tem suma importância para se relacionar o algoritmo na notação de fluxograma com a textual correlata.

3.2.1.2 Realce de Elementos

Durante a execução do algoritmo, são realçados o elemento de fluxograma e a linha de código onde se encontra o fluxo controle. Intenciona-se que isto ajude o aprendiz a assimilar a correspondência entre as notações, bem como a entender o funcionamento das estruturas empregadas (condicionais e de repetição).

Difere-se do Clique de Correspondência por ter caráter dinâmico, ou seja, o destaque é feito automaticamente de acordo com a progressão do fluxo de controle.

3.2.1.3 Recolhimento e Expansão de Blocos de Elementos

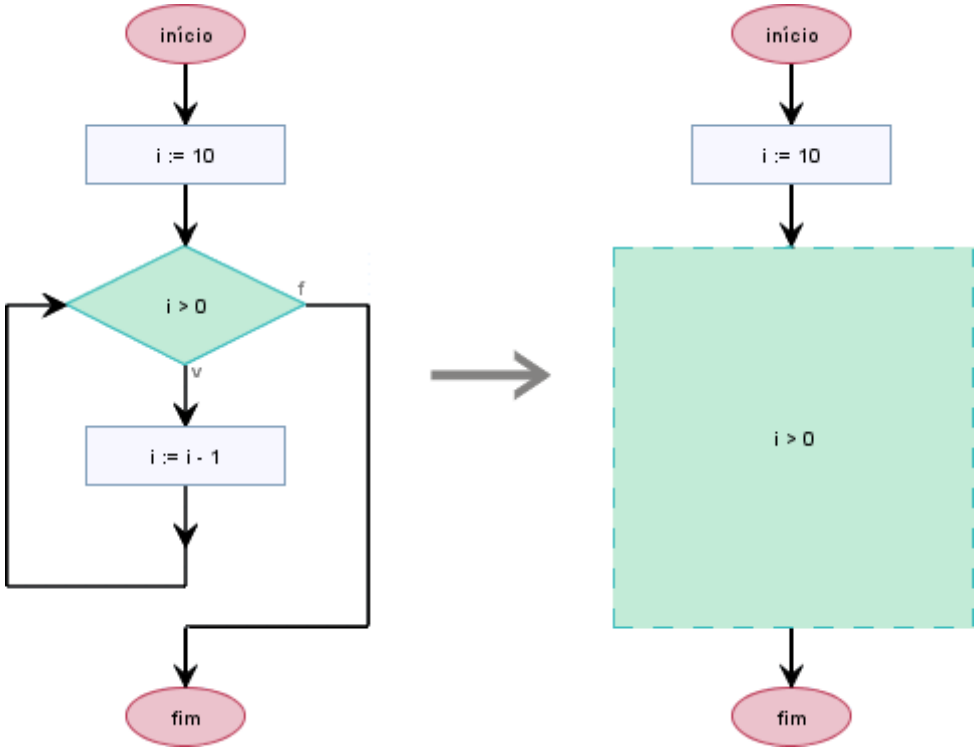


Figura 3.5: Recolhimento e expansão de blocos de elementos

Há situações em que se tem algumas estruturas condicionais e de repetição aninhadas e deseja-se limitar a visualização, por exemplo, à estrutura mais externa (que contém

todas as outras). Esta representação (figura 3.5) permite encapsular visualmente o que está contido dentro da estrutura escolhida, transfigurando-a numa espécie de "caixa-preta" (pois não se terá acesso ao que se passa dentro dela). Desta forma, pretende-se focar a atenção do aprendiz ao que lhe convém sem que ele se atenha aos elementos e ao fluxo que ocorre dentro da estrutura recolhida.

Conforme algumas observações, aponta-se melhorias à implementação dessa RE no protótipo. A mais substancial destas consiste em recolher (e expandir) não somente o elemento no algoritmo gráfico, mas encontrar maneiras condizentes de fazê-lo também no textual. Uma solução que se antecipa consiste no emprego de reticências e/ou sinais de adição e subtração ao lado do início das estruturas, no algoritmo textual, indicando as possibilidades de expansão e recolhimento. Outro aperfeiçoamento possível seria o redimensionamento de elementos recolhidos a um tamanho menor (padronizado e não proporcional).

3.2.1.4 Cores

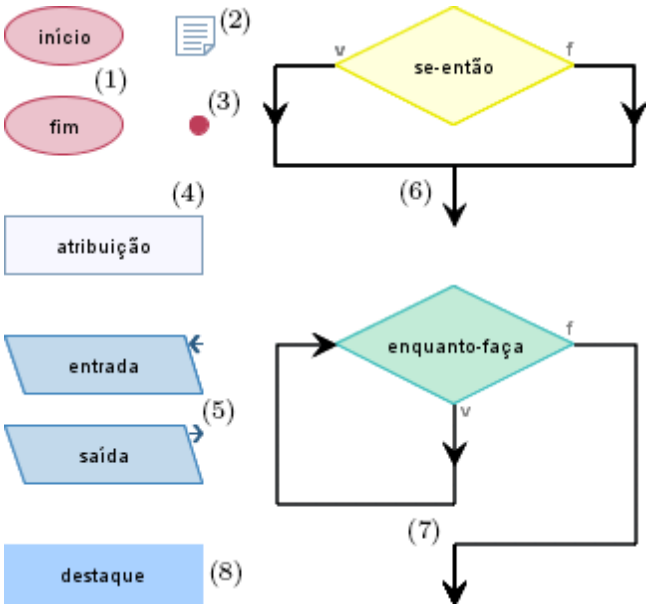


Figura 3.6: Cores

Investigou-se cores que atuassem como elo indireto entre o algoritmo textual e o representado pelo fluxograma. Considerou-se a associação de cores a categorias de elementos

representados, com a intenção de que o aprendiz fosse induzido a classificar e agrupar elementos semanticamente próximos. Além disso, preferiu-se cores brandas por, ao que se acredita, causarem uma menor exaustão visual ao aprendiz.

Conforme a figura 3.6, a seguinte relação foi estabelecida: (1) demarcadores de início e fim, róseo; (2) comentário textual, branco com contorno azul; (3) ponto de interrupção, vermelho escuro; (4) atribuição, variante de cinza; (5) entrada e saída, azul; (6) estruturas condicionais, amarelo; (7) estruturas de repetição, verde; e (8) qualquer elemento selecionado, azul claro.

Este último, a cor de destaque, necessitou de mudanças quanto à concepção inicial. Primeiramente, seria utilizado como realce para cada elemento, o complemento da cor original no padrão RGB[3] (efeito de "negativo"). Com isto, as instruções de entrada e saída, por exemplo, adquiririam um tom laranja escuro, quase marrom. Todavia, além das cores resultantes serem discrepantes à interface, requeriam uma maior carga cognitiva por parte do aprendiz. Como são 7 categorias de elementos, com o destaque feito desta maneira, haveria 14 cores para elementos de fluxograma. Evitando tal situação, optou-se pelo azul claro, semelhante àquele das canetas marca-textos, como cor padronizada para todos os destaques. Dessa forma há somente 8 cores para elementos do fluxograma.

3.2.1.5 Comentários Textuais

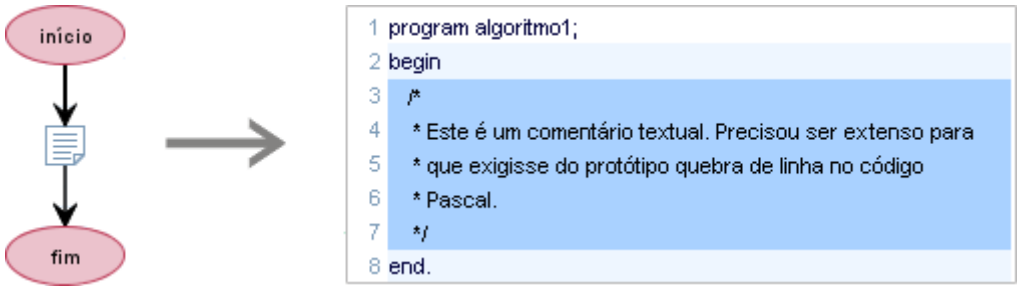


Figura 3.7: Comentários

São uma prática comum, tratada pela sintaxe das maioria das linguagens, que possibilita ao aprendiz inserir texto a fim de comentar linhas de código (aqui também elementos

[3]Do inglês *Red, Green, Blue* (Vermelho, Verde, Azul).

do fluxograma). Foram incorporados ao algoritmo gráfico, sendo representados por ícones (figura 3.7), inseridos em arcos, que podem ser expandidos para exibirem (ou permitirem edição) dos comentários digitados. Estes, juntamente com os demais elementos do fluxograma, são transpostos para o código em Pascal.

3.2.1.6 Pontos de Interrupção (*Breakpoints*)

Presente na generalidade dos depuradores, consiste aqui em permitir ao aprendiz apontar previamente elementos do fluxograma (ou de código) para que a execução do algoritmo seja interrompida quando o fluxo de controle atingir estes elementos demarcados. Têm grande utilidade porque o aprendiz pode verificar, entre outras coisas, se há blocos inatingíveis do algoritmo, inspecionar o valor de variáveis em determinados trechos e mesmo perceber erros semânticos bastante difíceis de serem detectados por compiladores (e.g. *loops* infinitos).

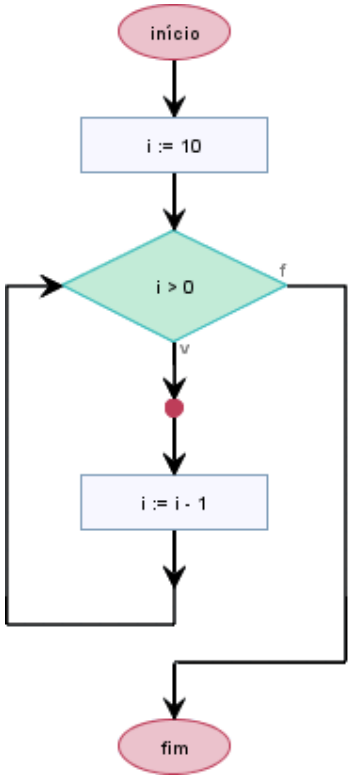


Figura 3.8: Ponto de interrupção precedendo uma atribuição

Inicialmente, esta representação perfazia-se de um ícone que era adicionado ao lado

do elemento de fluxograma demarcado. Contudo, isto implicava em ambigüidade, pois não ficava explícito se a parada ocorreria antes ou depois da execução daquele elemento demarcado. Levando isso em conta, alterou-se o ponto de interrupção de modo que fosse associado a um arco e não a um elemento particular (figura 3.8). Desta forma, transparece que, se a parada precede uma dada instrução, o fluxo de controle será interrompido antes que esta última seja alcançada. Convém lembrar que o ponto de interrupção não possui equivalente sintático no algoritmo textual, portanto fica somente expresso no fluxograma.

3.2.1.7 Destaque de Variáveis com Valores Alterados

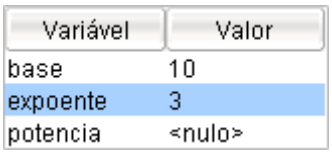


Figura 3.9: Destaque de variáveis com valores alterados

Cresce, juntamente com o número de variáveis empregadas, a dificuldade do aprendiz em gerenciá-las. Quando poucas variáveis são utilizadas, fica fácil atentar a quando cada uma delas tem o respectivo valor alterado. Porém, dependendo da quantidade, tais ocorrências tornam-se difíceis de ser percebidas. Cabe a esta RE evidenciar, durante a execução, a variável que teve valor alterado num dado instante. Faz-se isso através de uma tabela constando os nomes das variáveis seguidos dos valores (figura 3.9), sendo sempre destacada a variável que sofreu a última atribuição. Variáveis apenas declaradas são demarcadas com *<nulo>*.

Cogita-se também, como melhoria, a representação destas informações como Teste-de-Mesa, em que as variáveis são representadas por colunas de uma tabela cujas linhas são os valores em determinados instantes. Porém, quando implementada, provavelmente deva ser exibida mediante alternância com a área ocupada pelo algoritmo textual. Ou seja, opcionalmente pode-se monitorar as variáveis através da tabela Teste-de-Mesa ao invés de se observar a correspondência dinâmica do algoritmo em fluxograma com o textual.

3.2.2 Correspondência com as Funções de MREs

Compete, neste ponto, uma tentativa de remeter à taxonomia funcionalista proposta por Ainsworth [2] alguns dos recursos semânticos das REs que compõem o conjunto anteriormente categorizado. Faz-se conveniente esclarecer que não se trata de uma classificação excludente. Portanto, uma mesma representação pode desempenhar mais de um papel (ou função) em situações de aprendizado.

3.2.2.1 Construção de Compreensão Aprofundada

A função que mais notoriamente embasa este trabalho de mestrado classifica-se como **abstração** [2], visto que a RE de fluxogramas apresenta mais baixo nível abstrato do que o código em Pascal. Com isso, ao passo que o aprendiz desenvolva habilidades, mesmo que parciais, de construção de algoritmos em fluxogramas, o micromundo (protótipo) o subsidiará na aquisição de conhecimento sobre a tarefa correlata numa linguagem de alto nível (aqui, Pascal). Para isto, o aprendiz contará com uma série de instrumentos, como a geração automática do algoritmo textual equivalente e a visualização de aspectos dinâmicos (e também estáticos) de correspondência entre estas duas REs[4]. Desta forma, a compreensão de uma linguagem de alto nível a partir de uma outra de mais baixo pode ser considerado um importante exercício destinado à expansão da capacidade de abstração apoiado pelo micromundo.

Como exemplo (figura 3.10), pode-se relembrar a ocorrência de um desvio regressivo em um trecho de algoritmo. Uma linguagem de alto nível não explicita a noção de que o bloco de instruções contido na estrutura iterativa de pré-condição consiste em um subcontexto ciclado. Conseqüentemente, a percepção e entendimento do desvio regressivo, por parte do aprendiz, ficam condicionados à compreensão semântica daquela instrução em particular. A notação de fluxograma equivalente deverá auxiliar o aprendiz porquanto representa os desvios através de arcos direcionados. Potencialmente, a visualização de um arco retornando à avaliação condicional da estrutura iterativa, colaborará por elucidar esta

[4]Realce de elementos e cliques de correspondência.

importante característica ao aprendiz.

```
1  program contagem_regressiva;
2  var
3    i : integer;
4  begin
5    i := 10;
6    while (i > 0) do
7       begin
8          writeln(i);
9          i := i - 1;
10      end;
11 end.
```

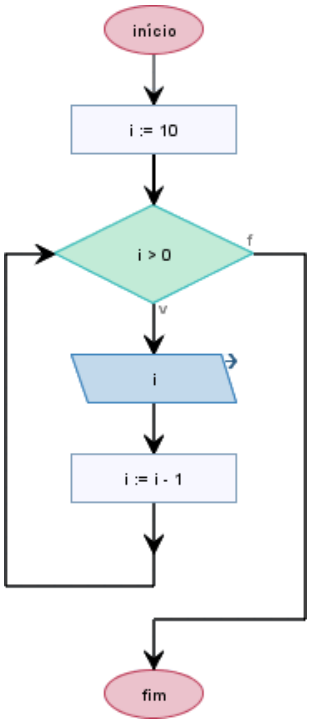


Figura 3.10: Abstração em um subcontexto ciclado

As outras funções que tangem à construção de compreensão aprofundada também ocorrem dentro do conjunto de MREs estabelecido, porém não desempenham papel tão importante quanto a abstração. A **relação** [2] entre o algoritmo gráfico e o textual, por exemplo, fica corroborada também pelos instrumentos associativos recém descritos[5]. A **extensão** [2], menos evidente, ocorrerá fora do micromundo, quando o aprendiz começará a escrever linhas de código Pascal paralelizando mentalmente esta tarefa com o conhecimento prévio da construção de algoritmos gráficos.

3.2.2.2 Restrição de Interpretação

Nesta classe, a função que sobressai denomina-se **restrição por propriedades herdadas** [2]. Neste sentido, mesmo na ocorrência do pior caso, ou seja, quando o aprendiz não tem familiaridade com nenhuma das duas formas de representação de um algoritmo, a mais específica destas (notação de fluxograma) deverá ajudar a restringir as ambigüidades daquela mais genérica (texto em Pascal).

[5]Geração automática de código e correspondência entre as notações

Assim, considerando o exemplo anterior (figura 3.10), existem dúvidas típicas quanto ao último ciclo da repetição. Havendo a atribuição $i := 10$ e o condicional $i > 0$, pode parecer ambíguo, ao aprendiz, se a seqüência impressa finaliza com o termo 1 ou com 0. Neste caso, não somente o fluxograma restringiria a interpretação do código Pascal, como também ambos teriam esta ambigüidade esclarecida pelo realce sincronizado de elementos durante a execução e pelo destaque de variáveis com valores alterados.

3.2.2.3 Papéis Complementares

A abordagem adotada por este trabalho enquadra-se, nesta classe, como **informações complementares diferentes** [2]. Isso acontece porque, conforme dito, os processos intrínsecos a qualquer uma das duas formas de representação adotadas é exatamente o mesmo, mas as informações contidas em cada uma das descrições de algoritmo são fundamentalmente diferentes.

O conjunto de cores adotado e o destaque de variáveis com valores alterados também se incluem nesta mesma subclasse. O primeiro, por complementar a notação de fluxograma através do agrupamento semântico de elementos por meio de cores. O último, por denotar aspectos dinâmicos apresentados pelo algoritmo, em particular o conteúdo de determinadas células de memória e o momento em que sofrem alteração.

CAPÍTULO 4

ARQUITETURA FUNCIONALISTA

Neste capítulo descreve-se a arquitetura proposta para a consolidação da abordagem apresentada. Esta arquitetura foi validada com a implementação de um protótipo e até então demonstrou-se adequada e suficiente. Contudo, espera-se que trabalhos futuros indiquem a demanda e o prospecto de alterações.

Conforme representado na figura subseqüente (4.1), seis módulos compõem a arquitetura: Interface, Seletor de Eventos, Gerente de Tarefas, Manipulador de Objetos de RE, Entradas Permanentes do Aprendiz e Gramática do Domínio de Pascal. Todos estes módulos serão tratados na seqüência.

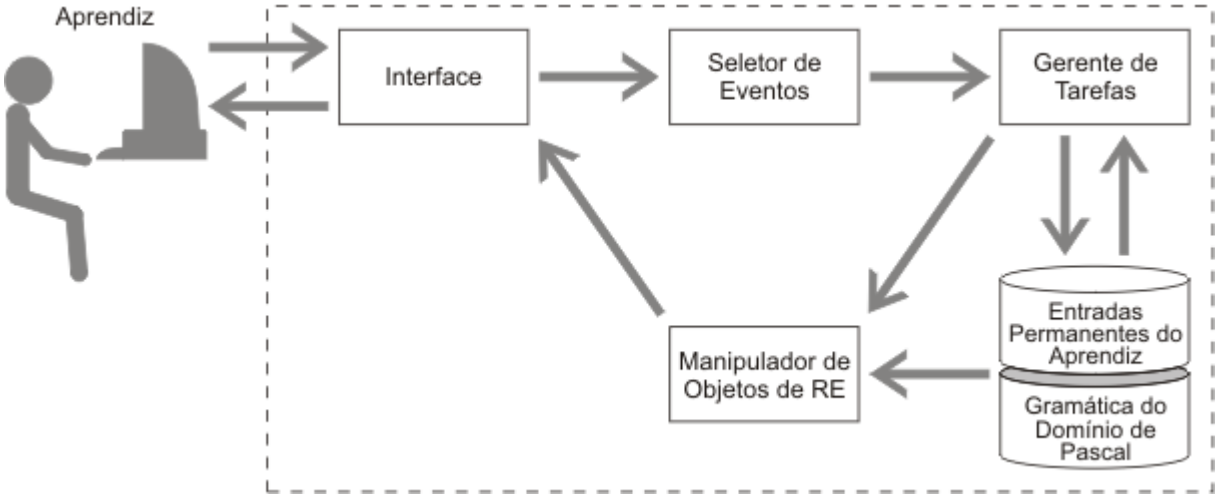


Figura 4.1: Arquitetura funcionalista

A respeito do protótipo, a última compilação realizada somou 216 classes (cada qual corresponde a um arquivo-fonte) e 19.061 linhas de código. Destas, 136 tiveram o código integralmente desenvolvido como parte deste trabalho e as outras 80 foram adaptações de trabalhos preexistentes devidamente creditados.

Deixa-se também registrado que se utilizou **Java**[1] como linguagem de programação

[1]Especificamente, **Java 2 Platform Standard Edition**. Disponível em http://java.sun.com.

para o desenvolvimento do protótipo. Fatores primordiais que fundamentaram esta escolha foram os seguintes:

- Trata-se de uma linguagem de programação orientada a objetos, paradigma cujos princípios e técnicas favorecem o domínio em questão. Além disso, há os padrões de projeto[2] [34, 46], que legam conhecimentos substanciais para projetistas e desenvolvedores neste paradigma;
- A comunidade Java dispõe de uma extensa gama de componentes de *software* para as mais diversas finalidades. Destes, a parcela voltada a aplicações científicas vem a ser bastante significativa;
- A implementação nativa da linguagem para estruturas de dados, além de ampla, é padronizada e integralmente documentada;
- Alta portabilidade, pois a compilação gera um código que será interpretado pela plataforma de execução (denominada Máquina Virtual Java[3]). Este código, teoricamente, independe do *hardware* e do sistema operacional utilizados;
- Disponibiliza vasta documentação *on-line* e favorece, através de mecanismos próprios, que o desenvolvedor documente o projeto através do código que está sendo escrito;
- Interfaces personalizáveis, qualidade que pode subsidiar a Interação Humano-Computador e atrair visualmente os aprendizes do ambiente desenvolvido;
- Familiaridade do proponente com a linguagem.

Antes de elucidar os módulos componentes, convém expor que cada um destes corresponde a um pacote de classes distinto. Consta-se o módulo de Entradas Permanentes do Aprendiz como única exceção (adiante esclarecida) a esta premissa.

[2]Do inglês *Design Patterns*.
[3]Do inglês *Java Virtual Machine* (JVM).

4.1 Interface

A interface gráfica[4] (figura 4.2) atua como mediador único entre o aprendiz e as funcionalidades do ambiente prototipado. Basicamente, consiste de uma barra de *menus* (superior) e da região central destinada aos algoritmos. Esta última ocupa a maior parte da tela, sendo subdividida verticalmente em duas outras partes: uma designada à construção do fluxograma e outra reservada à exibição do algoritmo textual equivalente.

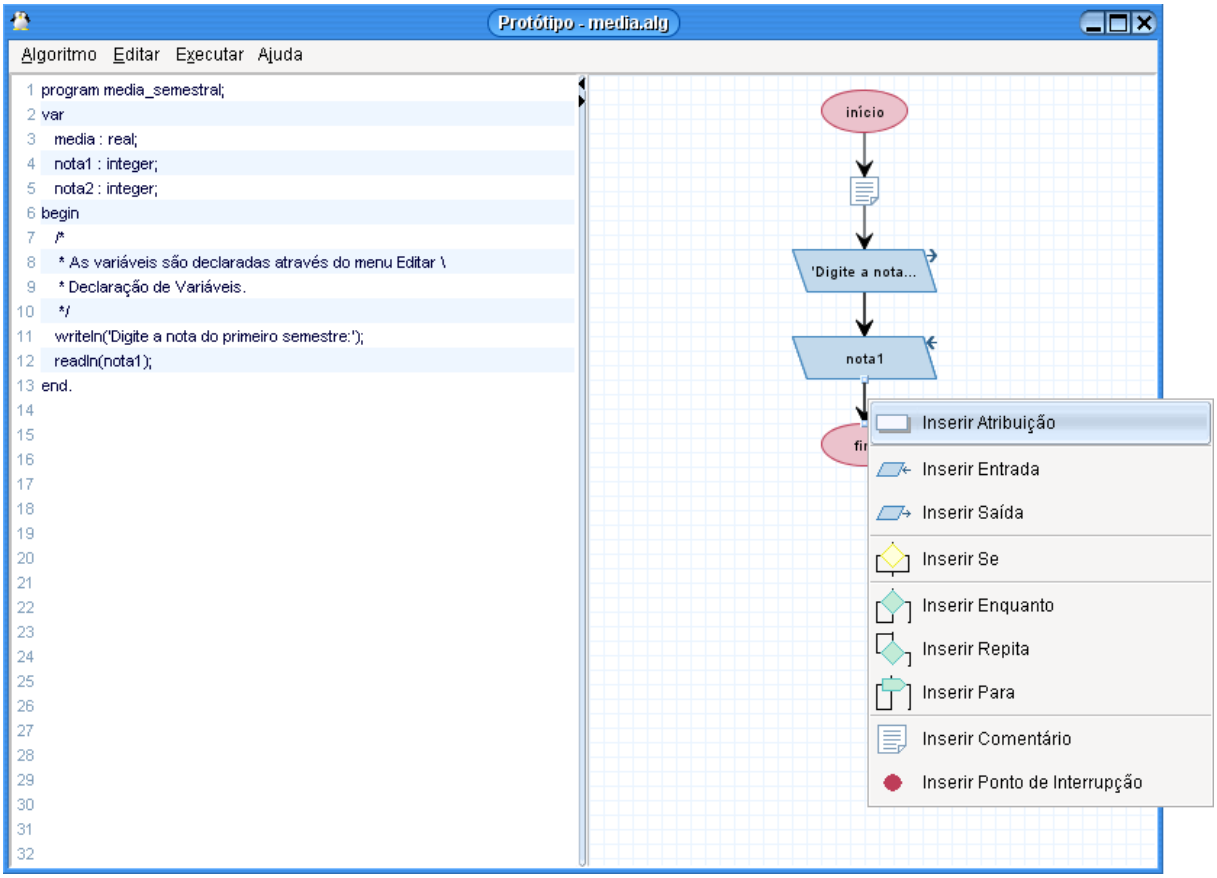


Figura 4.2: Interface do protótipo

Inicialmente, estava prevista uma barra de ferramentas (inferior) que comportaria os botões relacionados à construção do fluxograma (e.g. inserção, edição e exclusão de instruções de entrada, instruções de saída, estruturas condicionais e de repetição), bem como botões referentes às outras REs (e.g. comentários, pontos de interrupção, recolhimento e expansão de blocos de elementos). Contudo, foi notado que, além do espaço em tela

[4]Implementada utilizando o pacote **Swing**, atualmente incorporado à linguagem Java.

requerido, a exibição de muitos destes controles é contextualizada com o elemento selecionado do fluxograma, exigindo com que a barra freqüentemente fosse transmutada. Assim, na implementação, optou-se por *menus* de contexto em detrimento desta barra de ferramentas, pois estes não requerem local fixo para a exibição e podem ser adaptados aos elementos correlatos antes de desenhados na tela (a mudança não fica explícita aos olhos do aprendiz).

Portanto, delegou-se muita atenção para que não se sobrecarregasse visualmente (e semanticamente) a interface. Em se tratando de um ambiente de ensino, isso geralmente implica em excessos dos quais o aprendiz raramente conseguiria tirar proveito e ainda comprometeriam seu desempenho.

Ademais, apesar de toda a cautela, salienta-se que a interface aqui exposta precisa ser aperfeiçoada e respaldada pelos preceitos da área de Interação Humano-Computador (IHC).

4.1.1 Barra de *Menus*

A barra de *menus* (figura 4.3) compreende o acesso às opções de caráter mais geral no que concerne aos algoritmos. Cada subitem contém um texto descritivo, bem como uma figura metafórica que ilustra a funcionalidade correspondente. Exibe-se uma descrição mais detalhada da tarefa quando se repousa o ponteiro do *mouse* sobre qualquer um dos subitens.

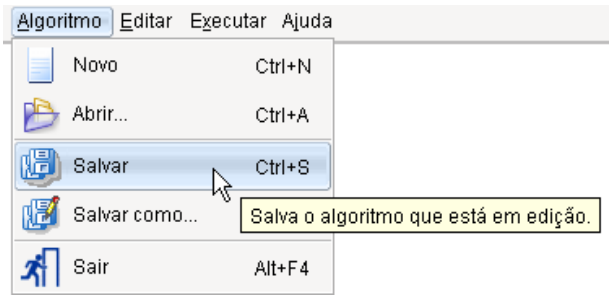


Figura 4.3: Barra de *menus*

Os itens de *menu* que compõem a barra são os seguintes:

- **Algoritmo:** comporta opções referentes à manipulação de arquivos de algoritmo: (a) novo, (b) abrir, (c) salvar, e (d) salvar como; bem como a opção de (e) encerramento do protótipo;
- **Editar:** diz respeito à edição de propriedades básicas do algoritmo, permitindo (a) a alteração do nome do algoritmo e (b) o acesso à tela de declaração de variáveis (descrita na seção 4.1.4);
- **Executar:** abriga as duas alternativas relacionadas à execução do algoritmo corrente, permitindo que este seja executado (a) em modo passo-a-passo ou (b) temporizado;
- **Ajuda:** reduz-se a uma única opção, que exibe a tela de informações sobre o protótipo. Esta tela não será explanada adiante, mas tem caráter bastante básico: compreende o logotipo da UFPR, o título desta dissertação, assim como os nomes do orientador, do proponente e *e-mail* para contato;

4.1.2 Área do Algoritmo

Conforme mencionado, esta região subdivide-se em duas, uma destinada à construção do fluxograma e outra à exibição do algoritmo textual correlato. Inicialmente, ambas as subregiões têm o mesmo tamanho, mas há a alternativa de redimensioná-las ou de suprimir uma delas.

O espaço reservado ao fluxograma, localizado à direita, inicializa com os demarcadores de início e fim do algoritmo, interligados por um arco direcionado. As opções de inserção de elementos são oferecidas através do *menu* de contexto respectivo a este (ou qualquer outro) arco. Analogamente, acessa-se as opções de edição de um determinado elemento através do *menu* de contexto relacionado.

Durante a implementação, foram incorporadas pequenas funcionalidades à área do fluxograma no intuito de facilitar a interação do aprendiz com o protótipo. Pode-se

citar, como exemplo, a rolagem e o redimensionamento da área de edição, bem como o reposicionamento do fluxograma em ações de inserção, remoção ou seleção.

A área designada à apresentação do algoritmo textual equivalente ao fluxograma, situada à esquerda, mantém-se atualizada a cada edição, inserção ou remoção feita no fluxograma. Possui, como todas as tabelas e listas da interface, cores distintas para linhas pares e ímpares. Além disso, conta também com uma margem vertical que numera cada uma destas linhas. Evidencia-se que os *menus* de contexto são exclusivos da área do fluxograma, não havendo *menu* ou botão associado a esta segunda área.

4.1.3 *Menus* de Contexto

Os *menus* de contexto, implementados em substituição à barra de ferramentas inicialmente prevista, perfazem uma solução que vem a fortalecer a intuitividade e a simplicidade da interface desenhada. Permitem que cada categoria de elementos do fluxograma tenha um quadro próprio de opções e que, mesmo aquelas comuns a outras categorias, apresentem ícones correspondentes a cada situação. Obtém-se acesso ao *menu* de contexto respectivo a um determinado elemento através de um clique com o botão contrário[5] no elemento desejado.

Abaixo categoriza-se os contextos presentes juntamente com as opções que concernem a cada um deles:

- **Arco:** abrange opções que possibilitam a inserção de elementos na imediata seqüência do arco em contexto. Todas estas são descritas pelo verbo "inserir" sucedido pelo nome do elemento, resultando respectivamente em: inserir atribuição, entrada, saída, se, enquanto, repita, para, comentário e ponto de interrupção;

- **Elementos editáveis:** dos elementos que podem ser inseridos, salvo o ponto de interrupção, todos disponibilizam em seus *menus* as opções de edição e remoção;

- **Estrutura condicional ou de repetição:** em acréscimo àquelas opções listadas

[5]Por padrão, o botão direito.

no item anterior, estas estruturas admitem outra duas: serem retraídas ou (exclusivamente) expandidas conforme o estado de visualização atual;

- **Ponto de Interrupção:** a semântica do ponto de interrupção típico não aceita edição, permitindo somente a inserção e a remoção. O *menu* relacionado ao arco trata da primeira, cabendo a este contexto especificamente a segunda opção;

- **Início e Fim:** visto que estes elementos não permitam edição ou remoção, teve-se a idéia de utilizá-los como atalho para as opções relativas à propriedades básicas do algoritmo: a edição do nome do algoritmo e o acesso à tela de declaração de variáveis[6];

- **Nenhum item selecionado:** adota-se a mesma abordagem do item anterior.

4.1.4 Tela de Declaração de Variáveis

Embora a notação de fluxograma não trate da declaração de variáveis, no nicho abordado por este trabalho tal recurso se faz necessário. A alternativa projetada para suprir esta falta consiste na alocação desta funcionalidade em uma tela independente do fluxograma.

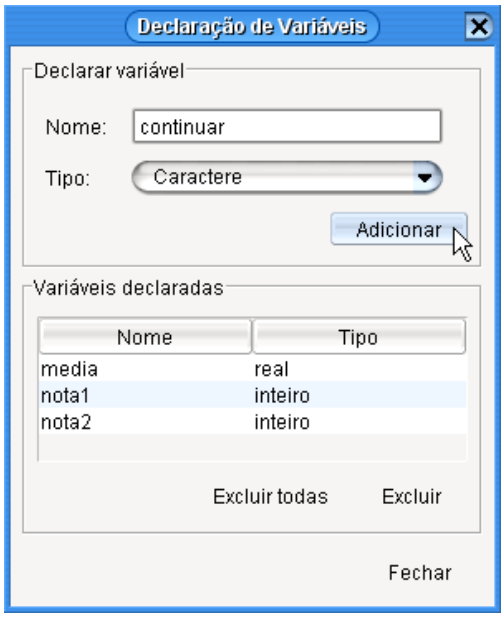


Figura 4.4: Tela de declaração de variáveis

[6]Ambos acessíveis através do *menu Editar* (abordado na seção 4.1.1).

A organização desta tela (conforme figura 4.4) dá-se em duas regiões claramente distintas pelos quadros que as envolvem. A primeira, comporta uma caixa de texto para digitação do nome da variável, uma caixa de seleção em que se opta pelo tipo correspondente (booleano, inteiro, real ou caractere) e um botão que adiciona a variável corrente àquelas previamente declaradas. A segunda região, por sua vez, contém uma tabela com o nome e o tipo das variáveis declaradas, bem como botões que possibilitam a exclusão destas variáveis.

Frisa-se que a inclusão de uma variável cujo nome já foi declarado implica na redefinição do tipo desta variável para o último escolhido na caixa de seleção.

4.1.5 Telas de Execução

O protótipo oferece duas opções de execução: em modo passo-a-passo e temporizada (esta última exibida na figura 4.5). Em ambas há a necessidade tanto de desabilitar os controles de edição quanto de obter espaço para a barra de tarefas de execução, a tabela de inspeção de variáveis e a janela de saída. Considerando estes fatores, preferiu-se não somente desabilitar a barra de *menus*, mas também retirá-la da interface durante o processo de execução, desocupando parte da tela. Partindo do mesmo raciocínio, todos os *menus* de contexto foram desabilitados.

De maneira geral, há bastante cabimento em inibir as funcionalidades acessíveis através dos *menus* acima citados. Não faz sentido, por exemplo, inserir uma estrutura de repetição em pleno funcionamento do algoritmo. Porém, existem tarefas desabilitadas que são menos controversas, como as opções de salvar o algoritmo ou de exibir a tela de informações sobre o protótipo. No entanto, realmente intencionou-se explicitar a alternância entre os modos de edição e de execução do algoritmo. Logo, a transmutação da interface e o estabelecimento de grupos disjuntos de controles vêm a corroborar este propósito.

A alternância para o modo de edição redimensiona verticalmente a área do algoritmo, requerendo uma parcela de 25% do espaço antes ocupado. Nesta área, são distribuídas a tabela para inspeção de variáveis e a janela de saída do algoritmo. A primeira, exibida somente se houver variáveis declaradas, consiste numa tabela bastante simples, contendo

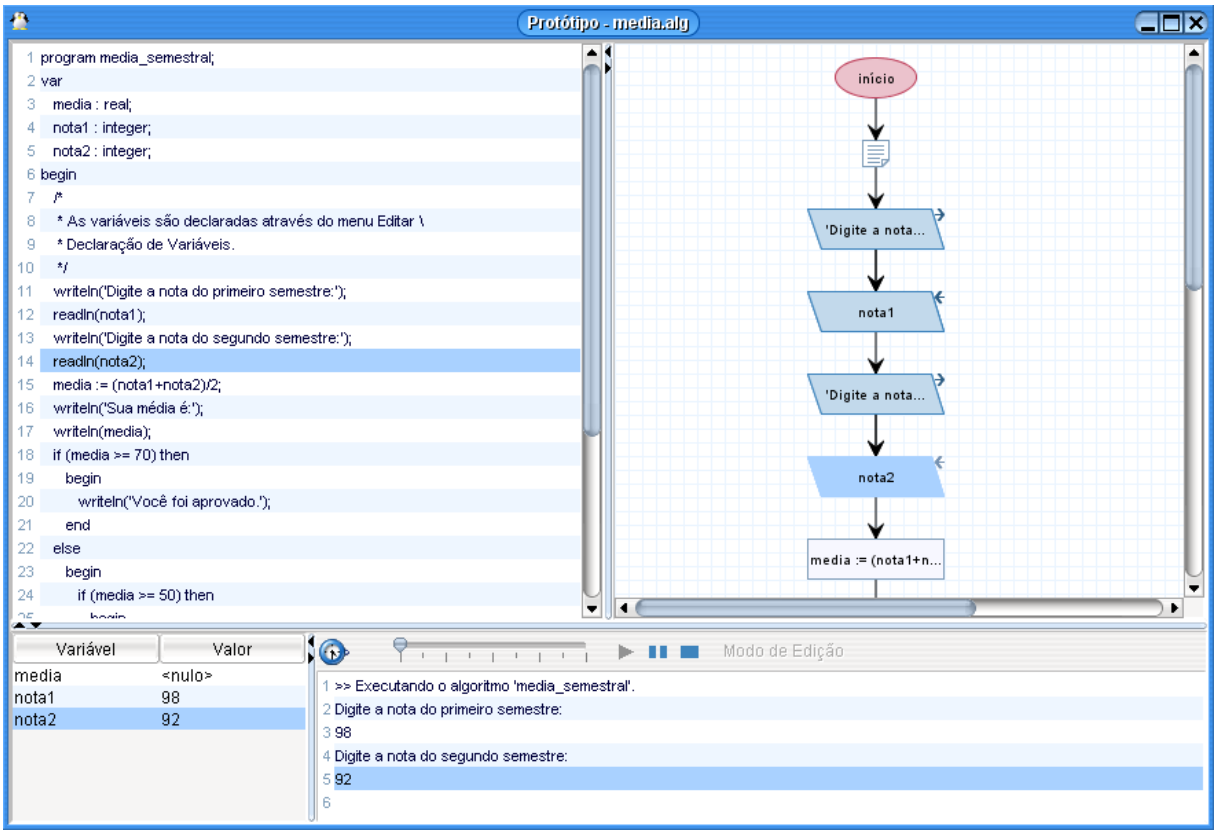


Figura 4.5: Execução temporizada em progresso

duas colunas: uma armazenando os nomes das variáveis e a outra os valores atribuídos. A janela de saída, que utiliza a maior proporção deste espaço, atua analogamente à saída padrão da maioria das linguagens de programação imperativistas. Duas pequenas distinções de implementação, que neste ponto residem, foram: a numeração das linhas desta janela e a utilização de caixas de diálogo próprias para instruções de leituras de tipagens diferentes. Neste último caso, aplicam-se validações e controles apropriados a cada tipo de dado (e.g. botões de opção, "verdadeiro" e "falso", para a leitura de variáveis booleanas).

Evidencia-se que, embora as áreas acima citadas sejam previamente dispostas e dimensionadas pelo protótipo, há a possibilidade tanto de redimensionar quanto de suprimir qualquer uma destas regiões.

No que diz respeito às duas opções de execução, a diferença na interface consiste tão somente nos controles alocados na barra de tarefas relacionada (conforme figura 4.6). A composição básica desta barra consta dos seguintes controles: iniciar, pausar e encerrar

a execução, assim como um botão que permite o retorno ao modo de edição. Exclusivo à execução passo-a-passo, existe um controle que executa a próxima instrução (um passo à frente). De forma semelhante, restrito à execução temporizada, há um controle deslizante que permite aumentar, ou diminuir, a velocidade com a qual o algoritmo é executado.

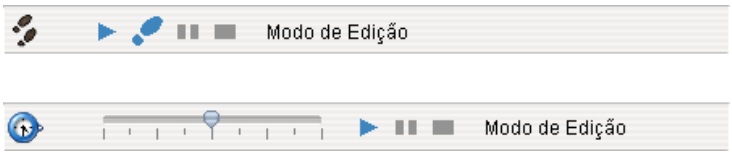


Figura 4.6: Barras de tarefas de execução (passo-a-passo e temporizada)

Observa-se, entretanto, uma funcionalidade emergente neste ponto. Seria proveitosa a alternativa de retornar ao passo anterior executado (passo para trás) com o objetivo de rever estados e instruções. Tem-se ciência da dificuldade de armazenar todas as informações necessárias para tal tarefa. Contudo, a possibilidade de retroceder poucos passos (julgando uma escala de três a cinco) ao atual já seria vantajosa.

Por fim, demarca-se outra característica de substancial importância para o projeto desta parte da interface: a utilização de processamento concorrente[7]. Através disto, foi possível o monitoramento dos controles da barra de tarefas enquanto ocorre a execução do algoritmo. Caso estes dois processos não fossem paralelamente executados, na melhor das hipóteses, seria necessário aguardar todo o procedimento de execução do algoritmo para se ter acesso aos eventos da barra de tarefas. Tal obstáculo impossibilitaria as funcionalidades de pausar e encerrar a execução através dos respectivos botões, que são bastante úteis em casos como laços infinitos.

4.2 Seletor de Eventos

Este módulo tem a responsabilidade de, mediante um evento da interface, definir as tarefas a serem executadas pelo Gerente de Tarefas. Foi concebido com o propósito de que atuasse como um multiplex em que um dado evento disparasse uma seqüência de tarefas então delegadas àquele módulo que as gerenciaria.

[7]Em Java, chamadas de *threads*.

Como exemplo, pode-se supor que o aprendiz tenha acessado a opção de *menu* referente à criação de um novo algoritmo. O evento disparado requisitaria ao módulo gerente a execução das três seguintes tarefas:

1. **Perguntar se deseja salvar:** em caso de alteração no algoritmo, pergunta se o aprendiz deseja ou não salvar as alterações, bem como se pretende cancelar a criação de um novo algoritmo;

2. **Salvar algoritmo:** caso o aprendiz tenha optado por salvar o algoritmo, solicita ao gerente a execução desta tarefa;

3. **Criar um novo algoritmo:** dada a circunstância do aprendiz não haver cancelado a criação de um novo algoritmo, requer a execução desta tarefa ao gerente.

Todavia, na implementação do protótipo, nenhuma tarefa executada diretamente a partir de um evento estendeu-se a ponto de precisar ser subdividida. Também quase não houve intersecção de tarefas que demandasse a criação de outras adicionais com a finalidade de especializar o código comum às primeiras. Assim, na maioria das situações, coube ao módulo gerente apenas o encargo de unificar em cada uma de suas tarefas o acesso aos outros módulos[8]. Nestes casos, somente compete ao módulo seletor, por intermédio de um evento, requerer do gerente a execução da tarefa associada.

Porém, dado o anseio de prosseguir com o projeto além do curso de mestrado, bem como o intento de consolidar um arcabouço para que outras pesquisas possam se beneficiar, decidiu-se por manter este módulo íntegro à concepção original. Dessa forma, mesmo na maioria das vezes associando um evento a uma única tarefa, o Seletor de Eventos está pronto para tratar do caso mais geral.

Conforme mencionado, este módulo consiste em um pacote Java distinto. Da mesma forma, cada tela do protótipo possui, alocado neste, um subpacote específico onde estão concentrados todos os eventos que a correspondem. Além disso, cada evento respectivo a um controle de interface possui uma classe responsável por tratá-lo. Exceções a este caso típico são os eventos automaticamente tratados pelos pacotes Swing ou JGraph[9].

[8] Entradas Permanentes do Aprendiz, Gramática do Domínio de Pascal e Manipulador de Objetos.
[9] Abordado na seqüência.

Isto posto, salienta-se que, como atalho de implementação, quando determinado evento dispara uma tarefa que consiste em apenas um redirecionamento por parte do módulo gerente, o Seletor de Eventos toma a liberdade de requerer a execução desta diretamente da classe que a abriga. Um caso comum que se menciona consiste na exibição de algumas telas do protótipo. Nestas situações, a única tarefa que o módulo seletor solicitaria ao Gerente de Tarefas seria a exibição de certa tela e, portanto, fracionar uma instrução tão simples eminentemente decrementaria a legibilidade do código.

Por fim, percebe-se que se trata de um módulo que inicialmente poderia ser suprimido, tendo suas capacidades distribuídas entre o módulo de Interface e o Gerente de Tarefas. Porém, intenciona-se consolidar uma arquitetura cujas responsabilidades sejam bem distribuídas e que suporte alterações, principalmente quanto à dimensão do ambiente proposto.

4.3 Gerente de Tarefas

Das tarefas requisitadas pelo Seletor de Eventos, será o Gerente de Tarefas quem efetivamente tratará da execução. Para isso, além de comportar os procedimentos para a realização de cada tarefa requerida, precisa se comunicar com os módulos de Entradas Permanentes do Aprendiz e de Gramática do Domínio de Pascal. Também faz-se necessário que o Gerente de Tarefas determine as subtarefas pertinentes ao Manipulador de Objetos de RE para que este as execute.

Conforme antedito, previa-se a subdivisão das tarefas aqui alocadas em outras menores. Antecipou-se que o fracionamento ocorreria basicamente em dois casos. O primeiro, seria caracterizado pela existência de tarefas demasiado extensas. Nestas, por prudência, seria necessária a identificação de subtarefas componentes cujo código permitiria realocação, originando novas tarefas. Mais tarde, com o incremento das funcionalidades do protótipo, outros elementos de código poderiam se beneficiar da utilização destas novas tarefas.

O segundo caso, por seu turno, consistiria no isolamento de intersecções (dos códigos) de tarefas. Assim, aquelas que tivessem uma parcela de código comum potencialmente originariam, a partir deste, uma nova tarefa mais específica.

Contudo, na implementação do protótipo, poucas foram as tarefas realmente executadas pelo Gerente de Tarefas que se enquadraram em algum dos casos acima. Ocorreu que, para o que se propôs, a quase totalidade de tarefas neste módulo compreendidas restringiam-se à unificação e coordenação de outras mais complexas estabelecidas em outros módulos. Daquelas restantes, todas apresentavam intersecções de código (sendo compreendidas pelo segundo caso determinado), demandando a criação de tarefas adicionais.

Como exemplo concreto, toma-se a tarefa de criar um novo algoritmo, relatando-se em seguida as respectivas subcomponentes solicitadas às classes responsáveis:

1. Criação de um novo fluxograma e conseqüente exibição deste na área devida;
2. Anulação do vínculo anterior entre o algoritmo e arquivo;
3. Atualização do título da tela principal, pois antes este possivelmente continha o nome do arquivo anterior.

4.4 Manipulador de Objetos de RE

Tendo em vista a importância das MREs para este trabalho, fez-se necessária a concepção de um módulo específico para gerenciá-las. Comparado aos demais módulos do protótipo, o Manipulador de Objetos de RE tem, de longe, a maior complexidade. Fundamentalmente, cabe a este módulo, através de informações provindas do Gerente de Tarefas, das Entradas Permanentes do Aprendiz e da Gramática do Domínio de Pascal, criar e coordenar as REs para que estas sejam exibidas através do módulo de Interface.

4.4.1 O Padrão de Projeto *Composite*

Muito apoiou este trabalho a descoberta de um padrão de projeto correspondente à situação aqui abordada. A dificuldade inicial que se tinha era encontrar uma modelagem adequada que representasse as possibilidades hierárquicas dentro de um fluxograma. Este, basicamente, consiste em uma seqüência de instruções em que cada uma destas pode ser

simples (entrada, saída e atribuição) ou composta (se, enquanto-faça, repita-até, para). Neste último caso, cada instrução composta reflete recursivamente a definição de um fluxograma como uma seqüência de instruções.

Ao encontro disso, o padrão *Composite* [34, 46] subsidia a composição de objetos em estruturas de árvore a fim de que representem hierarquias parte-todo. Dessa forma, grupos de objetos podem conter tanto itens individuais quanto outros grupos. Para isso, contribui a característica de que comportamentos comuns sejam definidos para ambos os casos, permitindo aos clientes tratarem uniformemente objetos individuais e composições de objetos. Alcança-se esta particularidade através da definição de uma superclasse abstrata (ou, possivelmente, uma interface) que represente tanto objetos individuais quanto compositores (aqueles que funcionam como recipientes).

Abaixo apresenta-se um diagrama de classes exemplificando o padrão *Composite*:

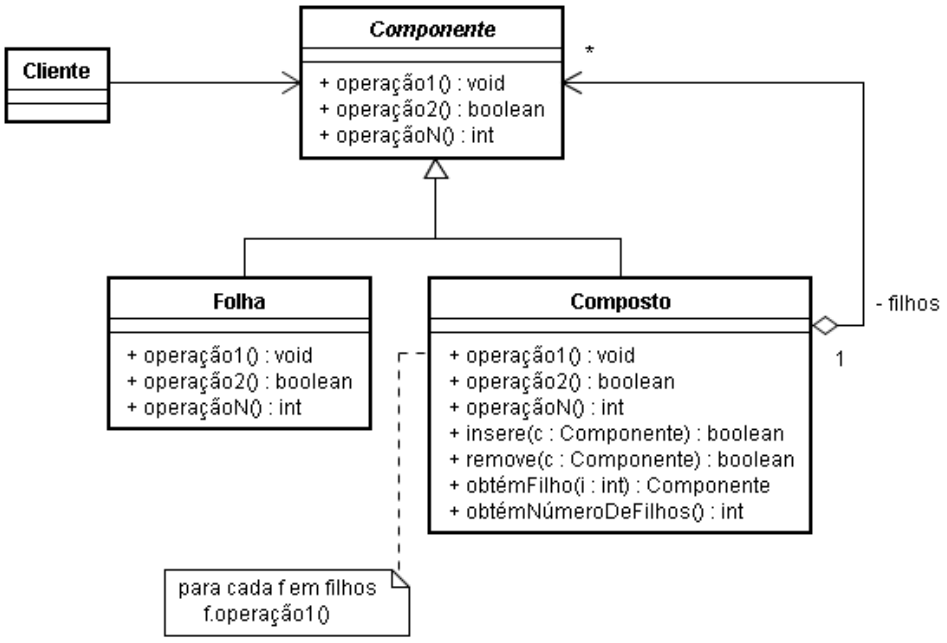


Figura 4.7: Exemplo de diagrama de classes para o padrão *Composite*

Diante deste exemplo de diagrama, aponta-se os seguintes participantes [34]:

- **Componente:**
  - declara a interface utilizada por todos os objetos na composição, sejam estes individuais ou compostos;

  - conforme apropriado, implementa o comportamento-padrão da interface comum a todas as classes que podem constituir uma composição.

- **Folha:**
  - representa objetos-folha (objetos individuais que não têm filhos) na composição;
  - mediante a interface especificada por *Componente*, define o comportamento para objetos primitivos da composição.

- **Composto:**
  - determina o comportamento para componentes que têm filhos;
  - armazena os componentes-filho constituientes;
  - declara e implementa operações relacionadas ao acesso e gerenciamento de componentes-filho.

- **Cliente:**
  - opera objetos na composição através da interface definida por *Componente*.

Portanto, de acordo com o diagrama de classes descrito, um objeto composto poderia assumir a seguinte estrutura (figura 4.8):

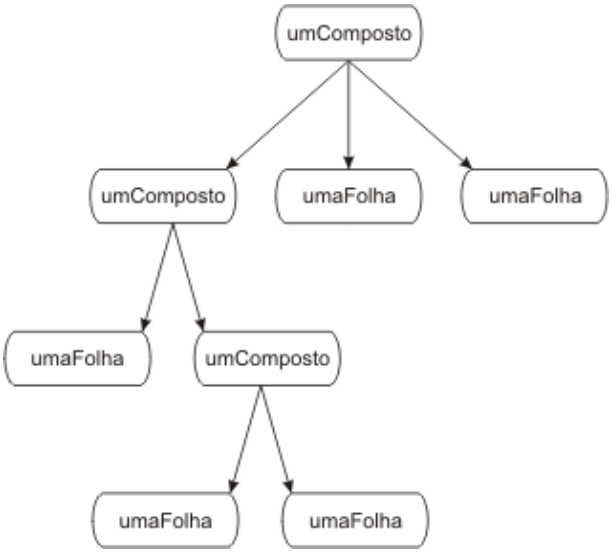


Figura 4.8: Estrutura de um objeto gráfico composto

Diante do apresentado, numa modelagem mais extensa pode-se ter várias subclasses imediatas à classe abstrata *Componente*, cada qual representando exclusivamente objetos individuais ou compostos. Não há restrições de proporção entre as classes que modelam ambos estes objetos.

O extremo contrário desse critério de modelagem consiste na declaração de duas subclasses abstratas, uma destinada a representar objetos individuais e outra denotando os compostos. Todas as classes concretas de uma composição proviriam (mesmo que indiretamente) de uma destas duas. Frisa-se que se adotou, no protótipo, esta última proposta.

Outra questão que merece evidência diz respeito a métodos (e, por analogia, atributos) exclusivos de algumas subclasses. A literatura da área oferece diferentes abordagens para este quesito. A primeira abordagem encontrada [34] consiste em definir estes métodos na superclasse (*Componente*) e inabilitá-los, mediante sobrecarga, nas classes que não os necessitam. Outra abordagem [46] dita que a abstrata *Componente* deve enunciar somente os métodos e atributos comuns a todas as subclasses, cabendo a cada qual a definição daqueles exclusivos que lhe couberem. Para este projeto em particular empregou-se a segunda abordagem, pois haveria debilitação da legibilidade do código por conta do engrandecimento da classe primária.

Considerando o exposto, os diagramas de classes das figuras 4.9 e 4.10 demonstram o padrão *Composite* adequado à modelagem dos elementos de fluxograma do presente protótipo. O primeiro diagrama explicita a classe abstrata primária, denominada *Elemento*, bem como as outras duas abstratas *ElementoSimples* e *ElementoComposto* que denotam, nesta ordem, objetos individuais e recipientes.

No segundo diagrama, houve a necessidade de suprimir métodos e atributos para que se visualizasse toda a hierarquia de classes descendentes. Neste, fica evidente a subordinação de instruções simples e compostas àquelas classes abstratas.

Levando em conta os diagramas apresentados, merece releitura prática a sobrecarga de métodos pelas subclasses. Utilizando o método *toCodigo()*[10] como exemplo, tem-se implementações distintas para ambas as subclasses. Na classe que representa objetos

[10]O método ocupa-se de transcrever em Pascal a instrução expressa por determinado objeto em fluxograma.

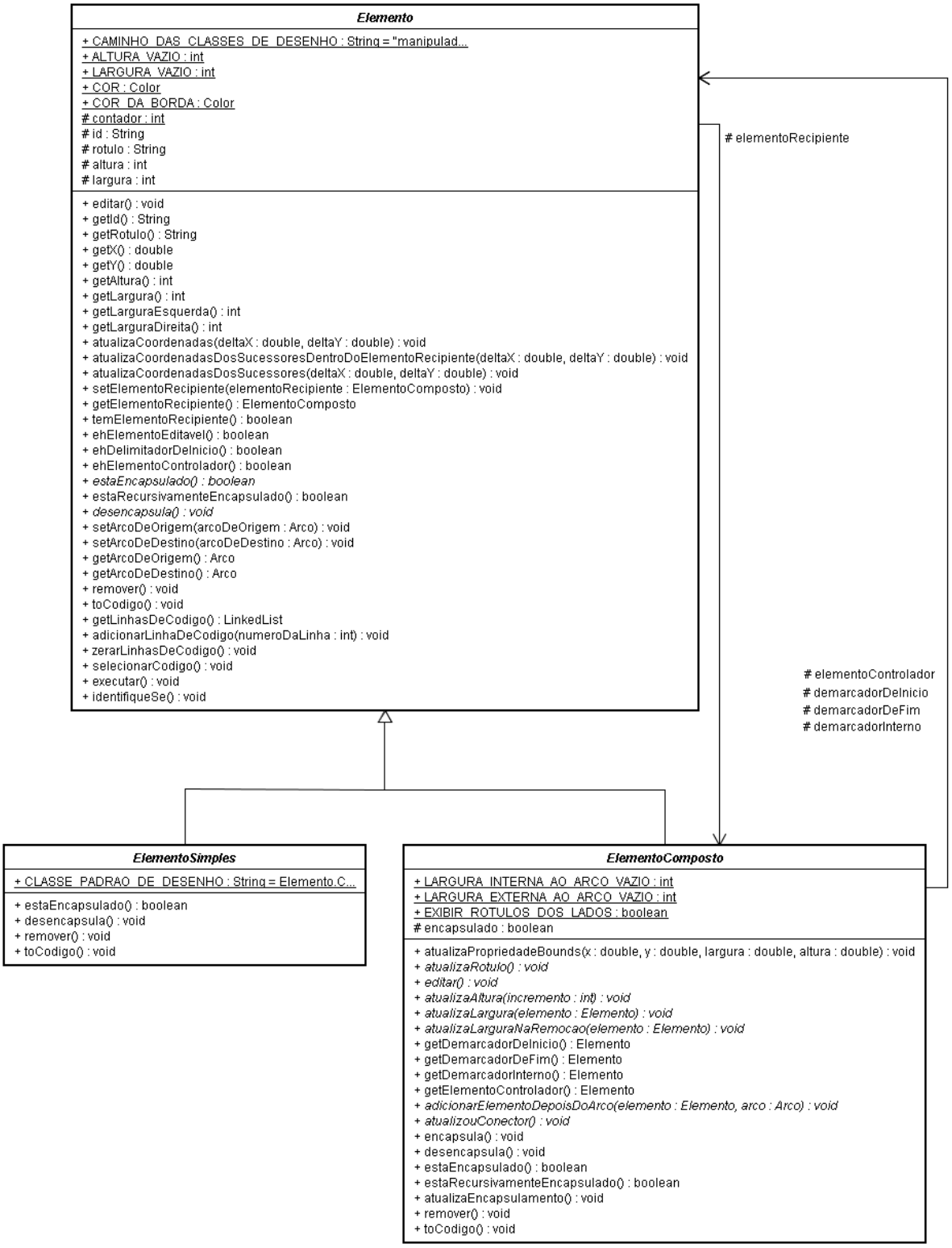


Figura 4.9: Diagrama das três classes principais dos elementos do fluxograma

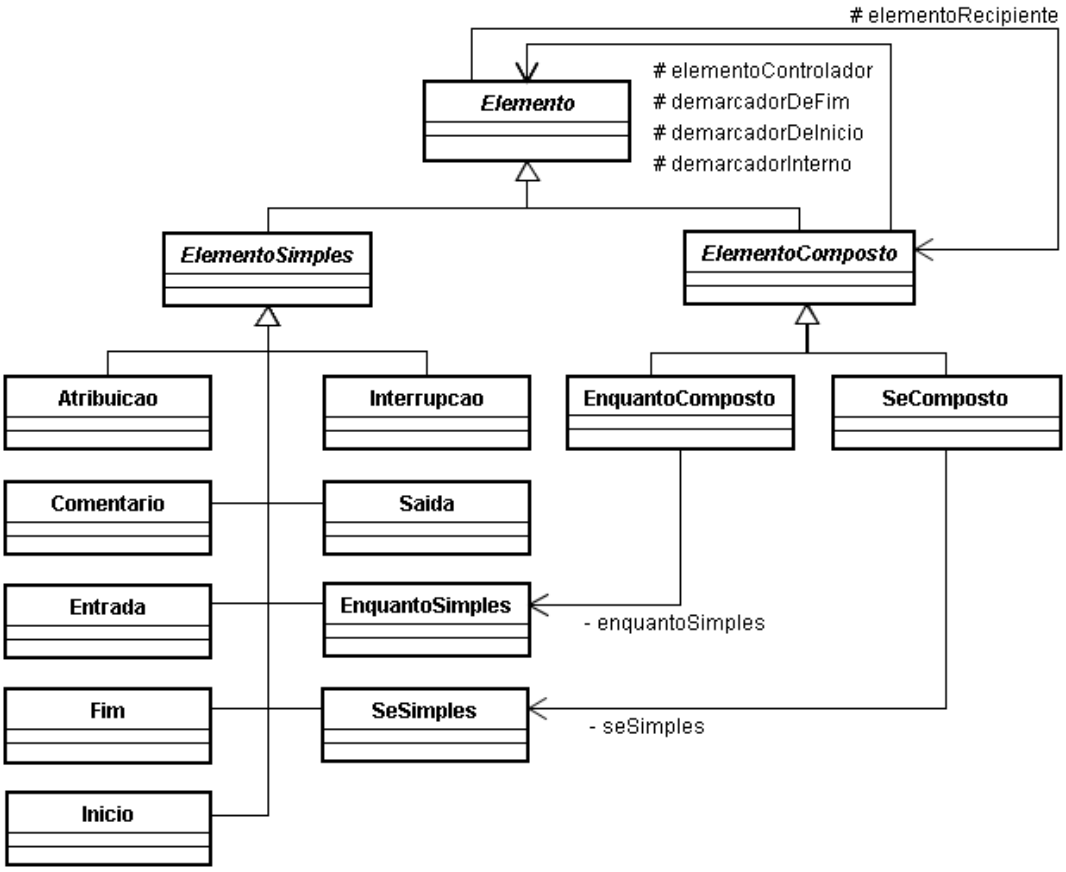


Figura 4.10: Diagrama de Classe simplificado dos elementos do fluxograma

simples, o método reduz-se a transcrever para o algoritmo textual o código correlato àquela única instrução. Em classes de objetos compostos, o método precisa, além de transpor a instrução individual, requerer a execução desta tarefa de todos os filhos imediatos alocados.

4.4.2 O Componente JGraph

Toda a representação visual do fluxograma foi fundamentada sobre a biblioteca de componentes **JGraph**[11]. Trata-se de uma ferramenta madura, rica em funcionalidades, com código-aberto e integralmente escrita em Java. Possui compatibilidade com Swing, tanto em aspectos visuais quanto no que se refere à arquitetura.

Sendo própria para se trabalhar com grafos, a biblioteca JGraph alicerça-se na teoria matemática de nome análogo (Teoria dos Grafos). Deste modo, o pacote descrito oferece mecanismos para visualização, interação, desenho e disposição de grafos; como também para o tratamento de operações que cabem à teoria que os envolve (e.g. determinação da menor rota entre dois nós).

Embora não caiba a este projeto prolongar-se neste quesito, convém mencionar que um fluxograma, por sua vez, pode ser considerado uma extensão de um tipo específico de grafo. Este, denominado Grafo de Fluxo de Controle (GFC), consiste em "um grafo direcionado e conexo utilizado como representação de um programa, em particular do fluxo de execução, no qual os vértices do grafo representam computações e os arcos do grafo representam o fluxo do controle" [28].

Isto posto, legitima-se a escolha dessa biblioteca para modelar, visual e internamente, muitos dos aspectos que concernem à construção do algoritmo gráfico. Salienta-se que, em vários aspectos, as funcionalidades da biblioteca precisaram ser estendidas a fim de que esta suportasse, especificamente, a manipulação de fluxogramas.

[11] Disponível em http://www.jgraph.com.

4.4.3 Tabelas de Variáveis

A estrutura de dados que armazena as variáveis declaradas reduz-se a uma tabela *hash*[12] nativa da linguagem Java. Esta estrutura caracteriza-se por proporcionar um mapeamento (e conseqüente armazenamento) de chaves a respectivos valores, bem como, por permitir o resgate de determinado valor mediante a chave associada. Não se admite valores ou chaves nulos, como também uma mesma chave não pode se referir a mais de um valor.

Portanto, neste projeto, as variáveis são descritas em termos de pares ordenados (nome, tipo[13]) que se relacionam, nesta ordem, ao valor e à chave requeridos pela tabela *hash*. As propriedades acima descritas garantem que não haja nome de variável ou tipo nulos. Evita-se, igualmente, que um mesmo nome de variável refira-se a mais de um tipo[14]

Evidencia-se que esta tabela consiste em um atributo da própria classe que corresponde ao fluxograma. Assim, esta informação também permanece em memória secundária quando um algoritmo for salvo (melhor detalhado na seção 4.5).

Analogamente, durante a execução, tem-se uma outra tabela *hash* que diz respeito aos pares ordenados (nome, valor atribuído) das variáveis. Esta apresenta o funcionamento semelhante àquela anteriormente descrita, com as exceções de que as variáveis são mapeadas aos valores atribuídos e a tabela não tem persistência em arquivo.

4.4.4 Execução do Algoritmo

Na possibilidade de executar um algoritmo construído residiu um dos maiores impasses desta dissertação. Considerou-se, primeiramente, duas alternativas, todavia nenhuma prosperou.

A primeira, limitava-se a chamar, a partir do protótipo, um compilador externo para verificar a ocorrência de erros e, feito isso, requerer a execução do programa gerado. Tratava-se de uma hipótese cabível, visto que se tinha, além do fluxograma, o algoritmo textual equivalente. Contudo, seria bastante difícil de reaver informações tanto do com-

[12] *Hashtable.*
[13] Representados, visando à eficiência, por constantes inteiras.
[14] Caso isso aconteça, a nova declaração da variável substitui a anterior.

pilador quanto do programa.

A atualização da tabela de valores, assim como a sincronização do algoritmo com os passos executados estariam sensíveis ao compilador e à plataforma utilizados. Não haveria garantias, por exemplo, de que determinada versão do compilador escolhido apresentasse o mesmo comportamento em sistemas operacionais distintos. Além disso, novas versões de tal compilador poderiam impactar em manutenção constante e profunda no protótipo.

A segunda opção consistia em efetivamente implementar um compilador que atendesse ao subconjunto de Pascal abordado. Embora se pudesse encontrar algum material correlato mediante pesquisa, este exigiria adaptação para acoplamento ao protótipo. Também estava evidente de que não seria difícil se deparar com uma mera formalização da linguagem Pascal, porém encontrar algo mais próximo de uma implementação (preferencialmente em Java) poderia tardar. Dessa forma, não seria raro de que o compilador superasse todo o restante do protótipo em magnitude e complexidade. Além disso, considerando que somente houvesse a compilação do código, ainda haveria a dificuldade de recolher o resultado de cada instrução executada para atualizar a tabela de variáveis e a sincronização com o algoritmo.

Sendo nenhuma das alternativas favoráveis, chegou-se à conclusão da possibilidade de executar o algoritmo através do próprio fluxograma, desprezando (internamente) o código em Pascal. Dentro disso, somente fazia-se necessário um avaliador de expressões para os tipos de variáveis abrangidos, pois o fluxo do algoritmo e o valores das variáveis seriam controlados pelos elementos do fluxograma.

Considerando o emprego do avaliador de expressões[15], abaixo segue uma descrição dos passos que cada elemento de fluxograma desempenha ao executar:

- **Início:** inicializa a janela de saída e a tabela de variáveis (certificando-se que o valor $<nulo>$ esteja atribuído a cada variável);
- **Atribuição:** avalia a expressão atribuída e fixa o valor resultante para a variável determinada;

[15]Frisa-se que o conceito de "expressão" adotado tem caráter geral, referindo-se tanto a expressões compostas quanto àquelas constituídas apenas de uma variável ou constante.

- **Entrada:** atribui à variável lida o valor de entrada;
- **Saída:** avalia a expressão dada como parâmetro e imprime o resultado na janela de saída;
- **Se:** avalia a expressão lógica do condicional e desvia o fluxo para o bloco de elementos correspondente ao resultado;
- **Enquanto-faça:** também avalia a expressão lógica do condicional. Caso o resultado seja verdadeiro, segue o fluxo para as instruções contidas nesta estrutura. Caso contrário, desvia o fluxo para a instrução seguinte ao *enquanto-faça*;
- **Comentário:** segue o fluxo, pois este elemento é ignorado durante a execução;
- **Ponto de Interrupção:** pausa a execução;
- **Fim:** encerra a execução e sinaliza este evento na janela de saída.

Procurando-se especificamente por materiais relacionados à avaliação de expressões, soube-se do projeto de código-aberto denominado **ePascal**[16]. Trata-se de um interpretador para um subconjunto da linguagem Pascal inteiramente implementado em Java.

Diante disso, somente os trechos que equivaliam ao avaliador de expressões foram adaptados e incorporados ao protótipo desta dissertação. Mesmo se tendo, naquele momento, o código-fonte de um interpretador em mãos, preferiu-se contemplar a generalização do protótipo implementado, absorvendo-se somente o relativo à avaliação de expressões. Assim, o fluxo do algoritmo e os valores das variáveis continuam sendo controlados pelos elementos do fluxograma (conforme descrito).

4.5 Entradas Permanentes do Aprendiz

Há a possibilidade do aprendiz armazenar em memória secundária qualquer algoritmo que venha a construir, bem como as REs associadas. Porém, sabendo que há intersecção de informações entre o conjunto representacional utilizado, seria inconveniente armazená-lo

[16]Disponível em https://epascal.dev.java.net.

por inteiro. Portanto, fez-se suficiente gravar informações das duas seguintes REs: o fluxograma estendido e a tabela de variáveis declaradas. Com a observação de que a primeira engloba tanto as instruções quanto os comentários e os pontos de interrupção, frisa-se que todos os demais elementos poderão ser gerados a partir das informações armazenadas.

O método de armazenamento adotado foi a serialização de objetos provida nativamente pela linguagem Java. Tal método permite que qualquer estado particular de uma determinada classe possa ser serializado e, posteriormente, recuperado.

Para conseguir isso, basta à classe implementar a interface[17] denominada *Serializable*. Salvo exceções que exijam tratamento especial, esta interface não possui métodos ou atributos a serem sobrescritos, auxiliando somente na identificação semântica de classes serializáveis.

Salienta-se que não há preocupação com o formato em que os objetos são serializados, pois a linguagem Java armazena uma representação fiel e compatível àquela utilizada internamente no aplicativo (aqui, protótipo). Contudo, como medida de segurança, a linguagem associa a cada classe serializável um número de versão[18]. Assim, durante a releitura de um objeto anteriormente serializado, faz-se a verificação de compatibilidade entre a classe lida e a esperada. O caso negativo incorre em uma exceção manifestando a invalidade da classe[19].

Preferiu-se convencionar a extensão dos arquivos de algoritmos gerados pelo protótipo como *.alg*. Conforme o investigado, apesar de existir aplicativos que utilizem tal extensão, estes abordam domínios muito específicos, tornando bastante remota a conseqüente ocorrência de conflitos quanto a isso.

Versões mais recentes do pacote JGraph permitem a persistência no formato XML[20]. Esta possibilidade vem a ser interessante, visto a consistência da padronização deste formato. Seria vantajoso, por exemplo, que diferentes pesquisas baseadas neste trabalho pudessem compartilhar um padrão unificado. Todavia, à altura em que se noticiou acerca desta nova funcionalidade, a incorporação ficou inviável devido à exigência de mudanças

[17] Fazendo a leitura deste termo no contexto do paradigma Orientado a Objetos.
[18] Internamente denominado *serialVersionUID*.
[19] *InvalidClassException*.
[20] Do inglês *Extensible Markup Language* (Linguagem de Marcação Extensível).

estruturais no projeto. Dessa forma, remete-se este tópico particular aos trabalhos futuros.

Convém lembrar que este é o único módulo que não corresponde a um pacote de classes Java. Tal fato se deve à persistência ser meramente requerida, pelo Gerente de Tarefas, à classe que representa o fluxograma. Assim, o módulo de Entradas Permanentes do Aprendiz restringe-se ao coletivo dos algoritmos armazenados em memória secundária.

Na prática, portanto, o aprendiz poderá então salvar um algoritmo para trabalhar nele posteriormente, manter um repositório particular de exercícios feitos e mesmo compartilhar algoritmos entre seu grupo de estudos. Este último constitui uma ferramenta pedagógica proveitosa, pois pode-se aprender observando-se erros e boas práticas de outrem.

4.6 Gramática do Domínio de Pascal

Refere-se ao módulo responsável por fornecer a gramática da linguagem Pascal para que se possa realizar a transposição do fluxograma para o algoritmo textual. A essência deste módulo concerne à definição de uma meta-linguagem para a transcrição e armazenamento em memória secundária da gramática de Pascal. Além disso, cabe-lhe conseqüentemente prover aos elementos do fluxograma mecanismos de acesso e tradução no que tange a esta meta-linguagem.

Conforme o diagrama de classes subseqüente (figura 4.11), a implementação deste módulo como um pacote Java consiste em duas classes, uma destinada à gramática e outra às instruções específicas que a compõem. A modelagem, que remete ao padrão de projeto *Singleton* [34], garante que exista somente uma instância da classe de gramática e que esta seja globalmente acessível pelo protótipo.

A idéia de implementação da meta-linguagem, por sua vez, tem caráter bastante conciso. Trata, basicamente, da definição de um conjunto fixo de rótulos[21] e da especificação de um grupo de atributos que, após valorados, podem descrever a gramática de uma linguagem de programação.

Os rótulos são todos pré-definidos e constantes, servindo somente para orientar a

[21] Conhecidos mais amplamente por *tags*, aqui empregados na forma <*%nome_do_rótulo*>.

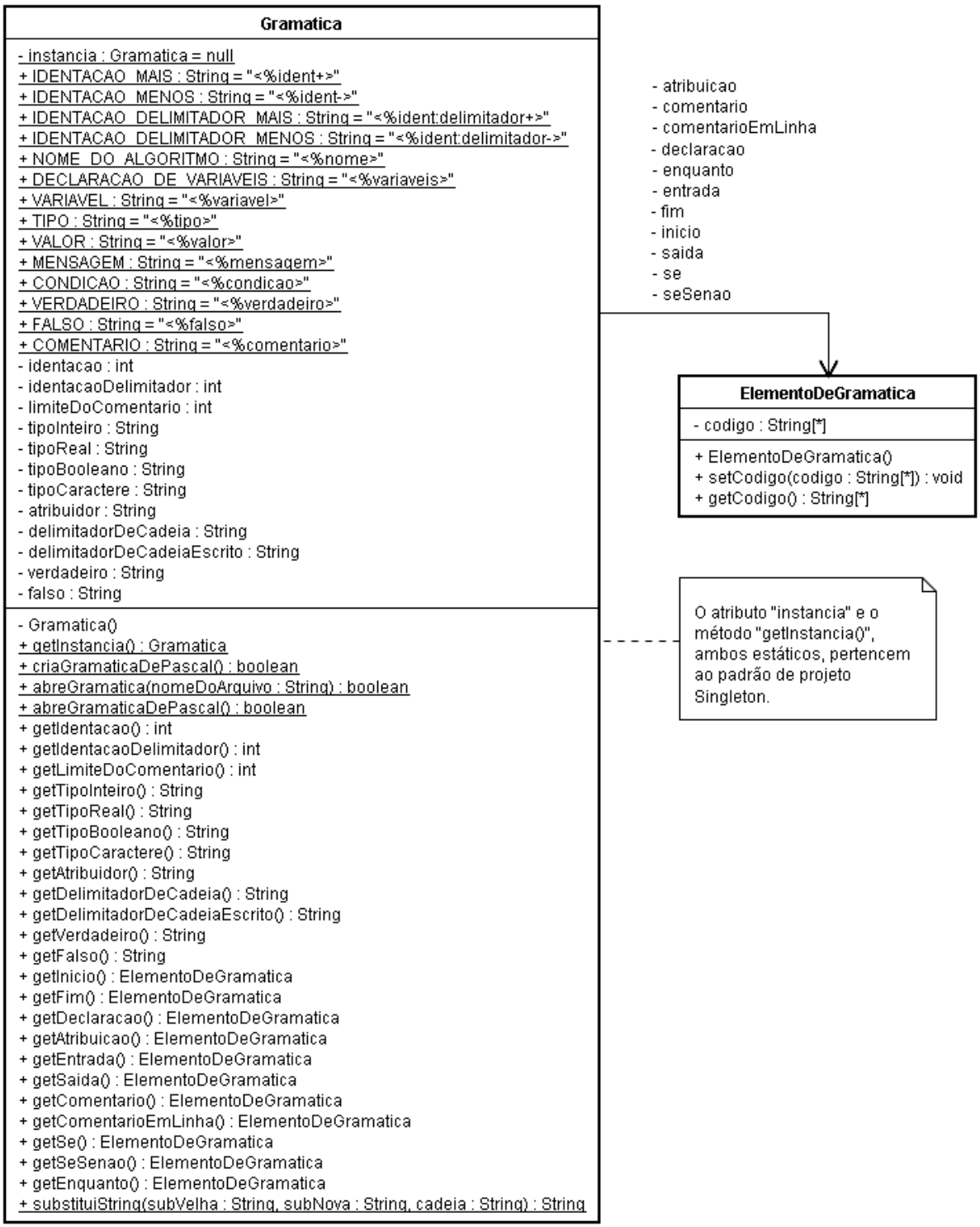


Figura 4.11: Diagrama de Classes do módulo Gramática do Domínio de Pascal

transcrição dos atributos. Estes últimos, por conseguinte, utilizam-se dos rótulos como auxiliares que demarcam elementos de substituição na gramática.

Como exemplo, pode-se citar a instrução *se*, associada a um atributo de mesmo nome, abaixo descrita[22]:

```
1  se.codigo =
2     [
3        "if (<%condicao>) then",
4        "<%ident:delimitador+>",
5        "begin",
6        "<%ident+>",
7        "<%verdadeiro>",
8        "<%ident->",
9        "end;",
10       "<%ident:delimitador->"
11    ];
```

Figura 4.12: Instrução *se* representada pela meta-linguagem adotada

Durante a transcrição para o algoritmo textual, processa-se cada um dos elementos que compõem o vetor correlato a determinada instrução. Vinculadas aos respectivos rótulos, neste caso, são realizadas as seguintes operações:

- *<%condicao>*: substitui este rótulo pela condição de desvio determinada;
- *<%ident+>* e *<%ident->*: respectivamente, avança e retrocede a identação;
- *<%ident:delimitador+>* e *<%ident:delimitador->*: idem ao anterior, porém refere-se à identação dos delimitadores de bloco;
- *<%verdadeiro>*: substitui este rótulo pelo bloco de instruções correspondente à condição verdadeira.

Evidencia-se que, tipicamente, cada elemento do vetor corresponde a uma linha do algoritmo textual. Como exceção ao caso geral, pode-se mencionar as identações, que

[22] A representação do subconjunto da linguagem Pascal abordado está inclusa no Anexo D deste documento.

simplesmente incrementam, ou decrementam, o número de espaços adicionados antes de cada linha.

Por seu turno, o comportamento configurado como padrão para este módulo ocupa-se primeiramente de abrir o arquivo de gramática da linguagem Pascal, determinado *pascal.gra*[23]. Em caso de insucesso, o módulo automaticamente tenta gerar a gramática em memória e então gravá-la neste arquivo. Se ambos falharem, põe-se em uso a gramática que durante a tentativa anterior foi armazenada em memória primária.

Embora o escopo desta dissertação tenha se restringido a um subconjunto de Pascal, destaca-se que futuramente poderá ser contemplada a generalização deste módulo a outras linguagens. No entanto, toma-se como indicativo do potencial de generalização a transcrição da gramática da linguagem ILA (anteriormente introduzida na seção 2.1.4), apresentada ao final do corrente documento (Anexo E).

[23]Optou-se por convencionar a extensão *.gra* aos arquivos de gramática gerados.

CAPÍTULO 5

RESULTADOS ESPERADOS

Neste capítulo são relacionadas as repercussões esperadas do trabalho que se desenvolveu, aqui resumidas a vantagens e limitações identificadas.

5.1 Coleta de Resultados a Partir do Uso Inicial

Ainda que muito se tenha pesquisado para a consolidação da arquitetura e implementação do protótipo, tem-se ciência de que ambos demandarão aperfeiçoamento. Será então através do uso inicial que alguns elementos do arcabouço se mostrarão incompletos e outros indicarão alterações menores. Evidencia-se que a arquitetura proposta está potencialmente apta a atingir os objetivos delimitados e o protótipo, embora não implementado em completude, valida toda a cobertura que a arquitetura provê.

5.2 Generalização da Abordagem Proposta

O mérito superior deste trabalho reside na consolidação de uma abordagem metacognitiva através de um conjunto multirrepresentacional e uma arquitetura correlata. Por tudo isso, pretende-se que a abordagem apresentada possa ser estendida ao ensino de outros tópicos da Ciência da Computação. Por exemplo, o domínio da Teoria da Computação poderia beneficiar-se de um ambiente que se concentrasse na construção de autômatos finitos. De outros tópicos que poderiam também aproveitar desse arcabouço cita-se: métodos de busca (Inteligência Artificial); listas, filas, pilhas e árvores (Estrutura de Dados); e métodos de ordenação (Programação de Computadores).

Ademais, seria bastante desejável que tanto docentes quanto discentes transportassem a experiência proporcionada pela abordagem ao uso pragmático dentro de outras disciplinas. Isto seria feito independentemente da existência de *software* educacional vinculado

à abordagem e influenciaria positivamente o desempenho e a autonomia de quem aprende.

5.3 Dinamização das Aulas e Colaboração

Um resultado bastante rápido que se espera é a dinamização das aulas em que o protótipo seja utilizado. Em geral, ministra-se a disciplina de Programação de Computadores inicialmente com algoritmos manuscritos e depois utilizando-se compiladores. Contudo, consiste em caráter limitativo a pouca motivação que os aprendizes têm para analisar as correções do instrutor ou as soluções de outros colegas. Ambas as tarefas requerem concentração e imersão demandando esforços que o aprendiz custa a despender. Logo, com a utilização do protótipo desenvolvido, pretende-se que este esforço seja reduzido e que o aprendiz possa facilmente examinar algoritmos alheios.

Resultado conseqüente disso seria o fato do instrutor valer-se deste potencial investindo numa abordagem colaborativa de ensino. Tal abordagem vem sendo positivamente relatada na literatura [39] e consiste na troca de conhecimentos e cooperação para se atingir um objetivo comum (no caso, programar melhor). Assim, o instrutor poderia incentivar o compartilhamento de algoritmos (arquivos do ambiente) para que os aprendizes tenham contato com boas práticas do grupo, bem como consigam sanar dúvidas e evitar conceitos errôneos. Com o tempo, os aprendizes podem gradualmente elaborar um catálogo de dúvidas e erros freqüentes para que este seja legado e incrementado pela turma do período/ano seguinte.

5.4 Limitações da Solução Proposta

Nesta seção descreve-se três importantes limitações que se observou até o presente. O suprimento destas será remetido a trabalhos futuros (seção 6.1).

5.4.1 Contemplação de Perícias Avançadas

Conforme a implementação do protótipo, este concerne ao aprimoramento dos princípios de programação e à evolução destes até, conforme [53], perícias básicas (e.g. análise do

problema, integração de estruturas, elaboração de planos primitivos, simulação mental dos estados do computador durante a execução). Não contempla, portanto, o desenvolvimento de perícias mais avançadas (e. g. generalização da solução). Com isso, supõe-se que a utilização do ambiente seja progressivamente abandonada conforme o progresso da aptidão do aprendiz.

5.4.2 Construção de Expressões Lógicas e Aritméticas

No tangente às expressões lógicas e aritméticas, efetua-se uma análise sintática básica (limitada ao balanceamento de parênteses, à verificação de operadores e tipos) a fim de encontrar erros antes da execução. Logo, salienta-se que não há RE ou recurso associado à construção destas expressões. Assim, se o aprendiz empregar uma estrutura condicional *se*, por exemplo, caberá a ele especificar a expressão lógica correspondente à condição de desvio (digitada em campo próprio e obedecendo-se a sintaxe da linguagem Pascal).

5.4.3 Linguagens Imperativistas de Programação

Da forma aqui concebida, o protótipo trata apenas de um subconjunto da linguagem Pascal. Contudo, seria interessante alcançar uma solução genérica que permitisse a alternância de linguagens imperativistas mediante substituição do arquivo que descreve a gramática. Embora a generalização deste módulo não tenha sido levantada por esta pesquisa, consta no Anexo E a transcrição na meta-linguagem criada, da sintaxe de outra linguagem imperativista.

CAPÍTULO 6

CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Apresentou-se, neste documento, uma abordagem metacognitiva utilizando múltiplas representações externas para tratar da aquisição de conhecimento na fronteira de princípios e perícias de programação de computadores. O presente trabalho decorreu da constatação de uma carência de pesquisas no que diz respeito a este nicho específico.

Em particular, enfocou-se a resenha em aspectos concernentes à Representação Externa e a Ambientes Interativos de Aprendizagem. Coube também à primeira elucidar, sob sua perspectiva, temas como "Múltiplas Representações Externas", "Abordagem Metacognitiva" e o "Domínio de Programação de Computadores".

No que tange aos formalismos adotados na solução do problema, apresentou-se o potencial de fluxogramas e do código em Pascal, bem como uma categorização do conjunto de MREs estabelecido. Procurou-se então remeter alguns elementos deste conjunto à taxonomia funcionalista proposta por Ainsworth [2].

Depois disso, descreveu-se cada módulo componente da arquitetura que dá suporte à abordagem deste trabalho. Permeando a explanação destes módulos, existem detalhes de implementação do protótipo que instancia e valida tal arquitetura.

Por fim, sinalizou-se as expectativas depositadas na contribuição desta pesquisa para com o estado da arte em foco. Demarcou-se, além disso, as limitações observadas.

Tem-se consciência de que os objetivos cumpridos perfazem uma contribuição relevante e que o trabalho corrente tem grande potencial, não obstante as limitações relatadas, para enriquecer o ensino do domínio em questão. Contudo, espera-se que esta dissertação seja apenas um ponto de partida no âmbito proposto e que, no futuro, tenha uma pequena dimensão dentro da escala de possibilidades e pesquisas que se espera surgir.

6.1 Trabalhos Futuros

Desde o início do desenvolvimento deste projeto, pôde-se entrever possíveis trabalhos futuros que venham a estendê-lo e a suprir as limitações apresentadas.

- Certamente, a melhoria mais visível consiste em permitir que o aprendiz possa elaborar algoritmos textuais e que, a partir deles, sejam gerados os fluxogramas correspondentes (caminho inverso). Desta forma, contribui-se para um estágio maior dentro do desenvolvimento de perícias do aprendiz, visto que ele poderá escolher uma das duas representações (fluxograma ou textual) para construir seu algoritmo e depois gerar a transposição equivalente na outra representação[1];

- Melhorar as funcionalidades de edição e execução do algoritmo criado. Para edição, faltam mesmo recursos básicos como os tradicionais "recortar e colar". Quanto à execução, a possibilidade de retroceder passos (passo para trás), mesmo que restrita, poderia auxiliar o aprendiz;

- Ampliar o alcance do ambiente para que trate, progressivamente, de outros tipos de dados e estruturas mais complexas. Pois, mesmo as estruturas de repetição *repita-até* e *para*, bem como as cadeias de caracteres, assuntos ainda básicos, não foram abordados. Provavelmente, resida nesta linha o potencial para um trabalho dirigido à disciplina de Estrutura de Dados (com ponteiros e funções);

- Tratar da generalidade do módulo de *Gramática do Domínio*, de modo que possa substanciar outras linguagens imperativistas além do Pascal;

- Dentro do módulo *Entradas Permanentes do Aprendiz*, estudar a persistência no formato XML;

- Implementar subsistemas próprios para auxiliar o aprendiz na construção de expressões lógicas e aritméticas. Para a primeira, especificamente, poderia ser utilizada

[1] Atualmente em desenvolvimento através do trabalho de graduação intitulado "Transposição de Algoritmos em Pascal para Representação Externa Gráfica no Contexto Educacional", proposto por Diego Marczal, acadêmico do curso de Análise de Sistemas da Universidade Estadual do Centro-Oeste.

a representação de portas lógicas. À última, seria interessante uma RE próxima das árvores de derivação, amplamente difundida pela Teoria da Computação, em que, basicamente, os operandos seriam os terminais e os operadores os não-terminais[2];

- Aperfeiçoar, de acordo com os resultados recolhidos, o conjunto de MREs. De imediato, menciona-se:
  - Estender a funcionalidade dos *Pontos de Interrupção*, de modo que possam também ser associados a outros eventos ao longo do conjunto de MREs. Uma melhoria imediata, por exemplo, seria permitir a interrupção vinculada a variáveis e valores atribuídos (e.g. deseja-se interromper quando $a \neq 0$ e $b \geqslant 1$);
  - Aprimorar o *Recolhimento e Expansão de Blocos de Elementos* de modo que não somente restrinja a visualização dentro do algoritmo gráfico, mas que também o faça no código Pascal. Além disso, seria interessante que o bloco de elementos fosse visualmente compactado, ou seja, redimensionado no fluxograma e, provavelmente, substituído por reticências no algoritmo textual;
  - No que diz respeito ao *Destaque de Variáveis com Valores Alterados*, pode-se implementar uma alternativa que represente tais informações como Teste-de-Mesa, em que as variáveis são representadas por colunas de uma tabela cujas linhas são os valores em determinados instantes.
- Abastecer o ambiente de uma galeria de exemplos comentados. Esta poderia não só estimular o aprendiz num primeiro contato, como também proporcionar uma linha-mestra de suporte para que o aprendiz possa conduzir seus estudos;
- Ainda nessa direção, pode-se investir em englobar algum trabalho que sujeite os algoritmos a uma solução de referência, prescrita pelo instrutor, para efeitos de comparação e correção. Um valioso estudo nessa linha foi cumprido por [28];

[2]Também em estágio de desenvolvimento através do trabalho de graduação intitulado "Uso de Múltiplas Representações Externas para o Ensino de Expressões Lógicas no Contexto de Programação de Computadores", proposto por Jabson Cândido, acadêmico do curso de Análise de Sistemas da Universidade Estadual do Centro-Oeste.

- Realizar testes prolongados com aprendizes no intuito de verificar os benefícios de utilização do protótipo. Para isso, haverá necessidade de encontrar ou estabelecer métricas adequadas para aferir o desempenho dos aprendizes frente ao trabalho desenvolvido;

- Conforme as indicativas de testes futuramente coletados, poderia averiguar-se sobre o suporte que a arquitetura fornece a uma abordagem, além de metacognitiva, também colaborativa. Nela poderia, por exemplo, haver comentários particulares e globais (estes, disponíveis para que todo o grupo tivesse acesso). Dependendo do quão fortes sejam os indícios, sugere-se investir também neste âmbito.

As idéias aqui expressas podem ser ambiciosas ou ainda sugerir sobrecarga da arquitetura e do ambiente propostos. Contudo, salienta-se que tais idéias podem evoluir para trabalhos distintos e concomitantes. Ademais, estes trabalhos podem surtir em resultados dos mais diversos, até mesmo, porventura, vantajosos enquanto abordados separadamente e fracos quando conjuntos.

BIBLIOGRAFIA

[1] Shaaron Ainsworth. Designing effective multi-representational learning environments. Relatório Técnico 58, ESRC Centre for Research in Development, Instruction and Training. University of Nottingham, março de 1999.

[2] Shaaron Ainsworth. A functional taxonomy of multiple representations. *Computers and Education*, 33(2/3):131–152, 1999.

[3] Shaaron Ainsworth. DeFT: A conceptual framework for considering learning with multiple representations. *Learning and Instruction*, 2006. (Em publicação).

[4] Shaaron Ainsworth e Nicolas Van Labeke. Using a multi-representational design framework to develop and evaluate a dynamic simulation environment. Dynamic Information and Visualisation Workshop, julho de 2002.

[5] Shaaron Ainsworth e Nicolas Van Labeke. Multiple forms of dynamic representation. *Learning and Instruction*, 14(3):241–255, 2004.

[6] Shaaron Ainsworth e Andrea Th Loizou. The effects of self-explaining when learning with text or diagrams. *Cognitive Science*, 27:669–681, 2003.

[7] Shaaron Ainsworth, David Wood, e Claire O'Malley. There is more than one way to solve a problem: evaluating a learning environment that supports the development of children's multiplication skills. *Learning and Instruction*, 8(2):141–157, 1998.

[8] Varol Akman e Paul J. W. ten Hagen. The power of physical representations. *AI Magazine*, 10(3):49–65, 1989.

[9] M. Beveridge e E. Parkins. Visual representation in analogical problem solving. *Memory & Cognition*, 15(3):230–237, 1987.

[10] J. B. Biggs e P. J. Moore. *The process of learning.* Prentice Hall, New York, 3 ed., 1993.

[11] Michael A. R. Biggs. Visual reasoning: I see what you mean. R. Cooper e V. Branco, editores, *4th European Academy of Design Conference Proceedings*, páginas 314–319. Universidade de Averio, Publisher, 2001.

[12] J. Bonar e R. Cunningham. *Intelligent tutoring systems: Lessons learned*, capítulo BRIDGE: Tutoring the programming process, páginas 409–434. LEA, Hillsdale, NJ, 1988.

[13] C. F. Boyle e J. R. Anderson. Acquisition and automated instruction of geometry skills. *Paper presented at the Annual meeting of the American Educational Research Association (AERA-84)*, 1984.

[14] Mary E. Brenner, Theresa Brar, Richard Durán, e Richard E. Mayer. Learning by understanding: the role of multiple representations in learning algebra. *American Educational Research Journal*, 34(4):663–689, 1997.

[15] A. Brown. *Metacognition, Motivation, and Understanding*, capítulo Metacognition, executive control, self-regulation, and other more mysterious mechanisms, páginas 65–116. LEA, Hillsdale, NJ, 1987.

[16] Jennifer Case e Richard Gunstone. Metacognitive development as a shift in approach to learning: an in-depth study. *Studies in Higher Education*, 27(4):459–470, 2002.

[17] P. Chandler e J. Sweller. The split-attention effect as a factor in the design of instruction. *British Journal of Educational Psychology*, 62:233–246, 1992.

[18] Mei-Hung Chiu. Algorithmic problem solving and conceptual understanding of chemistry by students at a local high school in Taiwan. *Proc. Natl. Sci. Counc. ROC(D)*, 11(1):20–38, 2001.

[19] Tom Conlon. A cognitive tool for classification learning. *International Journal of Continuing Engineering Education and Lifelong Learning*, 11(3):189–201, 2001.

[20] Richard Cox. Representation interpretation versus representation construction: a controlled study using switchERII. B. du Boulay e R. Mizoguchi, editores, *Artificial*

Intelligence in Education: Knowledge and media in learning systems (Proceedings of the 8th World Conference of the Artificial Intelligence in Education Society), páginas 434–441, Amsterdan, Holanda, 1997. IOS.

[21] Richard Cox e Paul Brna. Supporting the use of external representations in problem solving: the need for flexible learning environments. *Journal of Artificial Intelligence in Education*, 6(2/3):239–302, 1995.

[22] Richard Cox, Keith Stenning, e Jon Oberlander. Graphical effects in learning logic: reasoning, representation and individual differences. *Proceedings of the 16th Annual Conference of the Cognitive Science Society*, páginas 237–242. Lawrence Erlbaum Associates, 1994.

[23] Ministério da Educação. *Diretrizes curriculares de cursos da área de computação e informática*. Brasília, 1999.

[24] Jim Davies e Ashok K. Goel. Visual analogy in problem solving. Bernhard Nebel, editor, *Proceedings of the International Joint Conference on Artificial Intelligence*, páginas 377–382. Morgan Kaufmann, 2001.

[25] Jim Davies, Ashok K. Goel, e Nancy J. Nersessian. Visual re-representation in creative analogies. 2003.

[26] A. I. Direne. Intelligent training shells for the operation of digital telephony stations. B. du Boulay e R. Mizoguchi, editores, *Frontiers in Artificial Intelligence and Applications: Proceedings of World Conference on Artificial Intelligence in Education (AI-ED 97)*, páginas 71–78. IOS Press, 1997.

[27] Fehmi Dogan e Nancy J. Nersessian. Design problem solving with conceptual diagrams. B. Bara, L. Barsalou, e M. Bucciarelli, editores, *Proceedings of the 27th Annual Conference of the Cognitive Science Society*, páginas 600–605. Lawrence Erlbaum Associates, 2005.

[28] Gabriel dos Santos. Autoria e interpretação tutorial de soluções alternativas para promover o ensino de programação de computadores. Dissertação de Mestrado, Universidade Federal do Paraná, Curitiba, Paraná, 2003.

[29] Bennedict du Boulay. *Artificial Intelligence Tools in Education*, capítulo Intelligent Systems for Teaching Programming, páginas 5–15. Elsevier Science, 1988.

[30] Jaime Evaristo e Sérgio Crespo. *Aprendendo a Programar: Programando numa Linguagem Algorítmica Executável (ILA)*. Book Express, 2000.

[31] Ana Paula Mendes Correia Couceiro Figueira. Metacognição e seus contornos. *Revista Iberoamericana de Educación*, junho de 2003. http://www.campus-oei.org/revista/deloslectores/446Couceiro.pdf.

[32] J. H. Flavell. *The nature of intelligence*, capítulo Metacognitive aspects of problem solving, páginas 231–235. Lawrence Erlbaum Associates, Hillsdale, N.Y., 1976.

[33] Claudia Gama. Metacognition and reflection in ITS: increasing awareness to improve learning. *Proceedings of World Conference on Artificial Intelligence in Education (AI-ED 2001)*, páginas 492–495, San Antonio, Texas, EUA, maio de 2001. IOS Press.

[34] Erich Gamma, Richard Helm, Ralph Johnson, e John Vlissides. *Padrões de Projeto: soluções reutilizáveis de software orientado a objetos*. Bookman, 2000. Trad. Luiz A. Meireles Salgado.

[35] Merideth Gattis. Mapping relational structure in spatial reasoning. *Cognitive Science*, 28:589–610, 2004.

[36] Gabriela Goldschmidt. The dialectics of sketching. *Creativity Research Journal*, 4(2):123–143, 1991.

[37] Gabriela Goldschmidt. *Design Knowing and Learning: Cognition in Design Education*, capítulo Visual analogy - A strategy for design reasoning and learning, páginas 199–219. New York: Elsevier, 2001.

[38] Terry R. Greene. Children's understanding of class inclusion hierarchies: the relationship between external representation and task performance. *Journal of Experimental Child Psychology*, 48(1):62–89, agosto de 1989.

[39] Teresa Hübscher-Younger e N. Hari Narayanan. Features of shared student-created representations. *External Representations in AIED: Multiple Forms and Multiple Roles - AI-ED 2001 Workshop*, San Antonio, Texas, EUA, maio de 2001. IOS Press.

[40] Asha Jitendra. Teaching students math problem-solving through graphic representations. *Teaching Exceptional Children*, 34(4):34–38, 2002.

[41] J. J. Kaput. *Research issues in the learning and teaching of algebra*, capítulo Linking representations in the symbol systems of algebra. Lawrence Erlbaum Associates, 1989.

[42] Nicolas Van Labeke. Multiple external representations in dynamic geometry: a domain-informed design. *External Representations in AIED: Multiple Forms and Multiple Roles - AI-ED 2001 Workshop*, San Antonio, Texas, EUA, maio de 2001. IOS Press.

[43] Jill H. Larkin e Herbert A. Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11:69–99, 1987.

[44] Richard E. Mayer. *Advances in instructional psychology*, volume 4, capítulo Illustrations that instruct, páginas 253–284. Erlbaum, Hillsdale, NJ, EUA, 1993.

[45] D. C. Merrill, B. J. Reiser, R Beekelaar, e A. Hamid. Making processing visible: scaffolding learning with reasoning-congruent representations. C. Frasson, G. Gauthier, e G. I. McCalla, editores, *Intelligent tutoring systems, Proceedings of the 2nd International Conference, ITS'92*, páginas 103–110, Montreal, Canadá, 1992. Springer-Verlag.

[46] Steven John Metsker. *Padrões de Projeto em Java*. Bookman, 2004. Trad. Werner Loeffler.

[47] Tom Murray, Larry Winship, Roger Bellin, e Matt Cornell. Toward glass box educational simulations: reifying models for inspection and design. *External Representations in AIED: Multiple Forms and Multiple Roles - AI-ED 2001 Workshop*, San Antonio, Texas, EUA, maio de 2001. IOS Press.

[48] Brad. A. Myers. Visual programming, programming by example and program visualization: a taxonomy. *Proceedings of the SIGCHI'86*, páginas 59–66. ACM Press, 1986.

[49] Yukari Nagai e Hisataka Noguchi. How designers transform keywords into visual images. *Proceedings of the ACM Conference on Creativity & Cognition*, páginas 118–125, 2002.

[50] Jan M. Noyesa e Kate J. Garlandb. Solving the Tower of Hanoi: does mode of presentation matter? *Computers in Human Behavior*, 19(5):579–592, setembro de 2003.

[51] S. E. Palmer. *Cognition and categorization*, capítulo Fundamental aspects of cognitive representation, páginas 259–303. LEA, 1978.

[52] Roberto Pedone, John E. Hummel, e Keith J. Holyoak. The use of diagrams in analogical problem solving. *Memory & Cognition*, 29(2):214–221, 2001.

[53] Andrey Ricardo Pimentel e Alexandre Ibrahim Direne. Medidas cognitivas no ensino de programação de computadores com Sistemas Tutores Inteligentes. *Revista Brasileira de Informática na Educação (IE)*, 3:17–24, 1998.

[54] M. Pressley. The relevance of good strategy user model to the teaching of mathematics. *Educational Psychologist*, 21:139–161, 1986.

[55] Sara Price. Animated diagrams: how effective are explicit dynamics for learners? P. Bell, R. Stevens, e T. Satwitz, editores, *Keeping Learning Complex: The Proceedings of the Fifth International Conference of the Learning Sciences (ICLS)*, páginas 344–351, Mahwah, NJ, 2002. Erlbaum.

[56] Célia Ribeiro. Metacognição: um apoio ao processo de aprendizagem. *Psicologia: Reflexão e Crítica*, 16(1):109–116, 2003.

[57] Pablo Romero, Richard Cox, Benedict du Boulay, e Rudi Lutz. A survey of external representations employed in object-oriented programming environments. *Journal of Visual Languages and Computing*, 14(5):387–419, outubro de 2003.

[58] Eileen Scanlon. *Learning with multiple representations*, capítulo How beginning students use graphs of motion, páginas 9–40. Elsevier Science, Amsterdam, 1998.

[59] W. Schnotz. Commentary - towards an integrated view of learning from text and visual displays. *Educational Psychology Review*, 14(1):101–120, 2002.

[60] Robert W. Sebesta. *Conceitos de Linguagens de Programação*. Bookman, 5 ed., 2003.

[61] M. K. Singley, J. R. Anderson, J. S. Gevins, e D. Hoffman. The algebra word problem tutor. J. Breuker D. Bierman e J. Sandberg, editores, *Artificial intelligence and education*, Proceedings of the 4th International Conference on AI and Education, páginas 267–275, maio de 1989.

[62] M. W. Van Someren, P. Bozhimen Reimann, e T. T. de Jong. *Learning with multiple representations*. Elsevier Science, 1998.

[63] R. J. Spiro e J. C. Jehng. *Cognition, education, & multimedia: exploring ideas in high technology*, capítulo Cognitive flexibility & hypertext: theory & technology for the nonlinear & multidimensional traversal of complex subject matter, páginas 163–205. Lawrence Erlbaum Associates, 1990.

[64] Barbara Tversky. What do sketches say about thinking? *AAAI Spring Symposium Series - Sketch Understanding, Stanford University*, páginas 148–152, 2002.

[65] Han Vermaat, Henny Kramers-Pals, e Patricia Schank. The use of animations in chemical education. *Proceedings of the International Convention of the Association*

for Educational Communications and Technology, páginas 430–441, Anaheim, CA, 2003.

[66] Barbara Šketa e Saša Aleksij Glažarb. Using concept maps in teaching organic chemical reactions. *Acta Chim. Slov.*, 52:471–477, 2005. Pedagogical Paper.

[67] Etienne Wenger. *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge*. Morgan Kaufmann, 1987.

[68] Damien J. Williams e Jan M. Noyes. Effect of experience and mode of presentation on problem solving. *Computers in Human Behavior*, novembro de 2004. Publicado online.

[69] B. Winn. *The psychology of illustration*, capítulo Charts, graphs and diagrams in educational materials, páginas 152–198. Springer-Verlag, 1987.

[70] M. Yerushalmy. Student perceptions of aspects of algebraic function using multiple representation software. *Journal of Computer Assisted Learning*, 7:42–57, 1991.

[71] Benjamin Zayas. Learning from 3D VR representations: learner-centred design, realism and interactivity. *External Representations in AIED: Multiple Forms and Multiple Roles - AI-ED 2001 Workshop*, San Antonio, Texas, EUA, maio de 2001. IOS Press.

[72] Jiajie Zhang. The nature of external representations in problem solving. *Cognitive Science*, 21(2):179–217, 1997.

ANEXO A

EXEMPLO: CÁLCULO DA MÉDIA ANUAL

Dadas duas notas semestrais, calcula e exibe a média anual. Apresenta a situação do aluno como: aprovado ($média >= 70$), em exame ($média >= 50$) ou reprovado.

A.1 Código em Pascal

```
1  program media_semestral;
2  var
3     media : real;
4     nota1 : integer;
5     nota2 : integer;
6  begin
7     /*
8      * As variáveis são declaradas através do menu Editar\
9      * Declaração de Variáveis.
10     */
11    writeln('Digite a nota do primeiro semestre:');
12    readln(nota1);
13    writeln('Digite a nota do segundo semestre:');
14    readln(nota2);
15    media := (nota1+nota2)/2;
16    writeln('Sua média é:');
17    writeln(media);
18    if (media >= 70) then
19       begin
20          writeln('Você foi aprovado.');
21       end
22    else
23       begin
24          if (media >= 50) then
25             begin
26                writeln('Você está em exame.');
27             end
28          else
29             begin
30                writeln('Você foi reprovado');
31             end;
32       end;
33 end.
```

A.2 Fluxograma

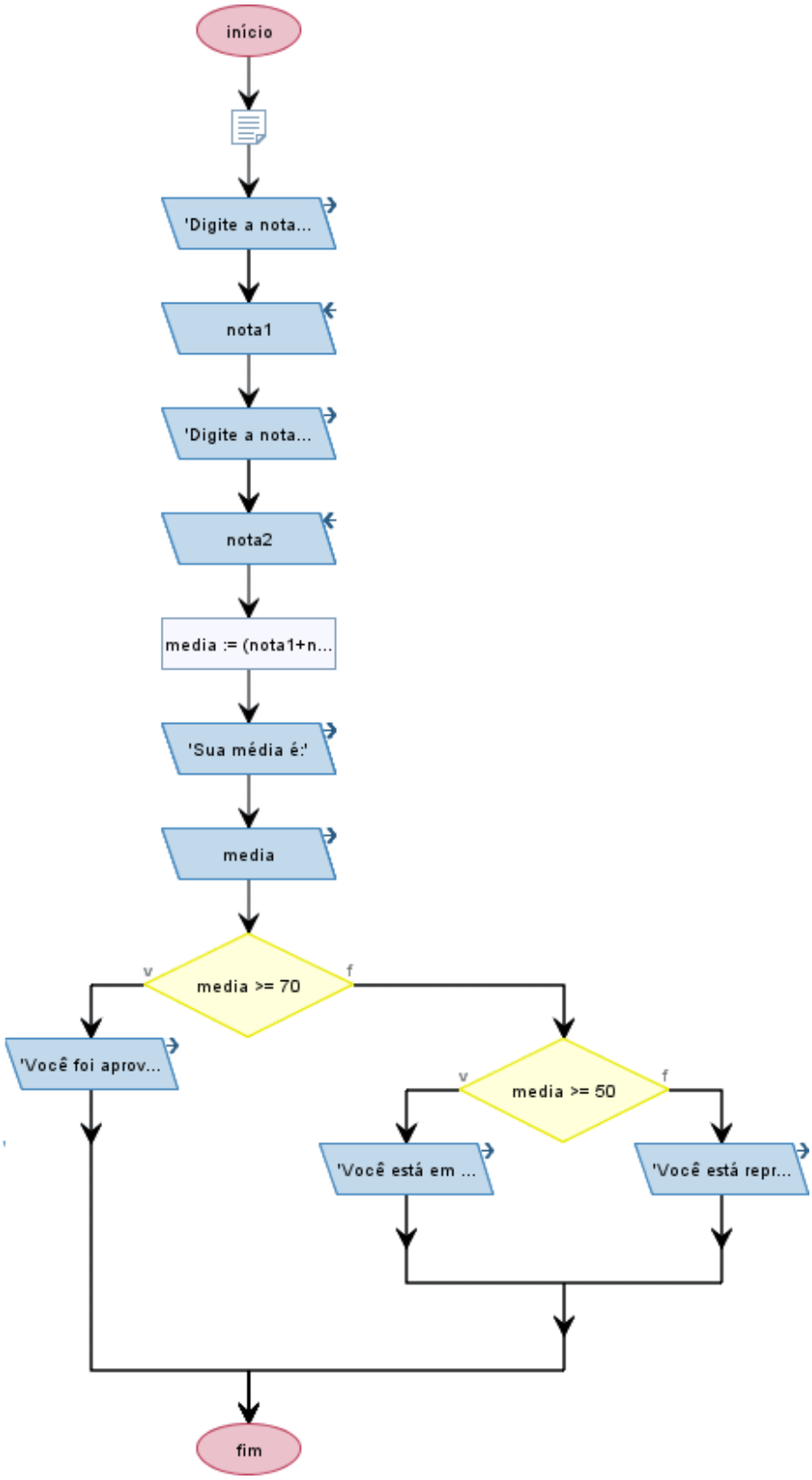


Figura A.1: Cálculo da média anual

ANEXO B

EXEMPLO: CONTAGEM REGRESSIVA

Faz a contagem regressiva de 10 até 1.

B.1 Código em Pascal

```
1  program contagem_regressiva;
2  var
3     i : integer;
4  begin
5     i := 10;
6     writeln('Iniciando contagem regressiva...');
7     while (i > 0) do
8        begin
9           writeln(i);
10          i := i - 1;
11       end;
12    writeln('Fogo!');
13 end.
```

B.2 Fluxograma

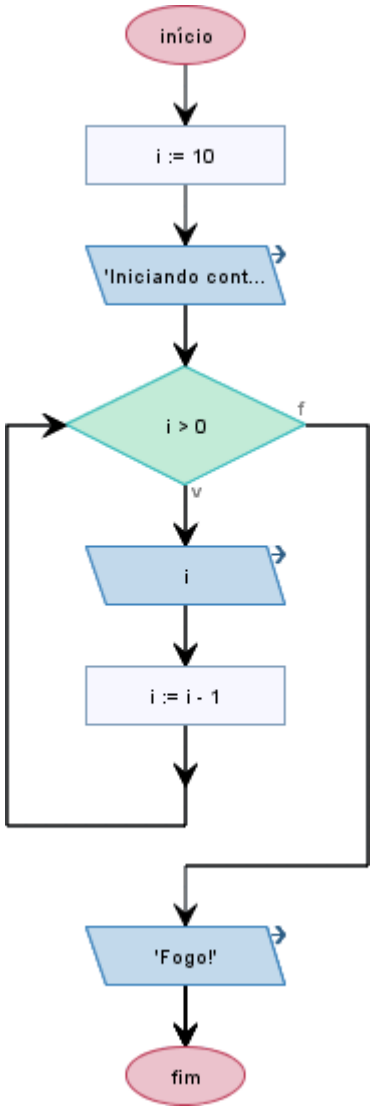


Figura B.1: Contagem regressiva de 10 a 1.

ANEXO C

EXEMPLO: CÁLCULO DE POTÊNCIA

Dados os inteiros *base* e *expoente*, calcula e exibe $base^{expoente}$.

C.1 Código em Pascal

```
1  program calcula_potencia;
2  var
3     base : integer;
4     expoente : integer;
5     potencia : integer;
6  begin
7     writeln('Digite a base:');
8     readln(base);
9     writeln('Digite o expoente:');
10    readln(expoente);
11    if (not(base = 0)) then
12       begin
13          potencia := 1;
14          if (expoente < 0) then
15             begin
16                // Não trabalharemos com números negativos.
17                expoente := -expoente;
18             end;
19          while (expoente > 0) do
20             begin
21                potencia := potencia * base;
22                expoente := expoente - 1;
23             end;
24       end
25    else
26       begin
27          // Evita iteração quando a base for zero. Imagine 0^50
28          potencia := 0;
29       end;
30    writeln('Resultado:');
31    writeln(potencia);
32 end.
```

C.2 Fluxograma

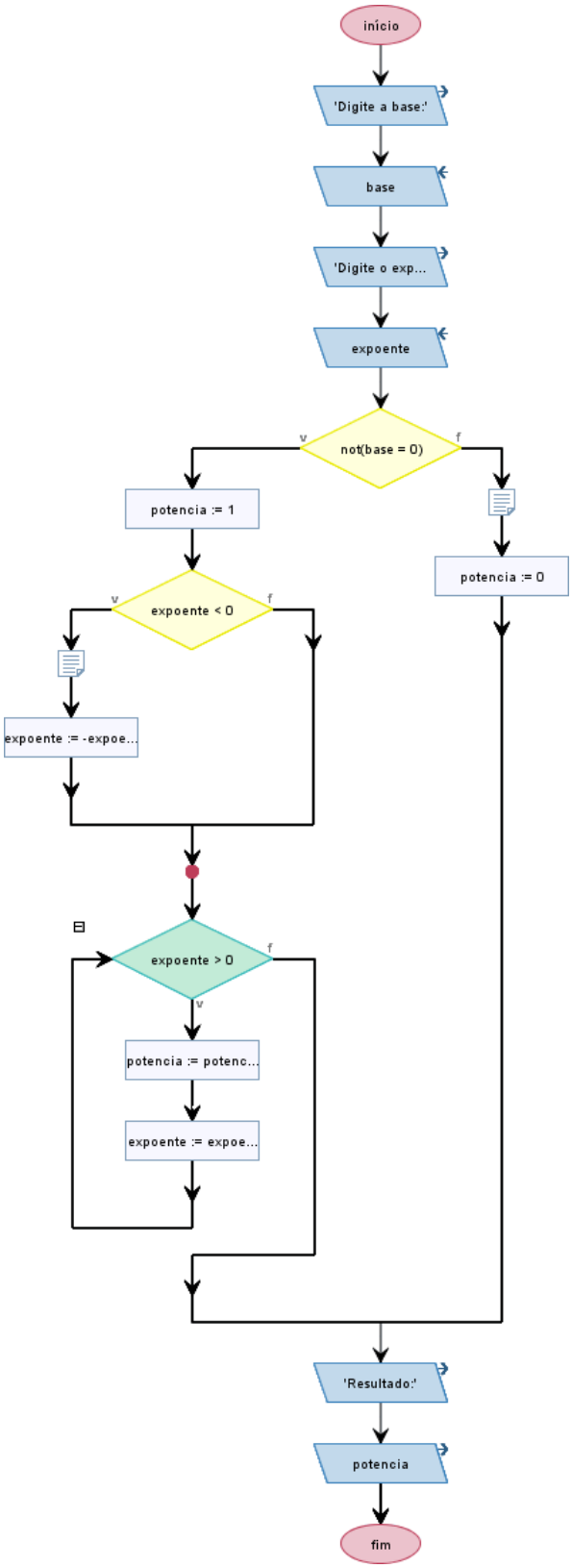


Figura C.1: Cálculo de potência

ANEXO D

GRAMÁTICA DA LINGUAGEM PASCAL

Relaciona-se na seqüência a gramática (do subconjunto abordado) da linguagem Pascal modelada conforme o especificado na seção 4.6. Coube à descrição somente os atributos e os respectivos valores assumidos.

```
identacao = 3;
identacaoDelimitador = 3;
limiteDoComentario = 60;

tipoInteiro = "integer";
tipoReal = "real";
tipoBooleano = "boolean";
tipoCaractere = "char";

atribuidor = ":=";

delimitadorDeCadeia = "'";
delimitadorDeCadeiaEscrito = "aspas-simples";

verdadeiro = "true";
falso = "false";

atribuicao.codigo =
   [
      "<%variavel> := <%valor>;"
   ];

comentario.codigo =
   [
      "/*",
      " * <%comentario>",
      " */"
   ];

comentarioEmLinha.codigo =
   [
      "// <%comentario>"
   ];

declaracao.codigo =
   [
      "var",
      "<%ident+>",
      "<%variavel> : <%tipo>;",
      "<%ident->"
   ];
```

```
42
43 enquanto.codigo =
44    [
45       "while (<%condicao>) do",
46       "<%ident:delimitador+>",
47       "begin",
48       "<%ident+>",
49       "<%verdadeiro>",
50       "<%ident->",
51       "end;",
52       "<%ident:delimitador->"
53    ];
54
55 entrada.codigo =
56    [
57       "readln(<%variavel>);"
58    ];
59
60 fim.codigo =
61    [
62       "<%ident->",
63       "end."
64    ];
65
66 inicio.codigo =
67    [
68       "program <%nome>;",
69       "<%variaveis>",
70       "begin",
71       "<%ident+>"
72    ];
73
74 saida.codigo =
75    [
76       "writeln(<%mensagem>);"
77    ];
78
79 se.codigo =
80    [
81       "if (<%condicao>) then",
82       "<%ident:delimitador+>",
83       "begin",
84       "<%ident+>",
85       "<%verdadeiro>",
86       "<%ident->",
87       "end;",
88       "<%ident:delimitador->"
89    ];
90
91 seSenao.codigo =
92    [
```

```
93          "if (<%condicao>) then",
94          "<%ident:delimitador+>",
95          "begin",
96          "<%ident+>",
97          "<%verdadeiro>",
98          "<%ident->",
99          "end",
100         "<%ident:delimitador->",
101         "else",
102         "<%ident:delimitador+>",
103         "begin",
104         "<%ident+>",
105         "<%falso>",
106         "<%ident->",
107         "end;",
108         "<%ident:delimitador->"
109     ];
```

ANEXO E

GRAMÁTICA DA LINGUAGEM ILA

Segue abaixo a representação da gramática da linguagem ILA (mencionada na seção 2.1.4) equivalente ao subconjunto de Pascal abordado. Este exemplo tem por finalidade, além de ilustrar o emprego da meta-linguagem criada, atuar como um parâmetro de comparação para a modelagem da gramática de Pascal anteriormente listada.

Salienta-se que a única característica não tratada pela meta-linguagem proposta foi a convergência, na linguagem ILA, dos tipos *inteiro* e *real* para um único, denominado *numérico*.

```
identacao = 3;
identacaoDelimitador = 0;
limiteDoComentario = 60;

tipoInteiro = "numerico";
tipoReal = "numerico";
tipoBooleano = "logico";
tipoCaractere = "caractere";

atribuidor = ":=";

delimitadorDeCadeia = "\"";
delimitadorDeCadeiaEscrito = "aspas-duplas";

verdadeiro = "verdadeiro";
falso = "falso";

atribuicao.codigo =
   [
      "<%variavel> := <%valor>"
   ];

comentario.codigo =
   [
      "// <%comentario>"
   ];

comentarioEmLinha.codigo =
   [
      "// <%comentario>"
   ];

declaracao.codigo =
   [
      "variaveis",
      "<%ident+>",
      "<%tipo> <%variavel>",
      "<%ident->"
```

```
39      ];
40
41  enquanto.codigo =
42      [
43          "faca enquanto <%condicao>",
44          "<%ident+>",
45          "<%verdadeiro>",
46          "<%ident->",
47          "fim_enquanto"
48      ];
49
50  entrada.codigo =
51      [
52          "ler <%variavel>"
53      ];
54
55  fim.codigo =
56      [
57          "<%ident->",
58          "fim"
59      ];
60
61  inicio.codigo =
62      [
63          "// Algoritmo <%nome>",
64          "//-------------------------------------------------------------",
65          "<%variaveis>",
66          "inicio",
67          "<%ident+>"
68      ];
69
70  saida.codigo =
71      [
72          "escrever <%mensagem>"
73      ];
74
75  se.codigo =
76      [
77          "se <%condicao> entao",
78          "<%ident+>",
79          "<%verdadeiro>",
80          "<%ident->",
81          "fim_se"
82      ];
83
84  seSenao.codigo =
85      [
86          "se <%condicao> entao",
87          "<%ident+>",
88          "<%verdadeiro>",
89          "<%ident->",
```

```
90          "senao",
91          "<%ident+>",
92          "<%falso>",
93          "<%ident->",
94          "fim_se"
95      ];
```

ELEANDRO MASCHIO KRYNSKI

UMA ABORDAGEM METACOGNITIVA ATRAVÉS DE MÚLTIPLAS REPRESENTAÇÕES EXTERNAS PARA O ENSINO DE PROGRAMAÇÃO DE COMPUTADORES

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Alexandre Ibrahim Direne

CURITIBA

2007