UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

QUALIFICA: UMA FERRAMENTA DE SUPORTE AO DESENVOLVIMENTO DE ALGORITMOS

JEAN FABIO FUCHS

BLUMENAU
2006

2006/1-23

JEAN FABIO FUCHS

QUALIFICA: UMA FERRAMENTA DE SUPORTE AO DESENVOLVIMENTO DE ALGORITMOS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Mauro Marcelo Mattos

BLUMENAU
2006

2006/1-23

QUALIFICA: UMA FERRAMENTA DE SUPORTE AO DESENVOLVIMENTO DE ALGORITMOS

Por

JEAN FABIO FUCHS

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Mauro Marcelo Mattos, Dr. – Orientador, FURB
Membro:	Prof. Everaldo Arthur Grahl, Ms. – FURB
Membro:	Prof. Paulo Roberto Dias, Ms. – FURB

Blumenau, 18 de Junho de 2006

Dedico este trabalho à minha família que me apoiou e deu forças para concluir a realização deste.

AGRADECIMENTOS

À Deus, pelo seu imenso amor e pela sua infinita graça.

À minha família, que sempre me apoiou mesmo longe, sempre esteve presente.

Ao meu orientador, professor Mauro, por ter acreditado na conclusão deste trabalho.

Não se aparte da tua boca o livro desta lei; medita nele dia e noite, para que tenhas o cuidado de fazer conforme tudo o que nele está escrito. Então tu farás prosperar o teu caminho e serás bem-sucedido.

Josué 1:8

RESUMO

O presente trabalho descreve os fundamentos para a utilização de técnicas de desenvolvimento estruturado de programas na construção de uma ferramenta de apoio ao ensino de desenvolvimento de algoritmos. O sistema está baseado em dois conceitos principais: uma especificação detalhada do problema por parte do professor e uma análise detalhada da especificação por parte do aluno. Tanto a especificação do problema como a análise por parte do aluno será conduzida através de um processo automatizado. O resultado da análise por parte do aluno é um pseudocódigo contendo uma possível proposta de solução para o problema.

Palavras chaves: Algoritmos. Lógica de programação.

ABSTRACT

This work describes the main aspects for the use of structured development techniques in the construction of a tool for teaching introductory programming classes. The system is based on two main concepts: a detailed specification of the problem by the teacher and a detailed analysis of the specification by the student. The process of the analysis is driven through an automated process by the tool. The result of the analysis is a pseudo code that contains a possible solution for the problem.

Key-words: Algorithms. Programming logic.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE SIGLAS

CLIPS – C Language Integrated Production System

COBOL – Common Business Oriented Language

DOS – Disk Operating System

FURB – Fundação Universidade Regional de Blumenau

MIT – Massachussets Institute of Technology

RBC – Raciocínio Baseado em Casos

SUMÁRIO

1 INTRODUÇÃO

Os recursos de informática vem sendo gradativamente incorporados na educação, na medida em que se verifica uma tendência de assimilar as novas tecnologias de informação e comunicação disponíveis. Conforme Mantovani et al. (2000) as tecnologias da informação e da comunicação apontam, nos meios educacionais, novos rumos de trabalho. Cabe aos educadores utilizar adequadamente os recursos dessas tecnologias e explorar seu potencial pedagógico, tendo em vista a configuração de novos ambientes de ensino e aprendizagem.

De acordo com Casas (1999), a pedagogia em ciências de educação está baseada em dois princípios: (a) a instrução pode desenvolver as habilidades do aprendiz para que compreenda intuitivamente como funciona o mundo natural em vez de inculcar-lhe a representação formal e as habilidades de raciocínio que os cientistas usam; (b) a instrução que pode ajudar o aprendiz a desenvolver o seu modelo mental (existente) para uma concepção mais exata da realidade.

Segundo Mattos, Fernandes e Lopez (1999), os estudantes que iniciam um curso de Graduação em Informática, normalmente encontram uma primeira dificuldade relacionada com a disciplina de Introdução a Programação (ou com nome similar), cujo principal objetivo é o de introduzir os conceitos básicos de lógica de programação. Esta dificuldade é na maioria das vezes decorrente da falta de experiência com os aspectos relacionados a ambientes industriais e/ou comerciais, pois é a partir destes ambientes que são caracterizados os exercícios propostos.

> Analisando-se o perfil dos alunos, verifica-se que em sua maioria, são oriundos do 2º Grau, e portanto, possuem conhecimentos abstratos sobre áreas científicas (matemática, física, biologia, e outros.). Porém, quando deparam-se com a descrição textual dos enunciados dos problemas apresentados nesta disciplina introdutória, geralmente encontram dificuldades em identificar como extrair as informações necessárias para iniciar a solução destes problemas. (MATTOS, 2002).

Esta constatação é corroborada em Kovacic (2003, p. 794).

Conforme Carvalho e Chiossi (2001, p. 19), quando os requisitos não são totalmente compreendidos, registrados e comunicados para a equipe de desenvolvimento, pode haver discrepância entre o que o sistema construído faz e o que deveria fazer. Estas discrepâncias no contexto de introdução à programação conduzem ao desenvolvimento de soluções erradas (na melhor das hipóteses) ou mesmo à dificuldade no desenvolvimento de uma solução (mesmo que incorreta).

As pesquisas do Prof. Mauro Mattos na área de ferramentas de apoio ao ensino de introdução a programação vêm sendo desenvolvidas desde 1999 (MATTOS, 1999); MATTOS; WANGENHEIM, 1999; MATTOS, 2000ab; MATTOS, 2002). Neste período as pesquisas avançaram no sentido da utilização de técnicas de Inteligência Artificial (IA) aplicadas como ferramental de apoio ao desenvolvimento de um método de ensino de algoritmos a partir da análise de enunciados de problemas específicos e da condução do aluno no sentido da construção de um esboço da solução algorítmica.

Kovacic (2003) apresenta os resultados de um estudo empírico realizado em 2003, na *Open Polytechnic of New Zeland*, sobre estilos de aprendizagem aplicado a um grupo de estudantes de computação e os estilos de ensino dos respectivos professores. As conclusões do autor corroboram os resultados obtidos por Felder (FELDER, 1993 apud KOVACIC, 2003, p. 795) enfatizando que, estudantes cujo estilo de aprendizagem é compatível com o estilo de ensinar de seus professores tendem a reter melhor as informações, obter melhores notas e manter interesse no curso. Ainda segundo Kovacic (2003, p. 802), a discrepância entre o estilo de aprendizagem e o estilo de ensino deve ser considerada como um obstáculo a ser superado e conduzir a mudanças nas práticas de ensino.

Conforme Mattos (2005), um outro obstáculo ao aprendizado de introdução à programação está relacionado também à ausência de um método padrão de ensino em diferentes turmas desta mesma disciplina, o que limita a possibilidade de interação entre alunos de turmas diferentes da mesma fase (interação horizontal) e a possibilidade de interação entre alunos de semestres diferentes (interação vertical).

A hipótese de pesquisa consiste em analisar a eficácia da adoção de uma ferramenta de análise de dados que permita a especificação de um esboço de solução para problemas de introdução à programação na perspectiva da construção de um método unificado de ensino de desenvolvimento de algoritmos nos cursos de computação da FURB.

A presente proposta possui dois aspectos motivadores: os métodos de especificação de programas COBOL e o método de programação estruturada de Jackson.

O primeiro aspecto está relacionado ao método de especificação de programas COBOL (Figura 1). Na época era comum a utilização de gabaritos de descrição de leiaute de registro e gabaritos de descrição de leiaute de relatório de saída como método de especificação de programas.

A Figura 1 apresenta um exemplo obtido em Brown (2003) e caracteriza a especificação de um exercício da disciplina de Introdução a Programação COBOL.

P1: Program #1 Specifications

Due Thurs. Feb 13, 2003 at the beginning of class (week 5)
Reference: Chapters 1-6 in Stern & Stern; and, "COBOL Programming Standards" on my web page

Specifications:
Write a program for the Acme Hardware Company to create a formatted aging accounts receivable report from an input file containing customer data. Calculate the total balance due for each customer, including a finance charge on balances over 60 days old. The finance charge is a flat $1 on a balance over 60 days old.
Note on Finance Charge: Calculating a finance charge must be a separate module in your code. This will be a complex calculation in the next assignment.

Input: The customer record includes the customer number (the two leading digits are the year the customer account was created), customer name, billing address, 60+ day balance, 30-60 day balance, and new purchases.

Customer record (input)		length = 90	
Field	Size	Type	decimal positions
customer number	7	numeric	0
customer name	20	alphanumeric	--
address info	45	alphanumeric	--
balance over 60 days	6	signed numeric	2 (implied)
balance betw 30-60 days	6	signed numeric	2 (implied)
balance < 30 days	6	signed numeric	2 (implied)

a) especificação de leiaute de registro

Output: The formatted report will contain properly formatted output and the following:

- standard heading including your name and the date the report is generated.
- report title
- column headers
- enhancement lines (underlines, stars, dashes)
- column totals
- summary of counters for number of records read and number of customer records written
- end-of-report line

b) especificação de leiaute de relatório

The following is a sample of a possible report layout
(yours may vary slightly but must contain all components shown):

```
Analyst:  (your name)              COP2120-098 Spring 2003                      Report date:   mm/dd/yyyy
                                     Acme Hardware:  Aging Accounts Receivable
****************************************************************************************************
                                                                                                    new
Cust #         Name                       over 60          30-60          purchases     FC          Balance
-----------   ------------------------   ------------   ------------   ------------   -------   -----------------
---
99-99999     Jones Contracting           z,zz9.99-       z,zz9.99-      z,zz9.99-     z9.99      $z,zzz,zz9.99CR
99-99999     Smith Bros Inc.             z,zz9.99-       z,zz9.99-      z,zz9.99-     z9.99      $z,zzz,zz9.99CR
99-99999     xxxxxxxxxxxxxxxxxxxx        z,zz9.99-       z,zz9.99-      z,zz9.99-     z9.99      $z,zzz,zz9.99CR
----------------------------------------------------------------------------------------------------
--
(page break)

Analyst:  (your name)              COP2120-098 Spring 2003                      Report date:   mm/dd/yyyy
                                     Acme Hardware:  Aging Accounts Receivable
****************************************************************************************************
     Summary:
                             Totals:
                                        over 90 days                  $z,zz9,999.99-
                                        30-60 days                    $z,zz9,999.99-
                                        less than 30 days             $z,zz9,999.99-
                                        --------------------------------------------
                                        TOTAL Accounts Receivable: $*,***,***,**9.99CR

   Statisical Information:
                                        Number of customer records read:       999
                                        Number of customer records written:    999
                                        Number of records in error:            999


*************************************end of report**************************************************
```

The report will be in landscape format. There are 100 characters on each line; the columns are separated by at least 3 spaces. Use blank lines to make the data easier to read. There are two blank lines before the end-of-report line. Print no more than 30 lines per page. Begin the summary report on a new page. [Landscape = 100 chars x 30 lines]

Refer to the "COBOL Programming Standards" document on the class web page for the standards expected in this class.

Fonte: Brown (2003).

Figura 1 – Modelo de especificação de programas em COBOL

Na Figura 1 é possível identificar as duas características citadas anteriormente, ou seja: a) declaração explícita da estrutura de dados a ser utilizada (leiaute de registro) e, b) a declaração explícita do formato da saída (leiaute de relatório) esperada para o programa. Isto permite ao programador analisar o problema a partir de exemplos de dados de entrada e do leiaute de saída esperado.

Segundo Mattos (1995), estas duas características estão associadas com a estrutura da linguagem. Em COBOL há a necessidade de declaração de uma *File Section* no programa fonte COBOL onde o leiaute do registro do arquivo a ser utilizado deve ser descrito. Além disso, há uma seção na *Working-storage section* que permite estabelecer o leiaute de cada linha-detalhe (em COBOL a saída em relatório demanda que cada linha com cabeçalho ou com dados a ser escrita tenha que ser previamente definida em termos de campos e *pictures* de campos) a ser escrita no relatório de saída.

O segundo aspecto está relacionado ao método de programação estruturada de Jackson o qual pode ser resumido em três etapas (JACKSON, 1988, p. 39):

a) considere o ambiente do problema e anote a maneira como ele é compreendido, definindo estruturas para os dados a serem processados;
b) forme uma estrutura de programa baseada nas estruturas de dados;
c) defina a tarefa a ser executada em termos das operações elementares utilizáveis, e aloque cada uma dessas operações a componentes apropriados da estrutura do programa.

Como se pode observar, a estrutura da linguagem COBOL facilita a adoção da metodologia de Jackson uma vez que o programa é desenvolvido a partir da análise e detalhamento dos dados do programa.

O projeto apresenta a fundamentação teórica e a descrição da arquitetura de uma ferramenta que, de um lado, possibilite ao professor de introdução a programação construir uma especificação mais detalhada do problema a ser resolvido pelo aluno (envolvendo a definição das estruturas de dados a serem utilizadas e o leiaute de saída esperado) e, de outro lado, permita ao aluno realizar uma análise da especificação do problema e a produção de um esboço de solução algorítmica que solucione o problema proposto.

Para a produção do pseudocódigo do esboço da solução algorítmica, foi escolhido o Portugol, por ser muito utilizado na descrição de algoritmos, possibilitando o uso de comandos escritos na língua portuguesa facilitando o ensino, introduzindo o aluno iniciante

dentro do formalismo da lógica e da programação. Conforme Saliba (1992), a forma de representação de algoritmo em Portugol é muito rica em detalhes, como a definição dos tipos das variáveis e, por assemelhar-se bastante à forma em que os programas são escritos.

1.1 OBJETIVOS

O objetivo deste trabalho consiste na construção de uma ferramenta que permita a especificação detalhada de um problema de introdução a programação e a solução do mesmo por parte do aluno através da análise dos dados de entrada e das saídas esperadas.

Os objetivos específicos são:

a) construção de um módulo professor que permita ao mesmo não só descrever em termos textuais o enunciado do problema (em pelo menos 3 níveis de dificuldade – básico, intermediário e avançado) mas detalhar as estruturas de dados a serem utilizados na solução do mesmo;
b) construção de um módulo aluno que possibilite a carga da especificação do problema proposto pelo professor e conduza o mesmo através de uma série de passos de refinamento de modo a permiti-lo construir um esboço de solução algorítmica para o problema.

Quanto aos níveis de dificuldade, quando o exercício for de nível básico, os dados de entrada serão armazenados em variáveis locais de modo que o aluno não poderá qualificar duas ou mais entradas com o mesmo nome, quando o exercício for intermediário as entradas serão armazenadas em vetores e de nível avançado em estruturas do tipo *record*.

1.2 RELEVÂNCIA DO TRABALHO

As pesquisas realizadas pelo Prof. Mauro Mattos na área de ferramentas de apoio ao ensino de introdução à programação levaram à constatação de que na base do problema de dificuldade de aprendizagem nesta disciplina está a dificuldade dos alunos em visualizar exemplos concretos de dados de entrada e de saídas esperadas que facilitem o processo de análise. Neste sentido, a relevância deste projeto está em: (a) complementar um trabalho de pesquisa de longo prazo (1999-2005) que vêm sendo desenvolvido pelo Prof. Mauro Mattos atacando uma hipótese específica de pesquisa centrada na dificuldade de compreensão do enunciado dos problemas de introdução à programação e (b) a adoção de uma técnica de especificação com abordagem de dados como ferramental de apoio ao ensino.

Deve-se destacar que o objetivo das pesquisas do Prof. Mauro Mattos é o de desenvolver uma ferramenta que conduza o aluno no processo de descoberta do conhecimento de como desenvolver algoritmos computacionais e não obter uma ferramenta que construa automaticamente soluções corretas para tais problemas. Com isto pretende-se contribuir para melhorar o nível do aprendizado dos alunos de introdução à programação.

1.3 ESTRUTURA DO TRABALHO

Este primeiro capítulo de introdução apresentou uma contextualização do trabalho, destacando e apresentando o assunto correspondente bem como os objetivos almejados.

O capítulo 2 apresenta a problemática de ensino de introdução a programação de uma forma geral e mais especificamente no contexto do projeto do Prof. Mauro Mattos nesta área que é onde o projeto atual está inserido.

O capítulo 3 discorre sobre o desenvolvimento estruturado de sistemas tendo em vista apresentar o embasamento filosófico para o modelo construído.

O capítulo 4 apresenta a especificação do sistema construído e o capítulo 5 apresenta um estudo de caso descrevendo o funcionamento da ferramenta.

O capítulo 6 apresenta as considerações finais, limitações e futuras extensões.

Nos apêndices A e B estão apresentados os diagramas de classes detalhando, respectivamente, dos módulos Professor e Aluno e o apêndice C apresenta um exemplo das telas utilizando o módulo de internacionalização.

2 O CONTEXTO DO PROJETO: A PROBLEMÁTICA DE ENSINO DE INTRODUÇÃO A PROGRAMAÇÃO

Este capítulo discorre sobre a questão das dificuldades no ensino de introdução a programação.

2.1 INTRODUÇÃO

> É verdade que, de abstracção em abstracção, muita gente advoga que ao ensinar Programação, devemos mesmo abstrair-nos da linguagem e concentrar-nos nos aspectos de resolução de problemas. E a linguagem de programação torna-se assim um pormenor secundário. (GUERREIRO, 1986).
>
> Quem pensa desse modo, usa descontraidamente uma pseudo-linguagem informal para exprimir os algoritmos que vão aparecendo. É esta atitude que detectamos quando nos currículos de cursos de programação de algumas universidades encontramos, depois da enumeração dos tópicos a tratar, vem a indicação subsidiária de que a linguagem-veículo é a linguagem X ou Y. A idéia é que se poderia mudar de linguagem mantendo a essência do curso. Pensamos que esta atitude é errada, pois, para além de tolerar um certo desleixo conceptual e notacional, passa ao lado de uma série de questões fundamentais que não tem existência real fora das linguagens: como é o mecanismo de passagem de parâmetros? Que acontece quando o índice de um vetor está fora dos limites? Como se detecta o fim de um ficheiro? Como se avalia uma expressão lógica? Embora estas questões possam ser descritas em abstrato, elas só fazem sentido se estivermos a pensar nalguma linguagem. Se não dermos importância a isto logo na origem, os estudantes sub-valorizarão pontos fundamentais da programação, e adotarão indesejáveis hábitos de falta de rigor. (GUERREIRO, 1986).

Conforme Carvalho e Chiossi (2001, p.19), quando os requisitos não são totalmente compreendidos, registrados e comunicados para a equipe de desenvolvimento, pode haver discrepância entre o que o sistema construído faz e o que deveria fazer. Estas discrepâncias no contexto de introdução à programação conduzem ao desenvolvimento de soluções erradas (na melhor das hipóteses) ou mesmo à dificuldade no desenvolvimento de uma solução (mesmo que incorreta).

> Actualmente, muitos dos alunos que chegam à universidade em Portugal para os cursos de informática ou outros de engenharia já conhecem o Pascal. Aparentemente, é esta a linguagem mais usada no ensino secundário, nas escolas que têm Informática. Na realidade, verifica-se que, na quase totalidade dos casos, o conhecimento é bastante superficial: não só os estudantes ignoram aspectos importantes da linguagem, como a utilização que dela fazem é rudimentar: por exemplo, é freqüente encontrar longas cadeias de if-then-elses, quase sempre as variáveis aparecem todas declaradas globalmente, os procedimentos e funções são ad-hoc, há ciclos imbricados com uma profundidade excessiva, etc. Além disso, desconhecem quase completamente os principais algoritmos e estruturas de dados. E por fim, não só geralmente não têm nenhum método, como adaptam alegremente a perigosa filosofia de que ‘o que é preciso é que funcione’. Seja como for, os estudantes novatos pensam que, de Pascal, e, por conseguinte (nas suas cabeças...) de Programação, já sabem quase tudo. (GUERREIRO, 1986).

Esta problemática vem sendo estudada pelo Prof. Mauro Mattos na FURB desde 1998 e os resultados não diferem muito. Vale destacar que, a publicação do Prof. Guerreiro é de

1986 e as pesquisas do Prof. Mattos iniciaram-se em 1998, portanto, 12 anos após. Um aspecto relevante é que embora as realidades sejam ligeiramente diferentes: na FURB os alunos, via de regra, não conhecem nenhuma linguagem de programação ao ingressarem nos cursos de Computação. Contudo, os problemas apontados pelo Prof. Guerreiro como as baixas qualidades no aprendizado em programação, são constatados pelos professores de 2º semestre ao receberem os alunos oriundos da disciplina de introdução a programação. Portanto, na realidade da FURB em particular desloca-se o problema um semestre adiante em relação ao problema encontrado em Portugal.

> Os estudantes que iniciam um curso de Graduação em Informática, normalmente encontram uma primeira dificuldade relacionada com a disciplina de Algoritmos (ou com nome similar), cujo principal objetivo é o de introduzir os conceitos básicos de lógica de programação. Esta dificuldade é na maioria das vezes, decorrente da falta de experiência com os aspectos relacionados aos ambientes industriais e/ou comerciais, pois é a partir destes ambientes que são caracterizados os exercícios propostos. Analisando-se o perfil dos alunos, verifica-se que em sua maioria, são oriundos do 2º Grau, e portanto, possuem conhecimentos abstratos sobre áreas científicas (matemática, física, biologia, etc.). Porém , quando deparam-se com a descrição textual dos enunciados dos problemas apresentados nesta disciplina introdutória, geralmente encontram dificuldades em identificar como extrair as informações necessárias para iniciar a solução destes problemas. (MATTOS, FERNANDES e LÓPEZ, 1999).

Esta é uma tônica constante no que se refere a bibliografia relacionada nas ementas de disciplinas de Introdução a Programação – e na FURB não é exceção.

> Apesar de ser reconhecido pela comunidade de Engenharia de Software a importância da adoção de métodos (se não formais, pelo menos rigorosos) de desenvolvimento de sistemas de software, via de regra as ementas das disciplinas introdutórias de programação concentram seus esforços na construção das abstrações que conduzam a uma solução algorítmica sem preocupar-se com os aspectos formais. Além disso, os exemplos propostos, via de regra, são estanques e não fazem parte do contexto de um problema mais complexo. Assim, o aluno no primeiro semestre é treinado para resolver problemas simples e, neste contexto, não faz sentido para o aluno entender a necessidade da adoção de métodos de desenvolvimento de software. Quando o aluno chega no 4º semestre e é apresentado ao conjunto de formalismos (diagramas, metodologias), ele já está cheio de vícios e, como diz o Prof. Guerreiro, acredita que sabe tudo de programação. (MATTOS, 1999).

Após vários experimentos, conduzidos pelo Prof. Mattos na tentativa de construir uma ferramenta para auxílio ao ensino de programação, utilizando Sistemas Especialistas, Raciocínio Baseado em Casos e geração e animação de algoritmos foi possível constatar que, um dos grandes problemas que ainda não havia sido atacado era o da questão da adequada formalização dos enunciados dos problemas de introdução a programação.

A partir do projeto iniciado na FURB pelo prof. Mauro Mattos em 1998, permitiu o desenvolvimento de uma série de experimentos, os quais são resumidamente apresentados a seguir.

2.2 OS EXPERIMENTOS

Conforme citado anteriormente, foram conduzidos vários experimentos pelo Prof. Mattos na perspectiva de construir uma ferramenta para auxílio ao ensino de programação. A seguir são apresentados os resultados parciais destes esforços.

2.2.1 UTILIZAÇÃO DE SISTEMAS ESPECIALISTAS

Conforme Mattos, Fernandes e López (1999), analisando-se o perfil dos alunos de introdução a programação na FURB, verifica-se que em sua maioria, são oriundos do 2º Grau, e, portanto, possuem conhecimentos abstratos sobre áreas científicas (matemática, física, biologia, etc.). Porém, quando se deparam com a descrição textual dos enunciados dos problemas apresentados nesta disciplina introdutória, geralmente encontram dificuldades em identificar como extrair as informações necessárias para iniciar a solução destes problemas.

A partir da constatação do problema, observou-se que, quando induzidos a pensar sobre o problema através de perguntas direcionadas, em sua grande maioria, os alunos conseguem descrever a solução “intuitivamente”, ou seja, sem o formalismo necessário à área de computação (MATTOS, FERNANDES e LÓPEZ, 1999).

A partir da análise e observação do processo de indução acima citado, iniciou-se no 1º semestre de 1998, na FURB, um trabalho no sentido de formalizar-se através de uma metodologia, este processo de indução, com o objetivo de conduzir o aluno no processo de busca da solução dos problemas propostos. Este processo foi utilizado ao longo do referido semestre e ao final do mesmo, os alunos avaliaram a proposta como sendo muito útil, principalmente porque estabelecia uma receita de bolo (segundo alguns), ou seja, um procedimento formal a ser seguido.

> Em função disto, optou-se pelo refinamento da proposta e pelo desenvolvimento de um sistema especialista que a partir do conhecimento informal sobre o processo de indução, permitisse a construção de uma ferramenta de software que permitisse ao aluno, na ausência do professor, desenvolver o processo de aprendizado dos conceitos básicos da disciplina. (MATTOS, FERNANDES e LÓPEZ, 1999).

2.2.2 A AQUISIÇÃO DO CONHECIMENTO

Como citado anteriormente, na medida em que verificou-se que o processo de indução apresentava resultados, procurou-se identificar quais perguntas e em que seqüência elas eram apresentadas no sentido de conduzir a turma em direção a uma possível solução a algum dos problemas propostos. A partir daí confeccionou-se o que foi caracterizado pelos alunos como: a metodologia dos oito passos. Esta primeira proposta procurava fazer com que o aluno

respondesse às seguintes questões:

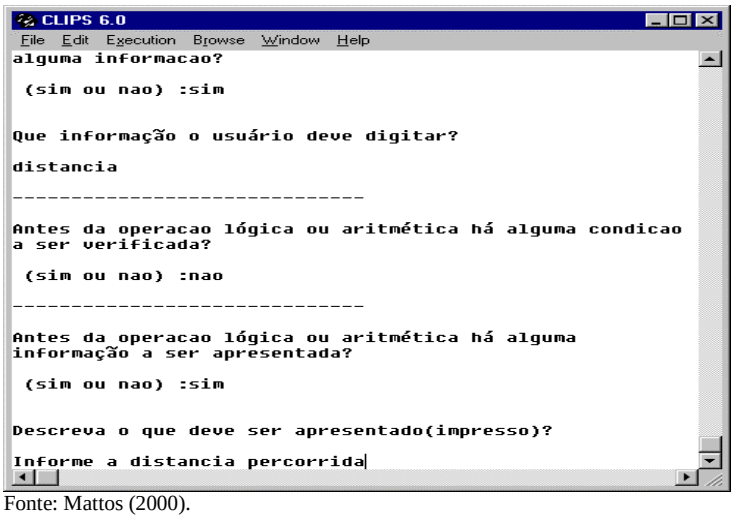
a) quais as “variáveis” conhecidas? – ou seja, quais as informações que podem ser obtidas a partir do enunciado do problema;
b) o que precisa ser calculado (ou executado)? – identificação do escopo da aplicação;
c) quais as “variáveis” desconhecidas? - ou seja, variáveis para as quais não há informação no enunciado do problema;
d) o que precisa ser informado (digitado) – ou seja, o que o usuário da aplicação precisa informar para que o mesmo realize suas funções;
e) o que precisa ser apresentado (impresso) – ou seja, o que a aplicação deve apresentar como resultado;
f) realizar um esboço da solução – ou seja, identificar em termos de macro passos, qual a estratégia a ser adotada para solução do problema;
g) construir um fluxograma – efetivamente elaborar um fluxograma, utilizando a notação gráfica para representar a lógica da solução;
h) construir o teste-de-mesa – excitar as variáveis a partir da execução dos comandos na seqüência em que aparecem no gráfico.

Conseqüentemente, após a realização do teste-de-mesa e, verificada sua correção, a solução está pronta para ser codificada em alguma linguagem alvo.

Conforme Mattos, Fernandes e López (1999), “embora facilitasse mais o encaminhamento da solução, a metodologia acima descrita ainda carecia de um detalhamento melhor, porque havia obviamente muitos passos "escondidos" entre a passagem das fases de (a) a (e) para a fase (f) – esboço da solução”.

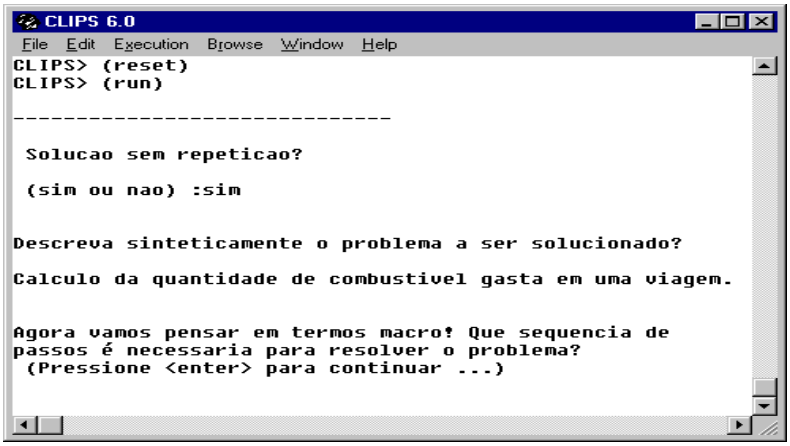
Relatam ainda Mattos, Fernandes e López (1999) que com esta estratégia, um número maior de alunos passou a entender o escopo da aplicação – isto porque, o primeiro passo para a solução de um problema é entendê-lo.

A Figura 2 apresenta uma tela onde é perguntado ao aluno que descreva sinteticamente o problema a ser resolvido. Neste momento já se procura induzi-lo a pensar em termos do contexto do problema.



Fonte: Mattos (2000).

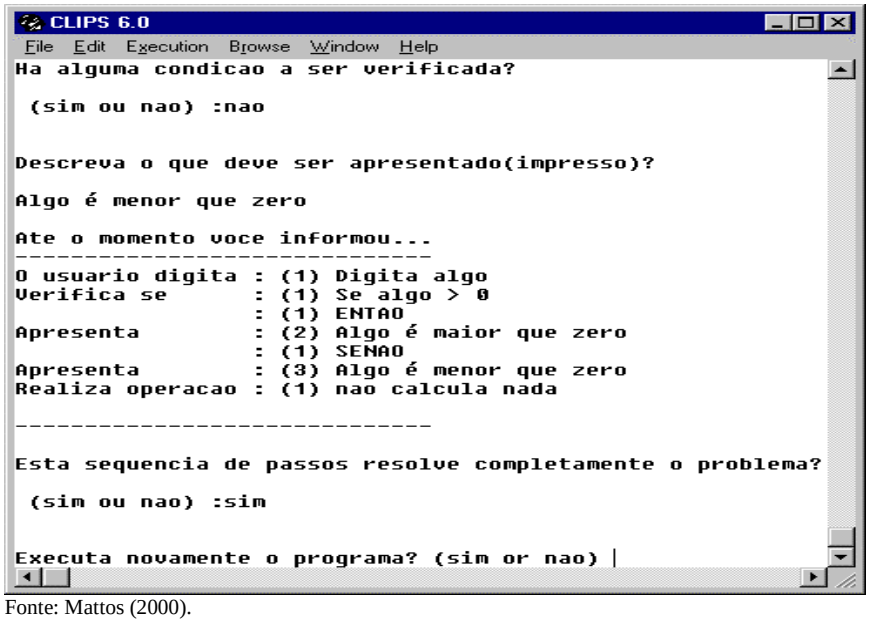
Figura 2 – Tela de contextualização do problema



Fonte: Mattos (2000).

Figura 3 – Tela de contextualização do problema

A Figura 3 continua apresentando questões ao aluno para que o mesmo vá aos poucos delimitando o escopo do problema, e aprofundando-se em direção a uma solução para o mesmo. Após cada rodada na árvore de decisões, na Figura 4 apresenta-se ao aluno um *feedback* do contexto delineado pelo mesmo até o momento e, dependendo do nível de refinamento necessário, automaticamente o sistema inicia um novo passo de refinamento de alguma estrutura que ainda requeira informações complementares.



Fonte: Mattos (2000).

Figura 4 – Tela de feedbak em andamento

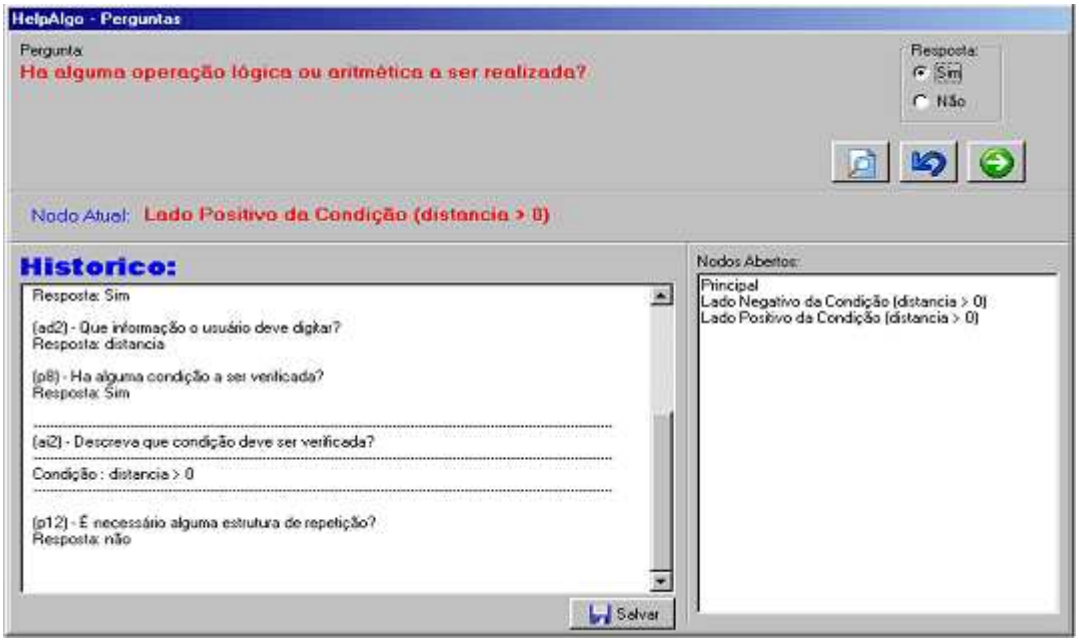
Conforme Mattos, Fernandes e López (1999), o uso de sistemas especialistas permitiu que fosse adquirido o conhecimento tácito do aluno, a partir do conhecimento declarativo, ou seja, o aluno sabe responder sim e não as perguntas que vão sendo realizadas. A ordem das perguntas está estabelecida na árvore de decisões. O que ele não sabe fazer é explicar como resolver o problema vai sendo armazenado na árvore auxiliar (*log*), de tal forma que, ao final do processo, é possível inferir a solução macro do problema através da conjugação dos vários tipos de conhecimento envolvidos no processo. Ainda segundo Mattos, Fernandes e López (1999), testes preliminares confirmaram os resultados obtidos através da execução do processo utilizando-se planilhas em papel, ou seja, os alunos conseguiram adquirir um maior nível de conhecimento do problema bem como gerar um esboço de solução bastante aceitável.

Cabe salientar que, esta proposta não conduz o aluno no sentido de gerar a solução correta para um determinado problema. O objetivo da ferramenta não é este. Tanto isto é verdade que, caso o aluno não responda corretamente (e o termo corretamente neste contexto é relativo) as questões, o sistema apresentará um esboço de solução de acordo.

Ou seja, a ferramenta não elimina a filosofia anteriormente descrita como "metodologia dos oito passos", ela simplesmente enquadra-se no que poderia ser denominado de fase de análise de requisitos de uma metodologia tradicional de desenvolvimento de sistemas, gerando como produto final um esboço de solução. Este esboço de solução terá que ser convertido em um fluxograma, o qual deverá ser avaliado quanto a sua correção através do teste-de-mesa. Somente após este teste é que a solução poderá ser considerada correta.

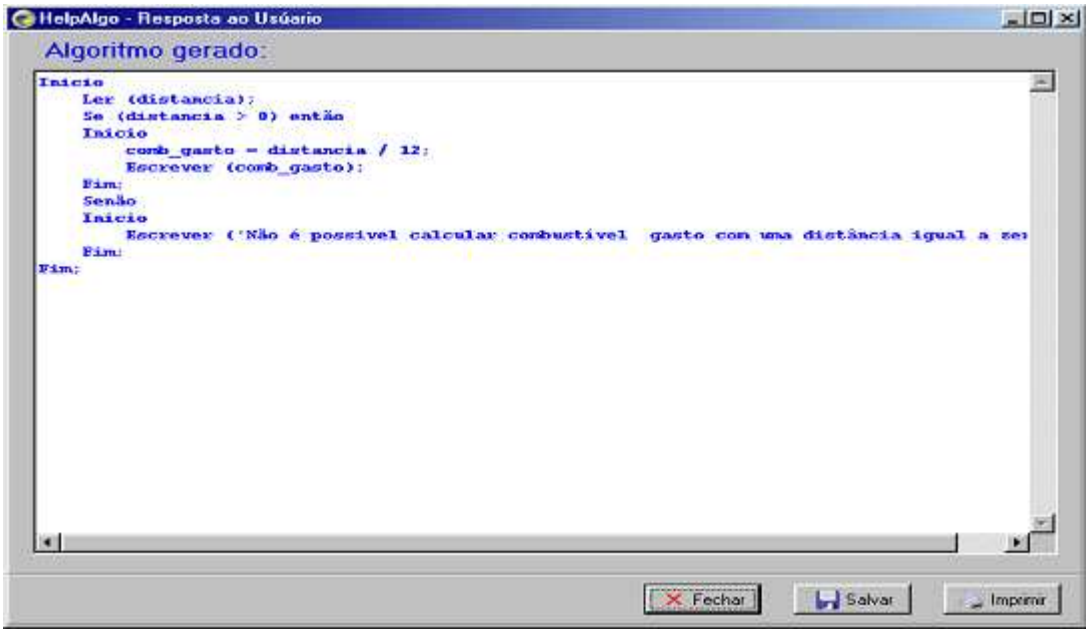
(MATTOS, FERNANDES e LÓPEZ, 1999).

Em 2002, o acadêmico André Iraldo Gubler converteu a aplicação desenvolvida em CLIPS produzindo uma versão em Delphi com a mesma funcionalidade. A Figura 5 apresenta a nova versão da interface onde o aluno está interagindo com o sistema e a Figura 6 apresenta a tela de resultado da interação.



Fonte: Gubler (2002).

Figura 5 – Interação do usuário com o sistema



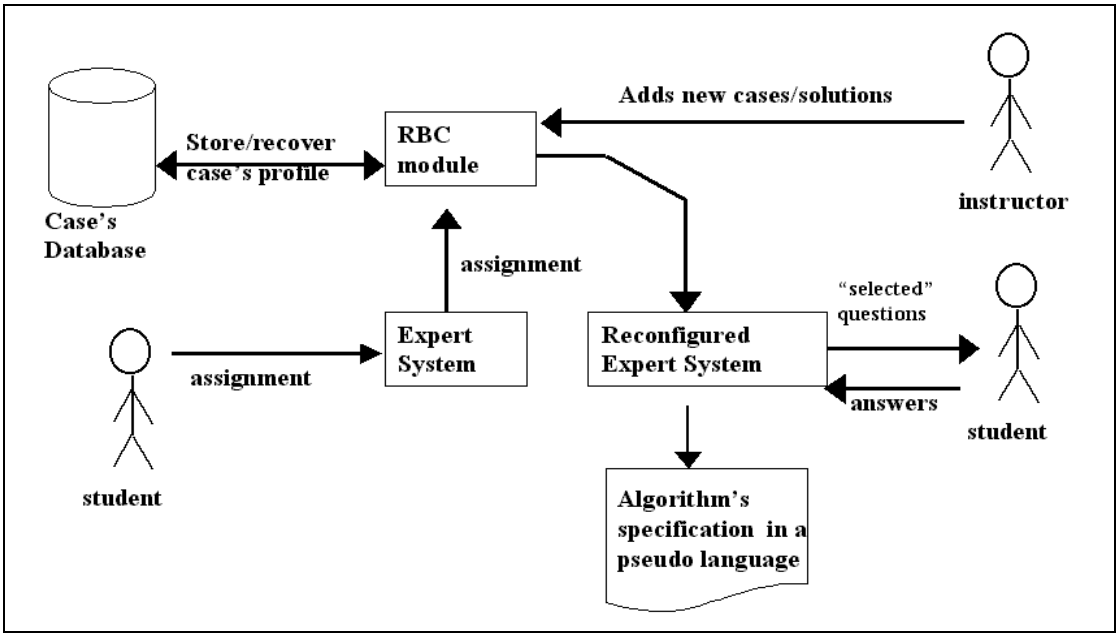
Fonte: Gubler (2002).

Figura 6 – Tela do algoritmo gerado

2.2.3 UTILIZAÇÃO DE RACIOCÍNIO BASEADO EM CASOS

Em Mattos e Wangenheim (1999), é apresentada uma solução utilizando sistemas especialistas, apesar de conduzir a um melhor entendimento do contexto do problema a ser solucionado, não contribuía significativamente para o processo de generalização da solução desenvolvida (processo de abstração). Nesta situação, alunos com dificuldades em abstrair novos conceitos continuavam excluídos do processo de aprendizagem.

A solução adotada envolveu a utilização de técnicas de processamento de linguagem natural e raciocínio baseado em casos (associada a atual solução a base de sistemas especialistas e RBC), a qual é delineada a seguir (Figura 7).



Fonte: Adaptado de Mattos (2002).

Figura 7 – Modelo arquitetônico da solução envolvendo sistemas especialistas e RBC

A maioria dos autores concorda que o RBC é um método de raciocínio baseado na proposta de utilizar experiências passadas encapsuladas em estruturas de dados como a base para lidar com novas situações similares. A abordagem parece ser intuitiva: quando uma nova situação acontece, deve-se tentar alguma coisa que já foi utilizada com sucesso. Usualmente, as soluções utilizadas em situações similares devem ajudar na solução do novo problema. É importante notar que o raciocínio pode funcionar também por contra-exemplos. Neste caso, dado uma nova situação, deve-se procurar descartar aquelas soluções utilizadas em situações similares que resultaram em insucessos. Os sistemas RBC utilizam um processo interativo

constituído genericamente por: identificação da situação atual, busca da experiência mais semelhante na memória e aplicação do conhecimento desta experiência na situação atual. Entretanto, a literatura usualmente não considera a identificação da situação atual como parte do processo RBC, adotando um modelo genérico baseado em quatro etapas: recuperar, reutilizar, revisar e reter.

2.2.4 REPRESENTAÇÃO DOS CASOS

Um caso no contexto do projeto foi definido através de um perfil de solução, o qual é formado pelos seguintes componentes: Palavras-Chave, Generalização, Número-de-passos, Número-de-variáveis, Número-de-constantes e Solução.

As palavras-chave são obtidas tomando-se por base o texto do enunciado, e a aplicação de um conjunto de heurísticas importadas da solução anterior (usando-se Sistemas Especialistas). A generalização constitui-se no resultado do processo de abstração da solução do caso em questão. Número-de-passos, Número-de-variáveis e Número-de-constantes são métricos utilizados para melhor definir o perfil deste caso, e usadas na fase de seleção dos melhores casos para um determinado contexto. A Solução contém a seqüência detalhada dos passos necessários para a solução do problema.

2.2.5 O PROTÓTIPO DA APLICAÇÃO EM RBC

O protótipo da aplicação foi desenvolvido por Luiz Ângelo Heinzen e possui as seguintes características:

a) implementa a fase de análise de enunciados, permitindo a identificação de: heurísticas (como explicado anteriormente), constantes, variáveis (se necessário ou não a sua inicialização), repetições, decisões, entradas e saídas;
b) implementa a fase de generalização do enunciado, a partir das informações coletadas na fase anterior.

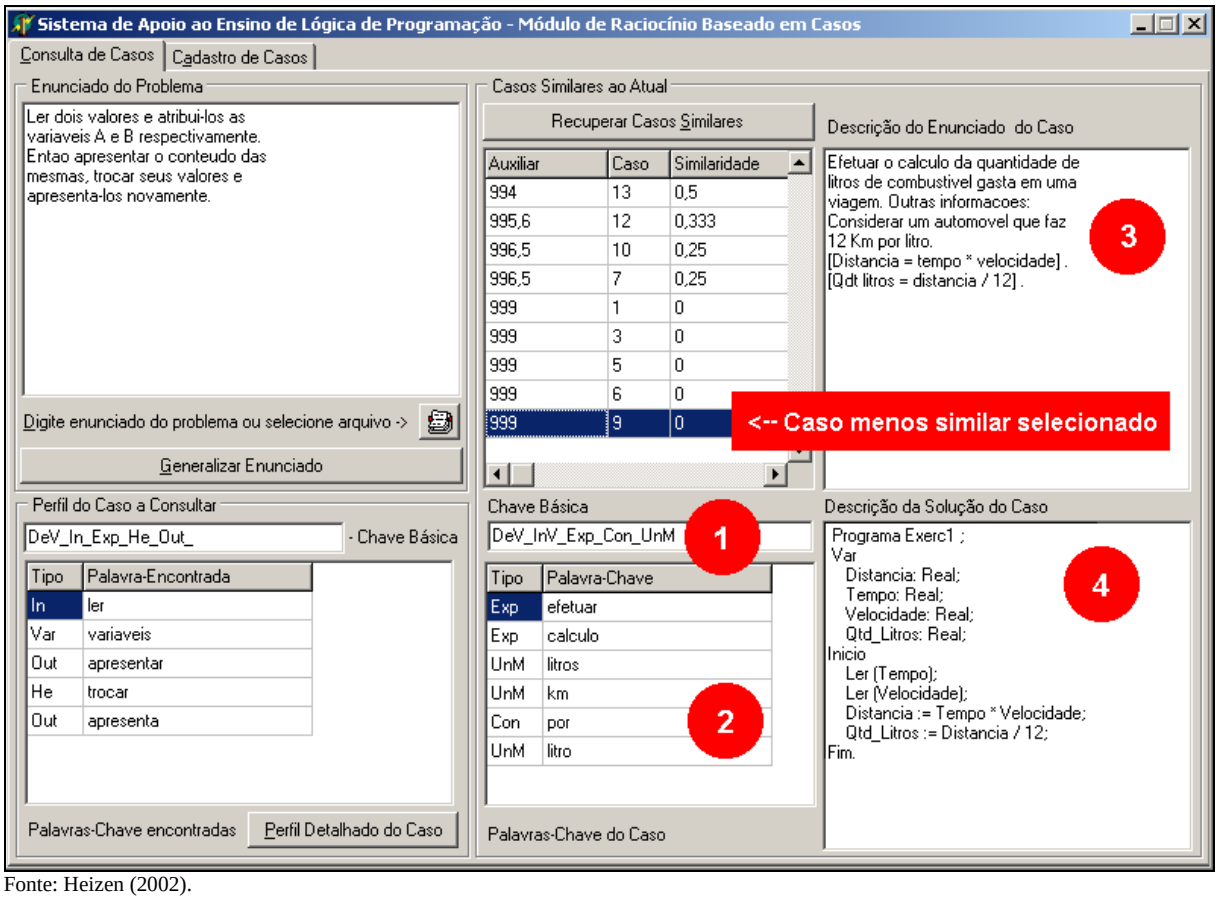
O funcionamento do protótipo é muito simples. Uma vez instalado, deve-se armazenar no diretório “Enunciados” as descrições dos problemas que venham a ser solucionados pelos alunos. Executando-se a aplicação, após uma tela de identificação do usuário (Figura 8).

Seleciona-se o caso clicando-se sobre o nome do arquivo que contém o enunciado e pressionando-se o botão “Enunciado”. Neste momento, aparece na janela abaixo, a descrição do enunciado do caso.

Clicando-se no botão “Generaliza”, o processo de análise do texto do enunciado é

disparado, e o resultado apresentado na janela “Perfil do Caso” como apresentado na Figura 8.

Na Figura 8 observa-se o caso menos similar selecionado - destaque em vermelho. E nas áreas 1, 2 3 e 4 tem-se, respectivamente, a chave básica deste caso, suas palavras-chave, o enunciado e a solução aplicada para resolução aquele problema.



Fonte: Heizen (2002).

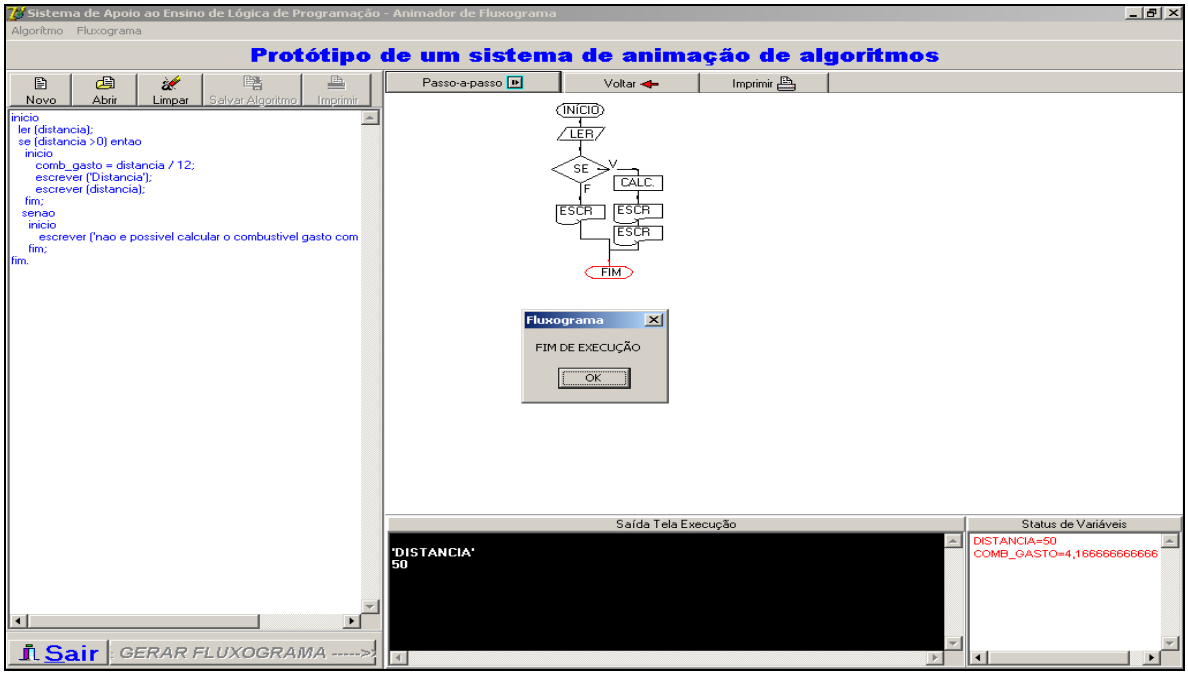
Figura 8 – Módulo RBC

2.2.6 GERAÇÃO E ANIMAÇÃO DE ALGORITMOS

As versões dos protótipos produzidas até então (baseadas em sistemas especialistas e raciocínio baseado em casos) estavam voltadas para o aspecto de aquisição de conhecimento do aluno. Neste sentido, faltava um módulo de animação do algoritmo proposto de tal forma a apresentar ao aluno um *feedback* do resultado da execução daquela proposta de solução construída.

Este módulo desenvolvido por Freitas (2003) lê um arquivo contendo o algoritmo produzido pelo aluno e desenha um fluxograma correspondente à especificação descrita no algoritmo. A fidelidade da conversão é de 1 para 1, ou seja, o módulo de animação não tenta

corrigir eventuais erros produzidos pelo aluno (Figura 9). Depois de desenhado o fluxograma, o aluno pode simular a execução do algoritmo e analisar os resultados produzidos pelo mesmo.

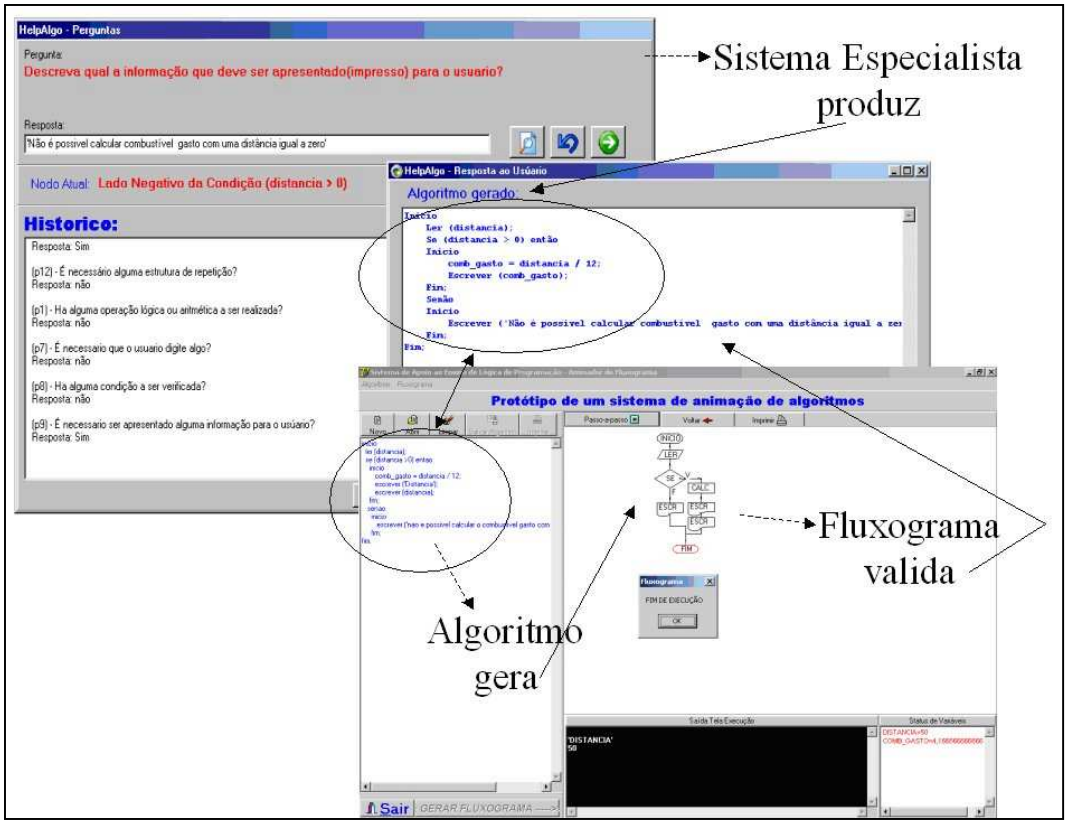


Fonte: Freitas (2002).

Figura 9 – Módulo de animação de algoritmos

Este passo de simulação faz às vezes de um teste-de-mesa, validando a especificação produzida pelo aluno. Posteriormente, o aluno pode solicitar a geração automática do código fonte em C ou Pascal do algoritmo especificado.

A Figura 10 apresenta uma visão geral da arquitetura do sistema. O aluno produz um algoritmo a partir da sua interação com o sistema especialista (e com o módulo de RBC). Este algoritmo é carregado pelo módulo de animação o qual permite ao aluno validar a especificação da sua solução. O passo final é a geração automática do código fonte.



Fonte: Freitas (2002).

Figura 10 – Arquitetura do sistema

3 DESENVOLVIMENTO ESTRUTURADO DE SISTEMAS

De acordo com Mattos (1999), a aprendizagem é um processo no qual experiências fomentam modificação do comportamento e aquisição de hábitos. Com o processo crescente no uso de tecnologias aplicadas à educação é importante ressaltar não só a contribuição dos recursos computacionais existentes, mas, também, a necessidade de estudar-se a forma ideal para sua aplicação. (MATTOS, 1999).

> É justamente pela abordagem abstrata no ensino da lógica de programação que justifica-se a pesquisa e o desenvolvimento de ferramentas e/ou aplicações tecnológicas para o aprendizado ideal. Os aspectos cognitivos relacionados a captação de conteúdos comprovam a eficácia de processos que relacionam a ferramenta computador em beneficio da produtividade intelectual (MATTOS, 2000a).

Conforme Guerreiro (1986) em seu trabalho intitulado “a mesma velha questão: como ensinar Programação?”, apresenta um retrospecto da problemática do ensino de programação relacionando a programação estruturada e a programação orientada a objetos e apresenta alguns argumentos que ainda continuam válidos.

> A programação como a praticamos começou com o primeiro compilador de FORTRAN, há quase quarenta anos. Aparentemente, no início, ensinar Programação não era uma questão que preocupasse demasiado os praticantes. O que era preciso era programas que funcionassem nas máquinas existentes, com os compiladores disponíveis. Nesses tempos, o método era o “code it now, fix it later”, e não havia muito para aprender em Programação além de dominar a linguagem usada e as peculiaridades do computador onde trabalhávamos. (GUERREIRO, 1986).

Segundo Guerreiro (1986), “A situação só mudou com o advento da programação estruturada, no final dos anos 60, sobretudo por influência de Dijkstra, para quem ensinar a programar era essencialmente ensinar a pensar“. (DIJKSTRA; 1972 apud GUERREIRO; 1986). Nesta seção são apresentados os principais conceitos relacionados ao desenvolvimento estruturado de sistemas.

3.1 CONSIDERAÇÕES INICIAIS

De acordo com Guerreiro (1986), a principal linguagem de programação estruturada é o Pascal. A partir desta linguagem é que os conceitos básicos foram assimilados pela comunidade de programadores.

> No entanto, a programação estruturada clássica, aquela que inspirou o Pascal, já não é o paradigma único, talvez nem sequer o dominante, e a função do Pascal ficam prejudicados. Com efeito, actualmente programar é programar com objectos, assim como há vinte anos programar era programar estruturadamente. (GUERREIRO, 1986).

Um outro aspecto importante citado por (GUERREIRO, 1986) é que a programação estruturada considerava a programação como uma atividade individual tendo em vista a realidade da época.

> Hoje, no entanto, se encaramos mais abrangentemente a Programação não apenas como a actividade de escrever programas, mas como a de desenvolver software, vemos que ela tem que ser compreendida, sobretudo como uma actividade de equipa, inserida na disciplica de Engenharia de Software. E, mais uma vez, o Pascal não é adequado para o trabalho em equipa. (GUERREIRO, 1986).

Todos os programadores sabem o que é a programação estruturada, ou se não sabem, pelo menos estão convencidos que é isso que praticam. Descontando as confusões acerca dos '*gotos*', programação estruturada é essencialmente o mesmo que programação por refinamentos sucessivos (GUERREIRO, 1986).

3.2 METODOLOGIA DE PROJETO ESTRUTURADO

> A metodologia de Projeto Estruturado, segundo as definições de Stevens, Constantine, Myers e Yourdon é um composto de técnicas, estratégias e ferramentas para se projetar sistemas e programas de software. Ele fornece um procedimento de projeto por etapas para o projeto do sistema e dos detalhes. (Martin e Mc'Clure, p.70).

O Projeto Estruturado é um refinamento do método de projeto *top-down*. Todos os princípios do projeto *top-down* são válidos no projeto estruturado, sendo que, o Projeto Estruturado acrescenta outras diretrizes para posterior sistematização do processo de projeto e para a avaliação da qualidade de um projeto.

Neste contexto, projetar programas e sistemas é um processo de tomada de decisões que envolvem muitas decisões técnicas. O objetivo do Projeto Estruturado é o de fornecer um procedimento que possibilita ao projetista tomar tais decisões de um modo sistemático.

As quatro etapas básicas no processo de projeto estruturado são (Martin e Mc´Clure, p.70):

a) representa o projeto como um fluxo de dados através de um conjunto de processos;
b) representa o projeto como uma hierarquia de funções (ou componentes de procedimentos;
c) avalia e aperfeiçoa o projeto;
d) prepara o projeto para a etapa de implementação.

3.3 METODOLOGIA DE PROJETO SEGUNDO JACKSON

Segundo (Martin e Mc´Clure), a metodologia de projeto de Jackson é um refinamento

do projeto *top-down*. Ele formaliza este processo através dos seguintes pontos:

a) etapas bem especificadas do processo de projeto;
b) técnicas de diagramação gráfica;
c) métodos de avaliação da exatidão do projeto.

Tanto a metodologia de projeto de Jackson quanto o Projeto Estruturado separam a fase de implementação da fase de projeto pela diferenciação das decisões sobre implementação e pela compleição total do projeto antes do início da fase de implementação A principal diferença entre ambas é que a metodologia de Jackson baseia-se na análise da estrutura de dados, enquanto que o projeto estruturado apóia-se na análise do fluxo de dados. Um é orientado a dados, o outro a processo. O enfoque de Jackson defende uma visão estática de estruturas, e o Projeto Estruturado de Yourdon defende uma visão dinâmica do fluxo de dados. (Martin e Mc´Clure, p.108).

3.4 PROJETO DE PROGRAMA ORIENTADO A DADOS

Segundo Martin e Mc´Clure (1985, p.109), o aspecto principal da metodologia de projeto de Jackson consiste em derivar a estrutura do programa da(s) estrutura(s) dos dados. A metodologia assume que o problema está totalmente especificado e que o programa será implementado numa linguagem de programação procedural de segunda ou terceira geração. Assim, as preocupações com a análise de sistemas e implementação do programa ficam foram do processo do projeto.

Ainda de acordo com Martin e Mc´Clure (1985, p.109), Jackson vê um programa como um processo seqüencial. Ele apresenta *inputs* e *outputs* que são vistos como correntes seqüenciais de registros. Ele sugere que pensemos em cada corrente de dados como um arquivo em fita para reforçar a idéia de desacoplamento de programas e limitação da comunicação a um protocolo simples e em série.

O processo de projeto consiste, em primeiro lugar, da definição da estrutura de correntes de dados e em seguida da ordenação lógica procedural para satisfazer as estruturas de dados. (Martin e Mc´Clure, p.109).

Assim sendo, o processo de projeto consistem de quatro etapas seqüências (Martin e Mc´Clure, p.110):

a) etapa de dados: descreve cada corrente dos dados de input e output com uma estrutura hierárquica;
b) etapa de programa: combina todas as estruturas de dados da primeira etapa numa única estrutura hierárquica de programa;

c) etapa de operações: faz uma lista das operações executáveis necessárias para produzir o output do programa a partir do input. Em seguida, designa cada operação a um componente de estrutura do programa;
d) etapa de texto: escreve as operações ordenadas, com lógica condicional, na forma de texto estrutural, uma versão formal do pseudo-código.

3.5 DESENVOLVIMENTO ESTRUTURADO DE PROGRAMAS EDUCATIVOS

Esta seção apresenta uma compilação do trabalho de Silva e Figueiredo (1994) tendo em vista ser uma referência explícita à adoção de técnicas estruturadas na concepção de ferramentas educacionais.

Segundo Silva e Figueiredo (1994), os métodos para o desenvolvimento de programas educativos baseiam-se tradicionalmente, em duas etapas independentes: concepção pedagógica e implementação. A deficiente comunicação entre estas duas fases conduz freqüentemente a inadequações, que se refletem negativamente na qualidade do produto final. Por outro lado, a qualidade do software educativo, em termos científico e pedagógico, torna conveniente que os projetistas estejam diretamente envolvidos no processo ensino/aprendizagem ao qual se destina esse software.

De acordo com Silva e Figueiredo (1994), embora os Métodos de Análise e Projeto Estruturados não tenham tradição de utilização no desenvolvimento de software educativo, a sua crescente aceitação nos meios industriais da informática não educativa, e os investimentos que vem sendo realizados, tanto em ferramentas como em recursos humanos, sugerem a conveniência de investigar a adoção de algumas das suas soluções, de forma a torná-las acessíveis a educadores não informáticos. O método proposto é inspirado na concepção pedagógica em torno do Modelo do Mercado[1,2], em vários aspectos dos métodos de Análise e Projeto Estruturados[3,4], e nos princípios mais recentes da Prototipação Estruturada[5]. O Modelo do Mercado enfatiza, ao nível da concepção, a componente pedagógica e a interatividade dos programas, ao mesmo tempo em que reduz a distância entre o projetista e o programador.

---

[1] Crossley, K., Green, L., Le Design des Didactiels: Guide Pratique pour la Conception de Scénarios Pédagogiques Interactifs, ACL - Editions, Paris, France, 1990.
[2] Minken, I., Stenseth, B., Vavik, L., Educational Software, ULTIMA – Gruppen A/S, Halden, Norway, 1988.
[3] Yourdon, E., Modern Structured Analysis, Prentice-Hall, Inc.
[4] Page-Jones, M., The Pratical Guide to Structured Systems Design, 2nd edition, Yourdon Press, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, U.S.A., 1988.
[5] Booch, G., Object-Oriented Design: With Applications, The Benjamin//Cummings Publishing Company, Inc., Redwood City, California, U.S.A., 1991.

Destina-se a ser utilizado por educadores não informáticos, e é suportada por um conjunto de conceitos que facilitam segundo este ângulo de visão, a transição para processos de desenvolvimento alternativos, capazes de incorporar soluções que atualmente apenas são usadas no desenvolvimento industrial de software.

3.5.1 UM MÉTODO HÍBRIDO PARA A CONCEPÇÃO DE PROGRAMAS EDUCATIVOS

O modelo descrito em Silva e Figueiredo (1994) é composto pelos seguintes passos:

a) idéia;
b) título;
c) objetivos;
d) perspectivas;
e) campo de aplicações;
f) metáforas e vivências;
g) tabelas de responsabilidades;
h) tabela de responsabilidades individual;
i) tabela de responsabilidades orientada;
j) diagrama de mercado;
k) tela principal (ecrã-chave);
l) telas secundárias (ecrãs secundários);
m) descrição das telas;
n) diagrama de fluxo de dados;
o) diagrama de entidade-relacionamento;
p) dicionário de dados;
q) tabelas de decisão;
r) árvores de decisão;
s) caixas de diálogo.
t) dicionário de conceitos no contexto;
u) ficha de utilização;
v) protótipo;
w) preliminares;
x) versão beta;
y) programa.

Segundo Silva e Figueiredo (1994) os pontos **a**, **b**, **c**, **d**, **f**, **g**, **j**, **k**, **s** e **y** são oriundos do Modelo do Mercado, os pontos **n**, **o**, **p**, **q** e **r** dos Métodos de Análise e Projetos Estruturados, e o ponto **v** dos Métodos Orientados para Objetos e Eventos. Foram introduzidos os pontos **b**, **e**, **h**, **i**, **l**, **m**, **t**, **u**, **v** e **w** face às vivências com os métodos estudados e à profissão de professores desempenhada em paralelo com este estudo.

> As etapas dos Métodos de Análise e Projeto Estruturados utilizadas no Método Híbrido apoiam e dão corpo à descrição dos *frames* e às caixas de diálogo, motivo pelo qual estão colocadas entre estes passos. Caso seja mais profícuo, os passos estruturados podem fazer parte do anterior Modelo do Mercado ou da descrição dos ecrãs. O entrosamento de passos provenientes de métodos distintos pode revelar-se proveitoso, quer para o conceptor, que assim realiza o seu trabalho de forma estruturada, quer para o programador, enquanto leitor do documento de concepção. Por exemplo, pode-se substituir por uma Tabela de Decisão ou uma Árvore de Decisão a especificação, em texto descritivo, da alteração de um ecrã. Também se podem utilizar estas duas ferramentas estruturadas para explicitar a aparição de caixas de diálogo tipificadas. (SILVA E FIGUEIREDO, 1994).

A seguir explicam-se, de modo resumido, os pontos novos introduzidos neste método:

a) Ponto **b**, *título*: contempla a apresentação do programa em torno da justificação para a escolha do título;
b) Ponto **e**, *campo de aplicações*: deve explicitar a população alvo do programa e sugerir algumas situações para as suas possíveis utilizações;
c) Pontos **h** e **i**, *tabela de responsabilidades individual* e *tabela de responsabilidades orientada*: tratando-se de programas educativos, estes podem ser utilizados nas atividades letivas, orientadas pelo professor, ou no estudo individual, cuja navegação depende do aluno ou do professor enquanto planifica a sua lição. Do exposto, surge a necessidade de diferenciar tabelas de responsabilidade, que, assim, permitem abranger diferentes situações de utilização do programa. Assim, quando a sua exploração é orientada, há mais um responsável, usualmente o professor;
d) Ponto **l**, *telas secundários*: surgem da impossibilidade de manter fidelidade a uma única tela principal, quando há mais do que um estado ou ambiente no programa. Às vezes podem ser pequenas alterações relativamente à tela principal, mas cujo tempo de vida possa ser significativo em face da duração do programa;
e) Ponto **m**, *descrição das telas*: tal como o supra referido deve ser articulada com métodos estruturados para evitar descrições longas e maçudas. Contudo, as telas merecem ser descritos para darem uma nova visão do programa ao implementador;

f) Ponto **t**, *dicionário de conceitos no contexto*: há palavras /conceitos com significados diferentes em contextos diferentes e que terão toda a conveniência em ser esclarecidos para eliminar deficiências por má comunicação. Por exemplo, processo pode significar uma parte de um DFD para um programador, ou um meio de separar os componentes de uma mistura para um projetista não-informático;

g) Ponto **u**, *ficha de utilização*: elabora-se uma ficha que pode guiar e exemplificar a utilização do programa. Esta deve ser utilizada pelo projetista para testar partes do protótipo ou ao apresentar este ao programador, ou ainda para testar a versão beta junto de uma população-amostra. Também deve ser reaproveitada para fazer parte do manual de instruções;

h) Ponto **w**, *preliminares*: são reuniões periódicas em que estão, frente-a-frente, o projetista e o programador para apresentarem os respectivos trabalhos, ouvirem sugestões, darem pareceres e reestruturarem todas as partes necessárias antes de se chegar ao produto final. Deste modo pretende-se diminuir os custos de alterações de fundo em fases muito adiantadas do produto final;

i) Ponto **x**, *versão beta*: trata-se de uma versão inacabada do programa que deve ser testada, numa situação de ensino/aprendizagem, mediante uma população-amostra da respectiva população alvo, se possível na presença do projetista, do programador e de outro(s) professor(es). Após esta lição, deve haver sempre uma reunião, com todos os intervenientes, para que sejam emitidas e discutidas opiniões de todos os sectores, a saber: alunos, professores, projetistas e programador. A versão beta deve ser encarada como um teste formativo do programa. Por fim, o projetista e o programador reúnem para as correções e os acertos finais.

3.6 CONSIDERAÇÕES FINAIS

A presente seção apresentou os conceitos principais que nortearam a concepção do projeto. Foi apresentada também uma compilação do trabalho de Silva e Figueiredo, o qual demonstra a viabilidade da adoção de técnicas estruturadas na construção de ferramentas didáticas.

Parte do presente projeto descreve o desenvolvimento de uma ferramenta de especificação de problemas baseado no modelo construtivo com abordagem de dados. Este módulo é utilizado pelo professor na elaboração de enunciados de problemas de introdução a programação.

O próximo capítulo apresenta o modelo de projeto estruturado.

4 DESENVOLVIMENTO DO SISTEMA

Neste capitulo será apresentado a estrutura do sistema e os componentes utilizados para a implementação da solução. Inicialmente são relacionados os componentes utilizados e posteriormente são apresentados os módulos professor e aluno.

4.1 INTRODUÇÃO

Nesta seção é apresentado o método de ensino de introdução a programação desenvolvido pelo Prof. Mauro Mattos (Mattos, 2005). O método está baseado em duas premissas: a construção de enunciados detalhados contendo exemplos de entradas e saídas esperadas e, um processo de análise detalhada dos dados de entrada e de saída. Tendo em vista facilitar a explicação, será utilizado um dos enunciados de problemas de introdução a programação utilizados em aula (Quadro 1).

Enunciado: Ler e imprimir um conjunto de dados contendo nome e uma nota referente a um grupo de n alunos. Parar a leitura quando o nome digitado for = “fim”.

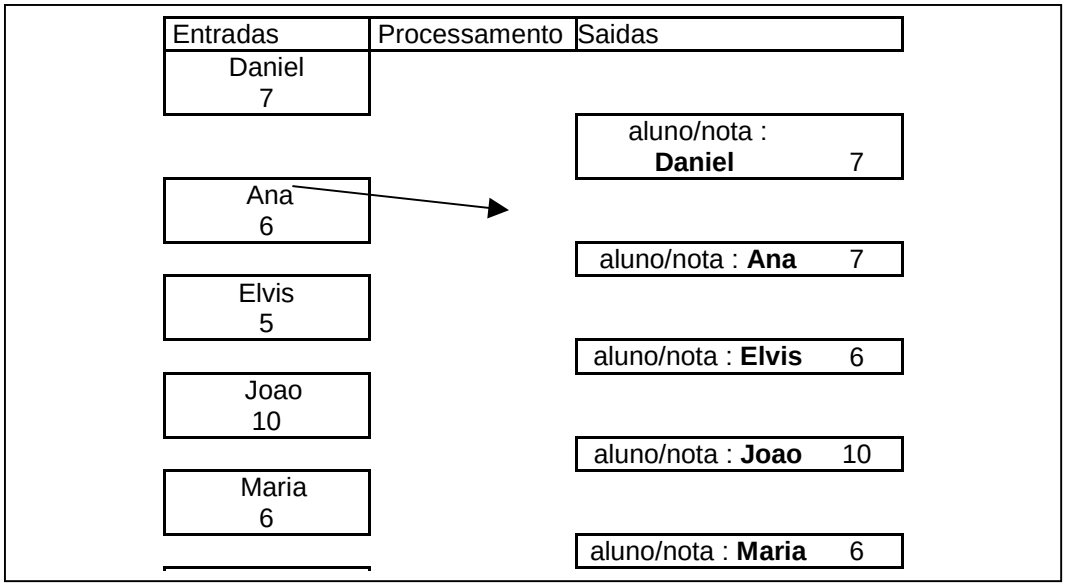
Exemplo:

BaseDados

aluno:	Daniel	Ana	Elvis	Joao	Maria	fim
Nota:	7	6	5	10	6	*

Quadro 1 - Exemplo de enunciado de problema

A partir deste enunciado o aluno deve construir uma planilha conforme apresentado no Quadro 2.



Quadro 2 - Relacionando saídas após as entradas

Nesta tabela o aluno registra, da esquerda para a direita e, de cima para baixo, a seqüência com que os dados de exemplo são entrados no sistema e qual a saída esperada para quela entrada. Observe-se que, a saída é registrada uma linha abaixo da última entrada. A partir do momento em que o aluno informou todas as entradas e saídas, o próximo passo é qualificar os dados de entrada e saída identificando o tipo de variável que deverá armazenar aquele valor entrado ou saído e um nome de variável para aquela entrada ou saída O Quadro 3 representa esta operação.

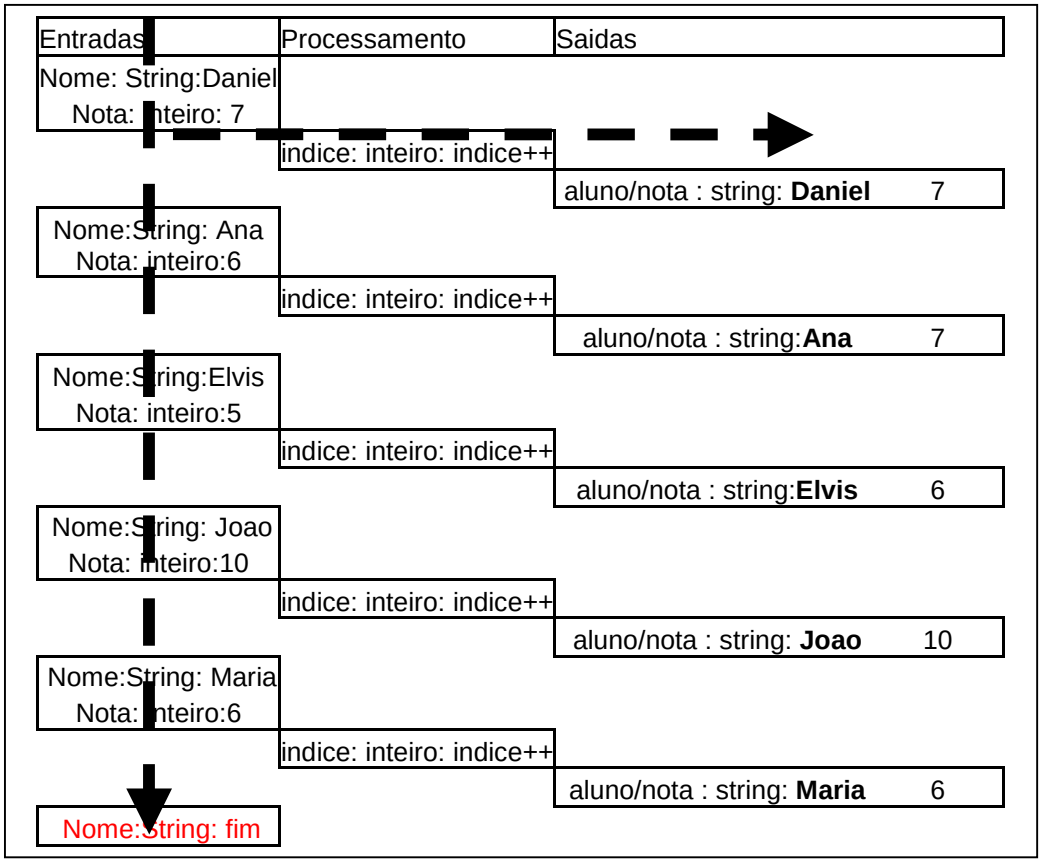
Entradas	Processamento	Saidas
Nome: String:Daniel Nota: Inteiro: 7		
		Linha: String: aluno/nota : **Daniel** 7
Nome: String: Ana Nota: Inteiro: 6		
		Linha: String: aluno/nota : **Ana** 7
Nome: String: Elvis Nota: Inteiro: 5		
		Linha: String: aluno/nota : **Elvis** 6
Nome: String: Joao Nota: Inteiro: 10		
		Linha: String: aluno/nota : **Joao** 10
Nome: String: Maria Nota: Inteiro: 6		
		Linha: String: aluno/nota : **Maria** 6
Nome: String: fim		

Quadro 3 - Qualificação das entradas e saídas

Tendo sido qualificadas as variáveis de entrada e saída o próximo passo é identificar se os nomes das variáveis estão repetidos. Por exemplo, na Quadro 3 os nomes de variáveis “nome” e “nota” repetem-se várias vezes. Isto pode indicar um procedimento repetitivo. Neste caso, uma possibilidade é, caso o aluno já domine o conceito de vetores, qualificar os nomes das variáveis através de um índice ou, decidir pela utilização de uma estrutura de controle do tipo “while” para controlar a repetição. Em uma alternativa ou outra, o aluno já depara-se com a necessidade de utilizar a estrutura de repetição e tem os elementos necessários para posicionar esta estrutura de repetição como controladora das entradas de cada sub-conjunto de dados de entrada (nome e nota) e dados de saída (linha).

O próximo passo consiste em identificar como os dados de saída são produzidos a partir do fornecimento dos dados de entrada. Supondo que o enunciado propusesse o armazenamento dos dados informados para um posterior processamento (exemplo, relação

dos alunos com nota maior que 6), o aluno deverá identificar na coluna processamento que açoes são necessárias para que os dados de saída sejam produzidos ou que ações são necessárias para que os dados de entrada sejam persistidos para posterior manipulação. O Quadro 4 apresenta um exemplo onde os dados de entrada deverão ser armazenados em uma matriz para posterior processamento. Neste caso, uma variável de índice precisa ser incrementada para permitir o armazenamento dos dados em posições corretas.



Quadro 4 - Detalhamento da coluna processamento

Seguindo o princípio de análise: de cima para baixo e da esquerda para a direita, o aluno consegue extrair, num primeiro momento, um pseudocódigo conforme apresentado no Quadro 5.

```
Passos para a resolucao do problema
//inicio de um bloco de processamento
  ler (aluno[i] );
  ler (nota[i] );
  escrever (aluno[i],nota[i]);
//fim de um bloco de processamento
```

Quadro 5 - Exemplo de um pseudo-código extraído a partir do detalhamento da planilha

Numa análise mais detalhada, o aluno pode perceber que o processo de repetição das

entradas produzindo as saídas implicaria na repetição do conjunto de linhas apresentado no Quadro 5 por exemplo, pelo menos quatro vezes. Isto não seria uma solução aceitável. Além disso, o destaque para a condição de parada (nome do aluno = “fim”) poderia levar o aluno a produzir uma solução conforme apresentado na Quadro 6.

```
//inicio de um bloco de processamento
inicializa índice;
ler (aluno[indice]);
  enquanto (aluno[indice]) <> 'fim' faça
    ler (nota[indice]);
    escrever (aluno[indice],nota[indice]);
    indice++
    ler (aluno[indice]);
  fim enquanto;
fim.
//fim de um bloco de processamento
```

Quadro 6 - Exemplo de um refinamento de um bloco de repetição.

Este processo de levar o aluno a detalhar as entradas e saídas, posicionando-as de forma deslocada nas linhas da tabela levam o aluno a pensar em termos temporais permitindo a ele identificar:

a) o que ocorre antes do que (exemplo, entra-se primeiro o nome depois a nota);
b) o que ocorre depois do que (exemplo: entra-se primeiro o nome e a nota e depois exibe-se no nome e nota).

4.2 CONSIDERAÇÕES FINAIS

O modelo apresentado acima foi executado em turmas de primeiro e segundo semestre durante o período de 2004/1 e 2004/2 tendo sido avaliado pelos alunos não só como um facilitador no processo de compreensão do enunciado de um problema como um facilitador na organização das idéias de como construir uma solução algorítmica para os problemas propostos.

4.3 ESPECIFICAÇÃO

A partir desta experiência propôs-se a construção de uma ferramenta que automatizasse o processo. A descrição desta ferramenta é apresentada no próximo capítulo.

O sistema foi concebido na forma de dois módulos: um módulo professor e um módulo aluno. O módulo professor permite que o professor configure um exercício enquanto o módulo aluno permite ao aluno solucionar o problema proposto.

A partir da apresentação do objetivo principal e dos objetivos específicos foram identificados os seguintes requisitos funcionais:

c) construção do módulo professor que viabilize:
   - especificação do enunciado do problema em três níveis de dificuldade (básico, intermediário e avançado),
   - cadastramento do leiaute da base de exemplos,
   - cadastramento dos exemplos de entrada,
   - cadastramento do modelo de relatório de saída,
   - persistência da especificação construída;

d) construção de um módulo aluno que viabilize:
   - a análise dos dados de entrada e a classificação dos mesmos em termos de entradas, processamento e saídas,
   - a qualificação dos dados utilizados em termos de nomes de variáveis e tipos destas variáveis,
   - ordenação temporal das subseqüências de entradas, processamento e saídas,
   - a identificação de laços de repetição e a qualificação em termos de estruturas de repetição ou procedimentos e funções (para nível intermediário e avançado),
   - geração de pseudocódigo em Portugol que represente a seqüência temporal de transformação de dados de entrada nas saídas esperadas.

Como requisitos não funcionais podem ser destacados:

a) a interface com o usuário deve facilitar a utilização por alunos inexperientes;
b) as mensagens de erro devem ser de fácil assimilação pelos alunos;
c) deve ser possível salvar a solução intermediária de um problema permitindo a restauração do contexto e a continuidade da solução em um momento posterior;
d) a ferramenta será construída em Borland Delphi 7.0;
e) o pseudocódigo será gerado em Portugol.

A seguir, são apresentados alguns diagramas de caso de uso dos dois módulos da ferramenta, bem como os diagramas de atividades de cada caso de uso, modelos estes que foram realizados com auxílio da ferramenta Rational Rose desenvolvida pela Rational Software Corporation.

O primeiro diagrama (Figura 11) mostra sob quais formas o professor pode interagir no programa, ou seja, o professor pode criar um exercício e/ou alterá-lo quando necessário.

Depois que o exercício é salvo, o mesmo pode ser carregado, e todos os estados, tabelas, estruturas, formatos e enunciado são apresentados no ponto em que foram persistidos.

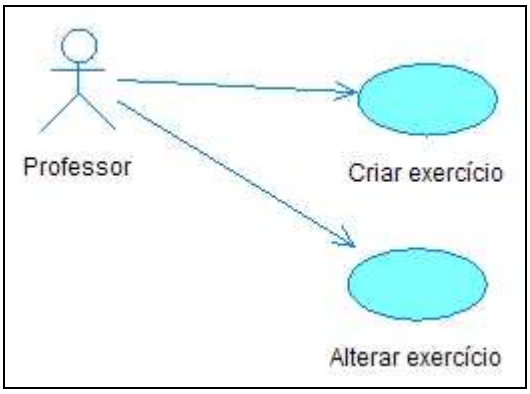


Figura 11 – Diagrama de caso de uso do módulo professor

A Figura 12 ilustra um diagrama de atividades do caso de uso ‘Criar Exercício’, ao criar um novo exercício, o programa irá perguntar se o usuário deseja salvar o exercício corrente, caso exista um exercício aberto.

Primeiramente o professor deve criar as tabelas envolvidas no exercício, deve em seguida configurar a estrutura das tabelas com seus atributos e incluir alguns registros de entrada nas tabelas.

Estruturada as tabelas com atributos e registros, pode-se modelar o formato de saída e salvar o exercício, se o exercício for de nível diferente de básico o programa verifica se todas as tabelas possuem um registro do tipo ‘Flag de Parada’, caso positivo o exercício poderá ser salvo.

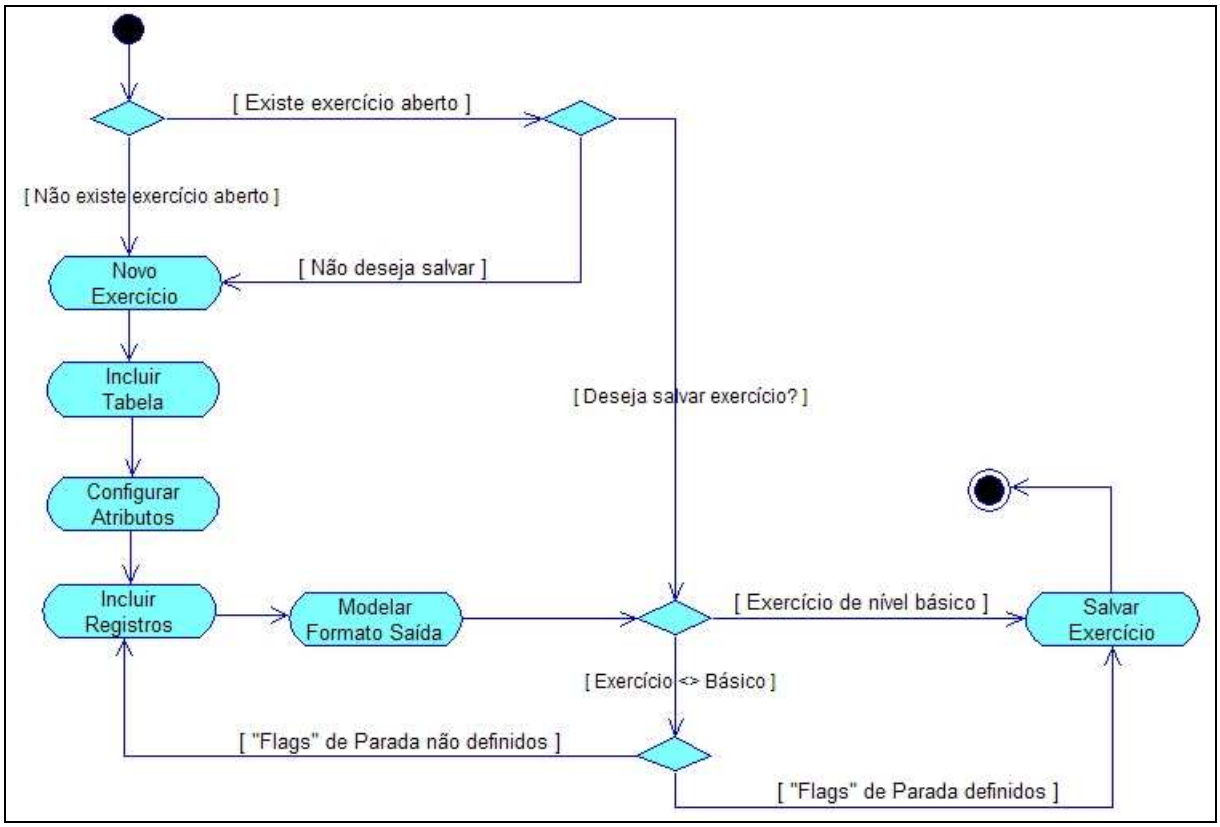


Figura 12 – Diagrama de Atividade do caso de uso 'Criar Exercício'

Assim, o programa executável denominado QualificaP.EXE é o resultado da compilação dos arquivos (*units*) assim dispostos[6]:

a) uQualificaP é o formulário principal do programa aonde o professor executa e realiza as principais funções;
b) ucExercicio é a classe principal que gerencia desde a criação do exercício até a solução do aluno;
c) ucEntrada é a classe que gerencia todos os campo dos registro de cada tabela do exercício;
d) ucSaida é a classe que gerencia as saídas que serão selecionadas para as entradas;
e) ucAtributo é a classe que especifica o nome, nome lógico, tipo e tamanho dos campos;
f) ucTabela é a classe que gerencia as tabelas do exercício;
g) ucTextoLivre é a classe que gerencia os textos livres do formato de saída;
h) ucVariavel é a classe que gerencia as variáveis que são geradas a partir dos processos das entradas e das saídas;

[6] Foram omitidas desta relação as *units* associadas aos formulários.

i) ucCampoEntrada é uma classe visual que gerencia as entradas que são selecionadas para o formato de saída e pelo aluno para serem classificadas e qualificadas;

j) ucCampoFormato é a classe que gerencia os campos de entrada e no formato de saída, cada campo de entrada selecionado gera um campo no formato de saída;

k) uIdiomas é uma *unit* auxiliar que gerencia a internacionalização do sistema, que pode ser apresentado também na língua inglesa, um exemplo pode ser apreciado pela Figura 81 e Figura 82 localizada no apêndice C.

A Figura 13 foi desenhada utilizando o Software Rational Rose 2000 e pode-se observar nesta figura como foi definida a estrutura de classes para o desenvolvimento do módulo professor da ferramenta.

O diagramas de classes completo está incluído no apêndice A.

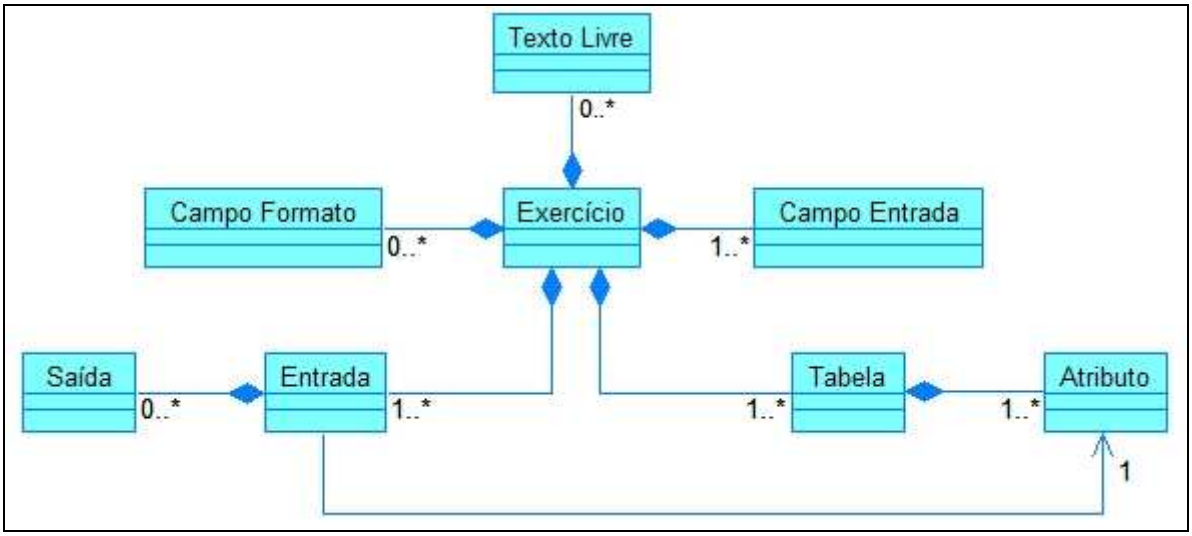


Figura 13 – Estrutura de classes do módulo professor

O diagrama apresentado na Figura 14 apresenta sob quais formas o usuário poderá interagir no módulo aluno, onde o mesmo pode abrir um exercício, resolvê-lo e alterar a solução encontrada.

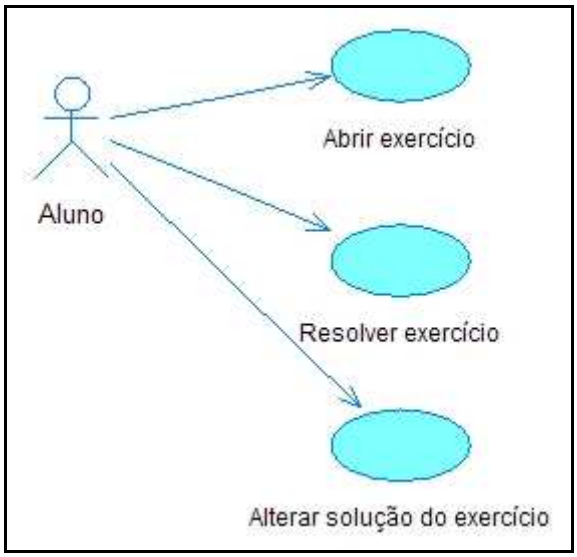


Figura 14 – Diagrama de caso de uso do módulo aluno

A operação **abrir exercício** consiste basicamente em abrir um arquivo com extensão PJA que foi desenvolvido no módulo professor.

No módulo aluno o usuário irá abrir o exercício proposto pelo professor, salvar a solução que ele encontrou e poderá abrir o exercício e a sua solução do ponto de onde parou.

O simples fato de o aluno resolver o exercício e salvá-lo irá gerar um arquivo dentro do mesmo diretório e com o mesmo nome do exercício, mas com a extensão SOL. Se o aluno decidir parar a solução do exercício e quiser continuar num outro momento, basta salvá-lo e carregá-lo posteriormente.

Deve-se ter cuidado ao carregar um arquivo de exercício (PJA) com versão superior ao utilizado na solução pelo módulo aluno, principalmente se o novo arquivo de exercício gerado tiver uma quantidade de registros de entradas e/ou campos de formato de saída menor que o arquivo utilizado anteriormente.

A Figura 15 ilustra um diagrama de atividades do caso de uso ‘Resolver Exercício’, ao abrir o exercício, o aluno começará a classificar as entradas e as saídas, depois irá qualificar as entradas e as saídas informando um nome, cada nome irá gerar um identificador que também deve ser informado um nome e um tipo, depois se o nível do exercício for diferente de Básico, o aluno poderá localizar a repetição de dados dentro da classificação e agrupá-las.

Após o agrupamento das informações o aluno poderá especificar processos para as entradas e saídas que poderão gerar variáveis que também deverão ser qualificadas devendo ser configurado o seu tipo.

Após a definição de processos a análise do exercício chega ao fim e o aluno pode então gerar o pseudocódigo em Portugol.

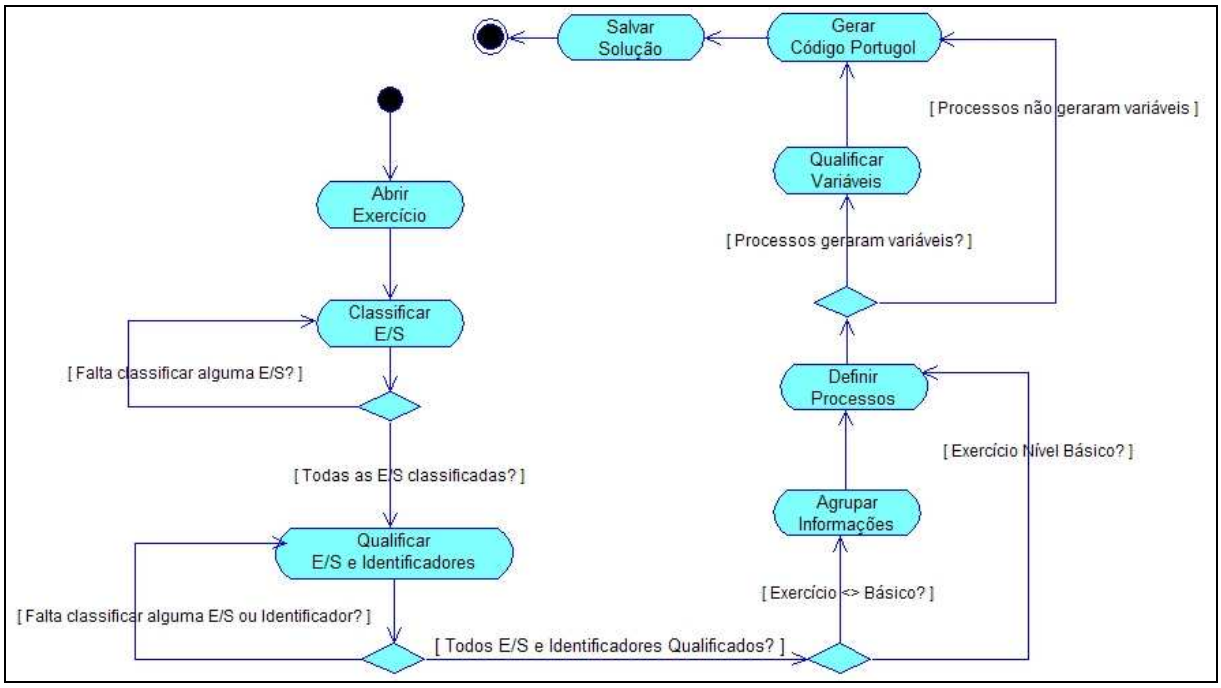


Figura 15 – Diagrama de atividades do caso de uso 'Resolver Exercício'

A Figura 16 apresenta o diagrama de classes do módulo aluno. O diagrama de classes completo está incluído no apêndice B.

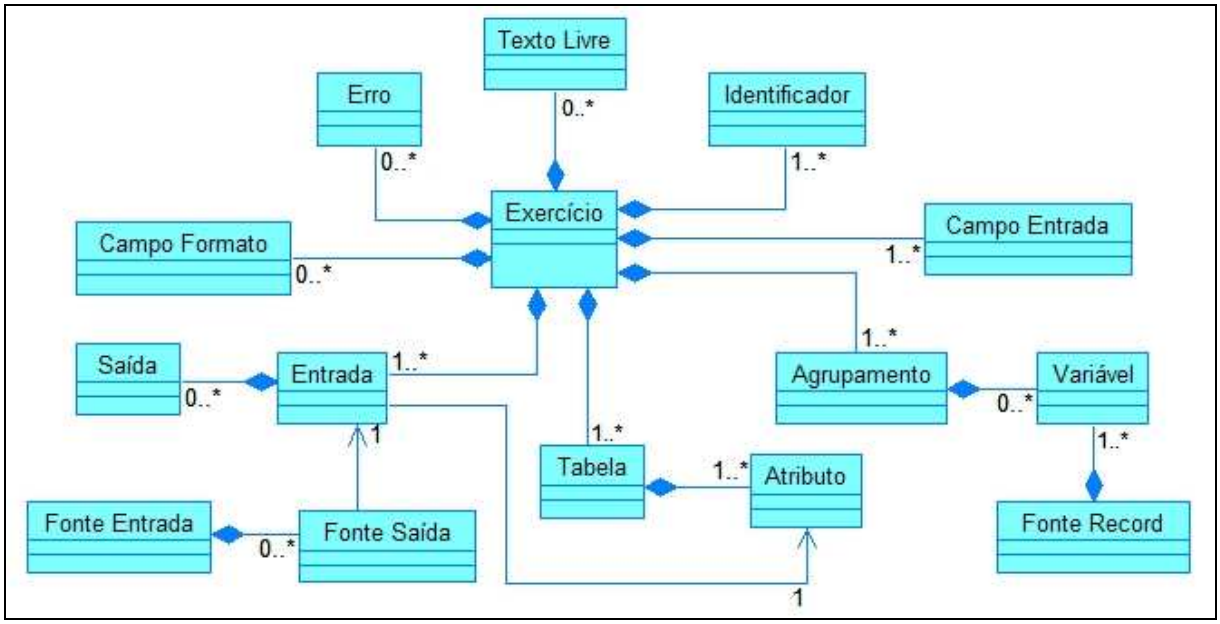


Figura 16 – Estrutura de classes do módulo aluno

O programa executável denominado Qualifica é o resultado da compilação dos arquivos (*units*) assim dispostos[7]:

a) uQualifica é o formulário principal do módulo aluno, onde o usuário poderá interagir com as entradas, saídas e resolver o exercício proposto;
b) uGerarFonte é onde o programa utiliza os registros de entrada selecionados, as saídas selecionadas, e os processamentos que foram definidos pelo aluno e gera o código fonte Portugol;
c) ucErro é a classe que gerencia as inconsistências e as mensagens para o aluno identificar qual o próximo passo que ele deve executar e ou corrigir;
d) ucAgrupamento é a classe que gerencia os agrupamentos de entradas e saídas para a geração de procedimentos e ou laços de repetição;
e) ucIdentificador é a classe que gerencia os nomes, rótulos, tipos e alias das entradas e saídas;
f) uFonteEntrada é uma classe auxiliar utilizada somente na geração do código portugol, para gerenciar a repetição das entradas dentro dos agrupamentos;
g) ucFonteSaída é outra classe auxiliar utilizada somente na geração do código portugol, para gerenciar a repetição das saídas dentro dos agrupamentos;
h) uFonteRecord é também uma classe auxiliar utilizada na geração do código portugol, para armazenar o código fonte das definições de variáveis do tipo registro, utilizado somente quando o exercício é do nível Avançado, para ser resolvido com a utilização de estruturas do tipo *record*.

4.4 COMPONENTES UTILIZADOS

Tendo em vista otimizar e facilitar o desenvolvimento do sistema, foi criado um pacote de componentes denominado “JFF” o qual agrupa todos os recursos necessários. Foram utilizados alguns componentes para a construção da interface com o usuário, que serão descritos a seguir.

4.4.1 TSTRINGGRIDSTRIP

Descendente da classe TStringGrid, exibe as linhas ímpares na cor diferente das pares dando um efeito listrado e com a possibilidade de exibir ícones. É utilizado para exibir a grade de classificação de entrada e saída, a lista de identificadores e a lista das variáveis (Quadro 7).

[7] Foram omitidas desta relação as *units* associadas aos formulários.

```
LeftBP := 2;
P := Pos('Ñ', Txt);
while P > 0 do
begin
  //Captura o nr. do bitmap
  NB := StrToInt(Copy(Txt, P+1, 2));
  Delete(Txt, P, Length('Ñ'+FormatFloat('00',NB)));

  if NB = 99 then  //Não é bitmap, são linhas verticais entre os agrupamentos
  begin
    Pen.Color := clBtnShadow;
    LB := aRect.Right - LeftBP - 9;
    Rectangle(LB, aRect.Top, LB+1, aRect.Bottom);
    Inc(LeftBP, 16);
  end else if NB > 0 then
  begin
    fBitmap := TBitmap.Create;
    Images.GetBitmap(NB-1, fBitmap);
    //Calcula posição na linha e desenha
    fBitmap.Transparent := True;
    fBitmap.TransparentColor := fBitmap.Canvas.Pixels[15,0];
    LB := aRect.Top + ((aRect.Bottom-aRect.Top) - fBitmap.Height) div 2;
    Draw(aRect.Right-(fBitmap.Width+LeftBP), LB, fBitmap);

    Inc(LeftBP, fBitmap.Width+2);

    fBitmap.Canvas.FloodFill(0,0, clBlack, fsBorder);
    fBitmap.Transparent := False;
    fBitmap.Free;
  end;
  P := Pos('Ñ', Txt);  //Procura por outros bitmaps no texto
end;
```

Quadro 7 – Exibindo ícones nas células do componente TStringGridStrip

4.4.2 TJFFPANEL

Descendente da classe TPanel, que possui uma barra de título que pode ser coberta com uma figura ou com o fundo chapado em uma cor, pode-se ainda configurar as suas bordas laterais com larguras variáveis (recurso utilizado para substituir o componente de painel convencional).

```
//Cria o bitmap e calcula a área a ser desenhada com a figura
fBitmap        := TBitmap.Create;
fBitmap.Width  := Self.Width - Options.BorderRight - Options.BorderLeft;
fBitmap.Height := Options.PanelTitle.Height;

//Cobre toda a área do título do panel com uma figura (JPG, BMP, etc)
with fBitmap.Canvas do
begin
  dX := fWallPaper.Bitmap.Width;
  dY := fWallPaper.Bitmap.Height;
  Y := 0;
  while Y < Self.Height do
  begin
    X := 0;
    while X < Self.Width do
    begin
      fBitmap.Canvas.Draw(X, Y, fWallPaper.Bitmap);
      Inc(X, dX);
    end;
    Inc(Y, dY);
  end;
end;
```

Quadro 8 – Preenchendo o título do painel com a figura

4.4.3 TBOTAOCAMPO

Descendente da classe TButton,. Quando o *mouse* é posicionado sobre ele dá-se um efeito piscante quando o mesmo não está selecionado, ao contrário o texto fica destacado em azul. Este recurso foi utilizado para exibir os registros de entrada que são selecionados e classificados pelo aluno e que também são selecionados para o formato de saída (Quadro 9).

```
//Se o mouse estiver posicionado sobre o Botão
if (fMouseInControl) then
begin
  if (Down) then  //Botão está pressionado
    Brush.Color := fColorOnMouseDown
  else if fValueColor = 0 then
    Brush.Color := fColorNormal
  else
    Brush.Color := fColorOnMouse;
end else  //Senão, alguém ativou o timer e mandou o botão ficar intermitente
begin
  if (Down) and (Enabled) then
    Brush.Color := fColorPressed
  else
    Brush.Color := fColorNormal;
  if fAtivarTimer then  //Timer intermitente ativado
    if fValueColor = 0 then
      Brush.Color := fColorNormal
    else
      Brush.Color := fColorTimer;
end;
```

Quadro 9 – Efeito piscante do compoente TBotaoCampo

4.4.4 TPANELGRID

Descendente da classe TPanel, é um painel com linhas desenhadas na horizontal e na vertical simulando uma espécie de grade onde os registros de entrada são posicionados, especificando assim a posição linha e coluna dentro do formato. O espaçamento entre as linhas e as colunas foi fixado nas medidas de um caractere configurado na fonte *Courier New* normal tamanho 10, que é de 16 *pixel* de altura por 8 *pixel* de largura para estabelecer um padrão semelhante aos caracteres em modo DOS. Este recurso foi utilizado para exibir a grade de formatação de saída (Quadro 10).

```
procedure TPanelGrid.DeslocarLinha(Todas: Boolean; Tipo: Char);
var
  Ok   : Boolean;
  I,Lin: Integer;
Begin
  //Varrer toda a lista de campos encontrados no formato de saída
  for I := 0 to fListaCampos.Count-1 do
  begin
    Lin := TPanelCampo(fListaCampos.Items[I]).Top;

    if not Todas then
      Ok := (Lin = fLinhaAtual)
    else
      if (Tipo in [' ','+']) then
        Ok := (Lin >= fLinhaAtual)
      else
        Ok := (Lin <= fLinhaAtual);

    if (Ok) then
    begin
      if (Tipo = '+') then
        Inc(Lin, 16)
      else
        Dec(Lin, 16);
      //Atualiza linha do campo do formato e chama método para ele se atualizar
      TPanelCampo(fListaCampos.Items[I]).Top := Lin;
      TPanelCampo(fListaCampos.Items[I]).RecalcularPosicoes;
      TPanelCampo(fListaCampos.Items[I]).Deslocar;
    end;
  end;
end;
```

Quadro 10 – Código para deslocar os campos inseridos no formato de saída

4.4.5 TPANELCAMPO

Descendente da classe TPanel, funciona juntamente com o componente *TPanelGrid*, que pode ser arrastado dentro do mesmo obedecendo as mesmas normas de posicionamento das linhas de divisão horizontais e verticais. Este recurso foi utilizado para apresentar os registros de entrada e os textos livres na área de formatação de saída (Quadro 11).

```
procedure TPanelCampo.RecalcularPosicoes;
var
  Resto: Integer;
begin
  //Calcular posição para ficar exatamente dentro da linha do Grid
  Resto := (Top mod fLin);
  if Resto > 0 then
    Top := (Top - Resto);
  //Calcular posição para ficar exatamente dentro da coluna do Grid
  Resto := (Left mod fCol);
  if Resto > 0 then
    Left := (Left - Resto);
  //Para não extrapolar pelo limite inicial de 0,0
  if Top < fLin then Top := fLin;
  if Left < fCol*2 then Left := fCol*3;
  Dica;
end;
```

Quadro 11 – Calcular posição relativa do campo dentro do formato de saída

4.4.6 TJFFSTRINGGRID

Outro componente descendente da classe TStringGrid, com opções de alinhamento nas

colunas. Este recurso foi utilizado para a apresentação da lista de atributos e dos registros das tabelas, onde os campos do tipo *real* e *integer* são exibidos com alinhamento à direta.

4.4.7 TLABELRISCADO

Descendente da classe TLabel, implementa um traço centralizado na cor vermelha. Este recurso foi utilizado no módulo aluno para exibir quais as fases da solução do exercício que o aluno já concluiu (Quadro 12).

```
procedure TLabelRiscado.Paint;
var
  Rect: TRect;
  Lin : Integer;
begin
  if not Enabled then
    Font.Color := fColorDisabled;
  //Chama o método paint do ancestral, para imprimir o texto normal
  inherited Paint;
  if fRiscado then with Canvas do
  begin
    Rect := ClientRect;
    Lin := (Rect.Bottom - Rect.Top) div 2;
    Pen.Color := fColorRiscado;  //Desenha uma linha vermelha no meio do texto
    MoveTo(Rect.Left, Lin+1);
    LineTo(Rect.Right, Lin+1);
  end;
end;
```

Quadro 12 – Código para “riscar” o texto

4.4.8 TJFFGRADBUTTON

Descendente da classe TBitBtn que implementa apresentações diferentes para quando o cursor do *mouse* estiver posicionado ou pressionado sobre o mesmo (recurso utilizado para substituir o componente de botão convencional).

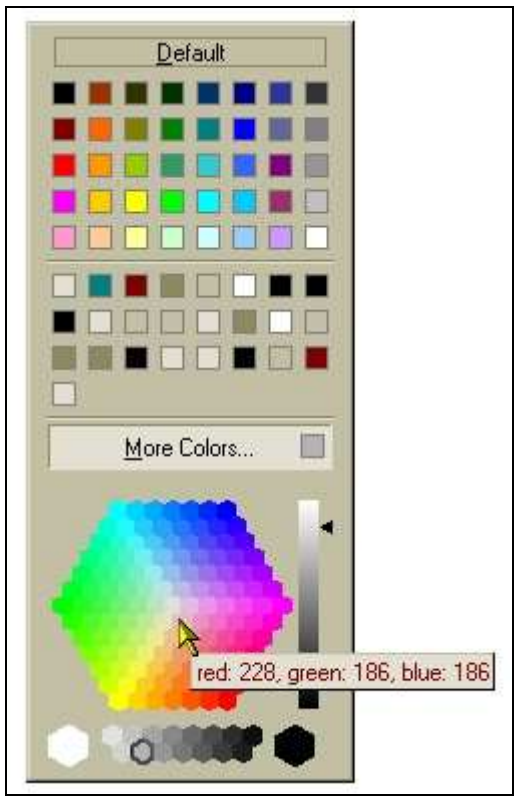
```
procedure TJFFGradButton.PaintFundo(Rct: TRect);
begin
  if not Enabled then  //Se botão estiver desabilitado
    FCanvas.Brush.Color := fOptions.fDisabledColor
  else if (FMouseInControl) and (Enabled) or (IsDown) and
          (csDesigning in ComponentState) then
    if IsDown then  //Se o botão estiver pressionado
      FCanvas.Brush.Color := fOptions.fColorPressed
    else
      FCanvas.Brush.Color := fOptions.fColorFocused
  else
    FCanvas.Brush.Color := FOptions.fColor;
  //Se utiliza borda simples
  if not FOptions.FSimpleBorder then
    InflateRect(Rct, -2, -2);

  FCanvas.FillRect(Rct);
end;
```

Quadro 13 – Exibindo cores diferentes no botão

4.4.9 TCOLORPICKERBUTTON

Componente externo que pode ser baixado pela internet, implementa um botão com opções de combinação de cores. Este recurso foi utilizado na área de formatação de saída, somente para configurar a cor de fundo e a cor das linhas divisórias.



Fonte: http://www.lischke-online.de/ColorPicker.php

Figura 17 – Exemplo do componente TColorPickerButton

O componente TColorPicker é liberado sob licença *opensource* padrão MIT a qual permite o uso do código em qualquer aplicação de software incluindo software comercial (SOFT GEMS SOFTWARE SOLUTIONS, 2006).

4.4.10 TSOURCEEDIT

Componente externo descendente da classe TMemoComponent. Conforme Reichelt (2006), TSourceEdit é um componente de editor de texto *opensource* com destaque para sintaxe que possui várias facilidades para o desenvolvimento de interface para programadores, exibindo cores diferenciadas para textos, números, comentários, palavras reservadas, etc.

Este recurso foi utilizado para apresentar os códigos Portugol gerados pelo módulo aluno e na edição das linhas de código de processo para as entradas e saídas.

4.5 MÓDULO PROFESSOR

O modulo professor permite que o professor modele um exercício a ser apresentado aos alunos e que será resolvido no módulo aluno. É de responsabilidade do professor configurar a estrutura das tabelas de entrada de dados bem como o leiaute do relatório de saída. Além disso, o professor deve redigir a especificação do enunciado que acompanhará o problema proposto. O enunciado do problema proposto é de caráter informativo e não exerce influência nas estruturas de entrada e saída, bem como na solução do exercício.

A Figura 18 mostra um exemplo da tela principal do programa com o editor do enunciado do problema e a definição das estrutura da base de dados, com suas tabelas, atributos e registros.

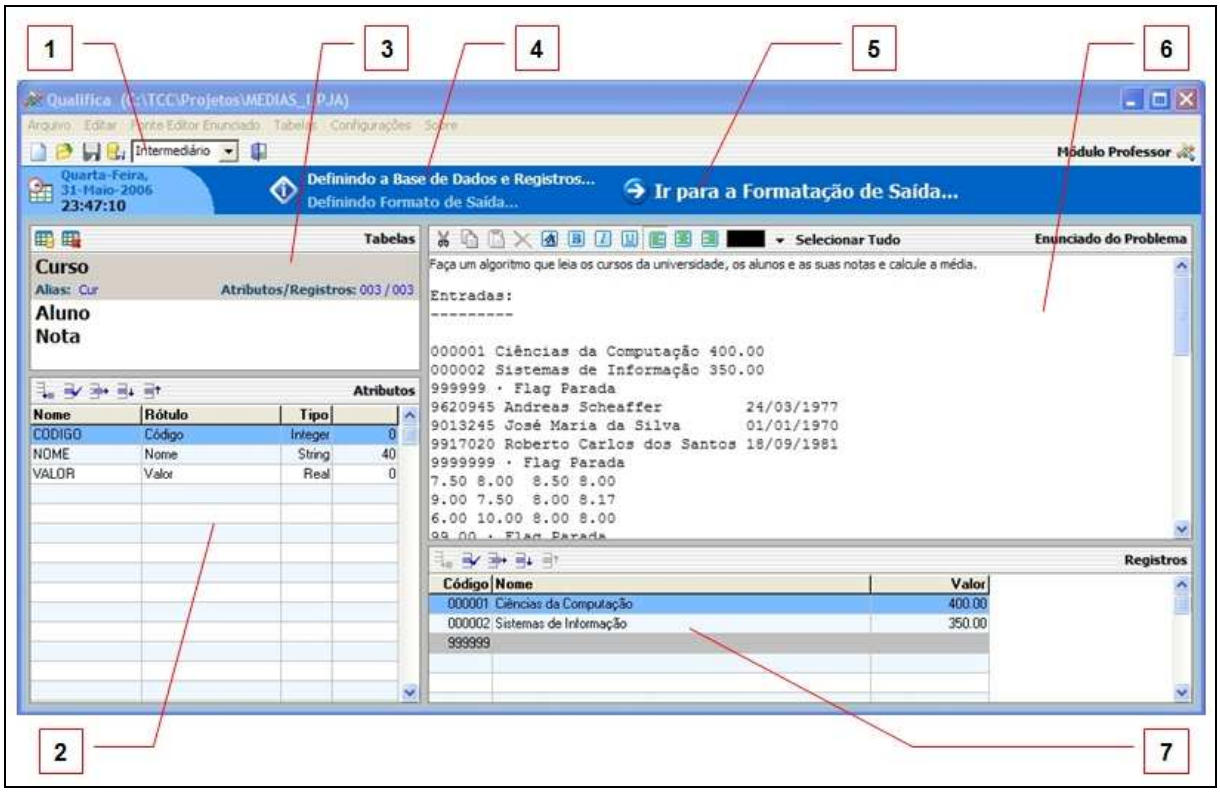


Figura 18 – Tela principal do módulo professor

a) o item 1: configuração do nível do exercício, no caso aqui é intermediário;
b) o item 2: exibe a lista de atributos da tabela atual (Curso);
c) o item 3: exibe a lista de tabelas do exercício, ficando a tabela atual (Curso) em destaque, exibindo detalhadamente o seu *alias* e a quantidade de atributos e registros;
d) o item 4: mostra a fase atual da criação do exercício, que no caso aqui o professor está Definindo a Base de Dados e Registros;

e) o item 5: um botão que permite o professor ir para a próxima fase de criação do exercício que é a Formatação de Saída;

f) o item 6: editor de textos aonde o professor deverá descrever o enunciado do exercício;

g) o item 7: mostra a lista de registros de entrada da tabela atual (Curso) do exercício.

A Figura 19 mostra o mesmo exemplo ainda na tela principal do programa, mas na área de formatação de saída dos dados, exibe os registros de entrada que são selecionados para o formato e logo abaixo a área dos textos livres.

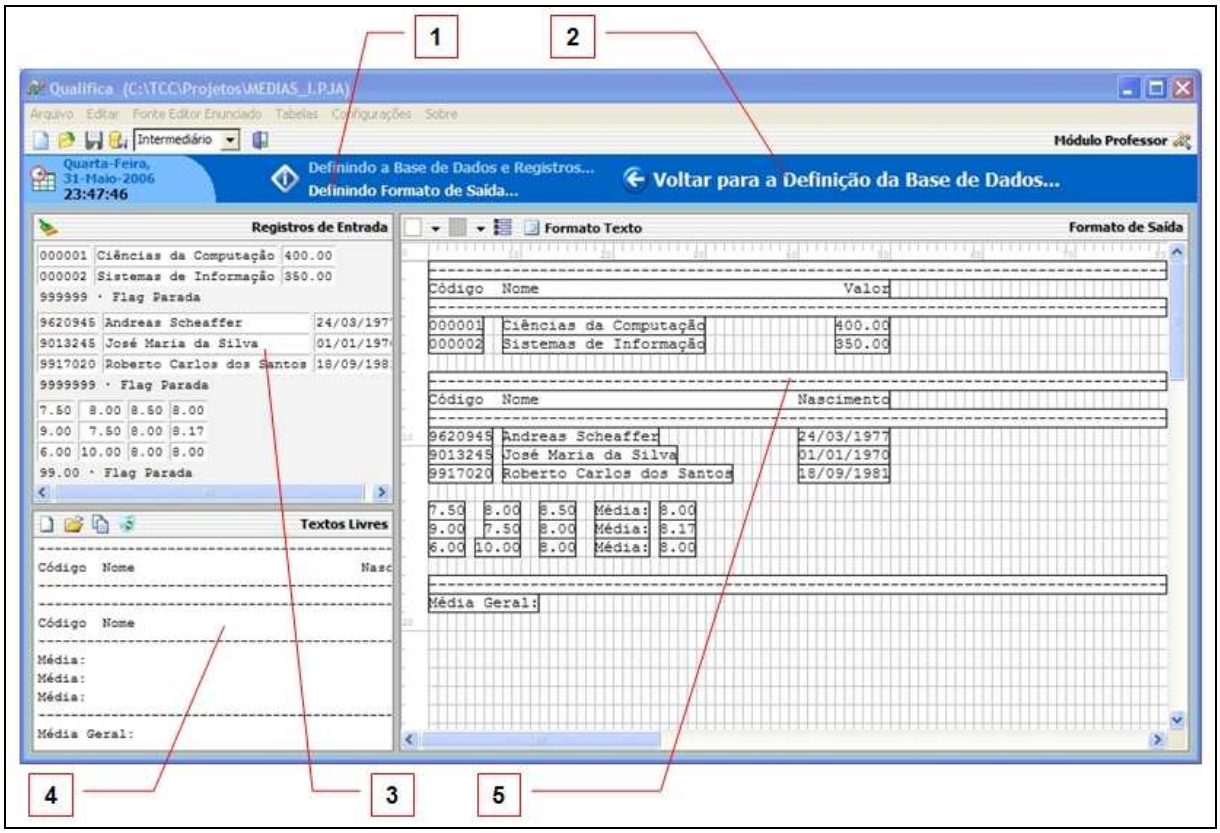


Figura 19 – Segunda tela principal do módulo professor

a) o item 1: mostra a fase atual da criação do exercício, que no caso o professor agora está Definindo a Formatação de Saída;

b) o item 2: um botão que permite o professor voltar para a fase anterior de criação do exercício que é a Definição da Base de Dados;

c) o item 3: mostra todos os registros de entrada que foram inseridos na fase anterior do exercício;

d) o item 4: uma lista contendo todos os textos livres inseridos no formato de saída;

e) o item 5: a área de formatação de saída propriamente dita, onde os registros de entrada e textos livres são posicionados.

A próxima seção detalha os requisitos para a criação de um exercício.

4.5.1 CRIAÇÃO DE UM EXERCÍCIO

O exercício consiste basicamente na descrição do enunciado de um problema ou estudo de caso, definição do grau de dificuldade do exercício que pode ser do tipo:

a) básico, sem a opção de agrupamento de entradas e saídas[8] e geração de procedimentos e laços de repetição. O código portugol será gerado com apenas um bloco principal de código (ou seja, permite configurar exercícios de pequena complexidade sem o uso de estruturas de dados homogêneas/heterogêneas e sem o conceito de modularização);
b) intermediário, com a possibilidade da geração de procedimentos, laços de repetição e com a utilização de estruturas do tipo *arrays* na construção da solução do problema;
c) avançado, também possibilita a geração de procedimentos, laços de repetição mas permite o uso de estruturas do tipo r*ecord* na construção da solução do problema.

Para definir o nível do exercício, deve-se selecionar uma opção na lista que encontra-se na barra de ferramentas na tela principal do módulo professor como ilustra a Figura 20 que no exemplo está configurado como Intermediário.

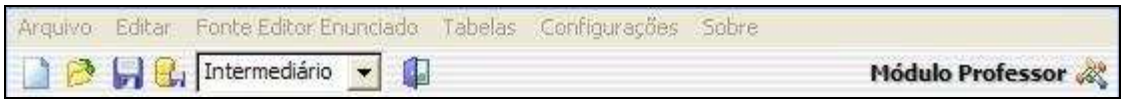


Figura 20 – A barra de ferramentas do menu principal

Nesta mesma barra de ferramentas encontram-se outras opções como: criar novo exercício, abrir um exercício, salvar o exercício atual, opção para salvamento automático do exercício em um intervalo de tempo que deve ser definido na tela de configurações gerais (Figura 21), acessada através da opção no menu principal <Configurações>.

Pode-se ainda configurar o idioma para apresentação do sistema, que além do português pode ser configurado para o idioma inglês, pode-se ver um exemplo conforme ilustram a Figura 81 e a Figura 82.

[8] Agrupamento de entradas e saídas: facilidade do sistema que permite ao professor estabelecer exercícios que envolvem o uso de matrizes e/ou registros tanto na entrada de dados como na saída dos mesmos.

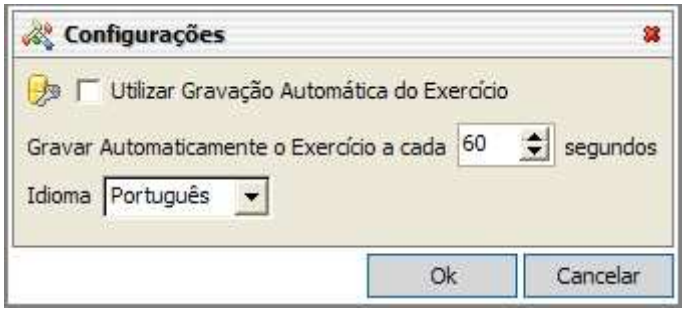


Figura 21 – Tela de configurações gerais

Em seguida deve-se definir o enunciado do problema conforme apresentado na próxima seção.

4.5.2 ENUNCIADO DO PROBLEMA OU ESTUDO DE CASO

O enunciado do exercício pode ser digitado no próprio programa ou em qualquer outro editor de textos e simplesmente colado na área do editor.

O editor de enunciado dispõe de uma barra com as ferramentas básicas para a edição de um texto, como recortar, copiar, colar, excluir, selecionar, alterar alinhamento, tipo, tamanho, cor e estilo de fonte, como é possível observar na Figura 22.

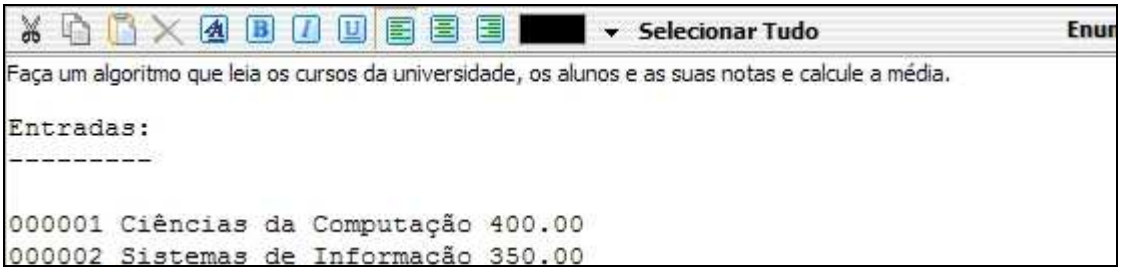


Figura 22 – O editor de enunciado com a sua barra de ferramentas

O enunciado do problema contempla também um exemplo de entradas e saídas esperadas para o programa a ser construído. O conteúdo das entradas é detalhado pelo professor na forma de tabelas. Com isto é possível obterem-se informações adicionais sobre o tipo dos campos a serem utilizados na solução do problema.

Portanto, o próximo passo é a definição da estrutura das tabelas o que consiste em inserir, para cada tabela, a quantidade necessária de registros e finalmente definir a formatação da saída dos registros com auxílio de textos livres.

4.5.3 DEFININDO A ESTRUTURA DAS TABELAS

Ao incluir uma tabela deve ser informando o seu nome e apelido (*alias*). Deve-se então especificar os seus atributos (Figura 25), informado o nome físico, nome lógico e tipo (se o tipo for *string*, deverá ser informado também o tamanho do campo).

Figura 23 – Inserindo uma tabela

Pode-se alterar (Figura 24) ou salvar a tabela com um outro nome e / ou outro *alias*, basta dar um duplo clique no nome da tabela desejada na lista de tabelas (Figura 25), e informar um outro nome e *alias* desde que não existe uma outra tabela com o novo nome ou o novo *alias*.

Figura 24 – Salvando a tabela com outro nome e/ou *alias*

Como pode ser visto na Figura 25, na alteração de algum atributo o nome do campo fica indisponível para edição. Quando o tipo for diferente de *string* o campo tamanho também é desabilitado.

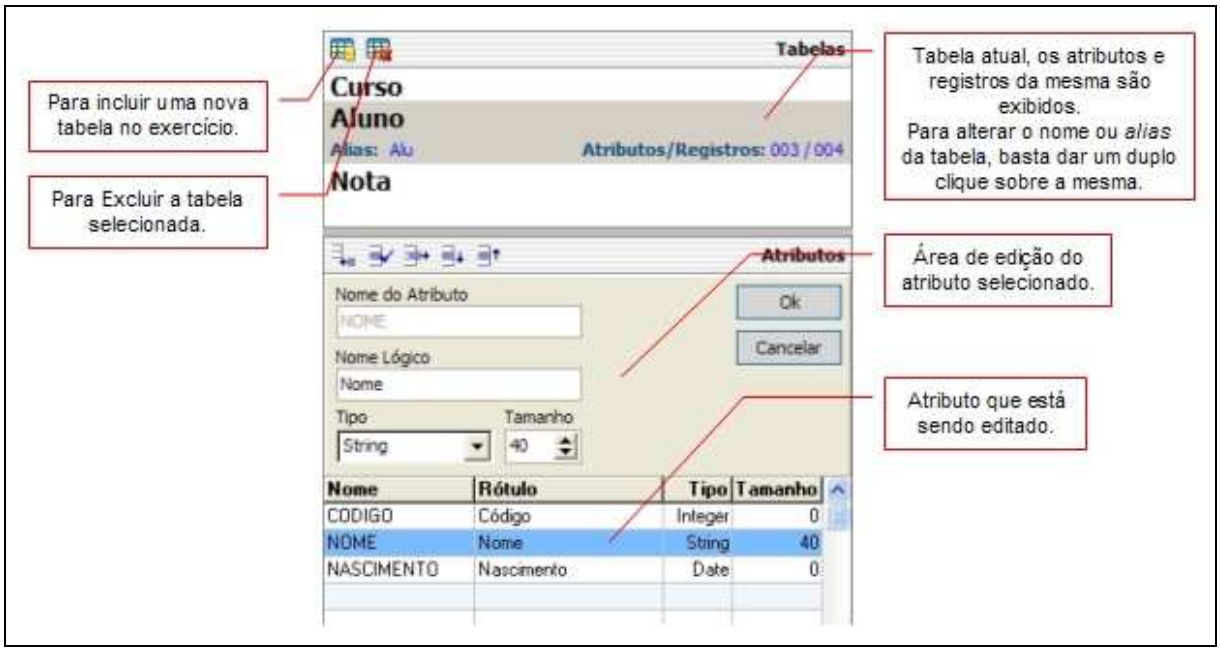
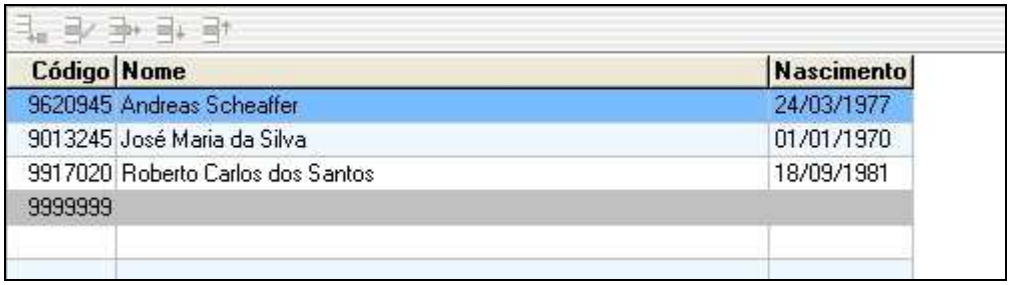


Figura 25 – Definindo a estrutura das tabelas

Para o nome físico não são permitidos nomes repetidos nem caracteres especiais ou de acentuação. O nome lógico é uma espécie de rótulo para o atributo, é a descrição que será apresentada na grade e na edição dos registros (ver Figura 26 para obter um exemplo de uso desta informação).



Código	Nome	Nascimento
9620945	Andreas Scheaffer	24/03/1977
9013245	José Maria da Silva	01/01/1970
9917020	Roberto Carlos dos Santos	18/09/1981
9999999		

Figura 26 – Grade dos registros de entrada

4.5.4 INSERINDO REGISTROS

Depois de definidas a tabela e a sua estrutura, o passo seguinte é inserir os registros de cada tabela, ou seja, criar uma base de exemplos de dados de entrada que podem ser utilizados pelo aluno como referência na solução do problema. Os campos são apresentados na tela de edição (Figura 27) conforme a ordem em que estiverem dispostos na grade de atributos (Figura 25). Além disso, conforme o tipo de cada atributo exige que o conteúdo de cada informação inserida respeite o tipo definido.

Todos os campos da tela de edição devem ser preenchidos, com exceção de atributos do tipo *date* e *real* e que podem ser persistidos com conteúdo vazio.

Na área inferior da tela de edição dos registros existe o botão <Ok> para gravar e sair da tela de edição, o botão <Cancelar> para cancelar a edição sem gravar e o botão <Aplicar> para gravar o registro corrente e criar um novo registro para edição sem sair da tela, como mostra a Figura 27. Na alteração de um registro, o botão <Aplicar> ficará desabilitado.

Figura 27 – Exemplo de uma tela de edição de registro

Caso o exercício seja de nível avançado é exibido um campo no canto superior direito da tela de edição denominado “Flag de Parada”, sendo obrigatória a definição de um registro desse tipo, pois em um exercício de nível intermediário ou avançado são definidos os agrupamentos de procedimentos e/ou laços de repetição que exigem uma entrada que identifique como sendo aquela que fará a quebra do *looping*.

Depois da inserção do registro de flag de parada não é mais permitido inserir novos registros naquela tabela. Se o usuário desejar inserir novos registros, deverá excluir o flag de parada, inserir os registros desejados e novamente inserir o flag de parada.

4.5.5 FORMATAÇÃO DAS SAÍDAS

Uma vez que, foram introduzidos os dados de exemplo de entrada de dados, o próximo passo é configurar o modelo de relatório de saída que o programa deverá apresentar.

Para avançar até a tela de formatação de saídas, na parte superior do sistema existe uma barra informando a data, dia da semana, hora atual e a fase atual da elaboração do exercício, basta clicar então no botão <Ir para a Formatação de Saída...> como mostra a Figura 28.

Figura 28 – Barra superior com informações sobre a ela

Um formato de saída é constituído basicamente dos registros de entrada que são selecionados e os textos livres que podem ser opcionalmente adicionados ao relatório de saída. Depois de selecionados os registros de entrada e inseridos alguns textos livres é necessário posicionar os mesmos dentro do formato de forma legível e ordenada.

4.5.5.1 Selecionando as entradas para o relatório de saída

Todos os registros de entrada que foram inseridos anteriormente devem ser utilizados para a geração do exemplo de dados de saída a partir dos dados de entrada (Figura 29).

Para selecionar uma entrada, basta clicar com o *mouse* e a mesma poderá ser visualizada na grade ao lado direito onde o leiaute de saída está sendo modelado (Figura 32).

Para desmarcar todos os registros de entrada que foram selecionados, basta clicar no botão localizado na parte superior esquerda da barra de título da área dos registros de entrada como mostra a Figura 29. Os registros de entrada que são selecionados ficam como a aparência de um botão pressionado ou aprofundado, como mostra o exemplo da Figura 29.

Nem todos os registros de entrada podem ou devem necessariamente ser selecionados para o formato de saída, alguns campos, como, por exemplo, os *flags* de parada podem não ser exibidos dentro do formato.

Cabe salientar que, o professor pode elaborar um exercício e não produzir nenhum leiaute de saída.

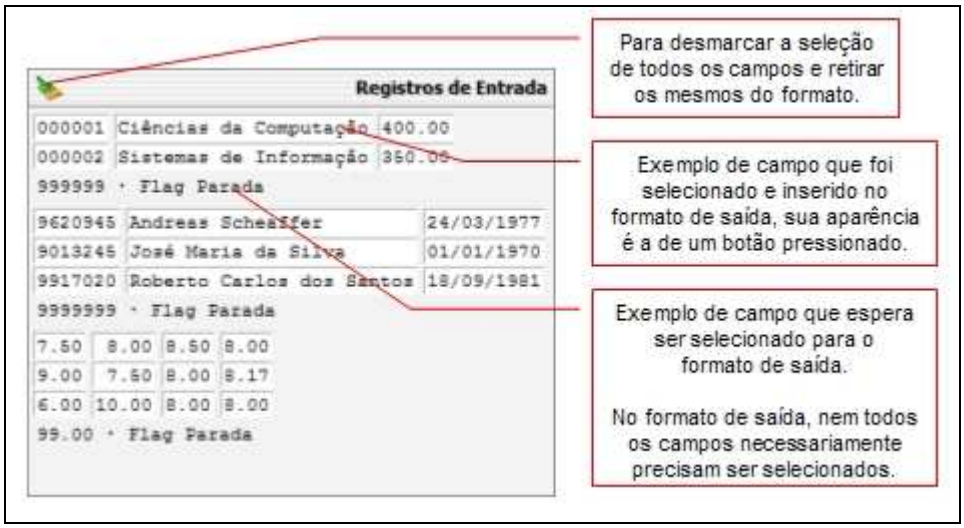


Figura 29 – Registros de entrada que são selecionados para o formato

4.5.5.2 Textos livres

Na modelagem do leiaute de saída além dos registros de entrada é possível introduzirem-se textos livres (além de ser possível alterá-los, duplicá-los ou excluí-los).

Um texto livre não possui nenhum vínculo com os registros de entrada, e serve apenas para elaborar e/ou incrementar uma melhor aparência visual ao formato de saída, como por exemplo, linhas delimitadoras horizontais, títulos de colunas e etc.

Todos os textos livres criados são automaticamente exibidos no formato de saída, não sendo possível configurá-los para estarem visíveis ou não (para não exibir mais um texto livre, o mesmo deve ser excluído da lista).

Quando o texto livre é selecionado, tanto na lista ou dentro da área de modelagem do formato de saída, a sua posição dentro da lista é expandida possibilitando a visualização de algumas informações sobre o mesmo como a sua posição linha,coluna dentro do formato, a sua largura em caracteres e o tipo (se é um texto normal, um cabeçalho ou rodapé).

Para duplicar um texto livre basta selecioná-lo na lista e clicar no terceiro botão da barra de título. Em seguida deve-se definir a estrutura das tabelas, inserir para cada tabela a quantidade necessária de registros e finalmente definir a formatação da saída dos registros com auxílio de textos livres. Dentro da área do formato o texto livre duplicado é apresentado próximo ao texto livre de origem.

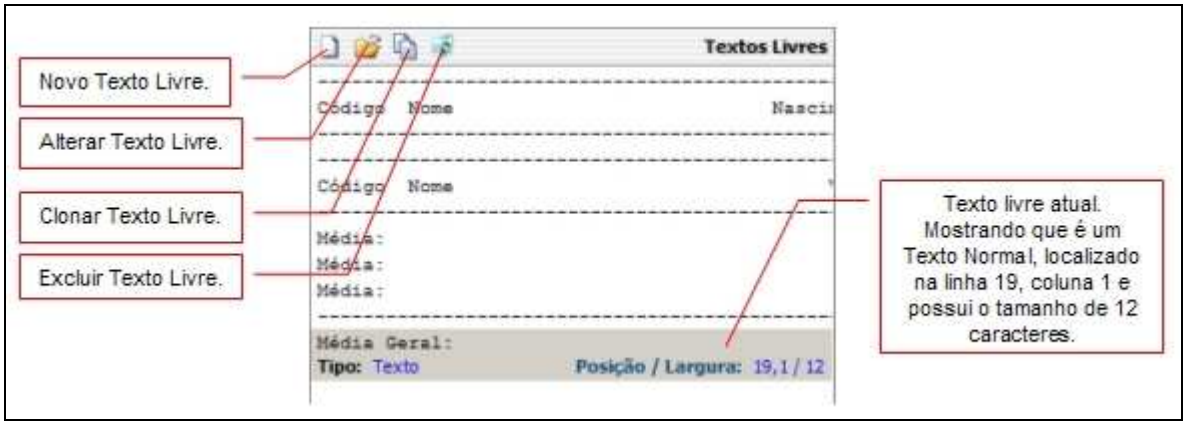


Figura 30 – Textos livres

Ao se criar um texto livre deve-se especificar o seu tipo. Se for um cabeçalho, quando for gerado o código fonte Portugol, esse texto livre será exibido sempre antes do agrupamento próximo ao qual estiver sendo inserido. Se for do tipo rodapé, será exibido no final do

agrupamento. Se o texto livre for do tipo Texto Normal ele será exibido como uma saída vinculada a uma entrada.

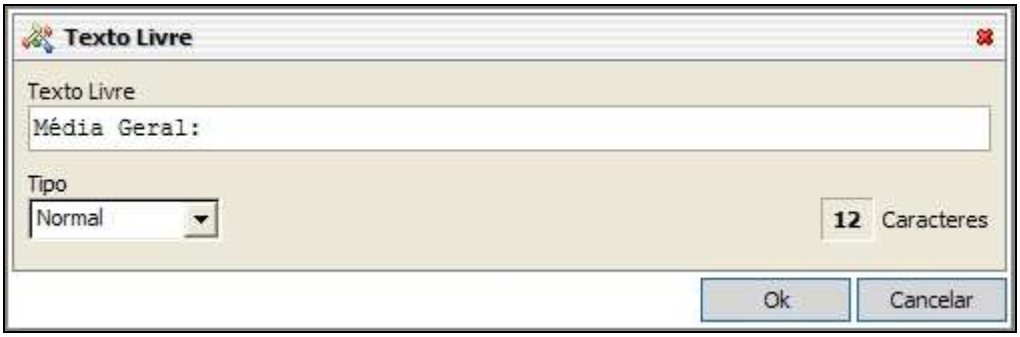


Figura 31 – Edição de um texto livre

Na medida em que o texto livre vai sendo digitado, um contador de caracteres é exibido no canto inferior direito da tela.

4.5.5.3 Modelando o formato de saída

Ao clicar com o *mouse* sobre qualquer entrada que se encontra na grade de formato de saída, desde que esta não seja um texto livre, é exibida uma dica informando qual a linha e coluna dentro do formato, ao mesmo tempo na área de registros de entrada (Figura 29) a entrada correspondente fica piscando de modo intermitente.

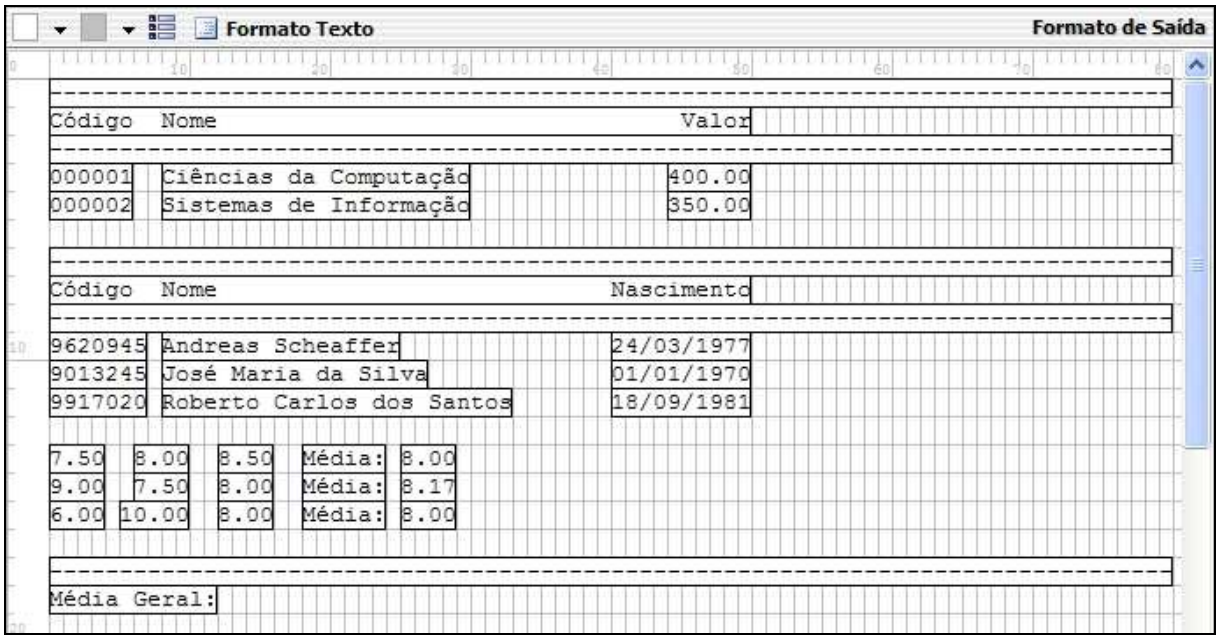


Figura 32 – Entradas selecionadas na grade de formatação de saída

Depois que os registros de entrada e os textos livres são selecionados, é necessário re-posicionar os mesmos corretamente, visto que, quando se seleciona uma entrada ou texto livre, o mesmo é inserido dentro do formato em uma posição *default*. Para isto basta clicar sobre a entrada ou texto livre e manter pressionado o *mouse* e arrastar e soltar sobre a linha e coluna desejada.

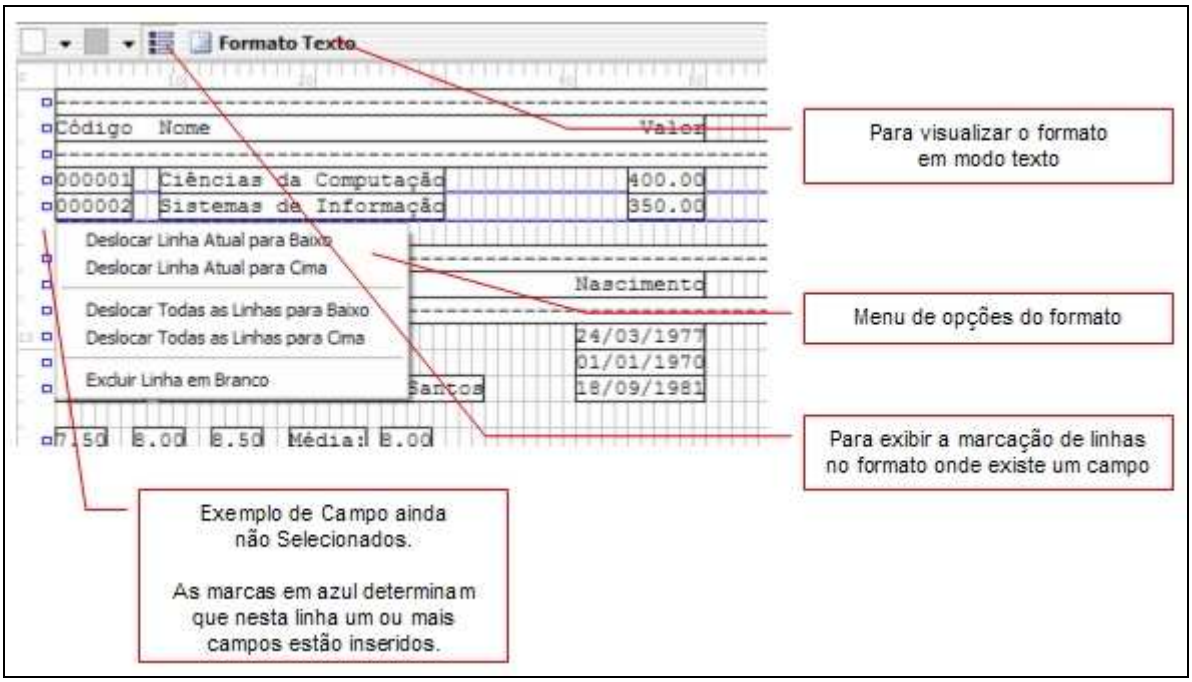


Figura 33 – Menu de opções

Existe um menu com algumas opções para movimentar um grupo de campos de uma mesma linha. Ao clicar na lateral esquerda da grade do formato de saída um menu é apresentado com algumas opções que podem auxiliar na montagem do formato como demonstrado na Figura 33.

a) Deslocar Linha Atual para Baixo: move todos os campos inseridos na linha em destaque uma linha abaixo;
b) Deslocar Linha Atual para Cima: move todos os campos inseridos na linha em destaque uma linha acima;
c) Deslocar Todas as Linhas para Baixo: move todos os campos inseridos em todas as linhas a partir da linha em destaque uma linha abaixo;
d) Deslocar Todas as Linhas para Cima: move todos os campos inseridos em todas as linhas a partir da linha em destaque uma linha acima;
e) Excluir Linha em Branco: não apaga os campos inseridos na linha corrente, mas, move todos os campos inseridos em todas as linhas abaixo da linha em destaque uma linha acima.

Para exibir o formato de saída em formato texto basta clicar no botão <Formato Texto> e uma tela será exibida (Figura 34). Existem dois botões no rodapé desta tela, para selecionar todo o texto e outro para copiar, este recurso é útil se no caso o professor desejar copiar o texto e colar no enunciado do exercício.

Figura 34 – O formato de saída visto em formato texto

4.5.6 SALVANDO O EXERCÍCIO

Para salvar o exercício, deve-se selecionar a opção <Salvar Exercício> ou clicar no terceiro botão localizado na barra de ferramentas (Figura 35) da tela principal.

A qualquer momento o professor pode salvar o exercício, interromper o desenvolvimento do mesmo, sair do módulo professor, e quando retornar poderá dar continuidade a partir do contexto anterior.

O arquivo com extensão PJA deve ser aberto no módulo aluno para o desenvolvimento da solução do exercício.

Depois de iniciado o desenvolvimento da solução do exercício por parte do aluno, o arquivo PJA pode ser alterado apenas no conteúdo dos registros, mas a estrutura das tabelas, bem como dos atributos não pode ser alterada, o que pode provocar o aluno a reiniciar a solução do exercício do ponto de partida.

4.6 MÓDULO ALUNO

Nesta seção é apresentada a arquitetura do módulo aluno. A estrutura da interface está organizada da seguinte forma (Figura 35):

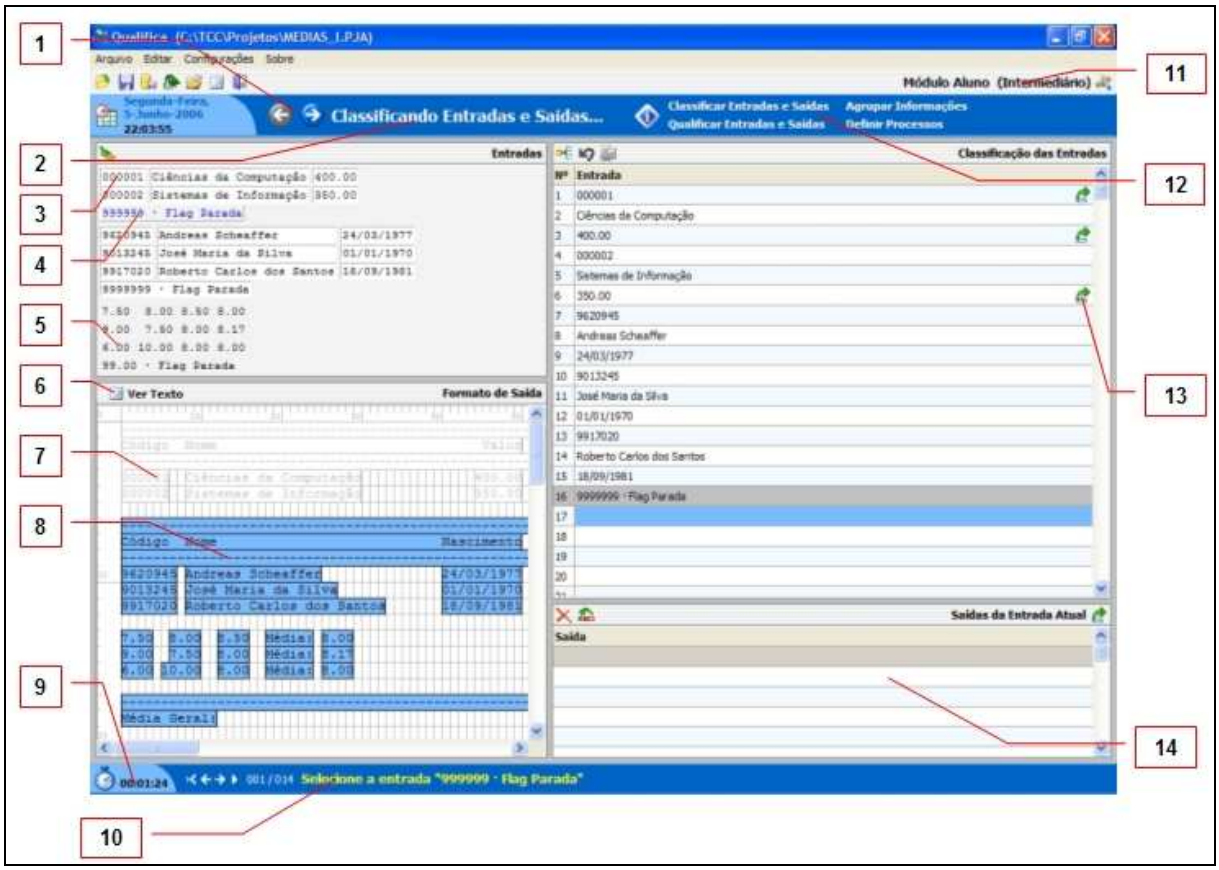


Figura 35 – As entradas na parte superior e as saídas no canto inferior

a) o item 1: botões para avançar e retornar as fases da solução do exercício, o aluno só poderá ir adiante a solução depois de concluir a fase atual;

b) o item 2: descreve a fase em que o exercício se encontra no momento: “Classificando as Entradas e Saídas...”, é fase em que o aluno terá que descobrir a seqüência de leitura das entradas e as saídas;

c) o item 3: os registros de entradas que foram selecionados, tem a aparência de um botão pressionado;

d) o item 4: um registro de entrada que está em destaque, quando o *mouse* é posicionado, a mesma se torna intermitente;

e) o item 5: as entradas que ainda não foram selecionadas;

f) o item 6: o botão para visualizar o formato de saída em formato texto;

g) o item 7: os itens do formato de saída, nas cores iguais a grade significam que já foram selecionados;

h) o item 8: os itens do formato de saída que ainda não foram selecionados;

i) o item 9: o tempo total que o aluno utilizou para resolver o exercício, a partir da primeira alteração em qualquer parte da tela o tempo é iniciado ou reiniciado caso o aluno esteja dando continuidade a um exercício que foi parcialmente resolvido;

j) o item 10: mensagem exibindo qual o próximo passo que o aluno deve tomar, no caso, qual o próximo registro de entrada que deve ser selecionado;

k) o item 11: exibe o nível do exercício, no caso aqui é intermediário;

l) o item 12: mensagens informando quais fases da solução do exercício que o aluno já completou;

m) o item 13: exibe a grade de classificação das entradas, aqui especificamente uma entrada que possui saídas vinculadas a ela;

n) o item 14: a grade de classificação das saídas, caso a entrada tenha alguma saída vinculada é então exibida.

4.6.1 RESOLVENDO UM EXERCÍCIO NO MÓDULO ALUNO

Para a utilização do sistema, o aluno receberá um arquivo contendo a especificação do exercício proposto. A resolução consiste basicamente na seleção das entradas, seleção das saídas que estão associadas àquelas entradas, definição do processamento a ser realizado sobre as entradas para a geração das saídas, agrupamento das informações (momento onde são definidos laços de repetição e/ou procedimentos – caracterizados pela repetição de entrada de dados) e por fim, a geração do pseudocódigo em Portugol.

4.6.2 SELECIONANDO AS ENTRADAS

Depois de abrir o arquivo com o exercício, deve-se selecionar os registros de entrada. Para isso, basta clicar sobre um dos campos dos dados de entrada (Figura 36). Ao clicar na entrada, a mesma é inserida na grade de classificação de entradas (Figura 37). Conforme as entradas vão sendo selecionadas, sua aparência toma a forma de um botão pressionado, e as entradas que esperam ser selecionadas tem a aparência de um texto normal. Ao posicionar-se o cursor sobre estas últimas é exibido um efeito piscante como demonstrado na Figura 37.

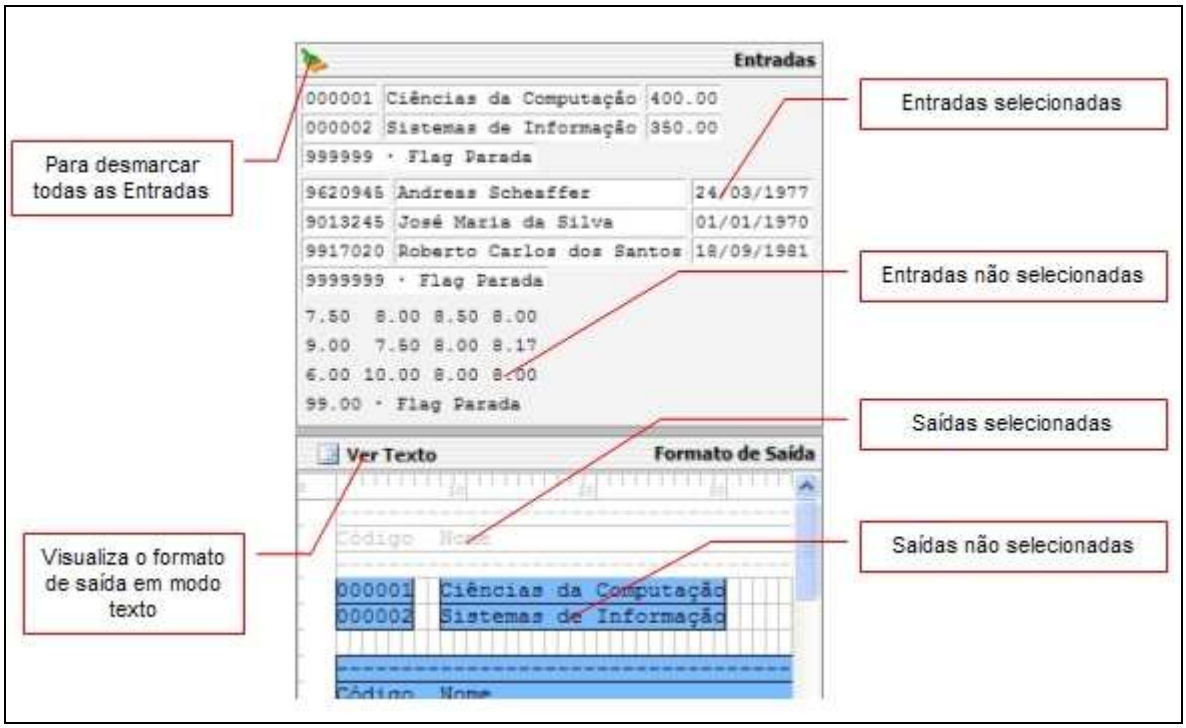


Figura 36 – As entradas na parte superior e as saídas no canto inferior

Todas as entradas deverão ser selecionadas para o aluno avançar para a próxima fase que é a de classificação das entradas.

Figura 37 – Entradas selecionadas para a classificação

4.6.3 SELECIONANDO AS SAÍDAS

Para selecionar uma saída, o processo é semelhante a seleção das entradas. Contudo o aluno deve associar (escolher) para qual entrada aquelas saídas serão inseridas. Para tanto

deve-se selecionar uma entrada na grade de classificação de entradas e selecionar as saídas correspondentes clicando-se com o *mouse* sobre as mesmas.

Depois de a saída ser selecionada a sua aparência se destaca sendo exibida com fundo azul. As saídas que esperam ser selecionadas são visualizadas com contornos azuis como mostra a Figura 36.

Após todas as entradas e todas as saídas serem selecionadas, uma mensagem no rodapé principal do programa (Figura 38) irá exibir uma mensagem para o aluno de que ele está apto a avançar na solução do exercício.

Figura 38 – Mensagem para avançar para a próxima fase

Para avançar para a próxima fase, deve-se clicar no botão descrito no item 1 conforme a (Figura 35), então o aluno poderá notar que as informações (Figura 39) na barra superior mudaram.

Figura 39 – Próxima fase: Qualificando entradas e saídas

A área onde estão localizadas as entradas e o formato de saída (Figura 36) torna-se invisível a partir deste momento, permitindo ao aluno se concentrar no próximo passo da solução.

4.6.4 QUALIFICANDO AS ENTRADAS E AS SAÍDAS

Depois de selecionar todas as entradas e saídas é necessário classificar as mesmas, ou seja, atribuir nomes de variáveis a cada uma delas. Para cada entrada e saída, deve-se dar um nome e especificar o tipo da variável que receberá aquele valor de entrada.

Para dar um nome a uma entrada basta clicar na célula da coluna nome e digitar o nome desejado, como demonstrado na Figura 40. Outra possibilidade é selecionar um nome da lista, lembrando que caso o exercício seja de nível básico não poderão existir entradas com nomes repetidos.

A Figura 40 e a Figura 41 ilustram o que foi descrito anteriormente, para confirmar o nome para a entrada ou saída é necessário o pressionamento da tecla <ENTER> e automaticamente o cursor será posicionado na linha seguinte para repetir o processo.

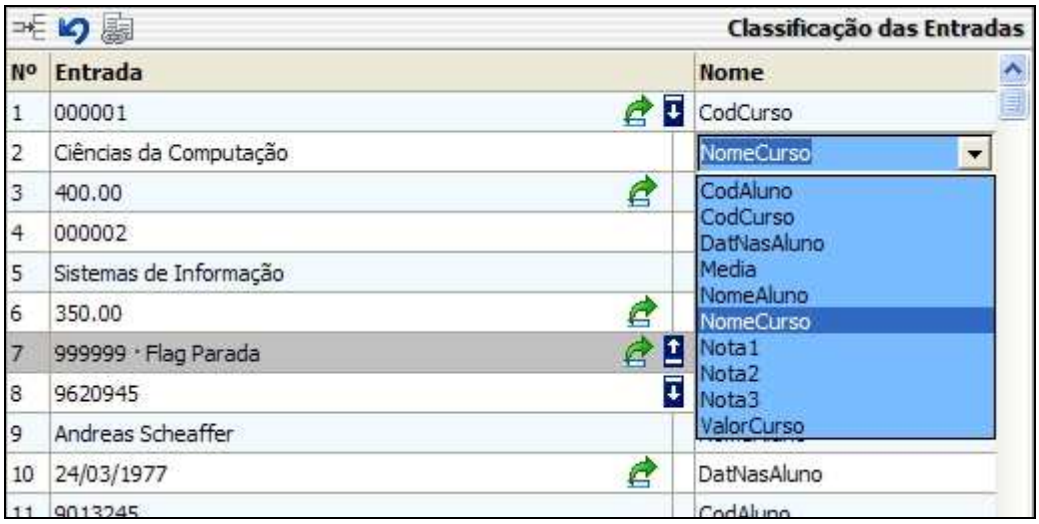


Figura 40 – Definindo um nome para a entrada

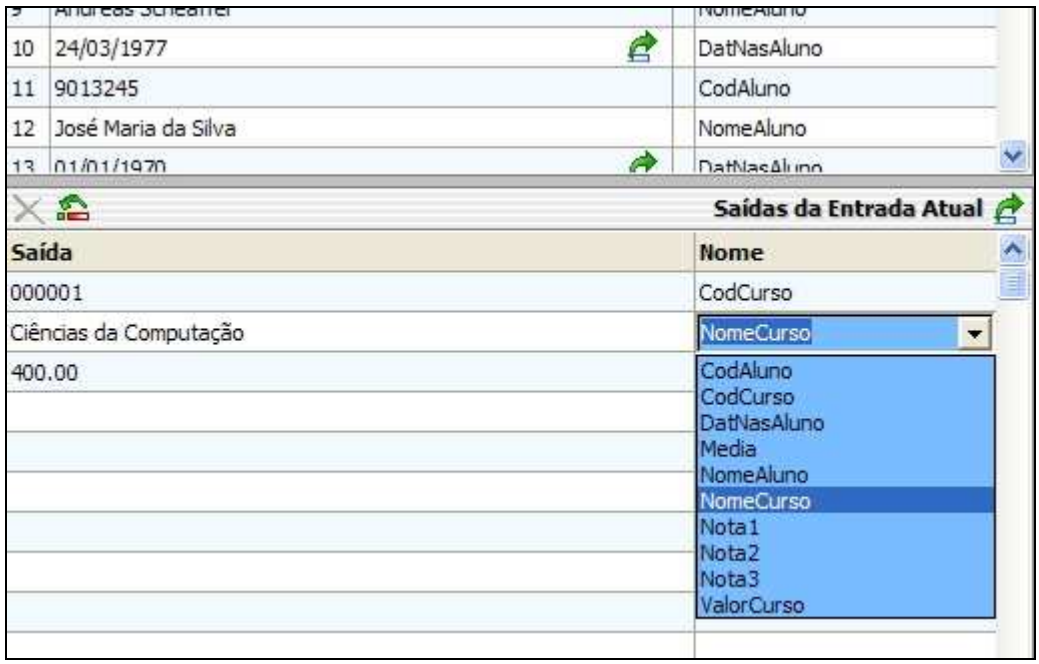


Figura 41 – Definindo um nome para a saída

Vale salientar que na definição dos nomes das saídas, quando esta for caracterizada por um texto livre, não será necessário nomeá-la.

Sempre que uma entrada ou uma saída é nomeada, o mesmo nome é inserido na lista de identificadores (Figura 42) para ser configurado o seu rótulo, tipo e caso o exercício seja do nível avançado, o *alias* a que pertence o identificador.

Figura 42 – Qualificando os identificadores das entradas e saídas

Para informar um rótulo para o identificador basta clicar com o *mouse* ou pressionar a tecla <F2> sobre a segunda coluna da lista e a mesma ficará em modo de edição, digita-se então a descrição e tecla-se <ENTER> para confirmar o rótulo para o identificador.

Para a especificação dos tipos foram utilizados como padrão os tipos da linguagem de programação Pascal, que são: *boolean*, *char*, *date*, *integer*, *real* e *string*.

A Figura 42 mostra como se define o tipo de uma entrada - clicando-se com o *mouse* na célula da coluna Tipo e selecionando-se uma das opções dentro da lista fixa.

Após todas as entradas e todas as saídas serem qualificadas, a mesma mensagem no rodapé principal do programa (Figura 38) irá informar ao aluno de que ele está apto a avançar na solução do exercício.

4.6.5 AGRUPANDO INFORMAÇÕES

Se o exercício proposto for de nível intermediário ou avançado, o mesmo possibilita a definição de agrupamentos de entradas (Figura 43), caso contrário a opção de agrupamento será desabilitada, ao passo que no momento que o aluno finalizar a fase de qualificação das entradas, saídas e identificadores, a próxima fase do exercício será a definição de processos para as entradas e saídas.

Estes agrupamentos podem ser classificados segundo o seu tipo: procedimento ou laço de repetição - tipo “procedimento” irá gerar dentro do código fonte uma *procedure* com a

parte do código que estiver entre a linha inicial e final; o tipo “laço de repetição” poderá gerar um *repeat until*, um *for do* ou um *while do*, conforme o tipo do laço que se deseja implementar. Um exemplo de uso deste recurso é apresentado na Figura 43.

O conceito de agrupamento de entradas diz respeito ao processo repetitivo de entrada de dados em um programa que é associado a um campo (ou a vários campos de um registro ou de um array) até que uma condição de fim seja detectada.

Quando o processo de associação entre entradas e saídas é encerrado, a janela que contém estas informações passa a ficar oculta e é desabilitada a possibilidade de qualquer alteração nas mesmas.

Contudo, o aluno pode consultá-las clicando no botão localizado na parte superior da barra lateral (Figura 45).

Figura 43 – Agrupamento informações

O processo de agrupamento (Figura 43) consiste em clicar na linha da entrada inicial e arrastar o mouse até a linha da entrada final (que deve ser uma entrada definida no módulo professor como “Flag de Parada”).

Depois de marcado o bloco, basta clicar no botão “Agrupar” (Figura 43) e a tela de cadastramento do agrupamento (Figura 44) irá aparecer para informar o nome do agrupamento, tipo do agrupamento – “procedimento” ou “laço de repetição”, tipo do laço de repetição e uma dica ou descrição sobre o que este agrupamento trata, esta descrição irá aparecer no código fonte no Portugol.

É mostrado também o intervalo das linhas inicial e final que foram selecionadas na grade de entradas (Figura 43) e que resultarão no novo agrupamento.

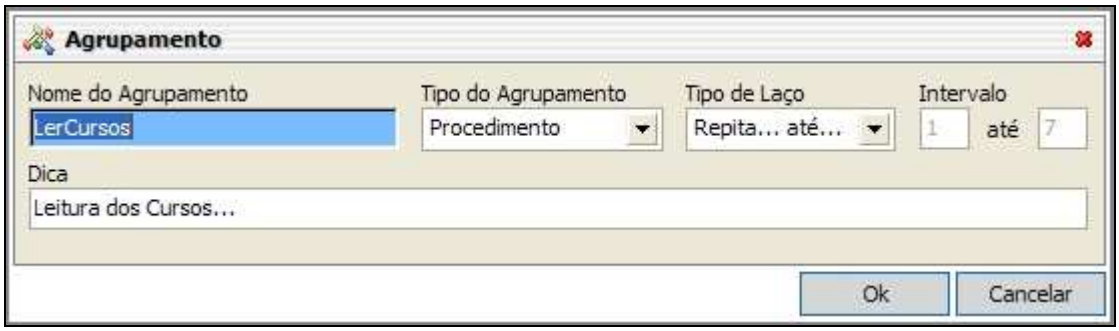


Figura 44 – Tela de cadastramento de agrupamentos

Ao confirmar os dados, o agrupamento é inserido em uma lista (Figura 45) onde é possível visualizar os seus detalhes, como: dica, tipo, tipo do laço e o intervalo de linhas.

Figura 45 – Lista de agrupamentos

Através dos dois botões localizados na área da lista dos agrupamentos, pode-se alterar os seus dados ou excluí-lo do exercício. Para cada agrupamento pode-se ainda visualizar o seu código fonte Portugol específico, conforme a Figura 46, o que ainda não é a solução final do exercício, apenas para que o aluno possa analisar separadamente o agrupamento das informações com o seu código gerado.

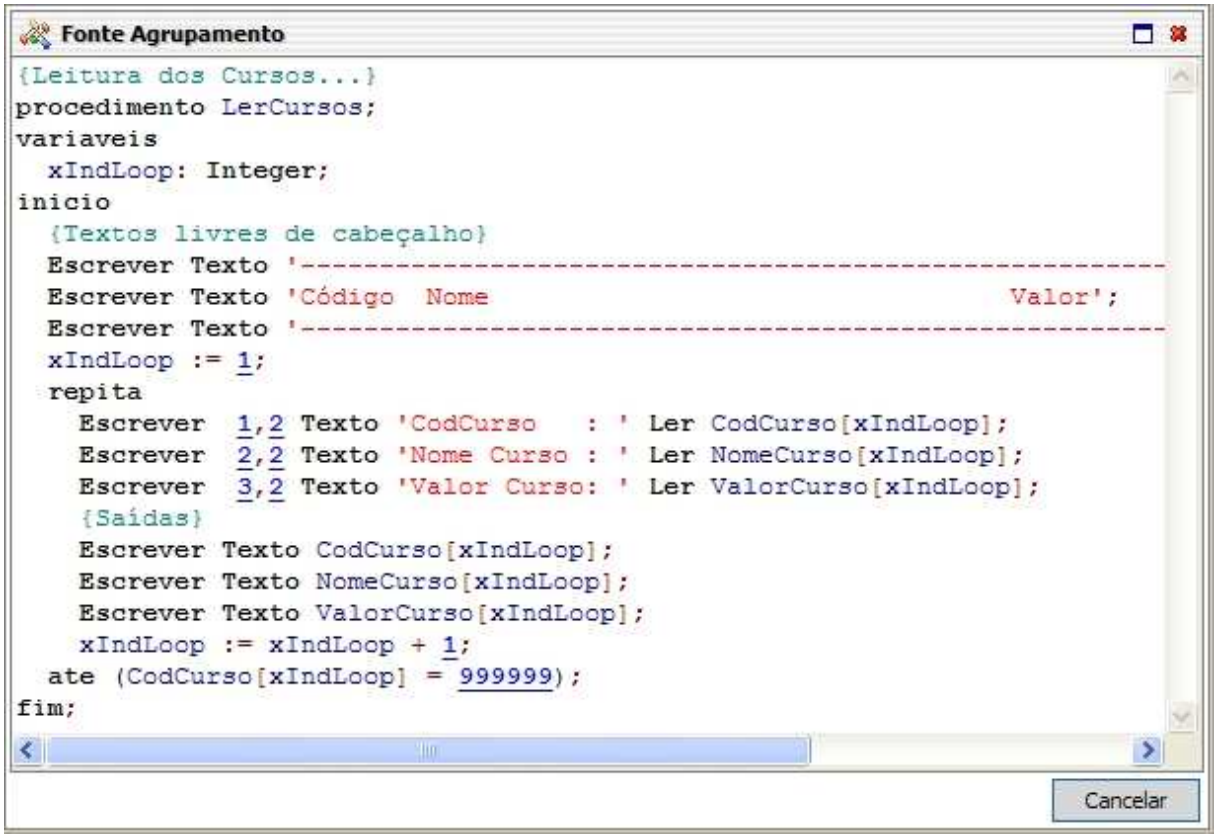


Figura 46 – Código fonte do agrupamento

Terminada a seleção dos agrupamentos pode-se avançar para a próxima fase do exercício que é a definição dos processos para as entradas e saídas.

Já nesta fase o aluno pode visualizar no rodapé principal do programa a mensagem final (Figura 47) informando que resta apenas definir os processos para as entradas e saídas e gerar o código fonte.

Figura 47 – Mensagem final

4.6.6 DEFINIÇÃO DE PROCESSAMENTO

Para a definição dos processos, deve-se selecionar clicando com o *mouse* sobre a entrada ou a saída desejada e digitar as linhas de código na área de processos, conforme a Figura 48.

Lembrando que o programa não realiza nenhuma consistência em relação aos processos, como análise da sintaxe e semântica de comandos.

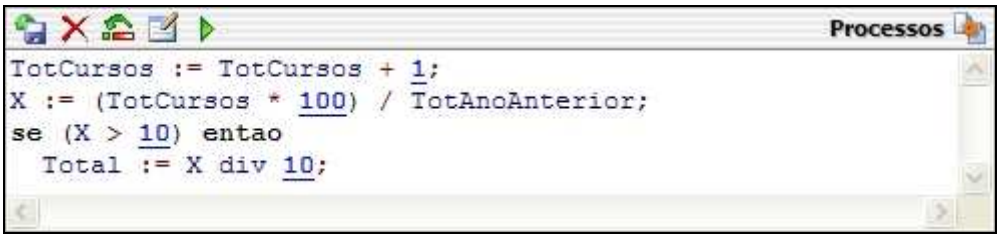


Figura 48 – Digitação de código de processamento

Depois de inserido o código de processamento das entradas e/ou saídas, o programa realiza uma análise em cada linha de código separando as variáveis, números, linhas de comentário, caracteres de pontuação, atribuição, comandos, funções para serem qualificados, conforme a Figura 51.

Após digitadas as linhas de código deve-se salvar as mesmas clicando-se no primeiro botão localizado na barra de título conforme ilustra a Figura 48.

Outros botões estão dispostos nesta mesma área, e servem respectivamente para excluir os processos, setar os processos como “condicionais”, digitar palavras reservadas e exibir informações sobre macros. Ao configurar um processo como “condicional”, no código fonte será gerado uma linha de condição para o(s) mesmo(s), e a condição será o conteúdo do identificador, que no exemplo será caso o Valor do Curso for igual a “400.00”, o esboço do código fonte de cada identificador pode ser visualizado logo abaixo da área dos agrupamentos e variáveis conforme ilustra a Figura 49 e a Figura 52 com um processo condicional.

A Figura 49 ilustra a descrição das macros disponíveis em estreita barra logo abaixo da barra dos botões, e um exemplo de código utilizando as macros. A macro ‘<@>’ é substituída pelo nome do campo atual da qual os processos foram inseridos, ‘NT_Media’, e a macro ‘<!>’ substitui o índice atual do laço de repetição, ‘xIndLoop’ é o nome de uma variável fixa usada para controlar os índices dos *arrays* e indexador dos laços de repetição.

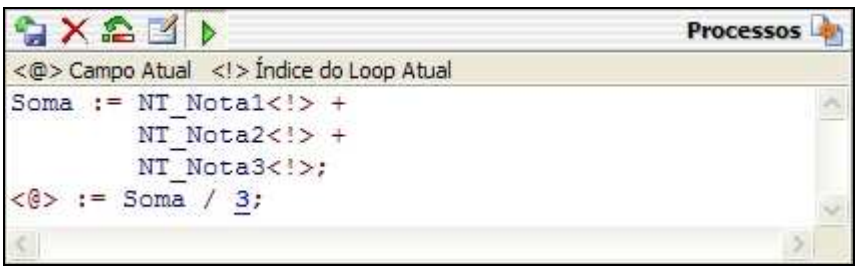


Figura 49 – Utilizando macros nos processos

Já a Figura 50 mostra o código fonte com o resultado da utilização das macros nas linhas de processos.

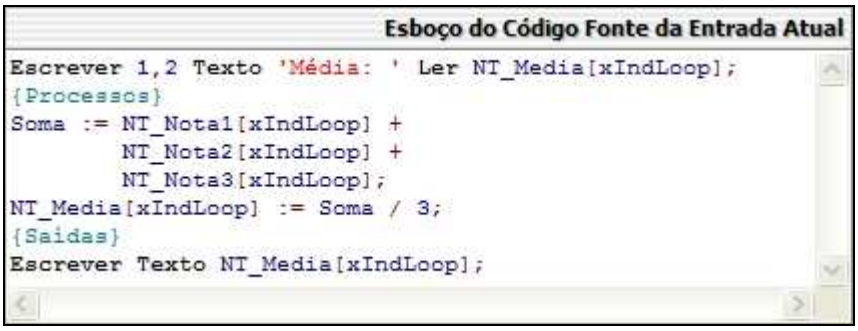


Figura 50 – Resultado do código fonte utilizando macros

Para associar um tipo a uma variável, o processo é semelhante a qualificação dos identificadores. Para tanto basta clicar com o *mouse* sobre a mesma e uma lista irá aparecer com os tipos disponíveis.

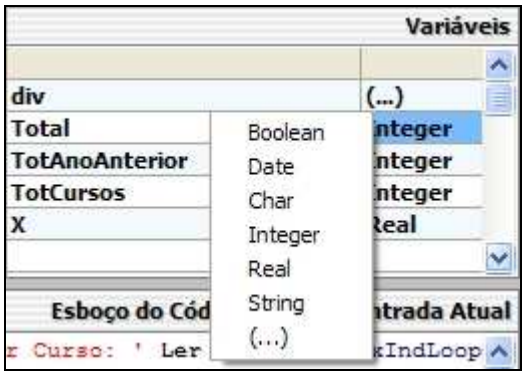


Figura 51 – Lista de variáveis

No exemplo da Figura 48 pode-se notar que o programa ignorou as palavras reservadas “se” e “entao”, essas palavras não aparecem para ser qualificadas, ao contrário da palavra “div” que é um comando para executar a divisão exata entre dois números, o aluno deverá identificar e selecionar na lista o tipo “(...)”, significando que não é uma variável, mas um comando ou função.

Figura 52 – Esboço do código fonte com processo condicional

Para gerenciar as palavras reservadas que serão utilizadas na digitação dos processos, basta clicar sobre o quarto botão (Figura 48) localizado na área de digitação de processos.

Na tela das palavras reservadas (Figura 53) encontra-se uma barra de botões para respectivamente o aluno poder incluir, alterar e excluir palavras reservadas.

Depois da palavra ser inserida na lista, a mesma será ignorada como variável a ser tipificada. Para confirmar a inserção, alteração ou exclusão de palavras da lista, deve-se clicar no botão <Ok> e a mesma é salva em um arquivo chamado “Qualifica.rwd” que fica localizado no mesmo diretório onde está o arquivo executável da aplicação.

Figura 53 – Lista de palavras reservadas

4.6.7 GERAÇÃO DO PSEUDOCÓDIGO

Depois de haverem sido selecionadas e classificadas todas as entradas e todas as saídas correspondentes e houver sido definidos os agrupamentos, processamentos associados e qualificadas as variáveis é possível a geração do pseudo-código em Portugol da solução construída pelo aluno, para isto, basta clicar no sexto botão localizado na barra de botões de atalho logo abaixo do menu principal como mostra a Figura 54.

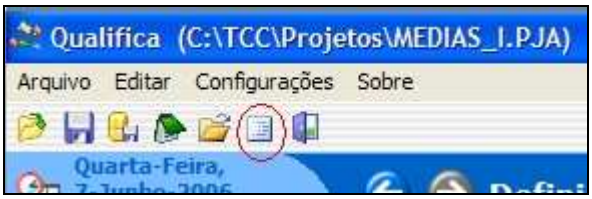


Figura 54 – Barra de atalhos do menu principal

A Figura 55 apresenta uma parte do código onde é possível visualizar um procedimento que foi gerado a partir da definição de um agrupamento na grade de classificação das entradas. Pode-se perceber que foram selecionadas todas as entradas referentes aos dados cadastrais da tabela "Aluno". Depois do comando de leitura da entrada "Data Nascimento Aluno" e das suas saídas, aparece os seus processos e uma chamada de procedimento "LerNotas" que é um agrupamento dos dados cadastrais da tabela "Notas".

Código Fonte

```
{Leitura dos Alunos...}
procedimento LerAlunos;
variaveis
  xIndLoop    : Integer;
  Soma        : Real;
  ListaAlunos: Integer;
  TotAlunos   : Integer;
inicio
  ListaAlunos := 0;
  TotAlunos   := 0;
  xIndLoop := 1;
  repita
    Escrever  1,2 Texto 'Código Aluno          : ' Ler CodAluno[xIndLoop];
    Escrever  2,2 Texto 'Nome Aluno            : ' Ler NomeAluno[xIndLoop];
    {Processos}
    ListaAlunos := ListaAlunos + 1;
    Escrever  3,2 Texto 'Data Nascimento Aluno: ' Ler DatNasAluno[xIndLoop];
    {Processos}
    TotAlunos := TotAlunos + 1;
    {Saídas}
    Escrever Texto CodAluno[xIndLoop];
    Escrever Texto NomeAluno[xIndLoop];
    Escrever Texto DatNasAluno[xIndLoop];
    LerNotas;   {Chamada de Procedimento}
    xIndLoop := xIndLoop + 1;
  ate (CodAluno[xIndLoop] = 9999999);
  {Saídas}
  Escrever Texto '-------------------------------------------------------------------------------
  Escrever Texto 'Média Geral:';
fim;

inicio
  LerCursos;   {Chamada de Procedimento}
  LerAlunos;   {Chamada de Procedimento}
fim.
```

Cancelar

Figura 55 – O Código fonte em Portugol gerado

Pode-se ver os processos que foram inseridos para a entrada "Nome Aluno".

O próximo capítulo apresenta um estudo de caso do emprego do sistema na solução de um exercício proposto.

5 ESTUDO DE CASO

Este capítulo tem como objetivo demonstrar a operacionalidade e as funcionalidades do software desenvolvido. Para tanto foi criado um exemplo de nível avançado que demonstra o funcionamento do mesmo, desde a sua concepção por parte do professor até a sua solução por parte do aluno.

5.1 EXEMPLO DE UM EXERCÍCIO DE NÍVEL AVANÇADO

Para viabilizar a análise, considere-se como exemplo um problema onde exista uma relação de bancos e os clientes relacionados a cada banco. A proposta de problema é construir um algoritmo que faça a leitura dos bancos até que o usuário informe ‘999’ para o código do banco.

Como exemplo para a especificação do exercício pode-se utilizar a relação de bancos e clientes conforme a Figura 56 e a Figura 57.

Código	Nome
001	Banco do Brasil
004	Besc
003	Bradesco
002	Caixa Econômica Federal
005	Unibanco

Figura 56 – Relação dos bancos cadastrados

Após a leitura do código e do nome do banco, o algoritmo deve ler as informações do cliente de cada banco até que o usuário digite o código do cliente igual a ‘999999’.

Código	Nome	Nascimento	Sexo	Salário	E-mail
000001	Woody Woodpecker	10/06/1940	M	5.000,00	woody@woodpecker.com
000002	Buzz Buzzard	02/06/1902	M	2.000,00	buzz@buzzard.com
000003	Andy Panda	04/07/1935	M	900.00	andy@panda.com

Figura 57 – Relação dos clientes cadastrados

Após a leitura dos bancos e dos clientes, o programa deverá imprimir um relatório conforme o leiaute de saída representado pela Figura 58.

```
-------------------------------------------------------------------------
Código  Nome
-------------------------------------------------------------------------
001     Banco do Brasil
004     Besc
003     Bradesco
002     Caixa Econômica Federal
005     Unibanco

000001  Woody Woodpecker  10/06/1940  M  5.000,00  woody@woodpecker.com
000002  Buzz Buzzard      02/06/1902  M  2.000,00  buzz@buzzard.com
000003  Andy Panda        04/07/1935  M    900,00  andy@panda.com
-------------------------------------------------------------------------
```

Figura 58 – Leiaute de saída das informações

5.2 CRIANDO O EXERCÍCIO

Para criar um novo exercício basta clicar no primeiro botão da barra de ferramentas do menu principal, caso exista um exercício aberto (Figura 59).

Se existir um exercício aberto e o mesmo conter tabelas, atributos e registros, o programa irá perguntar se o professor deseja salvar o exercício aberto.

5.2.1 INSERINDO TABELAS

O primeiro passo então é inserir todas as tabelas que farão parte do exercício, informando o seu nome e apelido *(alias)*, conforme ilustra a Figura 59.

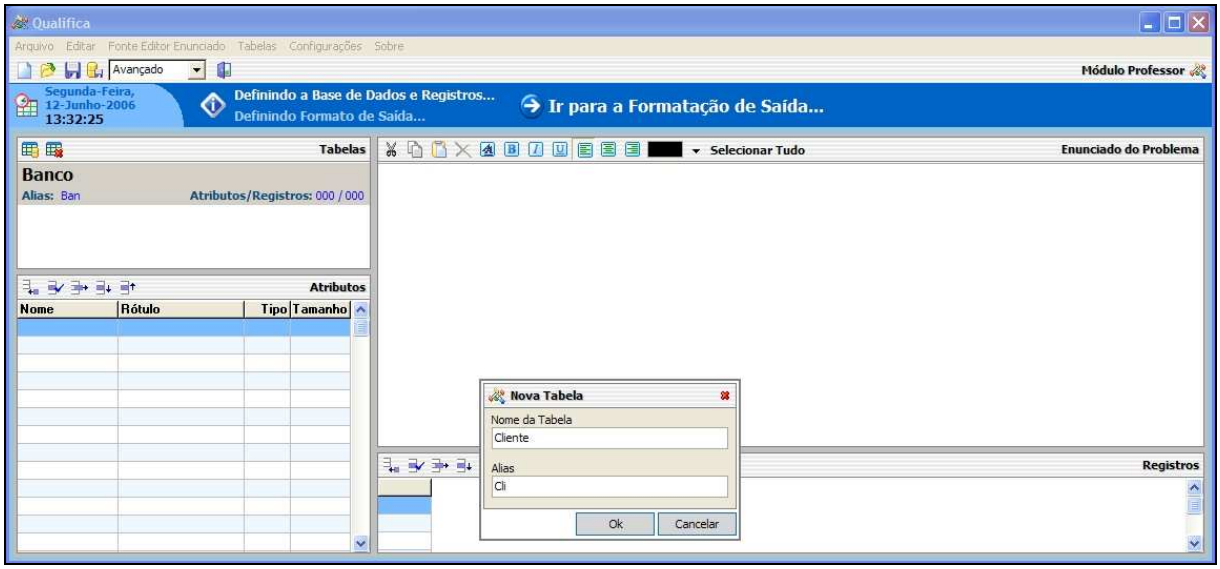


Figura 59 – Criando as tabela no exercício

5.2.2 CONFIGURAR A ESTRUTURA DE CADA TABELA

Depois de criadas as tabelas do exercício, deve-se selecionar cada tabela da lista e configurar a sua estrutura, ou seja, seus atributos, informando nome, nome lógico, tipo e tamanho caso o tipo seja *string*. A Figura 60 e Figura 61 ilustram esta operação.

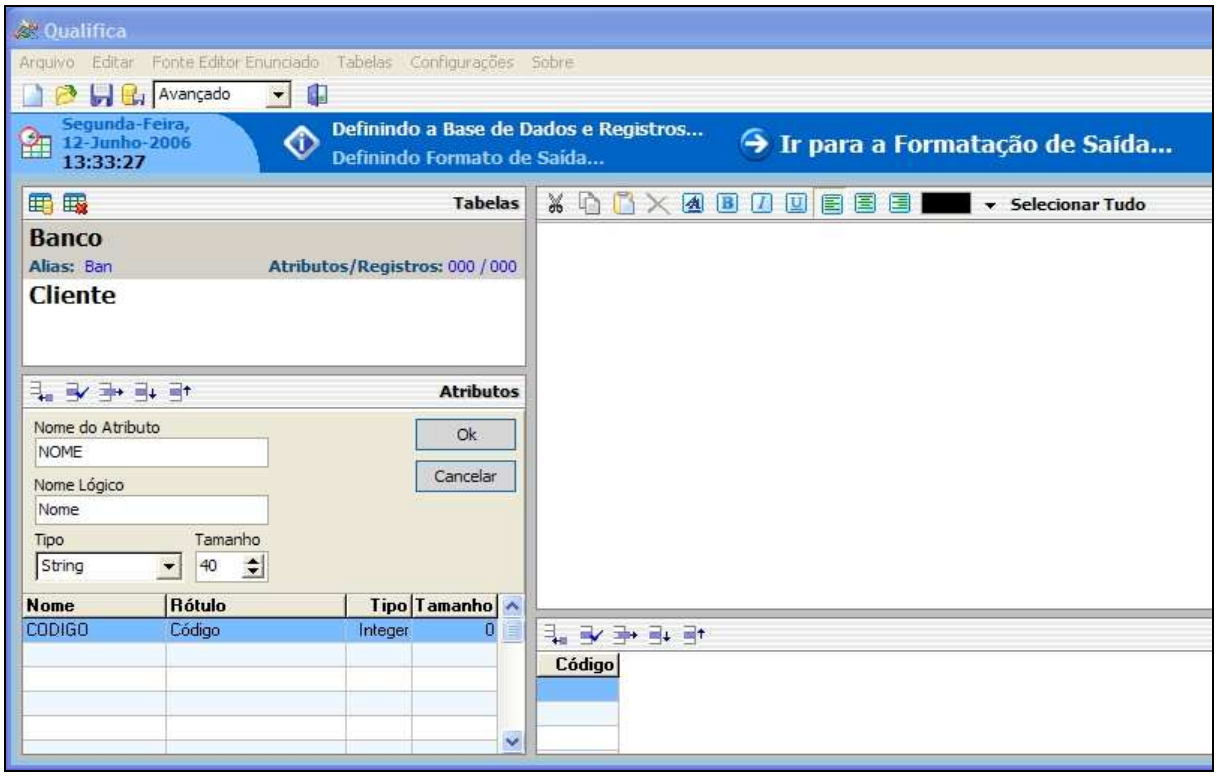


Figura 60 – Inserindo os atributos da tabela Banco

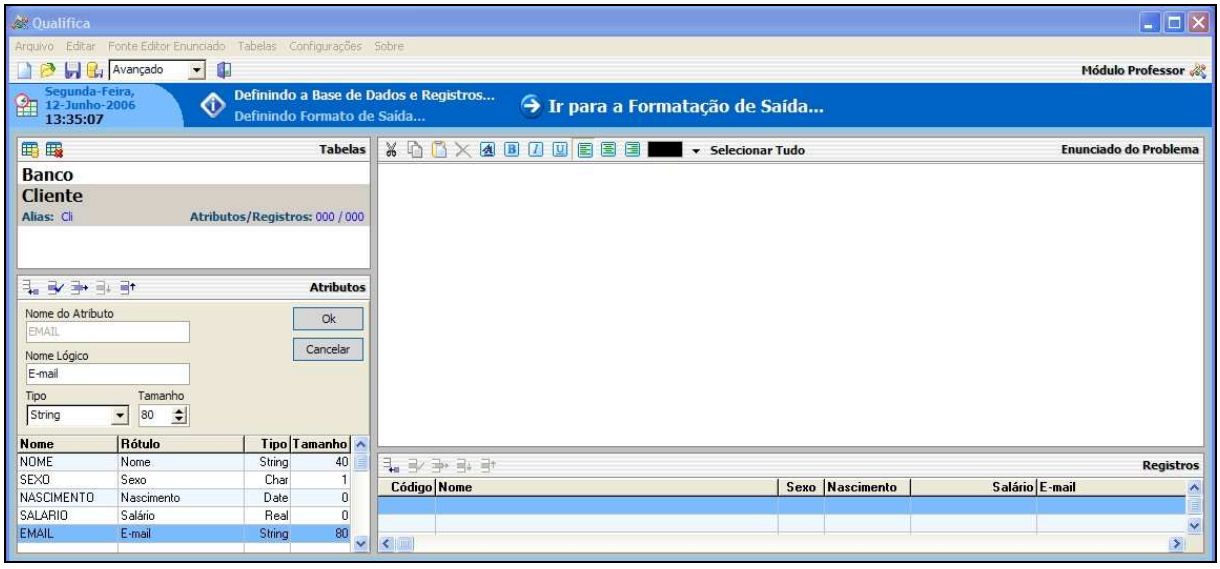


Figura 61 – Inserindo os atributos da tabela Cliente

5.2.3 INSERIR REGISTROS EM CADA TABELA

Uma vez estruturas as tabelas configuradas, é necessário inserir os registros nas mesmas. Para isso deve se escolher a tabela na lista e conforme a Figura 62 e a Figura 63 abaixo ir inserindo os registros, digitando o conteúdo de cada campo conforme o tipo configurado na estrutura da tabela.

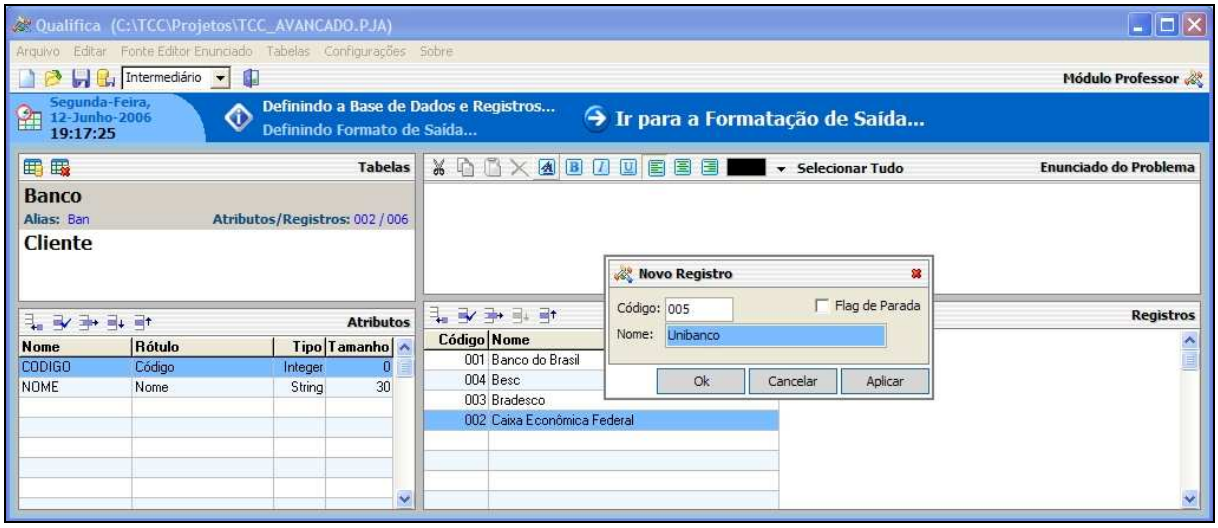


Figura 62 – Inserindo registros na tabela Banco

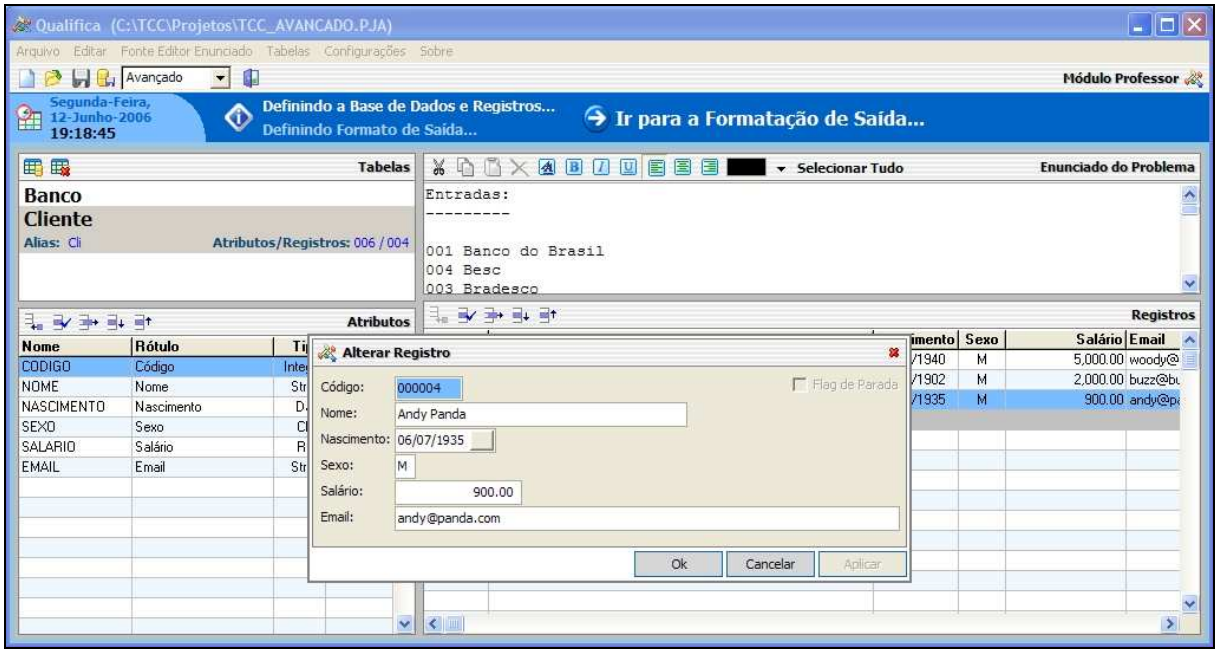


Figura 63 – Inserindo registros na tabela Cliente

Para a criação de um exercício de nível básico, o sistema não irá exigir a inserção de nenhum registro de ‘Flag de Parada’ em cada uma de suas tabelas, ao passo que no exercício

de nível avançado ou intermediário se o professor não inserir um registro de ‘Flag de Parada’ o sistema emitirá uma mensagem de erro (Figura 64) e não permitirá salvar o exercício.

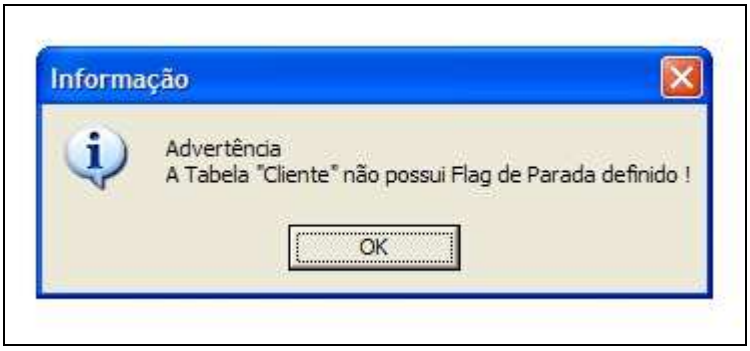


Figura 64 – Mensagem de erro de Flag de Parada indefinido

Para configurar o registro como ‘Flag de Parada’, basta clicar no campo “Flag de Parada” localizado no canto superior direito da tela de edição de registros, quando o exercício é de nível básico, este campo fica invisível. O registro do tipo ‘Flag de Parada’ determina o final do laço de repetição no momento da leitura das entradas. A partir do momento que se inclui um registro do tipo ‘Flag de Parada’ não se pode mais incluir novos registros, apenas alterar e excluir. Para inserir novos registros, deve-se excluir o registro de ‘Flag de Parada’ e inserir os novos registros.

5.2.4 MONTAR O FORMATO DE SAÍDA

Depois de estruturar todas as tabelas do exercício e inserir uma quantidade satisfatória de registros, é necessário então especificar o leiaute do formato de saída.

Para tal, usa-se na montagem do formato de saída os próprios registros que foram inseridos nas tabelas anteriormente e a criação de textos livres.

5.2.5 SELECIONANDO OS REGISTROS DE ENTRADA

Para selecionar os registros de entrada, basta clicar com o *mouse* e o mesmo é inserido na área de formato de saída. O registro de entrada imediatamente inserido fica visível com a sua borda e texto na cor vermelha (Figura 65).

Pode-se clicar e manter o *mouse* pressionado sobre o campo e arrastá-lo sobre a coluna e linha desejados.

Ao posicionar-se sobre o campo do registro de entrada com o *mouse*, o mesmo exibe uma dica revelando em qual linha e coluna o mesmo foi inserido na área do formato de saída.

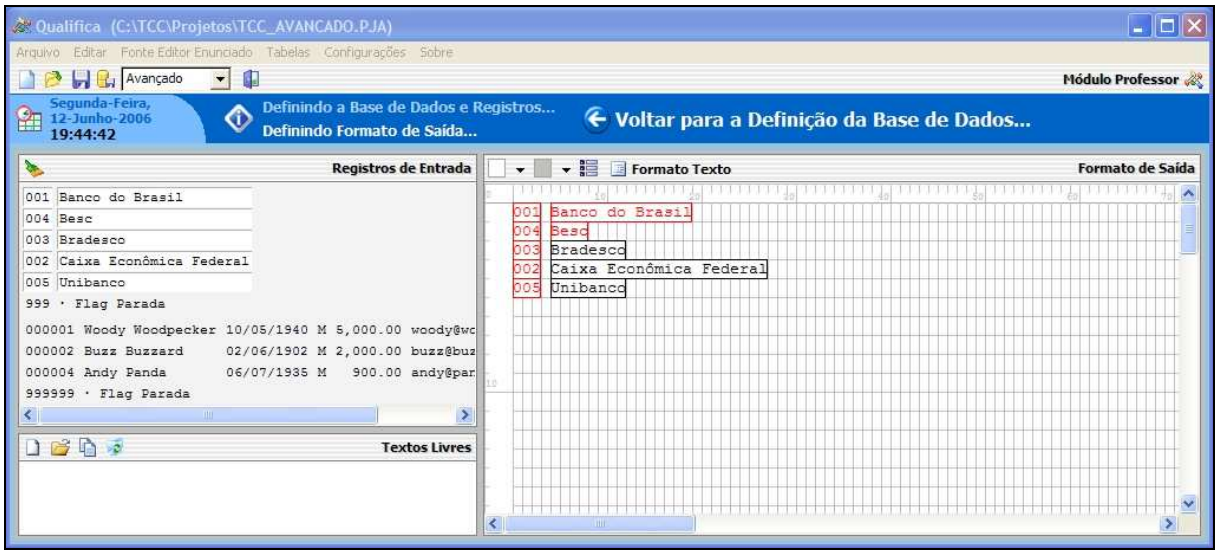


Figura 65 – Selecionando os registros de entrada para o Formato de Saída

5.2.6 INSERINDO TEXTOS LIVRES NO FORMATO DE SAÍDA

A criação de textos livres é opcional no formato de saída, é utilizado para inserir informações que não pertencem as tabelas, como títulos de colunas, cabeçalhos, rodapés, linhas delimitadoras, etc. Para inserir um texto livre, informa-se o texto propriamente dito e o seu tipo, se é um cabeçalho, um rodapé ou um texto qualquer que será impresso entre o cabeçalho e o rodapé (Figura 66).

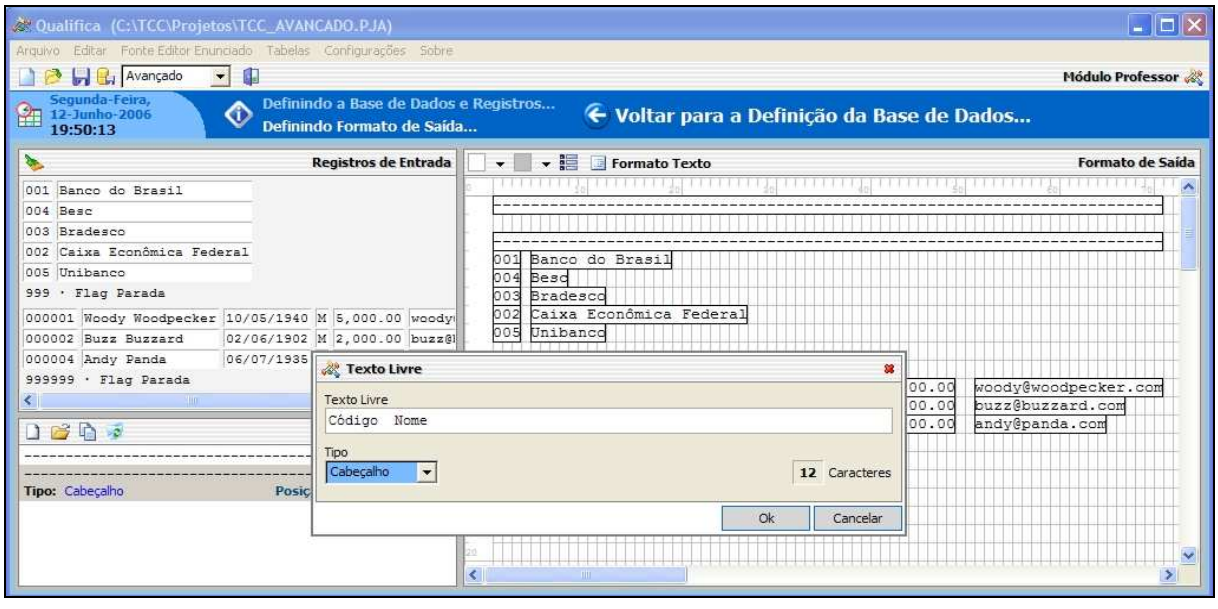


Figura 66 – Inserindo textos livres para o Formato de Saída

Ao escolher o texto livre da lista, o mesmo é exibido detalhadamente informando o tipo, a posição linha, coluna que se encontra no formato e a sua largura em caracteres.

5.2.7 VISUALIZANDO O FORMATO DE SAÍDA EM MODO TEXTO

Figura 67 – Visualizando o formato de saída em modo texto

Pode-se visualizar o formato de saída em modo texto, para ver se o mesmo satisfaz o leiaute desejado.

5.2.8 INSERINDO O FORMATO DE SAÍDA EM MODO TEXTO NO ENUNCIADO DO EXERCÍCIO

Esta facilidade foi incluída para permitir ao professor selecionar o texto e inseri-lo na descrição textual do enunciado do exercício proposta a ser distribuído aos alunos.

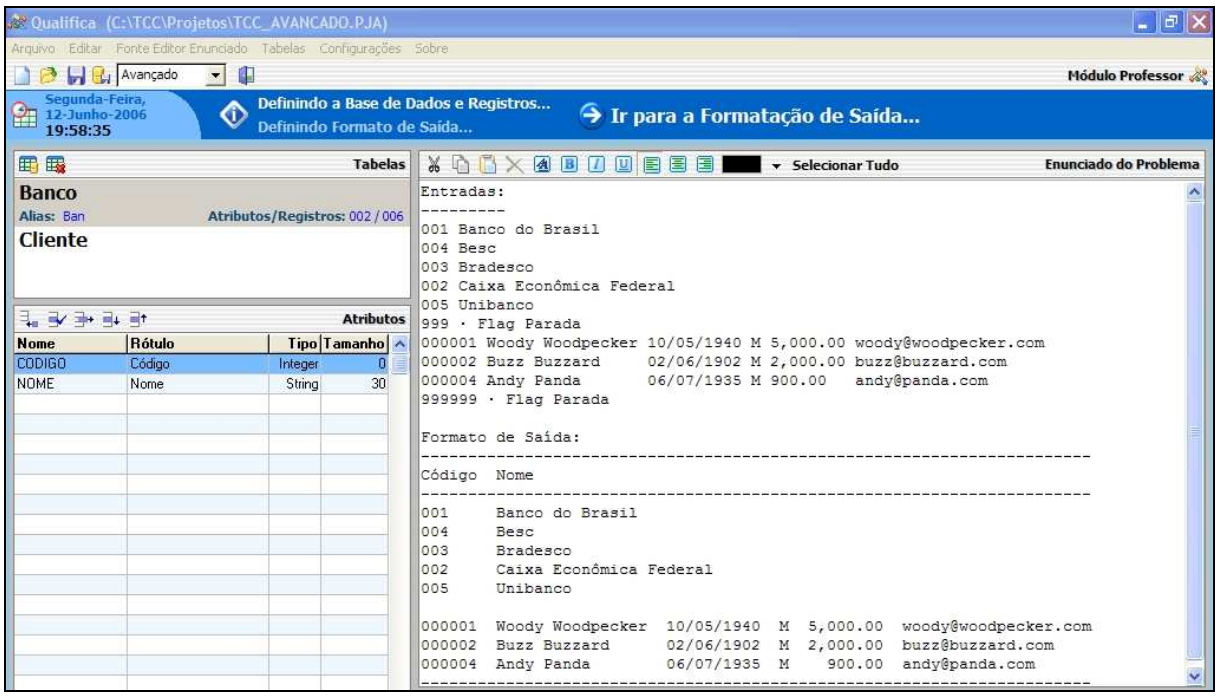


Figura 68 – Inserindo o formato em modo texto no enunciado do exercício

5.3 RESOLVENDO O EXERCÍCIO

Para iniciar a solução de um exercício, deve-se abrir o arquivo com extensão PJA que deve estar localizado dentro da pasta de nome “Projetos”, localizada no mesmo diretório onde está a aplicação.

5.3.1 SELECIONANDO OS REGISTROS DE ENTRADA

Depois de aberto o arquivo o aluno deve iniciar selecionando as entradas, e as mesmas são inseridas na grade de classificação das entradas, como ilustra a Figura 69.

O número de linhas da grade de classificação de entradas é igual a quantidade de registros de entrada que serão selecionados.

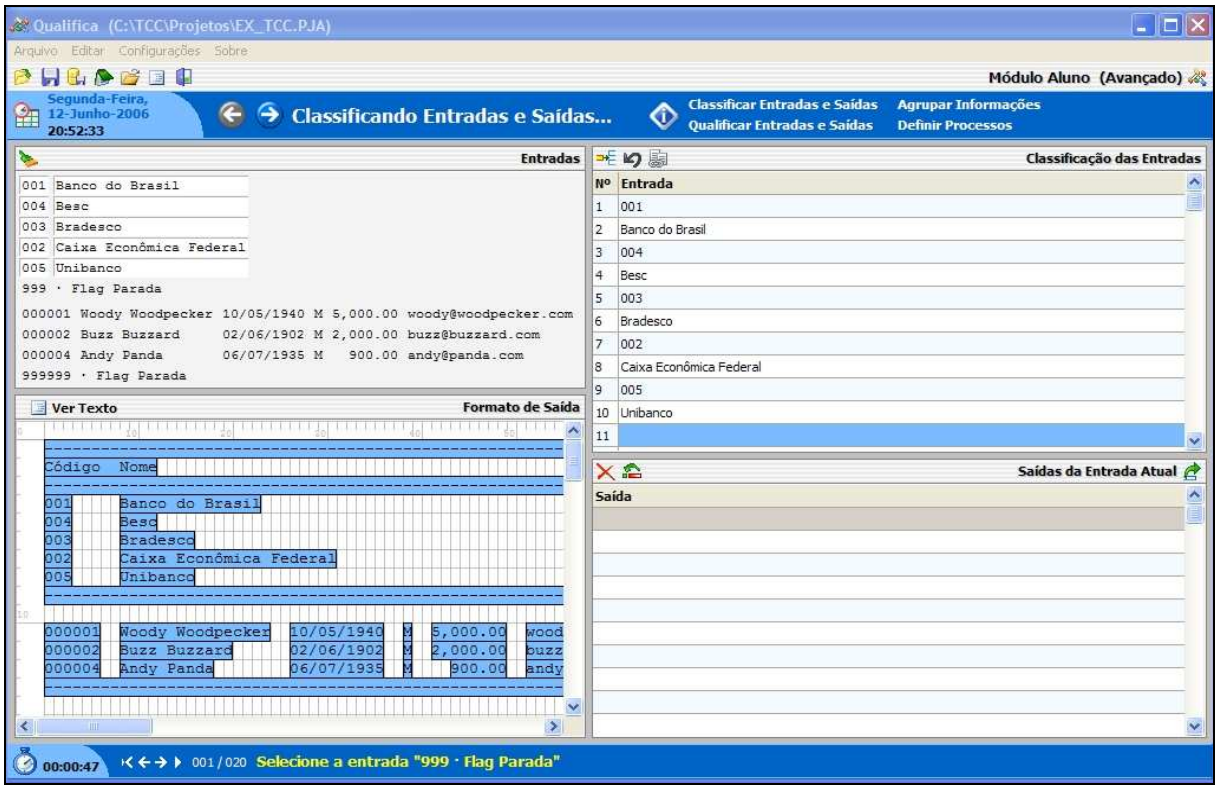


Figura 69 – Selecionando as entradas

5.3.2 SELECIONANDO AS SAÍDAS PARA AS ENTRADAS

Para selecionar as saídas, deve-se escolher a entrada desejada na grade de classificação das entradas e ir clicando com o *mouse* sobre as saídas localizadas dentro da área de formato de saída, como ilustra a Figura 70.

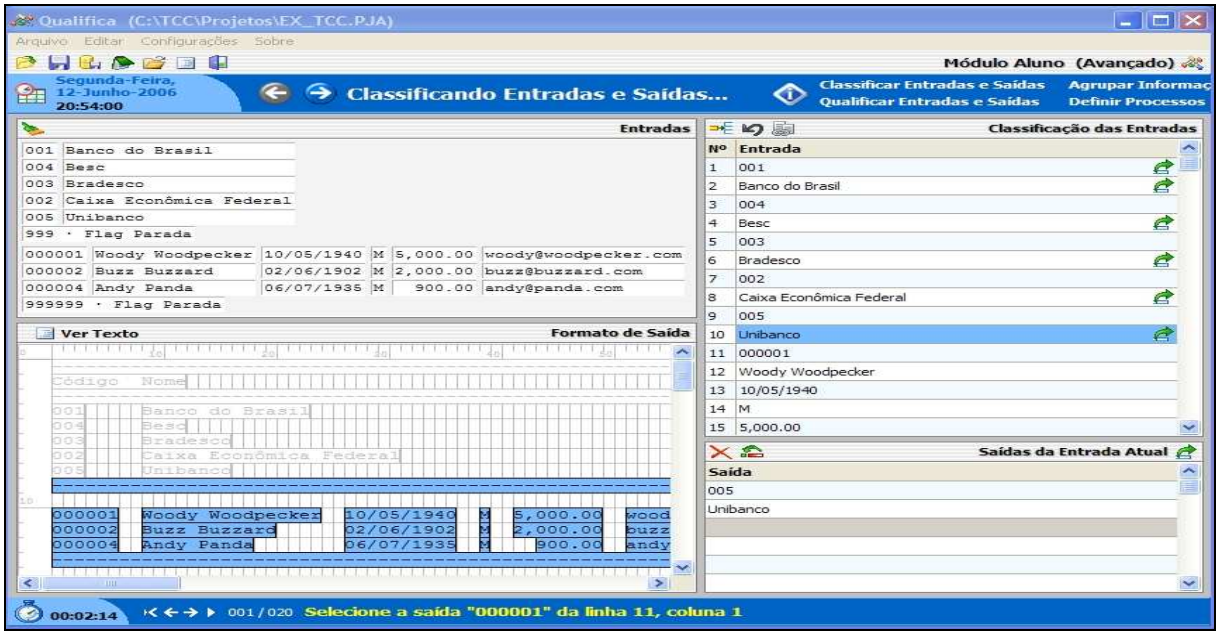


Figura 70 – Selecionando as saídas para as entradas

5.3.3 QUALIFICANDO AS ENTRADAS

Para qualificar as entradas, deve-se clicar sobre a linha e na coluna ‘Nome’ da grade de classificação das entradas e digitar um nome e confirmar com a tecla <ENTER>. O nome digitado é inserido numa lista e que pode ser escolhido para uma outra entrada.

Pode-se configurar o mesmo ‘Nome’ para várias entradas, para isso deve-se manter a tecla <Ctrl> pressionada e com o *mouse* ir clicando sobre as entradas desejadas, no final tecla-se <ENTER>, digita-se o nome desejado finalizando com a tecla <ENTER> e automaticamente o nome é inserido em todas as entradas selecionadas na grade como pode-se visualizar na Figura 71.

A mesma operação de qualificação deve ser feita na grade de saídas localizada logo abaixo da grade de classificação de entradas, lembrando que para textos livres não é necessário informar o nome.

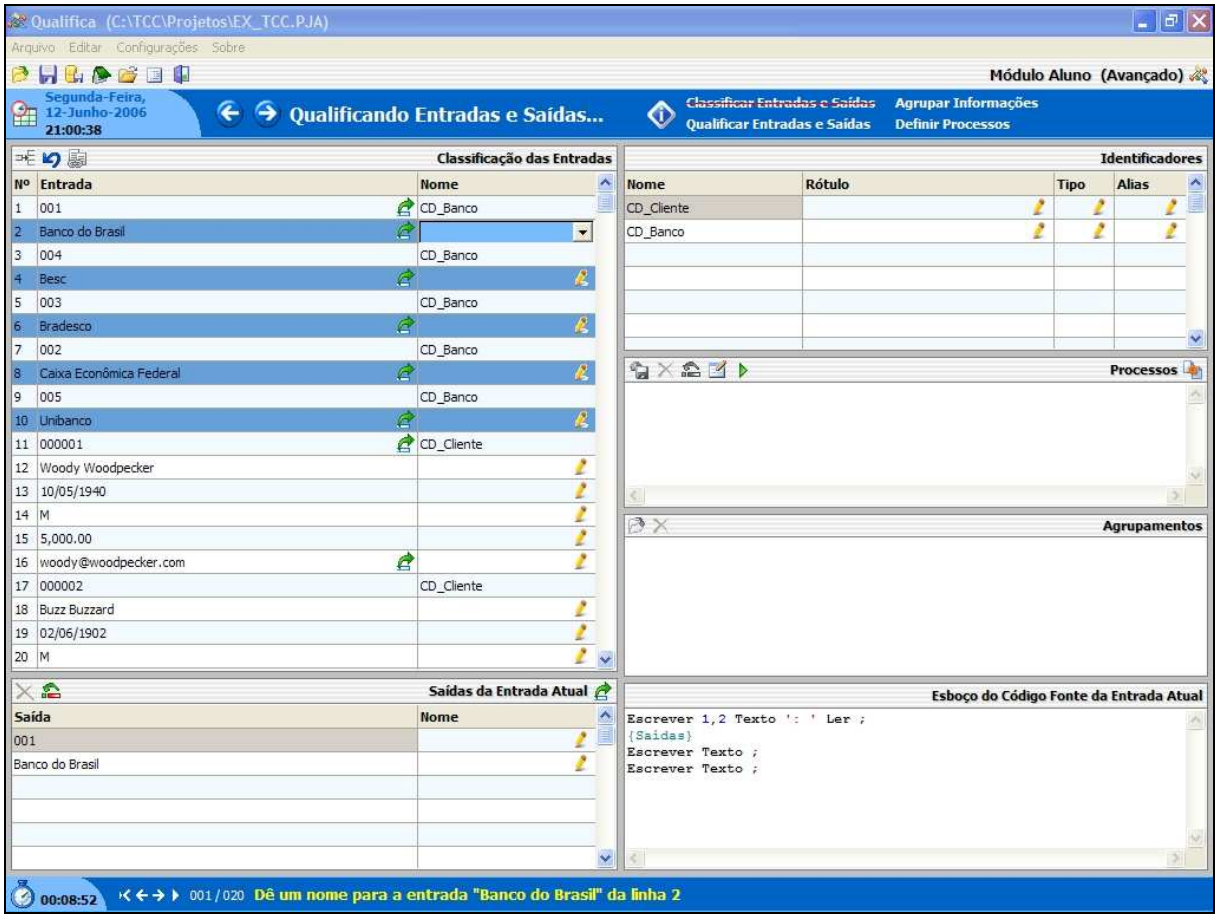


Figura 71 – Qualificando as entradas

5.3.4 QUALIFICANDO OS IDENTIFICADORES

Para qualificar os identificadores a operação é semelhante a qualificação das entradas, clica-se sobre a coluna ‘Rótulo’ e digitar o texto e confirmar com a tecla <ENTER>, depois selecionar um tipo e caso o exercício seja de nível avançado, o *alias* da tabela a qual o identificador está relacionado.

Figura 72 – Qualificando os identificadores

5.3.5 IDENTIFICANDO OS AGRUPAMENTOS NAS ENTRADAS

Para identificar os agrupamentos de entradas de dados comuns, basta marcar um bloco conforme apresentado na Figura 73.

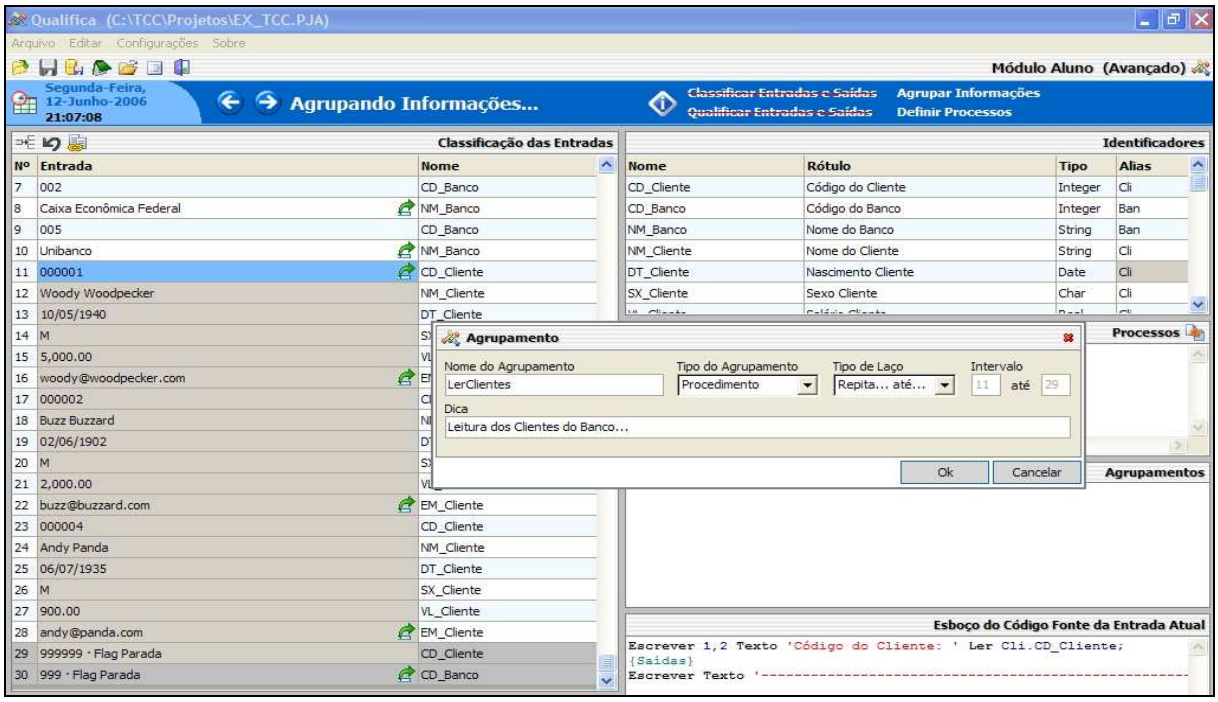


Figura 73 – Identificando e agrupando informações dos clientes

5.3.6 DEFININDO LINHAS DE CÓDIGO DE PROCESSAMENTO

Depois de criados os agrupamentos, deve-se incluir as linhas de código de procedimento para as entradas e saídas. Para isso, basta selecionar (escolher) a entrada ou a saída desejada e digitar as linhas de código na área de processos.

Deve-se observar que, o programa não realiza nenhuma análise sintática ou semântica nas linhas de código que forem inseridas, pois esta não é a finalidade do mesmo.

Digitando-se a macro '<@>', a mesma será substituída no código fonte pela entrada ou saída atual, a macro '<!>' é substituída no código fonte pelo índice atual do *array*, caso o exercício o exercício seja do nível intermediário.

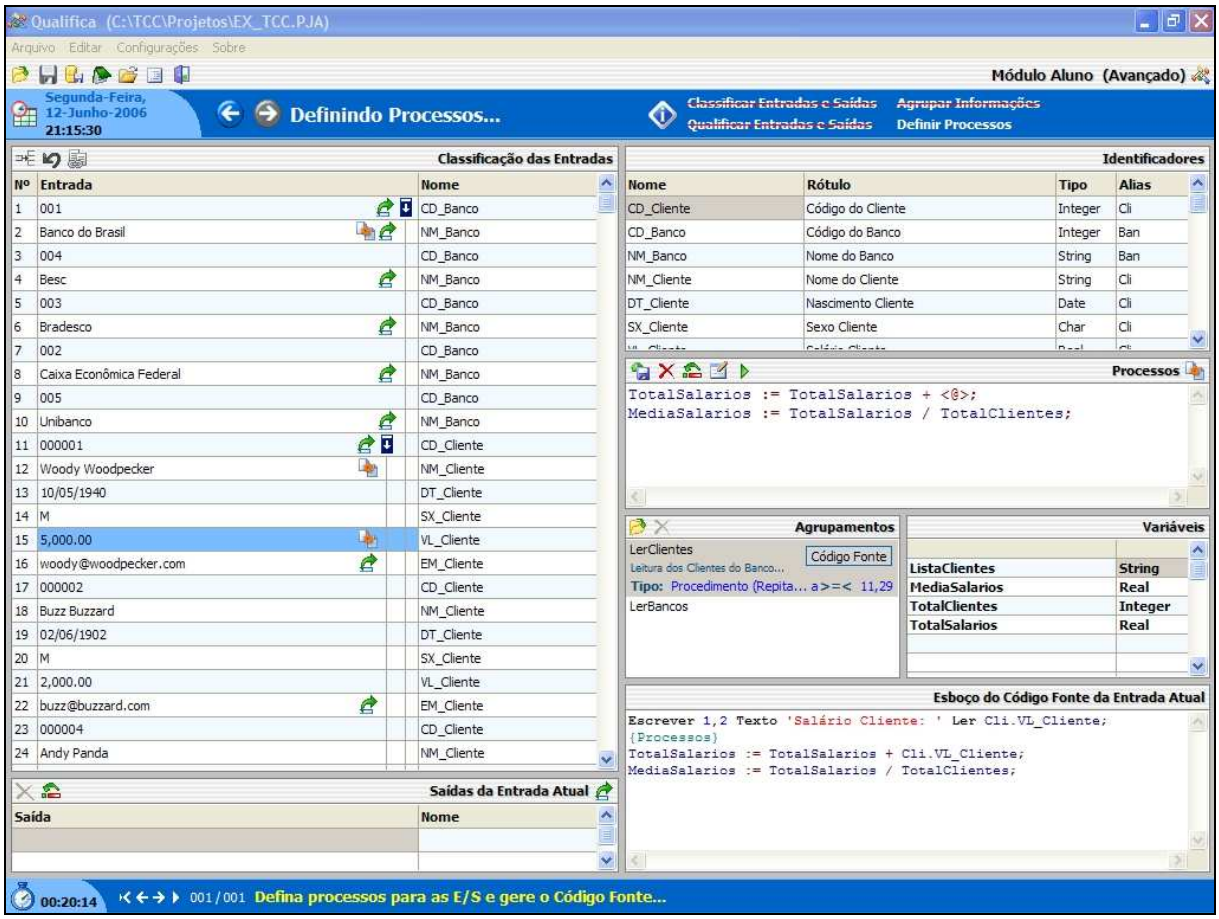


Figura 74 – Definindo processamentos para a entrada atual

5.3.7 QUALIFICANDO VARIÁVEIS

Depois de inserir as linhas de código de procedimentos, o aluno deve qualificar as variáveis. Pode-se notar que as linhas de código inseridas (Figura 74) geraram duas variáveis: TotalSalarios e TotalClientes.

Para configurar o tipo da variável, clica-se na coluna ‘Tipo’ e escolher o tipo desejado da lista (Figura 75). Deve-se escolher a opção ‘(...)’ caso a variável seja uma palavra que configure uma função ou comando.

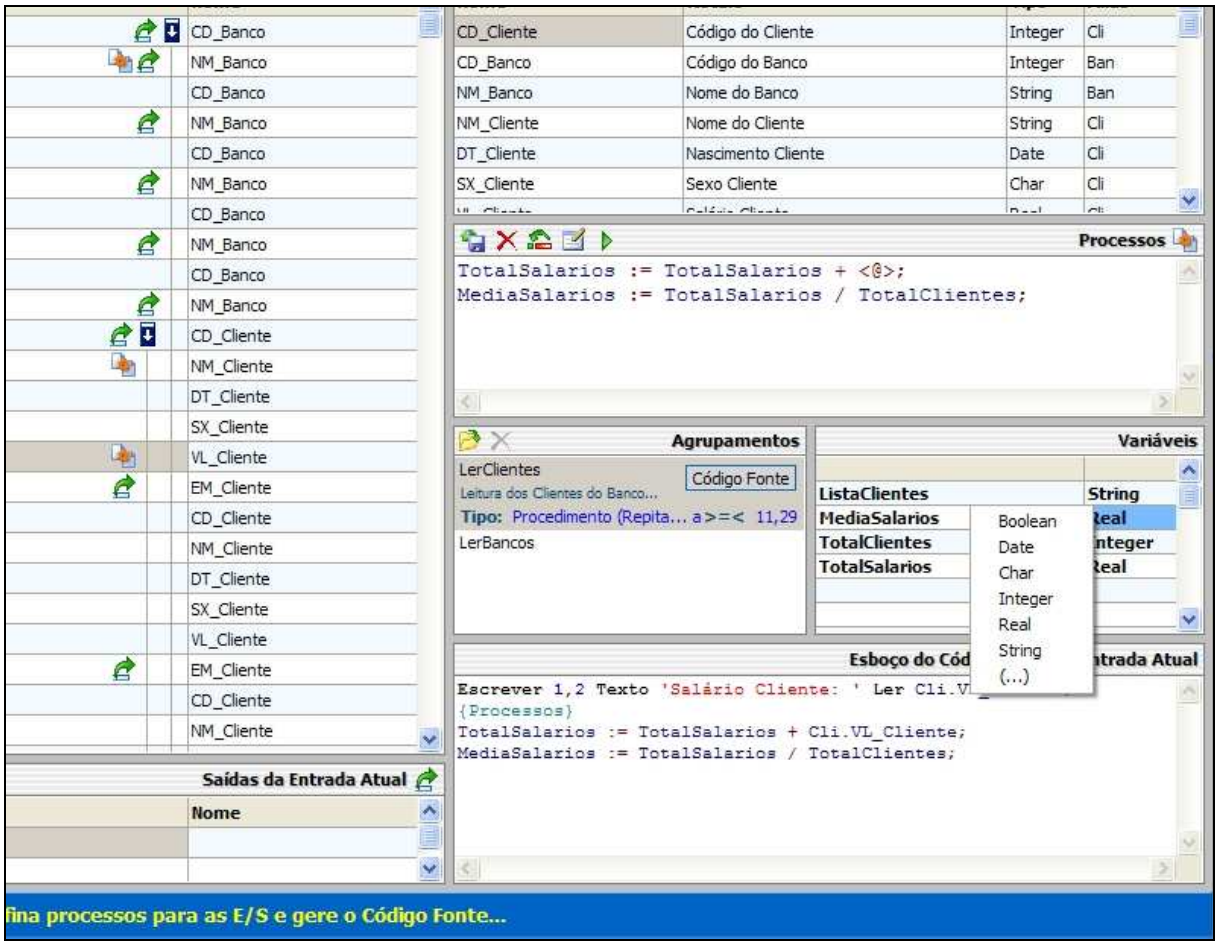


Figura 75 – Qualificando as variáveis geradas pelos processamentos

5.3.8 GERANDO O CÓDIGO FONTE PORTUGOL

As figuras Figura 76, Figura 77 e Figura 78 ilustram o esboço do código fonte Portugol gerado, dividido em três partes, onde é possível visualizar a definição dos tipos dos registros que serão utilizados, a declaração dos procedimentos (agrupamentos), o código do agrupamento para a leitura dos clientes, o código de leitura dos bancos e o código no bloco principal do programa onde pode-se ver uma chamada ao procedimento de leitura dos bancos.

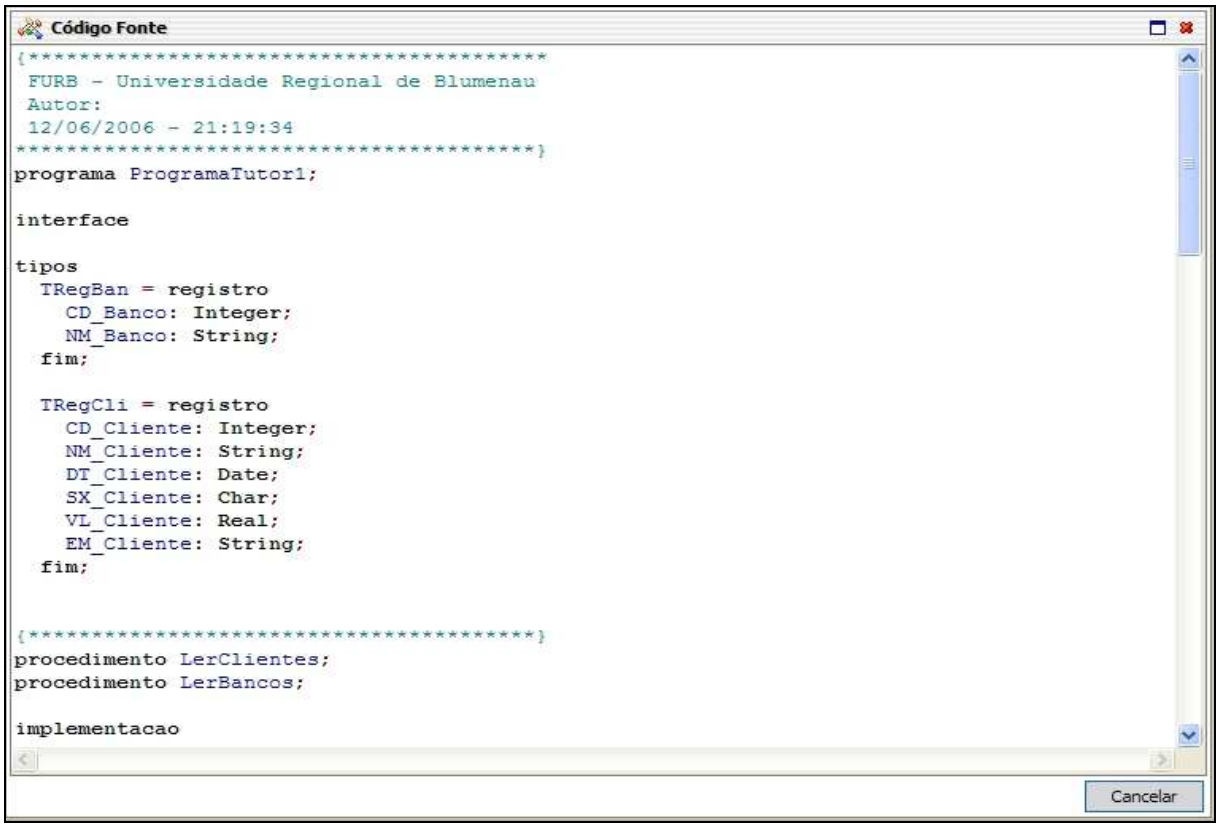


Figura 76 – Visualizando a 1ª parte do código fonte Portugol

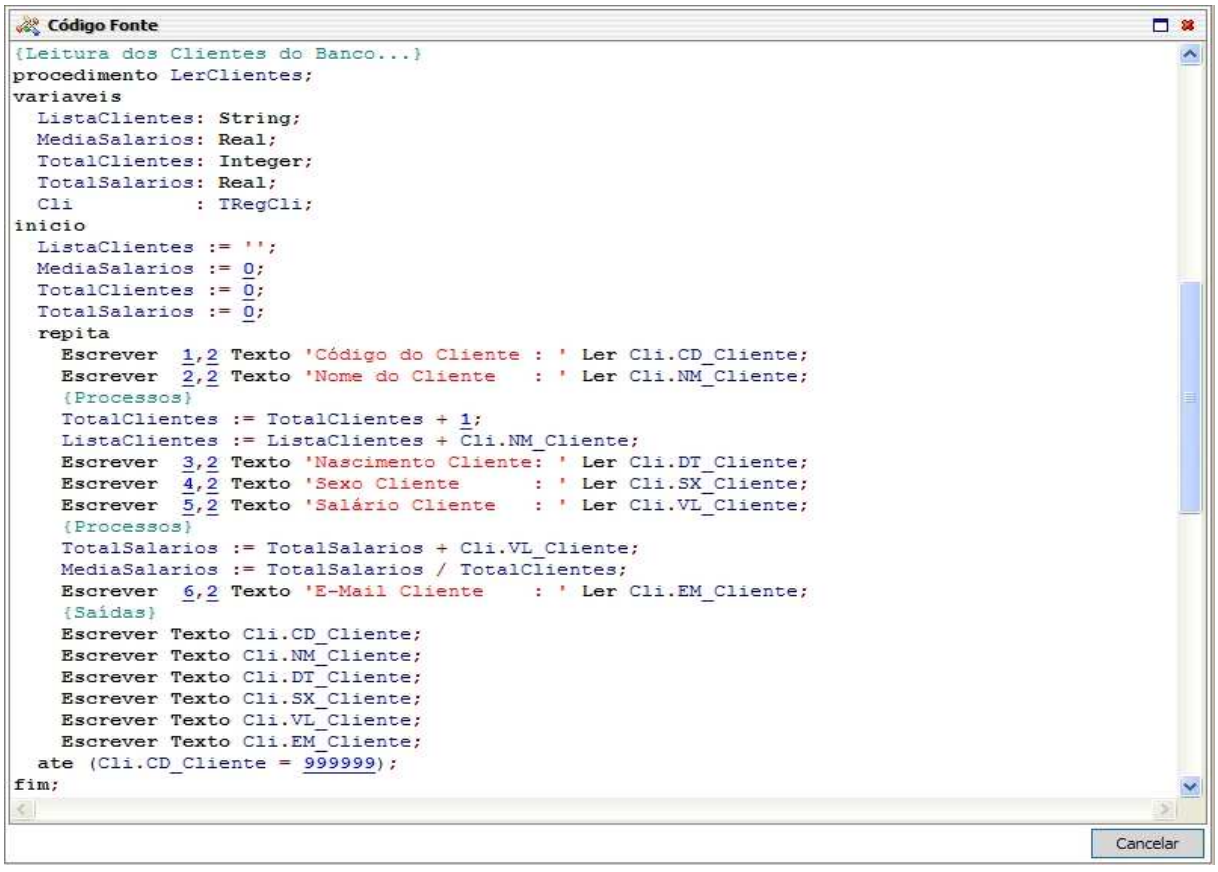


Figura 77 – Visualizando a 2ª parte do código fonte Portugol

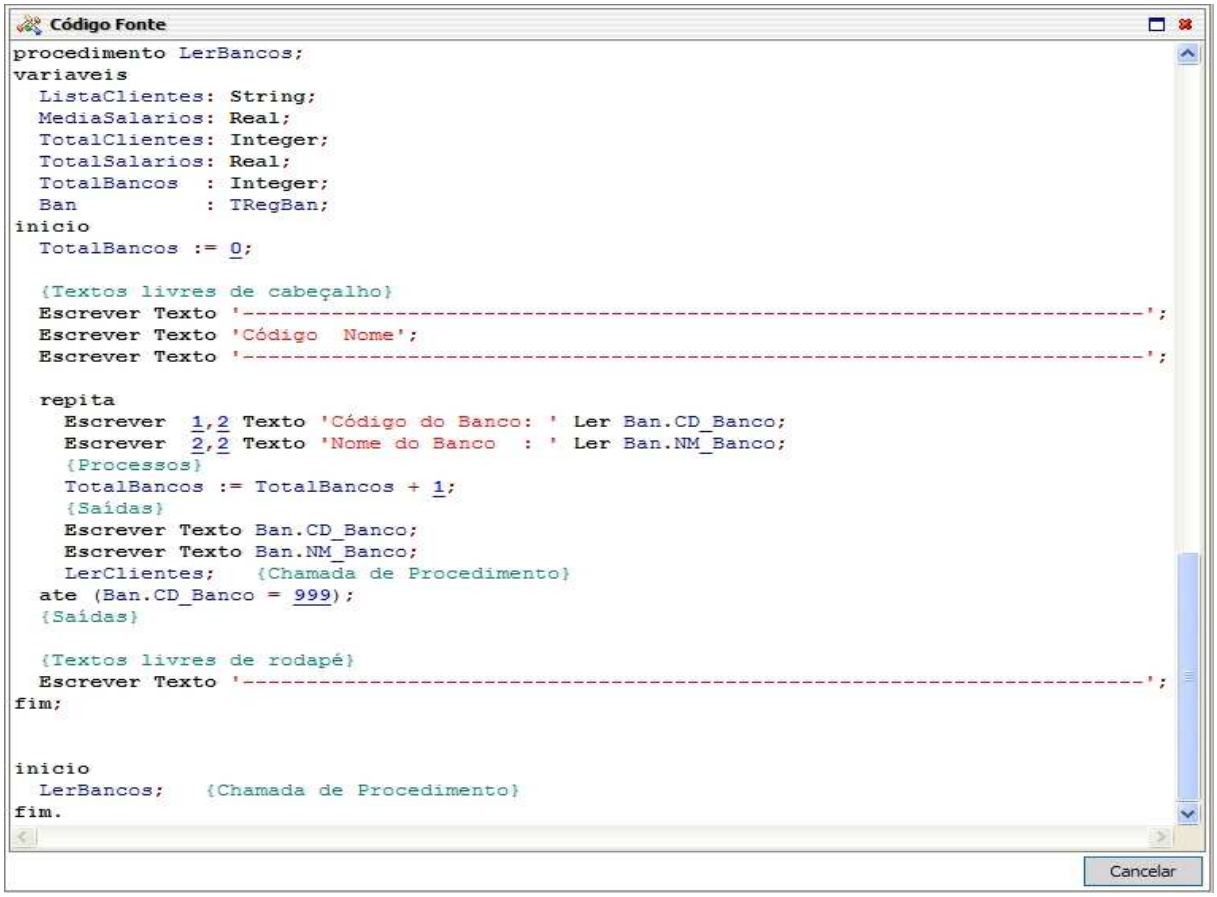


Figura 78 – Visualizando a 3ª parte do código fonte Portugol

5.4 RESULTADOS E DISCUSSÃO

O presente estudo de caso descreveu a forma de criação de exercícios de introdução a programação. Como pôde ser observado, os requisitos para a criação de um exercício vão além da simples digitação textual. Ela implica na especificação de leiaute de dados de entrada, da criação de dados de exemplo e da especificação do tipo de relatório de saída esperado.

Da mesma forma, a solução de um problema por parte do aluno envolve uma análise detalhada dos dados fornecidos como exemplos de entradas/saídas de dados. Com isto, espera-se diminuir a ambigüidade existente na declaração dos enunciados e facilitar a compreensão do problema por parte dos alunos contribuindo desta forma para melhorar a qualidade das especificações com conseqüente melhoria na qualidade do aprendizado.

6 CONCLUSÕES

O presente trabalho apresentou os fundamentos para a utilização de técnicas de desenvolvimento estruturado de programas na construção de uma ferramenta de apoio ao ensino de desenvolvimento de algoritmos. Como referido no texto, o projeto atual está inserido no contexto de um projeto mais amplo de desenvolvimento de uma ferramenta para ensino de algoritmos que vem sendo desenvolvida pelo prof. Mauro Marcelo Mattos da FURB.

Conforme apresentado, a filosofia sobre a qual o sistema foi construído está baseada em dois conceitos principais: (i) uma especificação detalhada do problema por parte do professor e (ii) uma análise detalhada da especificação por parte do aluno. Esta análise detalhada é conduzida pela ferramenta de tal forma que, após verificar todos os elementos da especificação o aluno obtém um pseudocódigo da solução algorítmica do problema a ser desenvolvido.

Deve-se destacar uma vez mais que, o objetivo das pesquisas do Prof. Mauro Mattos na área de Informática na Educação é o de desenvolver uma ferramenta que conduza o aluno no processo de descoberta do conhecimento de como desenvolver algoritmos computacionais e não de obter uma ferramenta que construa automaticamente soluções corretas para tais problemas.

Desse modo, a ferramenta desenvolvida possui inúmeros recursos que viabilizam no módulo professor a especificação de um problema de introdução à programação e, recursos que possibilitam o aluno a analisar e qualificar todas as informações necessárias para a possível solução do problema, inclusive, o detalhamento de procedimentos a serem realizados em laços de repetição. Além disso, o próprio detalhamento já vai conduzindo o aluno no processo de documentação do código gerado, contribuindo para que o conceito de qualidade de software possa ser trabalhado intuitivamente já nas primeiras fases do processo de aprendizagem.

6.1 LIMITAÇÕES

Um dos aspectos que merece atenção é que não houve tempo hábil para a aplicação do sistema em sala de aula para que se obtivesse uma leitura real por parte dos alunos. Isto

porque, esta ferramenta deve ser utilizada nos primeiros meses da disciplina de introdução a programação na perspectiva de treinar o aluno no processo de análise do enunciado.

Certamente a aplicação da mesma no final do semestre letivo não traria grandes avanços no processo de avaliação tendo em vista que os alunos já estariam treinados na forma de ensino/aprendizagem desenvolvida ao longo do semestre. Portanto, espera-se que a ferramenta possa ser aplicada em sala de aula já no início do primeiro semestre de 2006/2 para que seja possível coletar dados acerca da aplicação prática desta versão em sala de aula.

6.2 EXTENSÕES

Como extensões ao projeto sugerem-se:

a) implementação de uma facilidade para armazenamento/recuperação de informações do exercício em formato xml;

b) publicação dos enunciados em formato html;

c) possibilidade de inserir campos calculados dentro do formato de saída;

d) implementação de um interpretador do código fonte Portugol gerado, onde o aluno poderia visualizar passo-a-passo a execução dos comandos de leitura das entrada, processamento e a visualização das saídas;

e) desenvolvimento de um módulo de validação da implementação do aluno com base nos exemplos fornecidos.

REFERÊNCIAS BIBLIOGRÁFICAS

BROWN, Katharine. **Introduction to COBOL Programming - Main Page**. Jacksonville, Florida. 2003. Disponível em: <http://www.unf.edu/~kbrown/cobol/start.bak.html>. Acesso em: 4 abr. 2006.

CANTÙ, Marco. **Dominando o Delphi 5:** a bíblia. São Paulo: Makron Books, 2000.

CARVALHO, Ariadne M. B. R.; CHIOSSI, Thelma C. S. **Introdução à engenharia de software.** São Paulo: Unicamp, 2001.

CASAS, Alberto A. **Contribuições para modelagem de um ambiente inteligente de educação baseado em realidade virtual**. Florianópolis, 1999. Paginação irregular. Disponível em: <http://www.eps.ufsc.br/testes99/casas/>. Acesso em: 01 jun. 2004.

FREITAS, Gilberto. **Protótipo de um sistema de animação de algoritmos**. 2003. 60 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

GUBLER, André I. **Protótipo de um sistema especialista para auxiliar no ensino de algoritmos**. 2002. 55 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

GUERREIRO, Pedro. **A mesma velha questão**: como ensinar programação. Memoria del Quinto Congreso Iberoamericano de Educación Superior en Computación. Ciudad del Mexico: Ed. UNAM, 1986.

HEINZEN, Luiz A. **Módulo de raciocínio baseado em casos em uma ferramenta de apoio ao ensino de lógica de programação**. 2002. 62 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

JACKSON, Michael A. **Princípios em projetos de programas**. Rio de Janeiro: Campus, 1988. 273 p.

KOVACIC, Z. J. A comparison of learning and teaching styles. In: InSITE 2004 - INFORMING SCIENCE AND INFORMATION TECHNOLOGY EDUCATION, 4., 2004, Rockhampton, Australia. **Proceedings…** Santa Rosa, California, 2004. Paginação irregular. Disponível em: <http://proceedings.informingscience.org/InSITE2004/105kovac.pdf>. Acesso em: 1 abr. 2006.

MANTOVANI, Ana M. et al. L.I.S. – Learning in the Space: ambiente de aprendizagem computacional cooperativo. In: RIBIE 2000 - RED IBEROAMERICANA DE INFORMATICA EDUCATIVA, 5., 2000, Viña del Mar, Chile. **Actas...** Viña del Mar, Chile: Universidad de Chile, 2000. Disponível em: <http://www.c5.cl/ieinvestiga/actas/ribie2000/papers/119/index.htm>. Acesso em: 1 abr. 2006.

MARTIN, James; MC'CLURE, Carma. **Uma avaliação das metodologias estruturadas de análise e projeto**. São Paulo: Compucenter Sistemas, 1985. 224 p.

MATTOS, M. M. **Linguagem de programação COBOL.** 1995. Paginação irregular. Notas de aula (Disciplina de COBOL, Curso de Ciências da Computação), Departamento de Informática e Estatística, Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis.

MATTOS, Mauro M.; FERNANDES, Andrino; LÓPEZ, Oscar C. Sistema especialista para apoio ao aprendizado de lógica de programação. In: Congresso Ibero-americano de Educação Superior em Computação, 7., 1999, Florianópolis. **Anais...** Assunção: Universidad Autônoma de Asunción, 1999. p. 1-12.

MATTOS, Mauro M., WANGENHEIM Christiane G. Aplicação de raciocínio baseado em casos na fase de análise de requisitos para construção de abstrações em lógica de programação. In: SEMINCO - Seminário de Computação, 7., 1999, Blumenau. **Anais...** Blumenau: Universidade Regional de Blumenau, 2000. p. 119-129.

MATTOS, Mauro. M. Construção de abstrações em lógica de programação. In: SBC 2000, 10., 2000, Curitiba. **Anais...** Curitiba: Editora Universitária Champagnat, 2000a. p. 60-60.

MATTOS, Mauro M. Ferramenta Case para apoio a construção de abstrações em lógica de programação. In: SIPM'2000 - Simpósio de Informática do Planalto Médio, 2., 2000, Passo Fundo. **Anais...** Passo Fundo, 2000b. Paginação irregular.

MATTOS, Mauro M. Learning how to build abstractions in programming logics classes. In: IE 2002 – CONGRESSO IBEROAMERICANO DE INFORMATICA, 6., 2002, Vigo, Espanha. **Proceedings...** Vigo, Espanha, 2002. Paginação irregular.

MATTOS, Mauro M. **Linguagens para programação de sistemas.** 2005. Paginação irregular. Notas de aula (Disciplina de Linguagens para Programação de Sistemas, Curso de Ciências da Computação). Depto de Sistemas e Computação, Universidade Regional de Blumenau, Blumenau.

REICHELT, Sebastian. *Delphi pages*: TSourceEdit . Disponível em: <http://www.delphipages.com/result.cfm?ID=2108>. Acesso em: 19 jun. 2006.

SALIBA, W. L. C. **Técnicas de programação:** uma abordagem estruturada. São Paulo: 1992. 141 p.

SILVA, Paula C.; FIGUEIREDO, A. D. Bases para o desenvolvimento estruturado de programas educativos. In: CONGRESSO IBERO-AMERICANO DE INFORMÁTICA NA EDUCAÇÃO, 2., 1994, Lisboa, Portugal. **Anais eletrônicos...** Lisboa, Portugal, 1994. Paginação irregular. Disponível em: <http://www.niee.ufrgs.br/ribie98/cong_1994/index.html>. Acesso em: 01 jun. 1996.

SOFT GEMS SOFTWARE SOLUTIONS. **Color picker control**. 2006. Disponível em: <http://www.lischke-online.de/ColorPicker.php>. Acesso em: 19 jun. 2006.

APÊNDICE A – Diagrama de classes detalhado do Módulo Professor

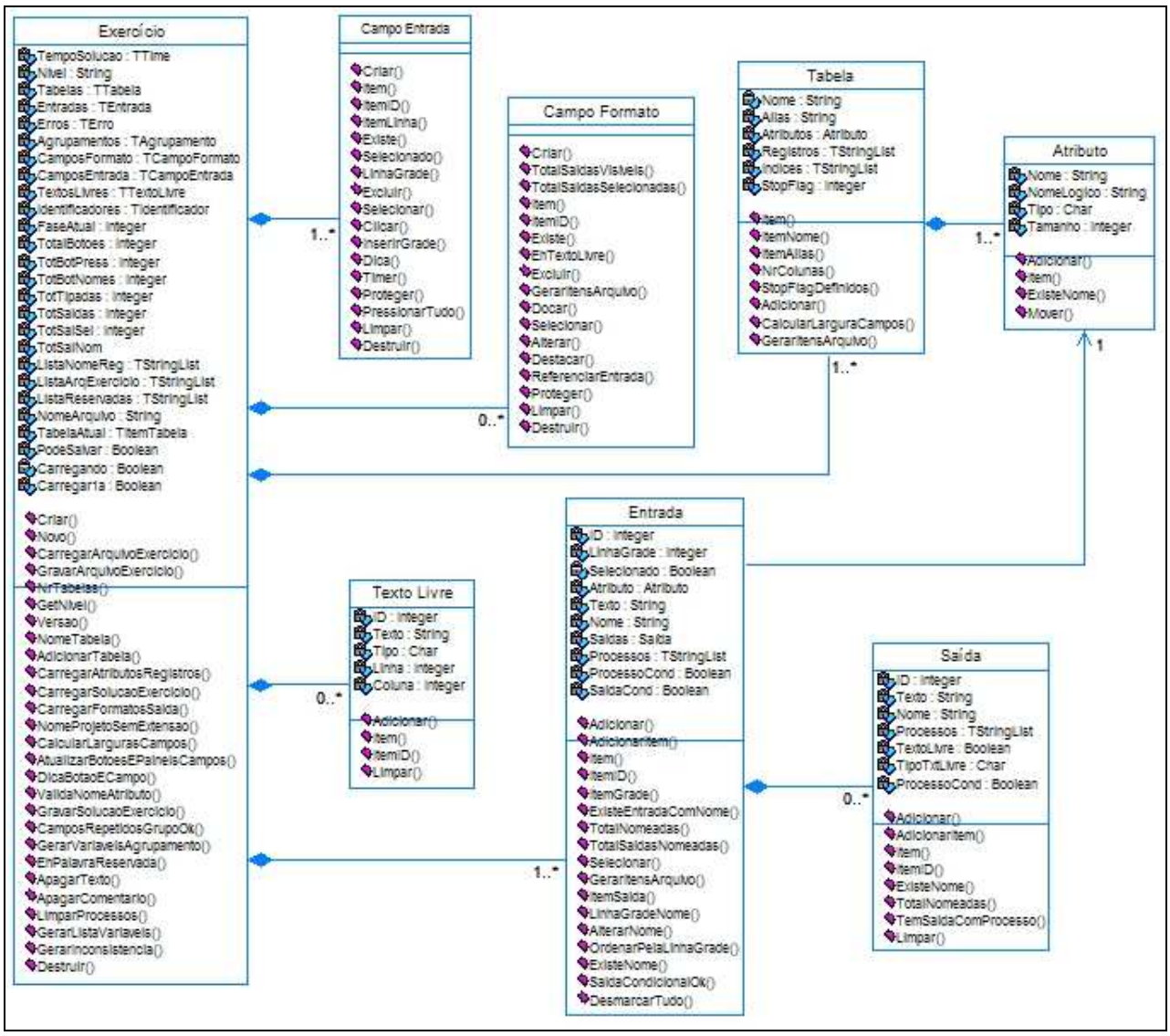


Figura 79 – Diagrama de classes detalhado do Módulo Professor

APÊNDICE B – Diagrama de classes detalhado do Módulo Aluno

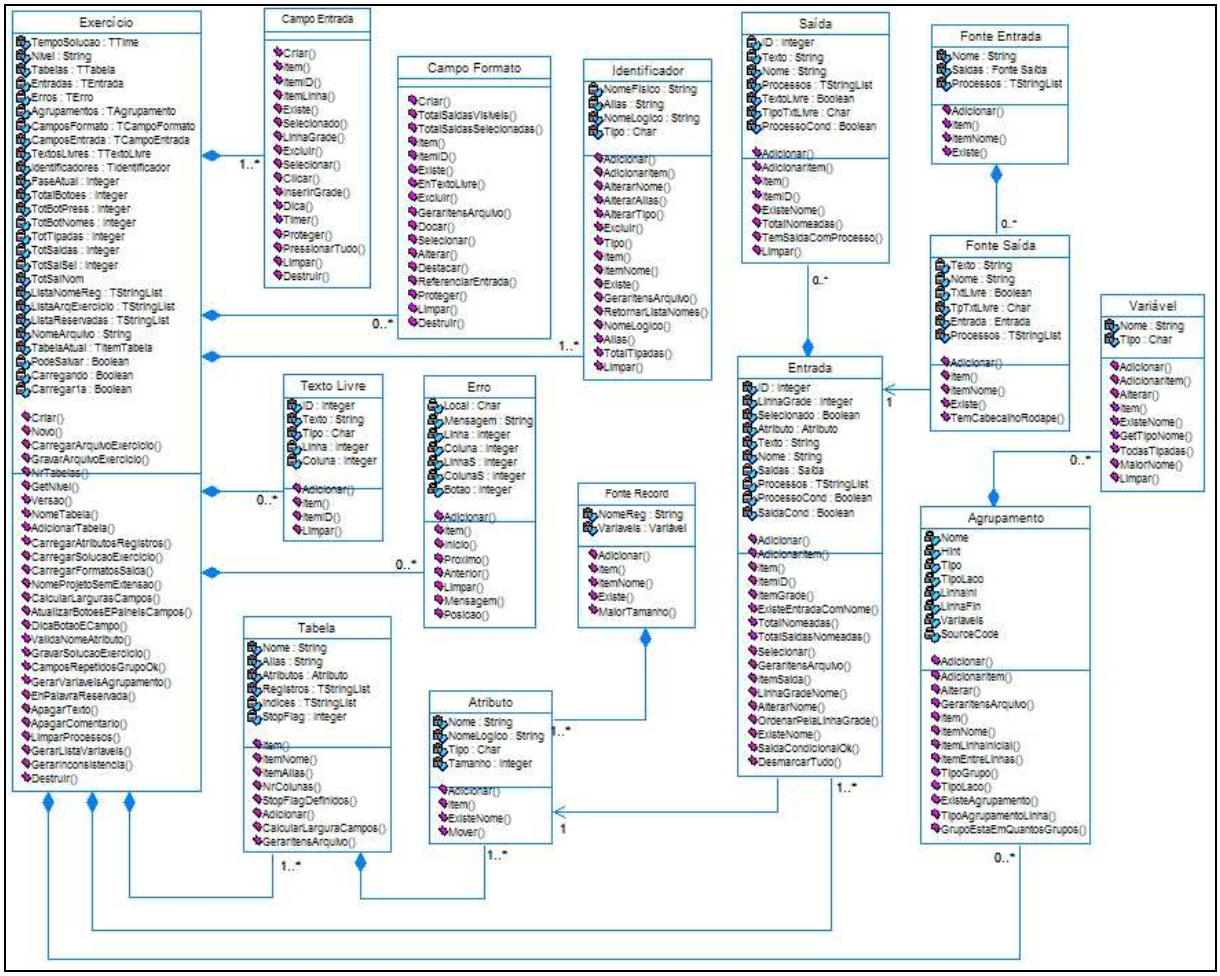


Figura 80 – Diagrama de classes detalhado do Módulo Aluno

APÊNDICE C – Exemplo de telas dos Módulos Internacionalizados

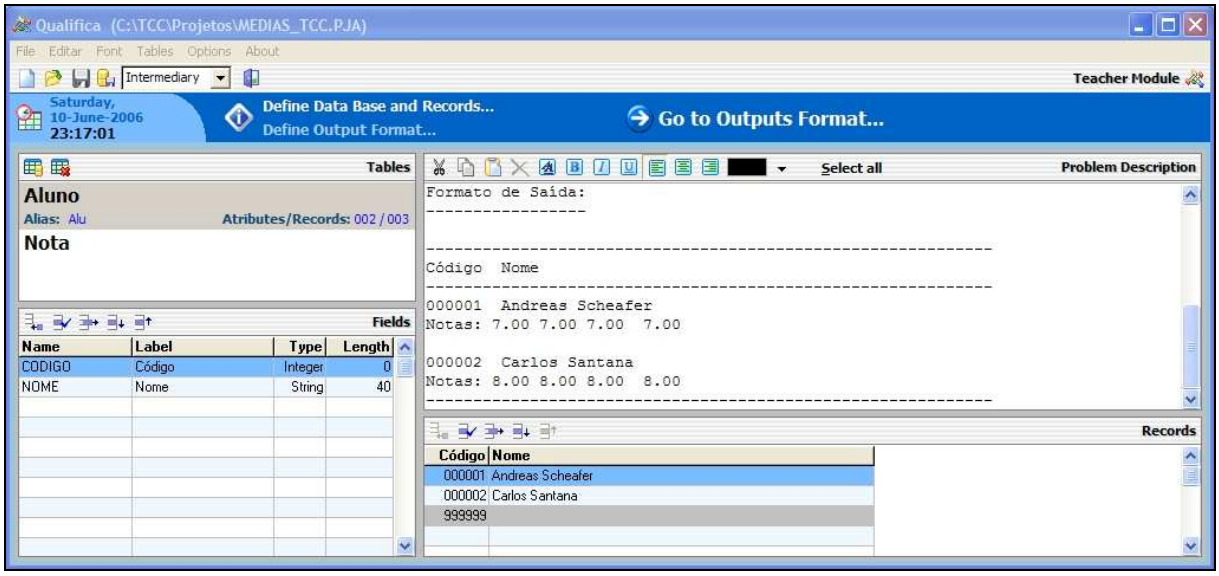


Figura 81 – Tela do Módulo Professor com apresentação em inglês

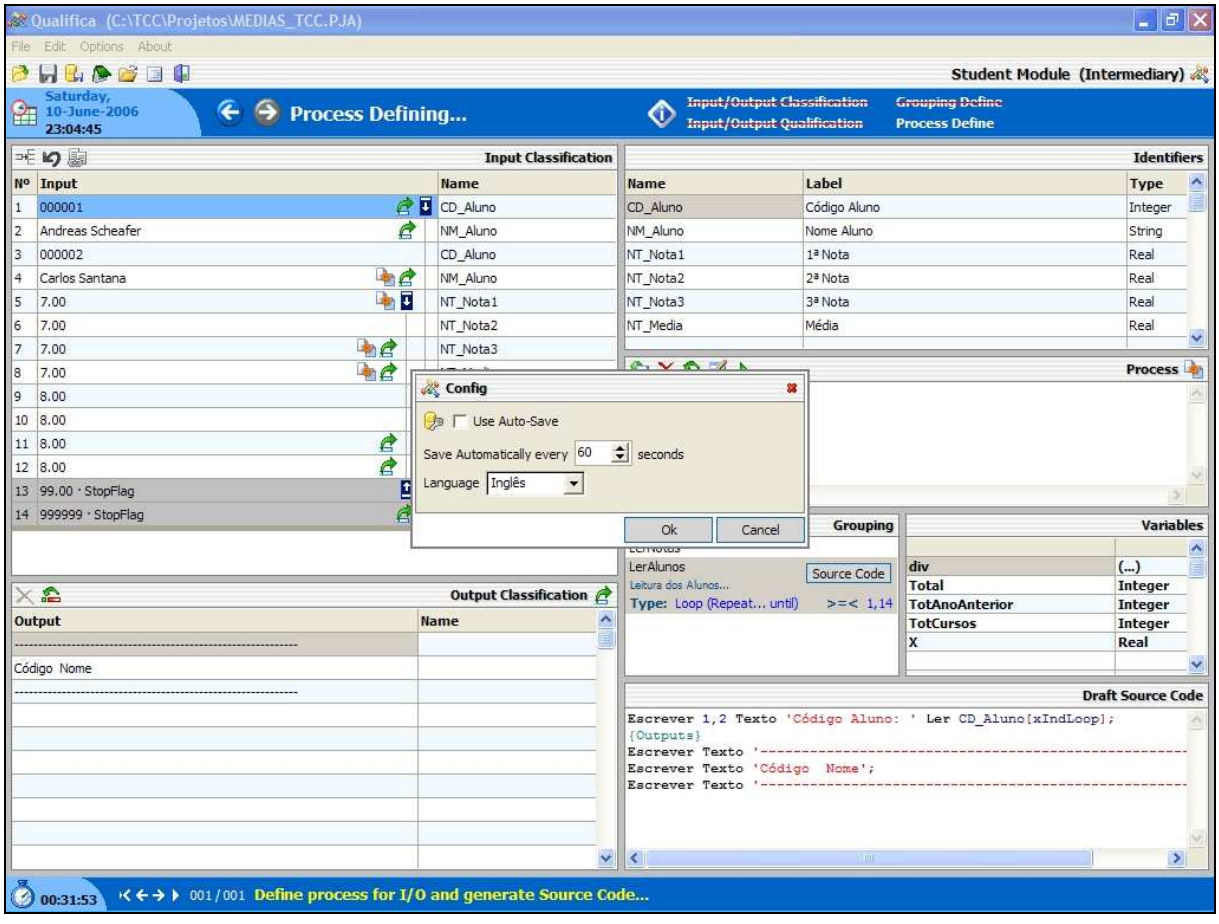


Figura 82 - Tela do Módulo Aluno com apresentação em inglês