UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

FERRAMENTA PARA TESTES DE PROGRAMAS UTILIZANDO COMPONENTES DA BIBLIOTECA CLX

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

DENISE SANTIAGO

BLUMENAU, JUNHO/2002

2002/1-21

FERRAMENTA PARA TESTES DE PROGRAMAS UTILIZANDO COMPONENTES DA BIBLIOTECA CLX.

DENISE SANTIAGO

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

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Maurício Capobianco Lopes — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Maurício Capobianco Lopes

_____

Prof. Everaldo Artur Grahl

_____

Prof. Wilson Pedro Carli

AGRADECIMENTOS

Agradeço principalmente ao meu pai, João Wilson, em quem eu me espelhei para escolher essa profissão. Sem o incentivo dele eu jamais teria concluído esse curso, pois ele sempre me deu oportunidades para realizar os meus objetivos.

Ao meu namorado, Fernando da Silva, que como Bacharel em Ciências da Computação, me auxiliou em vários momentos no desenvolvimento do trabalho e que teve muita paciência e compreensão aceitando os meus momentos de ausência e nervosismo no decorrer do mesmo.

Agradeço ao meu orientador, Professor Maurício Capobianco Lopes, que com muita sabedoria e paciência me ajudou na execução deste trabalho. Sem as suas palavras de incentivo, provavelmente eu teria desistido no caminho.

A todos os meus colegas de faculdade e trabalho que de alguma forma me ajudaram no decorrer deste curso, aos Amigos do Barney que sempre estiveram presentes me ajudando e me incentivando.

Agradeço a toda a minha família, que entendeu a minha ausência durante o desenvolvimento desse trabalho, e que eu amo muito.

Por fim, gostaria de dedicar esse trabalho e a conclusão desse curso à memória de minha mãe, Mirian, que infelizmente não está mais comigo nessa vida, mas que foi uma pessoa maravilhosa e que sempre fez de tudo para que não me faltasse nada, me proporcionando oportunidades que ela não teve.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

RESUMO

Este trabalho tem como objetivo desenvolver uma ferramenta de apoio para testes de programas utilizando o teste de validação, com o intuito de auxiliar os alunos no aprendizado de programação, utilizando os componentes da biblioteca CLX. A ferramenta é composta de três módulos distintos: módulo Servidor, módulo do professor e módulo do aluno, onde o módulo do Servidor é responsável pelas interfaces da tecnologia CORBA, o módulo do professor mantém as informações sobre alunos, exercícios e correções e o módulo do aluno disponibiliza os exercícios para o desenvolvimento bem como faz a transferência de arquivos entre o módulo e o Servidor. Como arquitetura de sistemas distribuídos foi utilizado o padrão CORBA e para implementação do trabalho foi utilizado o ambiente de desenvolvimento Delphi 6.0.

ABSTRACT

This paper aims the implementation of a program validation tool prototype developed by the programming discipline students, using the CLX library components. This prototype is composed by three distincts modules: server module and teacher's module and student´s module, where the server module is responseble by the CORBA technology interfaces; the teacher´s module keeps the information about the students, such as exercises and corrections; the student´s module disposes the exercises to the development as well the transference of the files bettween the server module. CORBA was used as distributed object architecture and the implementation enviornment was in Delphi 6.0.

1 INTRODUÇÃO

O teste de software é uma das áreas de pesquisa da Engenharia de Software que constitui um dos elementos principais para aprimorar a produtividade e ajudar a fornecer evidências da confiabilidade do software. A qualidade dos sistemas é assegurada através dos testes de software. A realização dos testes pode ser planejada e executada através de métodos que os auxiliam a se tornar mais eficazes e eficientes (Ayroso, 1998).

Aproximadamente 50% do tempo e mais de 50% do custo total da área de software são gastos no teste de programas ou sistemas em desenvolvimento. O teste é um processo de aquisição de confiança no fato de que um programa ou sistema faz o que se espera dele, ou seja, é o processo de se experimentar ou avaliar um sistema por meios manuais ou automáticos, de modo a verificar se ele atende às necessidades especificadas ou a identificar as diferenças entre os resultados esperados e reais (Hetzel, 1987).

Dentro dos conceitos da Engenharia de Software, encontram-se vários tipos de teste de software, entre eles o teste de validação, também conhecido como teste de aceitação. O objetivo deste teste é assegurar que as funções do sistema atinjam os objetivos esperados e de forma correta. Um plano de teste esboça as classes de testes a serem realizadas e um procedimento de teste define os casos de teste específicos que serão usados para demonstrar a conformidade com os requisitos (Pressman, 1995).

Deste modo, neste trabalho, foi desenvolvida uma ferramenta para auxiliar o professor nos testes dos exercícios resolvidos pelos alunos nas disciplinas de programação. Para isto foi disponibilizado um banco de dados de exercícios, através do qual o aluno pode selecionar e resolver um exercício, submetendo o seu programa à ferramenta para verificar se o mesmo está correto. A ferramenta disponibiliza ao aluno os resultados dos testes realizados em seu programa. No caso de erro o aluno pode corrigir o programa e imediatamente submeter a nova correção. A comunicação entre o módulo do professor e dos alunos foi feita utilizando a tecnologia CORBA.

A tecnologia CORBA permite que objetos de sistemas distribuídos comuniquem-se entre si de forma transparente, não importando em que plataforma ou sistema operacional eles

estejam rodando, em que linguagem de programação eles foram implementados e até mesmo qual protocolo de comunicação eles utilizam (Bósio, 2000).

Para a implementação da ferramenta proposta neste trabalho foram utilizados os componentes da biblioteca CLX que foi incluída na versão 6.0 do ambiente de desenvolvimento Delphi.

1.1 OBJETIVOS

O objetivo do trabalho proposto é desenvolver uma ferramenta de apoio para testes de programas utilizando o teste de validação, com o intuito de auxiliar os alunos de graduação no aprendizado de programação.

Os objetivos específicos do trabalho são:

a) verificar a funcionalidade da tecnologia CORBA utilizando os componentes da biblioteca CLX, para futuramente permitir a migração do protótipo entre diferentes plataformas de sistema operacional;
b) desenvolver exercícios de Programação e seus conjuntos de casos de testes, cadastrando-os em um Banco de Dados.

1.2 ESTRUTURA

Este trabalho foi estruturado em seis capítulos que estão descritos a seguir.

O primeiro capítulo apresenta a contextualização e justificativa para o desenvolvimento da proposta do trabalho.

O segundo capítulo define alguns conceitos de Engenharia de Software, abordando especificamente os testes de software e mostrando alguns aspectos pedagógicos da utilização destes testes.

O terceiro capítulo fala sobre objetos distribuídos, especificamente da tecnologia CORBA que será utilizada neste trabalho.

O quarto capítulo será sobre as técnicas e ferramentas de desenvolvimento utilizadas para a implementação do protótipo, o Borland Delphi 6.0, suas características e inovações e a biblioteca de componentes CLX.

O quinto capítulo descreve a especificação do protótipo, com toda a definição, casos de uso, diagrama de classes e diagrama de seqüência, mostrando os resultados obtidos com o uso dos componentes CLX com a tecnologia CORBA.

O sexto capítulo apresenta as considerações finais, abrangendo as conclusões do desenvolvimento deste trabalho, as dificuldades encontradas e as sugestões para próximos trabalhos.

2 ENGENHARIA DE SOFTWARE

A engenharia de software pode ser vista de forma objetiva, como o estabelecimento e o uso dos princípios básicos da engenharia, com a finalidade de desenvolver software de maneira sistemática e econômica, resultando em um produto confiável e eficiente. Ela abrange um conjunto de três elementos fundamentais: métodos, ferramentas e procedimentos, que possibilitam o controle do processo de desenvolvimento do software e oferece uma base para construção de software de alta qualidade (Rocha, 2001).

Os métodos da engenharia de software detalham como construir o sistema, envolvendo tarefas como: planejamento e estimativa de projeto, análise de requisitos de software, projeto da estrutura de dados, arquitetura de programa, codificação, testes e manutenção (Pressman, 1995).

As ferramentas proporcionam apoio automatizado ou semi-automatizado a todos os métodos citados anteriormente. Essas ferramentas podem ser integradas de forma que a informação criada por uma ferramenta possa ser usada por outra. Neste caso tem-se um sistema de suporte ao desenvolvimento de software chamado e*ngenharia de software auxiliada por computador* (CASE) (Pressman, 1995).

Os procedimentos da engenharia de software são o elo que mantém juntos os métodos e as ferramentas. Eles definem a seqüência em que os métodos serão aplicados, os produtos que se exige que sejam entregues, os controles que ajudam a assegurar a qualidade e coordenar mudanças e as referências que permitem avaliar o progresso do software (Pressman, 1995).

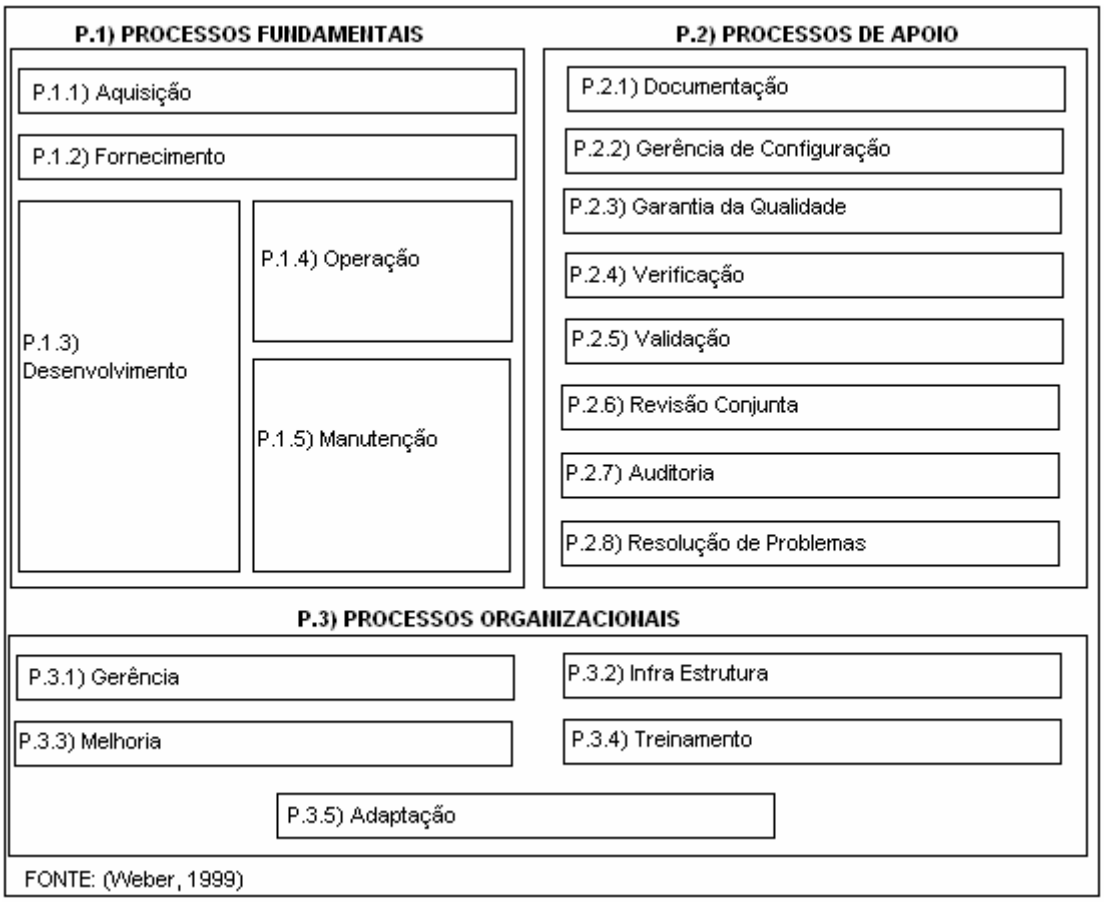
2.1 QUALIDADE DE SOFTWARE

Devido ao impacto causado pelo fenômeno da globalização, mais especificamente esta revolução tecnológica que fez com que a informática apresentasse um novo mundo, quebrando barreiras de comunicação e modificando a própria estrutura da indústria e do consumo, novas atitudes estão sendo tomadas, pelas organizações, para fornecer produtos e serviços com mais qualidade (Inthurn, 2001).

As empresas produtoras e prestadoras de serviços de software estão buscando a qualidade e a produtividade para alcançar o diferencial competitivo. A busca desse diferencial está sendo feita através da norma internacional NBR ISO / IEC 12207 – Tecnologia da Informação – Processos de Ciclo de Vida de Software. Essa norma tem como objetivo auxiliar os envolvidos na produção de software a definir seus papéis (Rocha, 2001).

A ISO / IEC 12207 agrupa as atividades que devem ser realizadas durante o ciclo de vida do software em cinco processos fundamentais, oito processos de apoio e cinco processos organizacionais. A Figura 1 mostra todas as atividades de cada um desses processos.

Figura 1 – Processos do ciclo de vida do software



Os processos fundamentais atendem ao início da vida do software. Fazem parte desses processos a contratação feita pelo cliente, a execução do desenvolvimento, a operação e a manutenção do software durante o seu ciclo de vida. Os processos de apoio contribuem para o sucesso e a qualidade do projeto de software enquanto que os processos organizacionais estabelecem e implementam uma estrutura constituída pelos processos de ciclo de vida e pelo

pessoal envolvido no desenvolvimento de software. Os ensinamentos dos processos organizacionais contribuem para a melhoria da organização (Rocha, 2001).

Segundo Pressman (1995), no desenvolvimento de um software, independente da área de aplicação, tamanho ou complexidade do problema, as três principais fases estão dentro dos processos fundamentais:

a) aquisição: nesta fase busca-se identificar quais informações devem ser processadas, qual função e desempenho são desejados, quais interfaces devem ser estabelecidas, quais as restrições e quais critérios de validação são exigidos para se definir um sistema bem sucedido;
b) desenvolvimento: nesta fase deve-se definir como a estrutura de dados e a arquitetura de software têm de ser projetadas, como os detalhes procedimentais têm de ser implementados, como o projeto será traduzido para uma linguagem de programação e como os testes têm de ser realizados;
c) manutenção: esta fase está ligada à correção de erros, adaptações exigidas à medida que o ambiente do software evolui e ampliações produzidas por exigências diversas do cliente.

Durante a fase de desenvolvimento, o desenvolvedor deve realizar os testes de qualidade de software verificando e validando os requisitos do sistema. Devem ser realizados testes de cobertura, atendimento aos resultados esperados e viabilidade de integração e operação (Rocha, 2001).

É exatamente na fase de desenvolvimento do software, mais especificamente a etapa de teste do software, que se encontra o objetivo principal deste trabalho. Dessa forma, a seguir, serão apresentados alguns conceitos gerais de teste de software.

2.2 TESTES DE SOFTWARE

Existem várias atividades que podem ser utilizadas no processo de desenvolvimento do software para garantir a qualidade do produto final, porém, apesar de todos os métodos, técnicas e ferramentas utilizadas, falhas no produto podem ocorrer. Dessa forma, dentro do processo de desenvolvimento, a etapa de teste é de grande importância para a identificação e eliminação de erros (Inthurn, 2001).

Os testes de software devem ser cuidadosamente desenhados e planejados, para garantir a eficácia dos mesmos. Durante e após a realização dos testes, deve-se inspecionar os resultados, comparando os resultados previstos com os obtidos, para garantir um nível adequado de confiança de que o software atinge os seus requisitos, contribuindo para a qualidade do produto (Paula Filho, 2001).

São quatro as etapas envolvidas pela atividade de teste: planejamento, projeto de casos de teste, execução e avaliação dos resultados. A definição de diferentes níveis de teste em relação às diversas fases do processo de desenvolvimento, enfocando classes distintas de erros, é uma das maneiras de se prevenir ou detectar erros cometidos no desenvolvimento do software (Rocha, 2001).

A atividade de teste é dividida, basicamente, em três fases: de unidade, de integração e de aceitação. Paula Filho (2001) descreve brevemente cada uma das fases da atividade de teste:

a) teste de unidade: tem por objetivo verificar um elemento que possa ser logicamente tratado como uma unidade de implementação, como uma sub-rotina, um módulo ou no caso da tecnologia orientada a objetos, uma classe;
b) teste de integração: tem por objetivo verificar as interfaces entre as partes de uma arquitetura de produto, ou seja, verifica se as unidades testadas de forma individual executam corretamente quando integradas;
c) teste de aceitação: também conhecido como teste de validação, tem por objetivo verificar se o produto atende os requisitos especificados.

Essas fases de teste de software podem ser classificadas utilizando os dois tipos de teste de software existentes que são denominados de método da caixa branca e método da caixa preta.

Paula Filho (2001) apresenta as definições dos tipos de teste de software:

a) método da caixa branca: também conhecido por teste estrutural, tem por objetivo determinar defeitos na estrutura interna do produto, através do desenho de testes que exercitem suficientemente os possíveis caminhos de execução;
b) método da caixa preta: também conhecido por teste funcional, tem por objetivo determinar se os requisitos foram total ou parcialmente satisfeitos pelo produto.

Esses testes não verificam como ocorre o processo, mas apenas os resultados obtidos.

Dessa forma os testes de unidade geralmente são de caixa branca, os testes de aceitação geralmente são de caixa preta e os testes de integração costumam misturar testes de caixa preta e caixa branca (Paula Filho, 2001).

A ferramenta proposta neste trabalho utiliza o teste de aceitação para verificar se os programas desenvolvidos pelo aluno estão corretos. A seguir serão apresentados alguns conceitos sobre o teste de aceitação e os tipos de testes existentes.

2.2.1 TESTE DE ACEITAÇÃO

O objetivo dos testes de aceitação ou validação é demonstrar que o sistema está pronto para ser colocado em operação. Estes testes são normalmente executados pelo usuário final do sistema a fim de verificar se a confiança atribuída ao sistema é justificada (Hetzel, 1987).

O teste de validação do software pode ser definido de várias maneiras, mas uma definição simples é que ele é bem sucedido quando o software funciona de uma maneira razoavelmente esperada pelo cliente (Pressman, 1995).

Para a realização dos testes, deve-se definir os casos de testes que serão submetidos ao software e os resultados esperados do software para as respectivas entradas. Isto é uma característica dos testes de caixa preta, que demonstram a conformidade do sistema com os requisitos, para que a validação do software seja realizada.

Conforme Paula Filho (2001) pode-se dividir o teste de aceitação em teste funcional e teste não funcional. Os testes funcionais têm como objetivo verificar se o produto implementado e os respectivos requisitos funcionais estão consistentes. Os testes não funcionais verificam se o sistema está consistente quanto aos aspectos não funcionais como desempenho, dados persistentes, confiabilidade, usabilidade, etc.

A seguir serão mostrados alguns métodos funcionais de teste com uma breve descrição de seu funcionamento, conforme Rocha (2001):

a) particionamento de equivalência: divide o domínio de entrada em categoria de dados, sendo que cada categoria revela uma classe de erro;

b) análise do valor limite: complementa o método anterior, pois exige casos de teste nos limites de cada classe de equivalência;
c) grafo de causa-efeito: verifica o efeito combinado de dados de entrada. As causas e efeitos são combinados em um grafo para montar uma tabela de decisão e a partir desta são derivados os casos de teste e as saídas.

2.2.1.1 PARTIÇÃO DE EQUIVALÊNCIA

O particionamento de equivalência é um método de teste de caixa preta, baseado em casos de teste. Um caso de teste ideal descobre sozinho uma classe de erros que poderia exigir muitos outros casos para que o erro geral fosse descoberto. Esse método procura definir um caso de teste que descubra classes de erros, reduzindo o número total de casos de teste que devem ser desenvolvidos (Pressman, 1995).

Para cada entrada do sistema, devem ser identificados os conjuntos de valores válidos e inválidos que definem as classes de equivalência para essa entrada. Uma entrada do sistema pode ser um valor numérico, um intervalo de valores, um conjunto de valores relacionados ou uma condição booleana.

Assim, Paula Filho (2001) define as classes de equivalência conforme a Tabela 1.

Tabela 1 – Desenho de classes de equivalência para teste funcional

Definição da entrada	Classes de equivalência
Intervalo de valores	Uma válida, para os valores pertencentes ao intervalo; duas inválidas, para os valores menores e maiores que o limite inferior e superior, respectivamente.
Lista de valores válidos	Uma válida, para os valores incluídos na lista; uma inválida, para todos os outros valores.
Valor numérico	Uma válida, que inclui o valor; duas inválidas, para os valores maiores e menores.
Lógica	Uma válida e uma inválida.

Aplicando-se as diretrizes acima, para a derivação de classes de equivalência, os casos de teste para cada item de dados do domínio de entrada podem ser desenvolvidos e executados.

2.2.1.2 ANÁLISE DO VALOR LIMITE

Os erros são mais freqüentes nos limites do domínio de entrada do que nas regiões centrais. Dessa forma a técnica de análise do valor limite tem como objetivo por à prova os valores fronteiriços, selecionando casos de teste que exercitam os limites (Paula Filho, 2001).

A análise do valor limite também não se concentra apenas nas condições de entrada, pois deve derivar os casos de teste também do domínio de saída (Pressman, 1995).

Segundo Paula Filho (2001), a seleção de casos de teste deve ser feita de acordo com as recomendações feitas na Tabela 2.

Tabela 2 – Desenho de casos de testes para análise de valor limite

Entradas válidas	**Casos de teste**
Intervalo delimitado pelos valores *a* e *b*	Valores *a* e *b*, logo acima e logo abaixo de *a* e *b*, respectivamente.
Série de valores	Valor imediatamente abaixo do mínimo, o mínimo, o máximo e imediatamente acima do máximo.
Estrutura de dados	Casos que exercite a estrutura em suas fronteiras.

No caso do exemplo demonstrado no item anterior, poderia se delimitar os valores limites da divisão como sendo o mínimo e o máximo possibilitado pelo tipo de dado atribuído às variáveis. Assim, o teste deveria ser feito com os valores limites do tipo, seus predecessores e seus sucessores.

2.2.1.3 GRAFO DE CAUSA-EFEITO

Essa é uma técnica que oferece uma representação resumida das condições lógicas e das ações correspondentes. Conforme Pressman (1995), a técnica segue quatro passos relacionados a seguir.

a) causas (condições de entrada) e efeitos (ações) são relacionados para um módulo e um identificador é atribuído a cada um;
b) um grafo de causa-efeito é desenvolvido;
c) o grafo é convertido numa tabela de decisão;
d) as regras da tabela de decisão são convertidas em casos de teste.

A técnica de grafo de causa-efeito é ideal quando se tem um número demasiadamente grande de casos, pois a tentativa de traduzir uma política ou procedimento especificado em linguagem natural para um algoritmo baseado em computador leva à frustração e ao erro.

2.3 TESTES DE SOFTWARE E APRENDIZAGEM

A potencialidade da informática e de seus recursos tornou-se fundamental para os processos de ensino. O uso do computador permite que os alunos realizem um processo interativo de aprendizado, uma vez que eles podem usá-lo no seu próprio ritmo e em qualquer momento (Sanderink, 2002).

Em um método tradicional de ensino, o professor fornece os exercícios em sala de aula para os alunos resolverem, estipulando um prazo de entrega. Após desenvolver e entregar o exercício o aluno não tem um *feedback* instantâneo do resultado. Muitas vezes, ainda, ele não tem experiência nem conhecimento para preparar casos de teste para validar o programa desenvolvido. Utilizando o computador e uma ferramenta específica para testes do software o aluno poderia receber o resultado do seu trabalho logo após o envio do mesmo.

Quando a informação de que existe algum erro no exercício chega ao aluno, a tendência é que ele efetue as correções e submeta novamente o exercício à ferramenta. Com isso a transferência de informações torna-se muito mais intensa quando comparada a um método tradicional (Sanderink, 2002).

3 OBJETOS DISTRIBUÍDOS

Uma arquitetura de objetos pode ser tratada como uma arquitetura cliente/servidor. Os objetos servidores são aqueles que oferecem um serviço para o resto do sistema, como a consulta a uma base de dados e os objetos clientes são aquelas aplicações que se utilizam destes serviços (Santos, 1998).

Um objeto distribuído é essencialmente um componente de software que pode interoperar com outros objetos distribuídos através de sistemas operacionais, redes, linguagens, aplicações, ferramentas e equipamentos diversos. Objetos distribuídos devem permitir que se distribuam objetos através de uma rede heterogênea e também que cada componente ofereça total interoperabilidade e acesso a outros objetos. (Capeletto, 1999).

Objetos distribuídos possuem as mesmas características principais dos objetos das linguagens de programação, tais como, encapsulamento, polimorfismo e herança, tendo, dessa forma, as mesmas principais vantagens: fácil reusabilidade, manutenção e depuração (Montez, 1997). Porém os objetos distribuídos não operam sozinhos. Eles são construídos para trabalhar com outros objetos e, para isso, precisam de uma espécie de barramento. Tais barramentos fornecem infra-estrutura para os objetos, adicionando novos serviços que podem ser herdados durante a construção do objeto, ou mesmo em tempo de execução para alcançar altos níveis de colaboração com outros objetos independentes (Colling, 2000).

Um exemplo desses barramentos de tecnologia que suportam a computação distribuída é o CORBA, tecnologia desenvolvida pela *Object Manegement Group* (OMG), que especifica a arquitetura completa necessária à comunicação entre objetos distribuídos.

3.1 CORBA – COMMON OBJECT REQUEST BROKER ARCHITETURE

Objetos distribuídos CORBA são pequenas partes de códigos de um sistema maior e permitem que objetos invoquem métodos e objetos distribuídos em redes, como se fossem locais. O desenvolvimento da tecnologia CORBA teve início em 1989 pela OMG.

De uma forma geral a arquitetura CORBA tem como objetivo permitir a interoperabilidade de objetos de software sobre sistemas distribuídos em plataformas heterogêneas e permitir sua composição em aplicações.

De acordo com Oss (2001), para garantir a interoperabilidade, a estratégia adotada pela OMG foi:

a) padronizar a “aparência” externa dos objetos (interface);
b) propor um mecanismo genérico de ligações entre objetos;
c) permitir uma programação que esconda a distribuição exata dos objetos (transparências);
d) oferecer um conjunto de serviços padronizados aos programadores.

A versão mais atual do CORBA disponível durante o desenvolvimento deste trabalho é a versão 2.6.

3.1.1 FUNCIONAMENTO

Para que um objeto cliente requisite algum serviço de um objeto servidor é necessário que exista uma interface que padronize a aparência externa dos objetos e suas operações. Existe no CORBA uma linguagem padronizada pela OMG, independente da arquitetura, para a definição de interfaces: a *Interface Definition Languag*e (IDL). Com a IDL cada objeto tem uma interface definida que deve conhecer e requisitar um serviço a outro objeto.

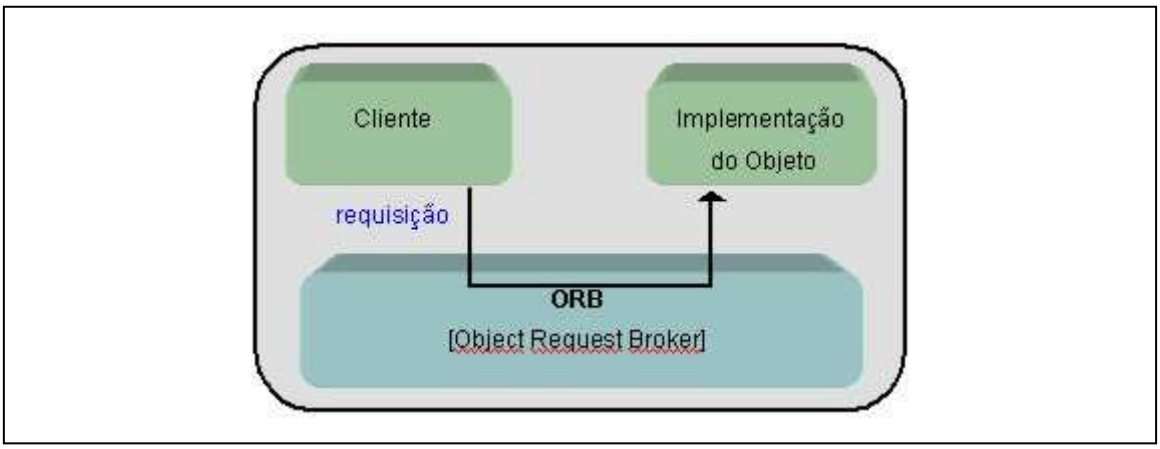
Nas implementações que utilizam a tecnologia CORBA o cliente enxerga apenas uma interface de um objeto que está distribuído na rede e, após executar a chamada, apenas aguarda pela resposta. Para o cliente todos os serviços são atendidos de uma forma transparente e para o objeto distribuído chamado, todas as requisições se comportam da mesma forma, como se fossem requisições locais.

O CORBA permite uma independência de linguagem de programação entre os objetos do cliente e do servidor. Cada um desses objetos pode ser desenvolvido em uma linguagem de programação diferente, pois as chamadas aos objetos distribuídos são feitas após um mapeamento da IDL para a linguagem de programação utilizada pelo objeto. Este mapeamento é feito automaticamente pelo pré-processador IDL e os módulos gerados são ligados ao código deste objeto (Oss, 2001).

Uma operação sobre um objeto é feita através de uma referência para o objeto. Esta interação é feita pelo *Object Request Brokers* (ORB). O ORB permite que os objetos clientes façam requisições para o servidor sem o conhecimento de onde estes objetos estão, em qual linguagem foram implementados ou em qual sistema operacional estão sendo executados.

A Figura 2 mostra o funcionamento básico do CORBA no caso de uma requisição feita pelo objeto cliente ao objeto servidor. Para que a comunicação seja feita de forma transparente, entre o cliente e a implementação do objeto desejado, o ORB é utilizado para simplificar a programação em ambientes distribuídos.

Figura 2 - Uma requisição enviada através do ORB.



FONTE: Brose (2001).

Conforme Capeletto (1999), para entender a arquitetura CORBA, é necessário entender o *Reference Model*, que consiste nos seguintes componentes:

a) *Object Request Broker*: permite aos objetos a transparência de fazer e receber solicitações e respostas em um sistema distribuído. Possibilita a interoperabilidade entre ambientes heterogêneos e homogêneos;
b) *Object Services*: são interfaces e objetos (serviços) que suportam funções básicas para usar e implementar objetos. São necessários para construir qualquer aplicação distribuída e são sempre independentes da aplicação;
c) *Common Facilities*: é uma coleção de serviços que muitas aplicações podem compartilhar, mas que não são tão fundamentais como o *Object Services*;
d) *Application Objects*: constituem a camada mais alta do *Reference Model*. Correspondem a tradicional noção de aplicações, então elas não são padronizadas

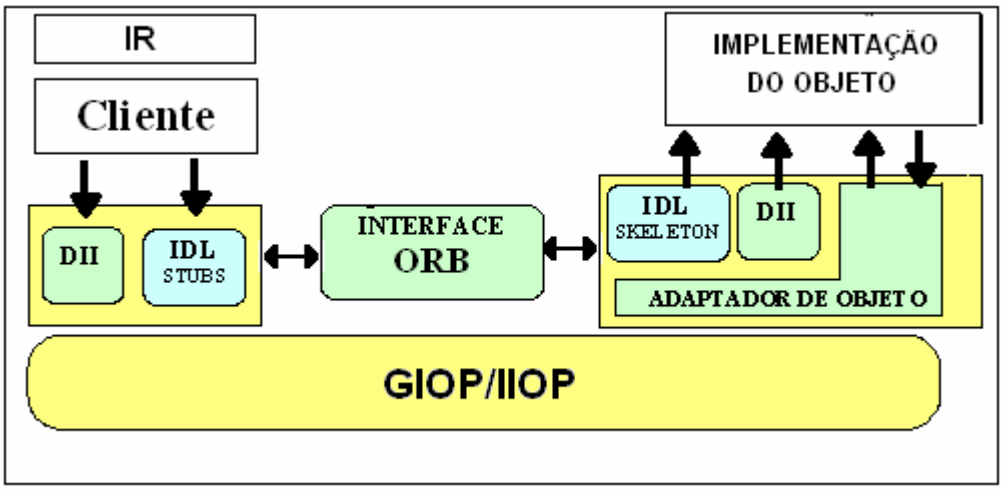
pela OMG.

3.1.2 ARQUITETURA

A arquitetura CORBA é formada por um complexo conjunto de partes com características próprias, que quando reunidas possuem um objetivo comum: promover um ambiente que atenda as especificações definidas pela OMG para o CORBA (Jacobsen, 2000).

Um objeto CORBA pode agir simultaneamente como cliente e servidor. Na Figura 3 são explicitados os componentes que fazem parte da estrutura CORBA.

Figura 3 – Arquitetura CORBA



FONTE: adaptado de Vasconcelos (1998).

A seguir estão enumerados os componentes da arquitetura CORBA com uma breve descrição de seu funcionamento:

a) cliente: pode ser chamado de cliente toda entidade responsável em solicitar uma determinada operação sobre uma implementação de objeto. Porém é um cliente em relação a um determinado servidor. Dessa forma um objeto servidor pode ser cliente de outros objetos servidores;
b) repositório de interface (IR): fornece objetos persistentes que representam a informação da IDL disponível em tempo de execução. Permite obter e modificar descrições de todas interfaces de componentes registrados, os métodos oferecidos, assim como sua assinatura;

c) interface de invocação dinâmica (DII): permite a descoberta em tempo de execução dos métodos a serem invocados;
d) *IDL stubs*: são interfaces estáticas que definem como os clientes invocam métodos nos servidores, sem preocupações com detalhes de comunicação;
e) *Object Request Broker* (ORB): é responsável pela simplificação da programação distribuída. É através do ORB que os clientes fazem chamadas de métodos como se fosse local. Ele intercepta a chamada de um objeto para o outro e fica responsável em encontrar um objeto que atenda as necessidades do pedido. Quando encontra o objeto chamado, o ORB passa os parâmetros para o mesmo, invoca os métodos necessários dele e retorna para o objeto que solicitou os resultados de todo esse procedimento;
f) implementação do objeto: provê a semântica do objeto definindo dados para a instância do objeto e definindo dados para os métodos do objeto;
g) *IDL skeletons*: disponibilizam os métodos dos objetos servidores para o resto do sistema. Tem como função encontrar os serviços que são solicitados pelo *stub*;
h) adaptador de objeto: tem como principal função auxiliar na comunicação entre o objeto servidor e o ORB, tanto para a ativação quanto para o recebimento de requisições;
i) *General Inter-ORB Protocol* (GIOP): é um padrão que especifica o formato das mensagens e uma representação comum de dados para permitir a interligação de ORB para ORB;
j) *Internet Inter-ORB Protocol* (IIOP): especifica como as mensagens de padrão GIOP são transmitidas por meio de uma rede TCP/IP.

4 FERRAMENTAS DE DESENVOLVIMENTO

Inicialmente, o desenvolvimento de sistemas para os ambientes Windows e Linux requeria a utilização da linguagem C, na qual estão implementadas as funções da *Application Programming Interface* (API) do Windows e do Kernel do Linux. Nessa época o desenvolvimento de uma aplicação extremamente simples requeria dezenas de linhas de código em linguagem C (Leão, 2001).

Em linguagens de programação como a linguagem C o desenvolvedor levava um tempo enorme apenas codificando a interface do sistema, cuja finalidade é simplesmente obter informações do usuário ou exibir informações referente ao resultado de algum processamento.

Essa realidade começou a mudar com o surgimento das primeiras ferramentas para o desenvolvimento rápido de aplicativos, denominadas ferramentas *Rapid Application Development* (RAD). Essas ferramentas permitiam associar, de maneira simples e rápida, um elemento de interface ao código de aplicação (Leão, 2001).

Em 1994 a Borland inovou o mercado de ferramentas de desenvolvimento RAD com o lançamento do Borland Delphi, uma ferramenta que aliava a facilidade do Visual Basic ao poder da linguagem Object Pascal e cujo compilador apresentava o mesmo desempenho do Borland C++. Hoje o ambiente de desenvolvimento lançado em 1994 já está na versão 6.0.

4.1 DELPHI 6.0 – INOVAÇÕES DO AMBIENTE

O Borland Delphi 6.0 é o primeiro ambiente de desenvolvimento rápido de aplicações (RAD) para Windows que suporta totalmente futuros *Web Services*, que são o próximo passo em desenvolvimento de aplicações na Internet (Borland, 2002).

O suporte a *Web Services* possibilita a integração imediata com plataformas como .NET e BizTalk da Microsoft e ONE da Sun Microsystems. Além disso, quando associado ao Borland Kylix, que é o primeiro ambiente RAD nativo para o sistema operacional Linux, os usuários de Delphi 6.0 podem criar aplicações de código fonte único, tanto para Windows quanto para Linux.

A Tabela 3 mostra os principais novos recursos do ambiente de desenvolvimento Delphi 6.0:

Tabela 3 – Novos Recursos do Delphi 6.0

Recurso	Descrição
BizSnap	É uma plataforma RAD para desenvolvimento de *Web Services*, que simplifica a integração *Business-to-business* (B2B) criando conexões e *Web Services* baseados em *Extensible Markup Language*(XML) e *Simple Object Access Protocol* (SOAP).
WebSnap	É uma estrutura de desenvolvimento de aplicações Web baseada em componentes que suporta os principais Servidores de Aplicações Web, inclusive Apache, Netscape e Microsoft Internet Information Services (IIS).
DataSnap	Permite aos clientes criar *middleware* de alto desempenho capazes de trabalhar com Web Services, possibilitando fácil conexão de qualquer serviço ou aplicação de cliente com os principais bancos de dados, por exemplo, Oracle, MS-SQL Server, Informix, IBM, DB2, Sybase e InterBase, através de Serviços Web padrão da indústria e XML, *Distributed Component Object* Model (DCOM) ou CORBA.
Biblioteca CLX	Biblioteca de componentes multiplataforma que permite a portabilidade das aplicações desenvolvidas em Delphi 6.0 para o Kylix.

Fonte : (Borland, 2002)

4.2 UTILIZANDO CORBA NO DELPHI

Para as aplicações CORBA, o ponto de partida é a interface comum (IDL) que as mesmas compartilham quando trocam informações. A IDL tem sua própria linguagem, apesar de sua sintaxe ser similar a Java e C++. Seu propósito é definir a interface para os objetos que serão passados entre aplicações CORBA. A implementação e o uso desses objetos são feitos na linguagem específica escolhida. A única condição é que esta linguagem possua recursos para mapear a arquitetura CORBA.

Aplicações CORBA podem ser implementadas em Delphi através do editor *Type Library* para criar facilmente interfaces IDL, através do MIDAS para conectar-se a dados CORBA e através do utilitário IDL2PAS para transformar código IDL em fontes Pascal.

O editor *Type Library* não está disponível quando se trabalha com uma aplicação CLX. Neste caso deve-se utilizar o utilitário IDL2PAS para a compilação das IDLs. Este utilitário pode ser acessado através do item *Corba Server Application*, na página CORBA do menu *File/New/Other* do Delphi 6.

Após a compilação das IDLs, o Delphi gera quatro arquivos (units) para cada IDL compilada. Essas units possuem as terminações _c, _s, _i e _impl que correspondem, respectivamente, ao servidor *skeleton*, ao cliente *stub*, a interface das classes e a implementação dos objetos.

Todos os métodos que deverão estar disponíveis para os clientes devem ser implementados nas *units* de implementação de objetos, ou seja, nas *units* com terminação _impl, geradas pela compilação da IDL.

As units são geradas com as variáveis e funções previstas na IDL já declaradas e semi-implementadas. Se após a implementação dos métodos previstos o programador decidir por implementar novos métodos, ele poderá alterar o código da IDL e compilá-la novamente, pois o utilitário IDL2PAS possui uma opção que indica se deve sobrescrever a *unit* de implementação de objetos. Assim todas as outras *units* (_c, _i e _s) serão geradas novamente contemplando o novo método e o programador precisará alterar apenas a unit _impl.

4.3 COMPONENT LIBRARY FOR CROSS PLATTAFORM – CLX

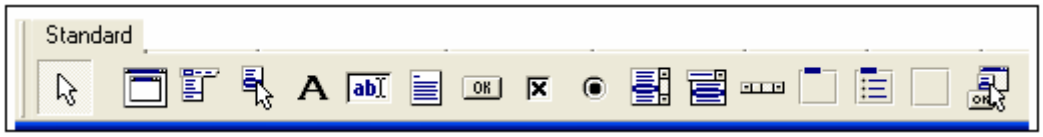
A *Visual Component Library* (VCL), biblioteca de componentes visuais do Delphi é completamente baseada na API do Windows e seria muito complicado portá-la para o Linux, pois, a menos que se use alguma biblioteca de emulação, como a WineLib, as funções da API do Windows não estão disponíveis no Linux(Leão, 2001).

A Borland optou então por reescrever completamente a VCL, mantendo seus aspectos básicos e alguns componentes, criando a CLX que é uma biblioteca para desenvolvimento de programas em Delphi e Kylix, que permite o aproveitamento de código fonte nos dois ambientes.. Para isso foi usada uma biblioteca de classes do C++, chamada Qt, criada por uma companhia norueguesa, chamada Troll Tech.

A CLX engloba as seguintes bibliotecas:

a) VisualCLX: trata-se da biblioteca onde encontram-se os componentes visuais da CLX, como por exemplo os componentes da barra *Standard* da Figura 4;

Figura 4 – Barra de componenetes Standard



b) BaseCLX: é a biblioteca onde estão as classes e rotinas básicas da CLX;

c) DataCLX: onde estão os componentes de acesso a bancos de dados, como por exemplo os componentes da barra *dbExpress*, da Figura 5;

Figura 5 – Barra de componentes dbExpress

d) NetCLX: nesta biblioteca estão os componentes de acesso à Internet, como por exemplo os componentes da barra *Internet*, como mostra a Figura 6.

Figura 6 – Barra de componentes Internet

A CLX é completamente independente da API do Windows, dependendo apenas da biblioteca Qt. Por exemplo, o componente TEdit da VCL nada mais é do que a herança do componente Edit do Windows. Na CLX o componente TEdit herda o componente QEdit do Qt. Para o desenvolvedor de aplicações isto não significa nada, uma vez que um TEdit pode ser usado da mesma maneira, tanto no Delphi quanto no Kylix. Para um desenvolvedor de componentes, porém, esta alteração é significativa internamente, pois o QEdit é muito diferente do TEdit da VCL e a maneira de modificar o comportamento deste componente não é a mesma.

5 DESENVOLVIMENTO DO TRABALHO

Neste capítulo serão apresentadas a especificação e implementação do modelo proposto neste trabalho.

5.1 REQUISITOS PRINCIPAIS DO PROBLEMA

A ferramenta desenvolvida neste trabalho aplica um método de teste de caixa preta para efetuar a correção dos exercícios submetidos pelos alunos em uma disciplina de programação.

Assim, ao concluir um exercício proposto pelo professor, o aluno o submeterá à correção. A partir de um conjunto de dados de entrada padrão, o sistema irá comparar os resultados alcançados pelo programa do aluno com os resultados esperados, informando ao aluno se o programa atende ou não as especificações do enunciado.

A ferramenta está dividida em dois módulos distintos:

a) professor: neste módulo são cadastrados os alunos da disciplina que poderão ter acesso aos exercícios. É neste módulo também que o professor cadastra os exercícios e verifica suas correções;
b) aluno: após a validação do seu usuário e senha o aluno terá acesso através deste módulo ao exercício disponibilizado pelo professor. O aluno poderá também, através deste módulo, enviar o seu exercício e verificar os erros existentes.

Dessa forma neste trabalho foram aplicados os conceitos de Objetos Distribuídos, utilizando a tecnologia CORBA para tornar possível a comunicação entre o módulo do aluno e o módulo do professor. Estes dois módulos enviam e recebem informações e podem estar sendo executados em diferentes plataformas, possibilidade esta alcançada através do uso dos componentes CLX para o desenvolvimento da aplicação.

5.2 ESPECIFICAÇÃO DO PROTÓTIPO

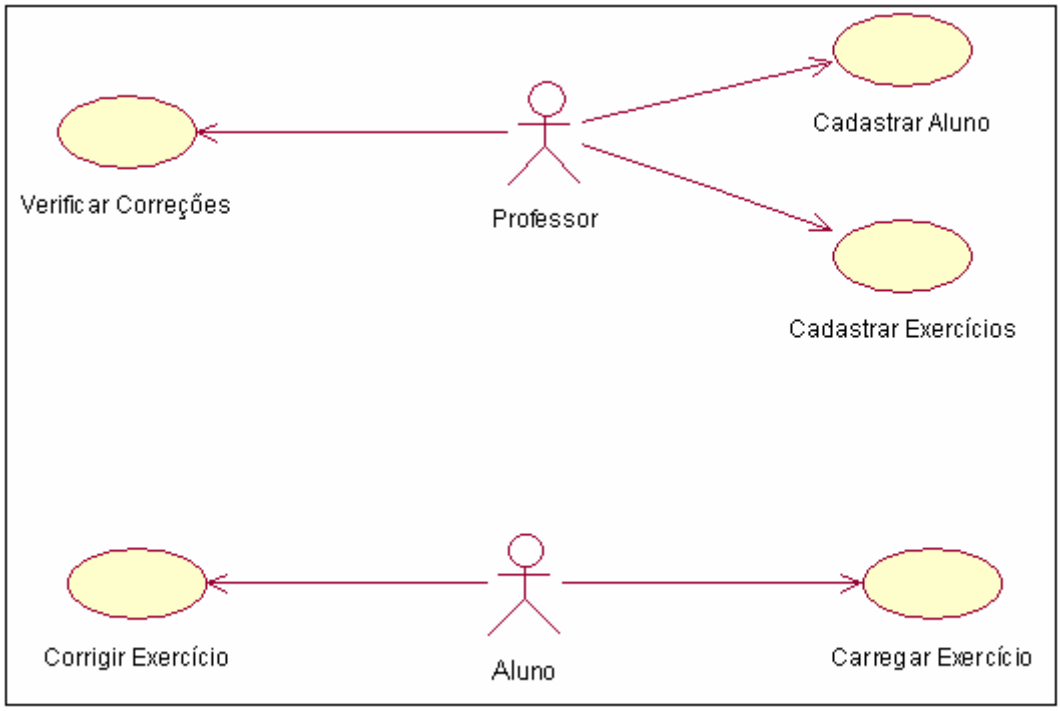
A especificação do sistema foi feita utilizando-se alguns diagramas da UML. A ferramenta CASE utilizada para esta especificação foi o *Rational Rose*, devido aos recursos

disponíveis para aplicar as representações da UML, como Diagrama de Classes, o Diagrama de Casos de Uso e o Diagrama de Seqüência que estão relacionados a seguir (Furlan, 1998).

5.2.1 DIAGRAMA DE CASOS DE USO

A Figura 7 apresenta os principais diagramas de casos de uso do sistema.

Figura 7 – Diagramas de Casos de Uso



São 5(cinco) os casos de uso identificados neste programa:

a) cadastrar Aluno: é a entrada dos dados referentes aos alunos da disciplina de programação e a gravação dos mesmos na base de dados. Somente através do módulo do professor é que os alunos poderão ser cadastrados;
b) cadastrar Exercício: é a entrada dos dados referentes aos exercícios e a gravação dos mesmos na base de dados. Somente através do módulo do professor é que os exercícios poderão ser cadastrados;
c) carregar Exercício: o aluno poderá consultar uma lista de enunciados para escolher o exercício que quer resolver;
d) corrigir Exercício: o Aluno envia a resolução do Exercício, isto é, o programa, para

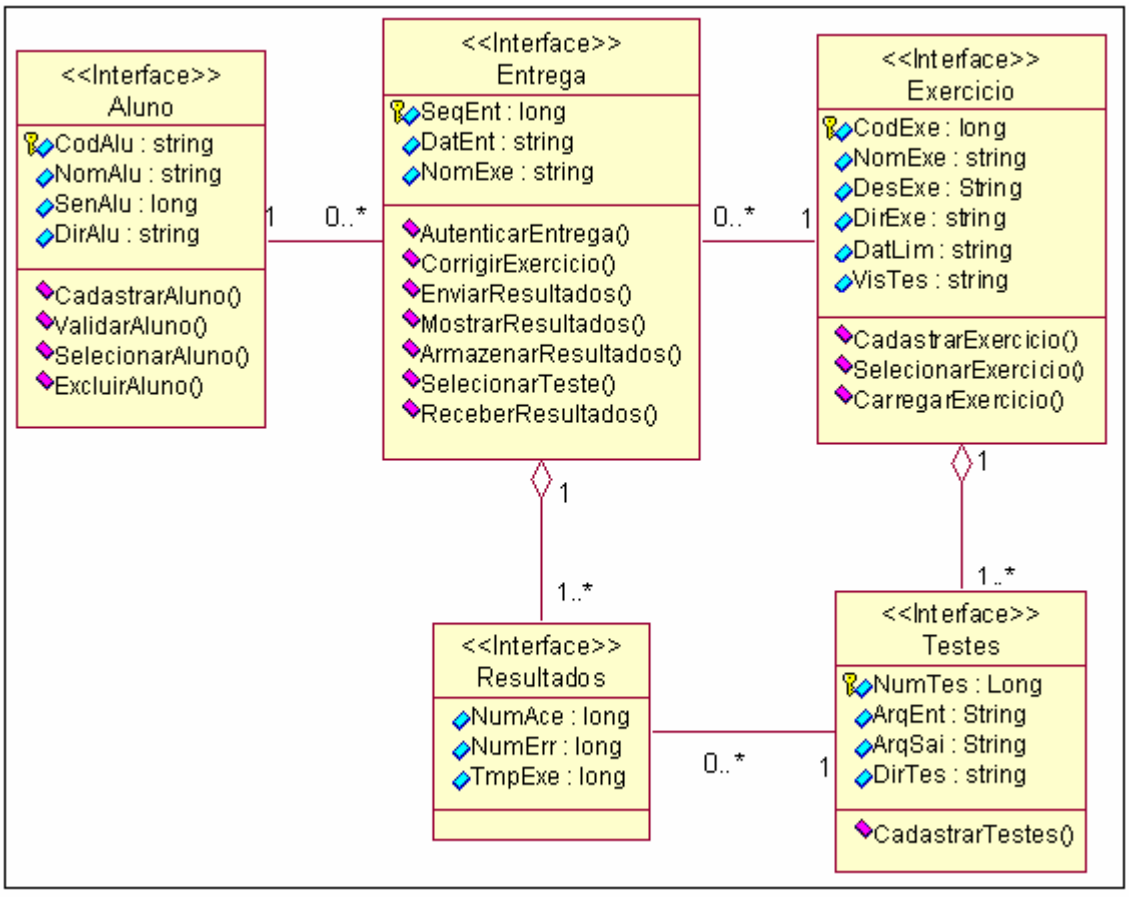
que a ferramenta possa corrigi-lo e recebe uma resposta com os casos de testes não aceitos pela ferramenta.

e) verificar Correções: após o sistema registrar a correção dos exercícios submetidos pelo aluno, o Professor pode consultar essas correções para verificar como anda o aprendizado do Aluno;

5.2.2 DIAGRAMA DE CLASSES

A Figura 8 demonstra o diagrama de classes.

Figura 8 – Diagramas de Classes do protótipo



Existem 5 (cinco) classes identificadas no sistema:

a) aluno: é a classe que mantém as informações dos alunos cadastrados no sistema através do módulo do professor. Apenas alunos cadastrados poderão ter acesso aos exercícios;

b) exercício: é a classe que mantém os exercícios cadastrados pelo módulo do professor;
c) testes: é uma agregação da classe exercício. Um exercício poderá ter vários casos de testes;
d) entrega: essa classe mantém as informações sobre os exercícios enviados pelo módulo do aluno. O aluno poderá enviar o mesmo exercício várias vezes;
e) resultados: é a classe que armazena as informações sobre a correção do exercício. Os resultados armazenados são específicos para cada caso de teste submetido ao programa.

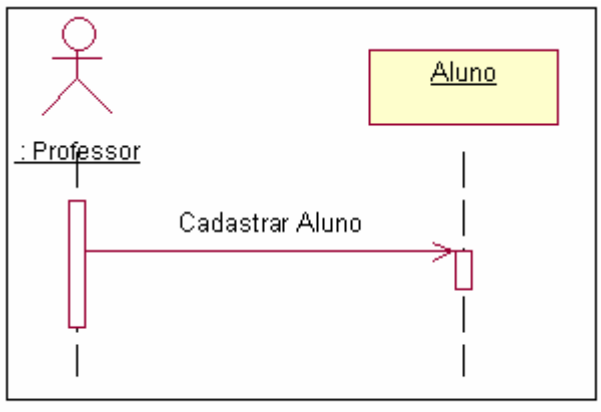
5.2.3 DIAGRAMAS DE SEQÜÊNCIA

Os diagramas de seqüências representam, como o próprio nome indica, a seqüência em que as ações ocorrem dentro do sistema. Eles demonstram como é feita a troca de mensagens entre as classes. Para cada caso de uso, há um diagrama de seqüência, conforme descrição a seguir.

5.2.3.1 CADASTRAR ALUNO

O professor através do seu módulo informa os dados dos alunos que serão gravados através da rotina “CadastrarAluno” da classe Aluno.

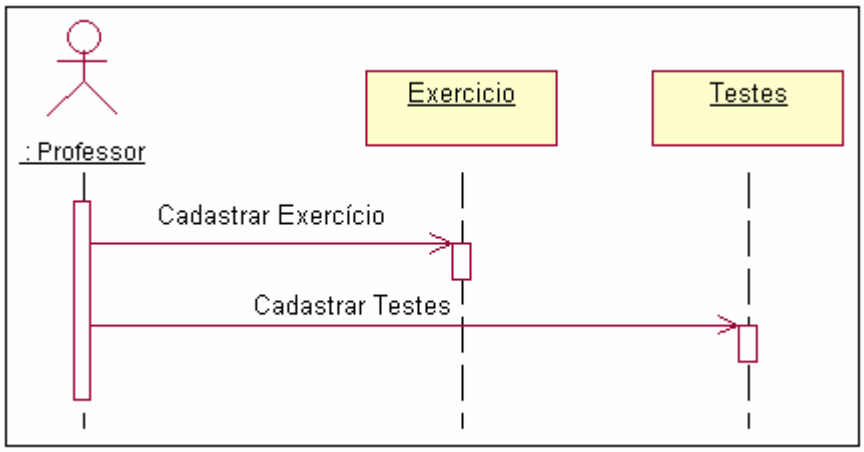
Figura 9 – Diagrama de Seqüência Cadastrar Aluno



5.2.3.2 CADASTRAR EXERCÍCIO

O professor cadastra os exercícios através do método “CadastrarExercício” da classe Exercício, informando o código do exercício, o nome do documento texto que contém o enunciado do exercício, o diretório onde está gravado esse documento, uma pequena descrição do exercício e uma data limite para a entrega. Depois cadastra os casos de teste para esse exercício através do método “CadastrarTestes”, informando uma seqüência para o teste, o nome do arquivo texto que contem os dados de entrada, o nome do arquivo texto que contem os dados esperados na saída e o diretório onde esses arquivos estão armazenados no servidor.

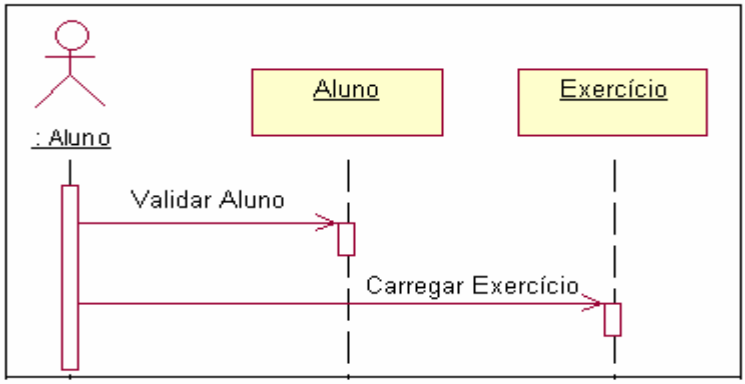
Figura 10 – Diagrama de Seqüência Cadastrar Exercício



5.2.3.3 CARREGAR EXERCÍCIO

O aluno acessa o sistema através do método “ValidarAluno”, esse método solicita o código do aluno e sua senha e verifica se o aluno está cadastrado no sistema e se a senha informada confere com a cadastrada, permitindo assim o acesso às funcionalidades do módulo. Acessando o módulo o aluno pode através do método “CarregarExercicio” verificar quais os exercícios estão disponibilizados pelo módulo do professor.

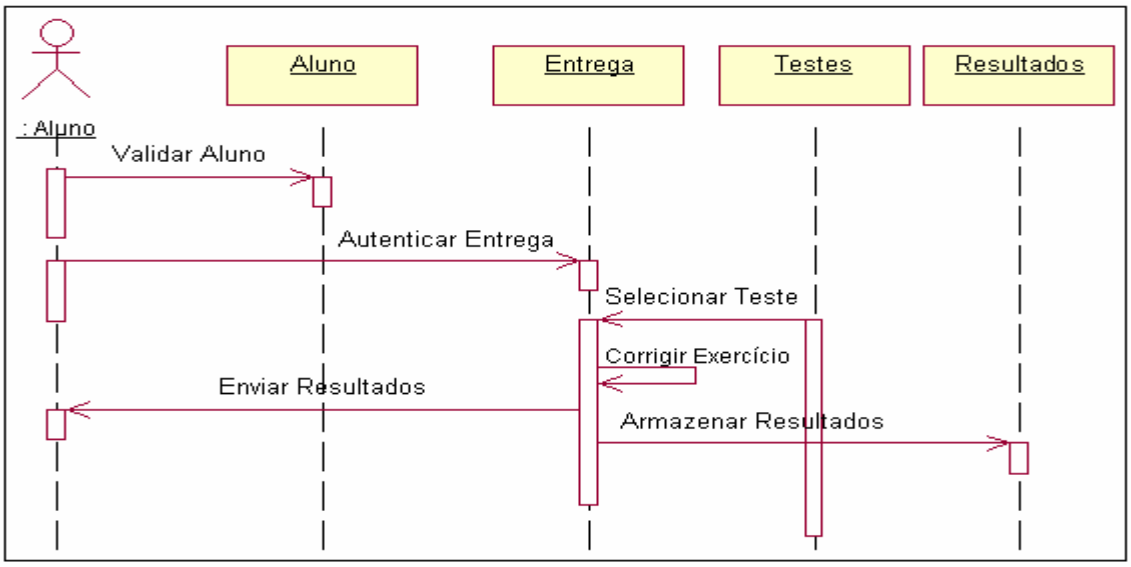
Figura 11 – Diagrama de Seqüência Pegar Exercício



5.2.3.4 CORRIGIR EXERCÍCIO

O aluno envia o exercício implementado e a classe “Entrega” grava as informações através do método “AutenticarEntrega”, em seguida a classe “Entrega” recebe as informações sobre os casos de testes que deverão ser aplicados ao programa. A ferramenta efetua os testes através do método “CorrigirExercício”, grava os resultados e envia uma resposta ao aluno indicando se o programa funcionou através do método “EnviarResultados”.

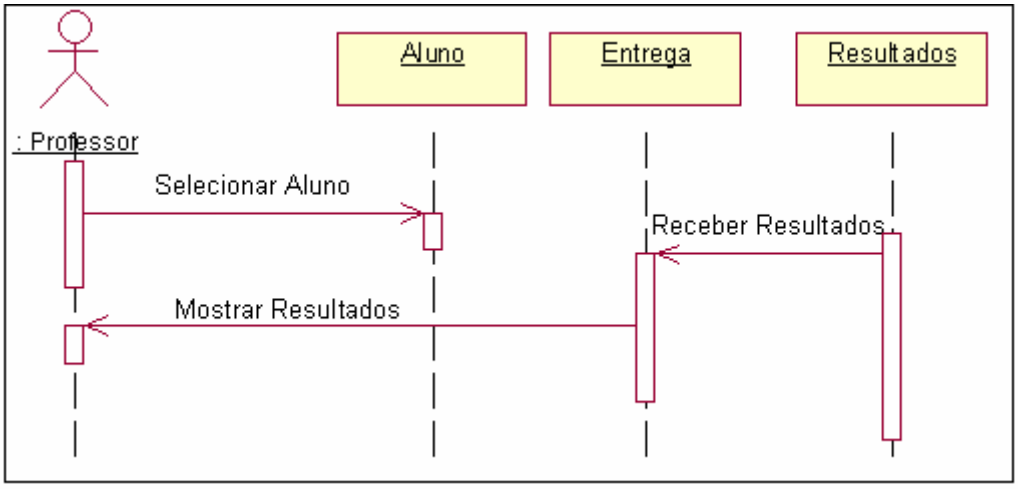
Figura 12 – Diagrama de Seqüência Corrigir Exercício



5.2.3.5 VERIFICAR CORREÇÕES

O professor seleciona o aluno sobre o qual deseja verificar as correções. Com a informação do aluno a classe “Entrega” recebe as informações sobre os resultados através do método “ReceberResultados”. Com essas informações a classe “Entrega” apresenta os resultados dos testes para o professor utilizando o método “MostrarResultados”.

Figura 13 – Diagrama de Seqüência Verificar Correções



5.2.4 IMPLEMENTAÇÃO

A seguir é apresentada a implementação do protótipo que foi feita no ambiente de programação Delphi 6 utilizando os componentes da biblioteca CLX, para verificar a funcionalidade da tecnologia CORBA para uma futura migração da aplicação para o sistema operacional Linux.

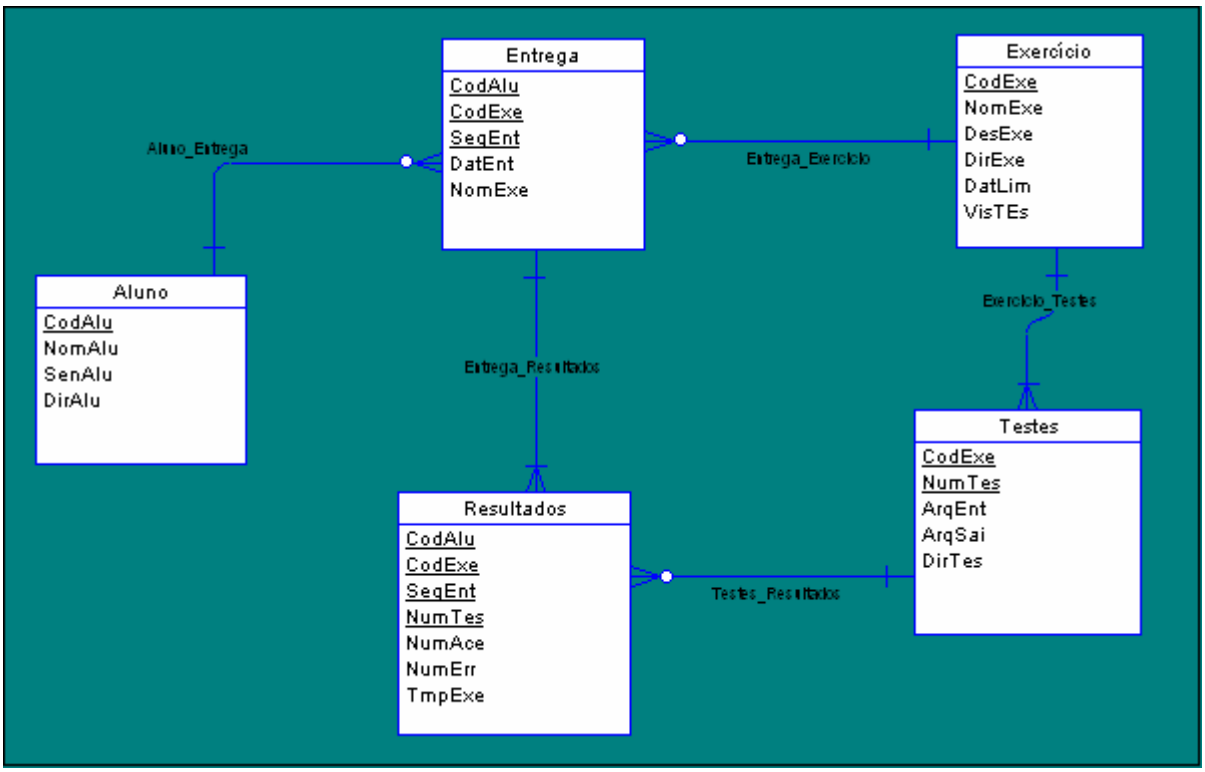
O protótipo foi dividido em três partes distintas:

a) servidor Corba : neste servidor foram disponibilizadas todas as interfaces, implementações, skeletons e stubs necessários para a comunicação entre os objetos através do CORBA. Este módulo foi denominado ServerCLX;
b) módulo do Professor: encontram-se nessa parte do protótipo as rotinas que apenas o professor tem acesso, como cadastro de alunos, cadastro de exercícios, etc.. Este módulo foi denominado ProfessorCLX;
c) módulo do Aluno: neste módulo encontram-se as rotinas para *Download* e *Upload* dos exercícios. Este módulo foi denominado AlunoCLX.

Estes módulos serão detalhados a seguir.

Para armazenar as informações utilizadas pelo sistema foi utilizado o banco de dados Interbase 6. A Figura 14 mostra o modelo de entidade e relacionamento criado a partir do Diagrama de Classes proposto para o trabalho.

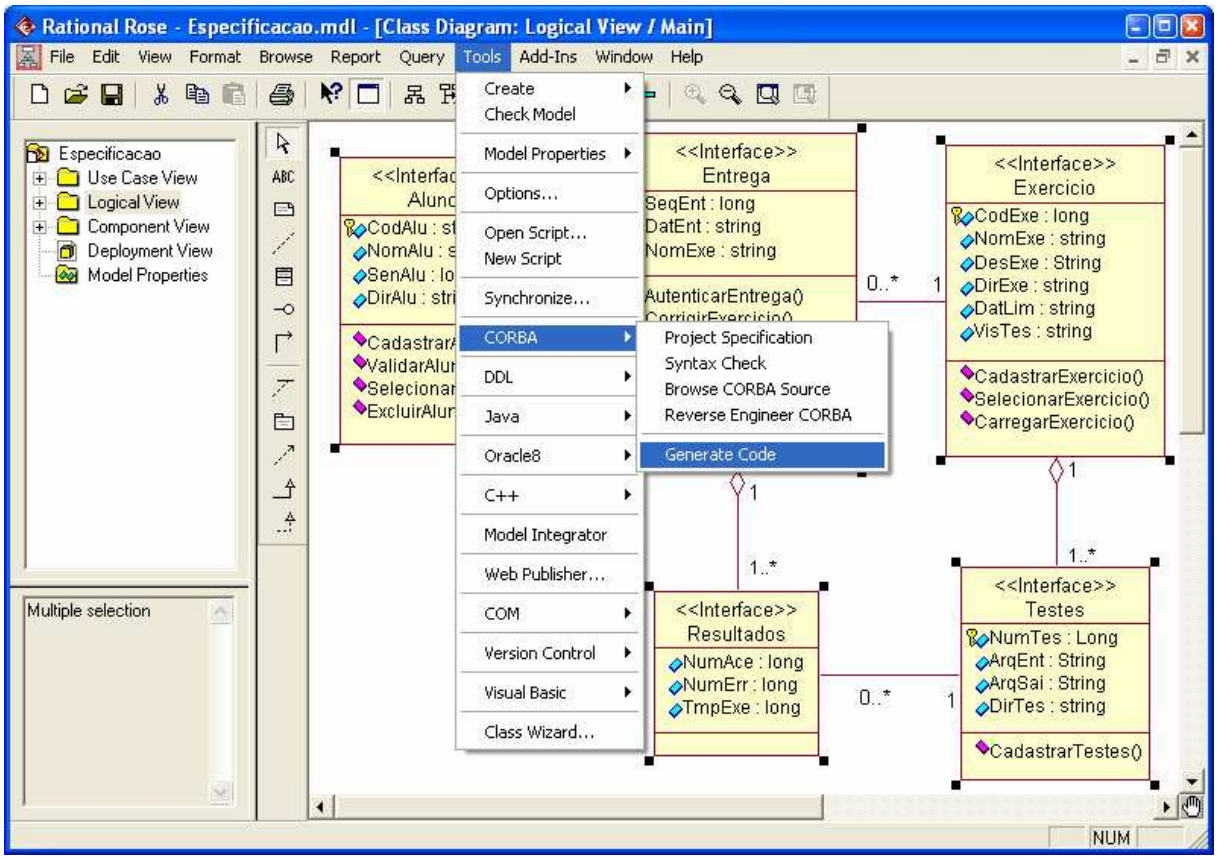
Figura 14 – Modelo Entidade e Relacionamento



5.2.4.1 SERVIDOR CORBA - SERVERCLX

O primeiro passo para a implementação de uma aplicação distribuída é a geração de sua IDL. Para a implementação do Servidor Corba foram necessárias cinco IDLs. Essas IDLs foram geradas através da ferramenta Rational Rose, através do menu “Tools, Corba, Generate Code”,como mostra a Figura 15.

Figura 15 – Geração das IDLs através da ferramenta Rational Rose



O Quadro 1 mostra a IDL gerada para a interface da classe Aluno.

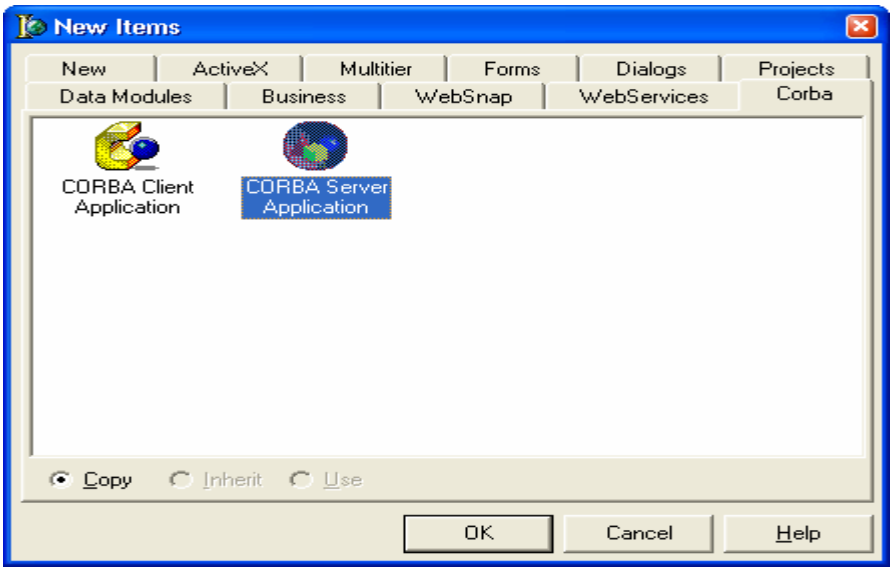
Quadro 1 - IDL Aluno

```
Interface Aluno {
attribute string CodAlu;
attribute string NomAlu;
attribute long SenAlu;
attribute string DirAlu;
boolean CadastrarAluno (in string CodAlu,in string NomAlu,in long SenAlu,in string DirAlu);
boolean ValidarAluno (in string CodAlu, in long SenAlu);
boolean SelecionarAluno (in string CodAlu,out string NomAlu,out long SenAlu,out string DirAlu);
};
```

Dessa mesma forma foram criadas as IDLs para as classes Exercício, Entrega, Testes e Resultados. Todas as IDLs podem ser encontradas no Anexo 1 desse trabalho.

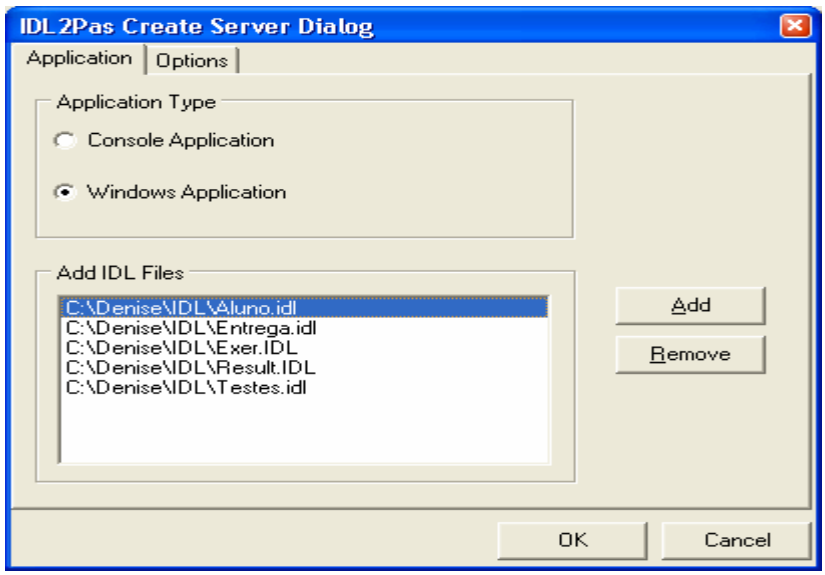
O próximo passo é a compilação das IDLs. Para isto o Delphi 6 disponibiliza a ferramenta *IDL2PAS Create Server Dialog*, que pode ser acessada através do menu *File\New\Other* posicionando na página CORBA e abrindo o item *Corba Server Application* (Figura 16).

Figura 16 – Corba Server Applicaton



Selecionando esse item, o Delphi abrirá a janela da *IDL2PAS Create Server Dialog*. Nesta janela deve-se adicionar todas as IDLs desejadas através do botão “Add” como mostra a Figura 17.

Figura 17 – IDL2PAS Create Server Dialog



Ao clicar em ok nesta tela o Delphi irá gerar os seguintes arquivos (units):

a) aluno_c.pas, entrega_c.pas, exer_c.pas, result_c.pas e testes_c.pas: estes arquivos são utilizados para a definição do servidor skeleton;
b) aluno_s.pas, entrega_s.pas, exer_s.pas, result_s.pas e testes_s.pas: estes arquivos são utilizados para a definição do cliente stub;
c) aluno_i.pas, entrega_i.pas, exer_i.pas, result_i.pas e testes_i.pas: estes arquivos são utilizados para a definição das interfaces das classes;
d) aluno_impl.pas, entrega_impl.pas, exer_impl.pas, result_impl.pas e testes_impl.pas: estes arquivos são utilizados para a implementação dos objetos no servidor.

Todas essas units geradas foram adicionadas ao módulo ServerCLX. O próximo passo foi desenvolver a implementação dos objetos distribuídos nas units aluno_impl.pas, entrega_impl.pas, exer_impl.pas, result_impl.pas e testes_impl.pas.

Após a implementação dos objetos deve-se criar a unit principal, onde é necessário inicializar o CORBA e associar as units da interface, implementação e skeleton do objeto distribuído (Quadro 2).

Quadro 2 – Declaração dos objetos

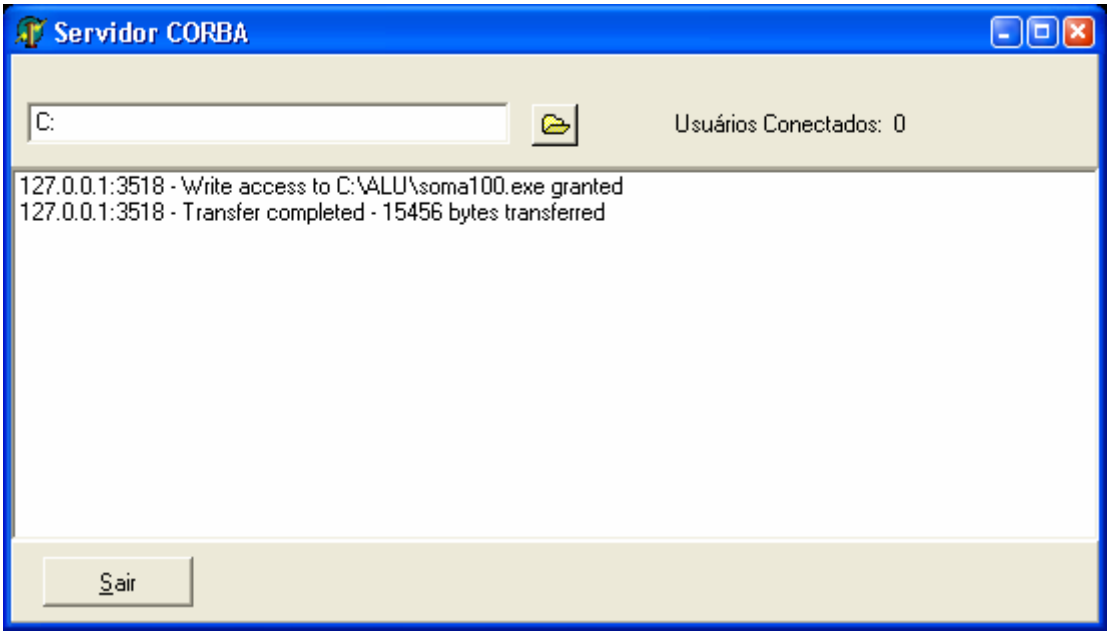
```
procedure TFMenuServerCLX.InitCorba;
begin
  CorbaInitialize;  //inicializa o orb

  // Cria o objeto servidor
  faluno := TAlunoSkeleton.Create('Aluno', TAluno.Create);
  BOA.ObjIsReady(FAluno as _Object);

  fExer := TExerSkeleton.Create('Exercicio', TExer.Create);
  BOA.ObjIsReady(FExer as _Object);

  fTestes := TTestesSkeleton.Create('Testes', TTestes.Create);
  BOA.ObjIsReady(FTestes as _Object);

  fEntrega := TEntregaSkeleton.Create('Entrega', TEntrega.Create);
  BOA.ObjIsReady(FEntrega as _Object);

  fResult := TResultSkeleton.Create('Result', TResult.Create);
```

```
  BOA.ObjIsReady(FResult as _Object);
end;
```

Além de disponibilizar as interfaces necessárias para que os clientes possam comunicar-se com o servidor o ServerCLX também executa o envio e o recebimento de arquivos, como por exemplo, os arquivos executáveis que serão enviados pelos alunos para a correção. Para isso foi utilizado nesse trabalho um componente FTP disponível no ambiente Delphi 6 com a utilização da biblioteca CLX.

Dessa forma a tela principal do ServerCLX ficou conforme mostra a Figura 18.

Figura 18 – Tela principal do módulo ServerCLX



Nesta tela o usuário informa qual o diretório no servidor que será considerado como o diretório raiz pela ferramenta. Abaixo do diretório selecionado serão geradas as estruturas de diretórios para armazenar as informações dos exercícios e dos alunos.

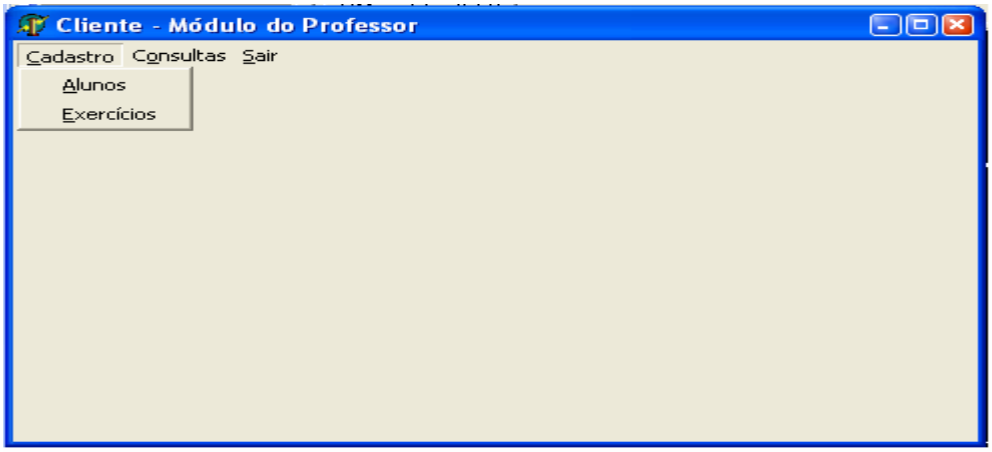
A tela mostra também o número de usuários conectados ao servidor e um “log” dos processos que estão sendo executados, conforme as solicitações dos clientes.

5.2.4.2 MÓDULO DO PROFESSOR – PROFESSORCLX

No módulo do professor foram implementadas as funções que não estarão disponíveis para os alunos, como cadastro de exercícios e seus casos de testes, cadastro dos alunos, etc.. Através deste módulo o professor poderá também verificar os resultados das correções feitas nos exercícios submetidos pelos alunos.

A tela principal do ProfessorCLX (módulo do professor) é um menu que permite o acesso a todas as telas desse módulo (Figura 19). As rotinas disponíveis neste módulo serão explicadas em seguida.

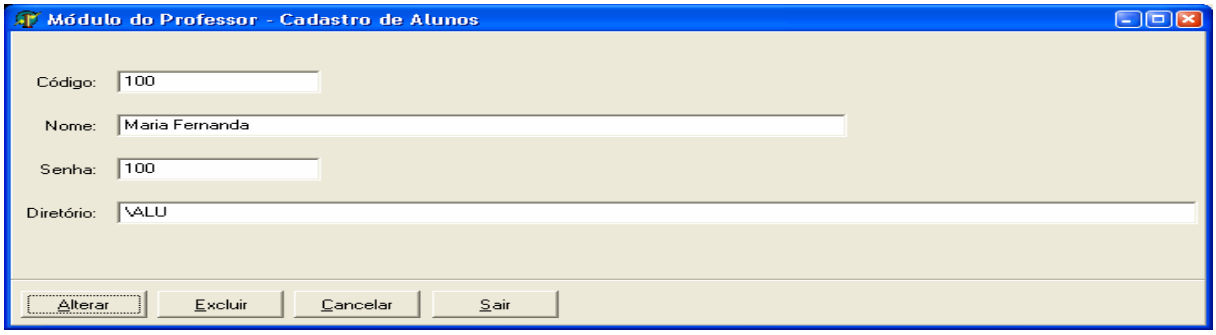
Figura 19 – Menu principal do módulo do Professor



5.2.4.2.1 CADASTRAR ALUNO

Nesta tela o professor deverá cadastrar todos os alunos que poderão ter acesso ao módulo do Aluno, conforme mostra a Figura 20.

Figura 20 – Cadastro de alunos do módulo ProfessorCLX



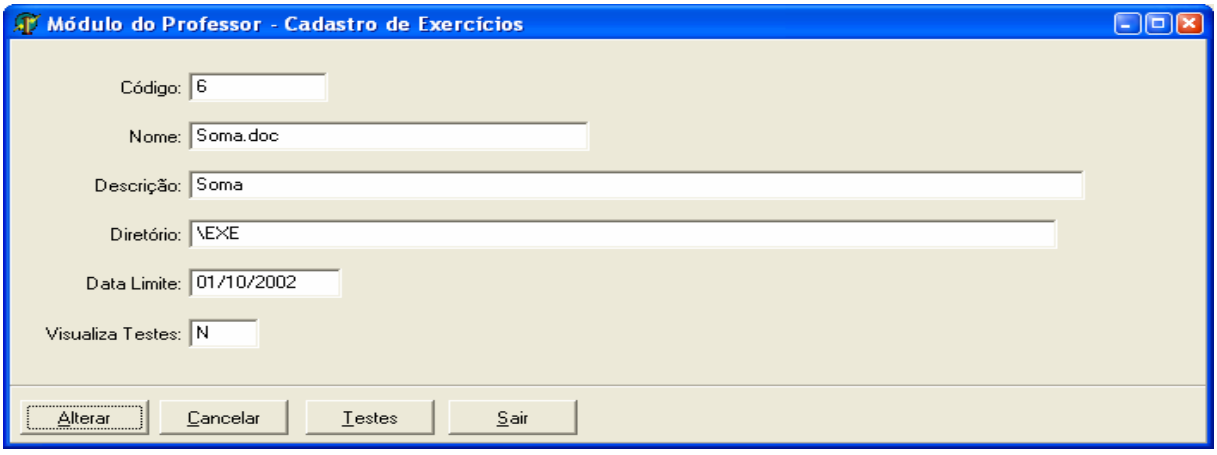
Nesta tela o professor informa o código do aluno, seu nome completo, uma senha e o diretório do aluno no servidor. Através desta tela o professor pode incluir, alterar e excluir alunos.

A informação do diretório é muito importante, pois quando o aluno submeter o seu exercício para correção o sistema vai gravar o código fonte, o executável e os resultados obtidos, neste diretório.

5.2.4.2.2 CADASTRAR EXERCÍCIOS

Nesta tela o professor poderá cadastrar os exercícios que ficarão disponíveis para todos os alunos cadastrados. O professor deverá informar um código para o exercício, o nome, uma descrição, o diretório, a data de entrega e se o aluno poderá visualizar os testes, conforme a Figura 21.

Figura 21 – Cadastro de Exercícios do módulo ProfessorCLX



Nesta tela, algumas informações são muito importantes para o funcionamento da ferramenta, como o nome do exercício. Esse nome corresponde ao nome do arquivo que contém o enunciado do exercício. Da mesma forma a informação do diretório é muito importante, pois corresponde ao diretório no servidor onde o enunciado do exercício está gravado. É através dessas duas informações que a ferramenta vai localizar o exercício para enviar ao módulo do Aluno.

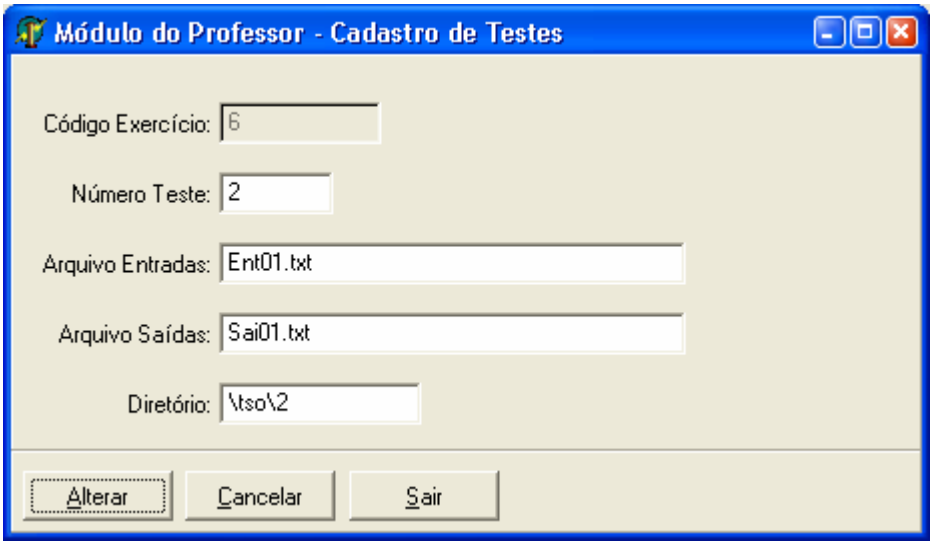
A data limite indica para o aluno o prazo que ele tem para a entrega do exercício. A ferramenta permite que o aluno entregue o exercício mesmo após a data especificada pelo

professor, porém este poderá verificar os alunos que entregaram o exercício com atraso através da consulta de resultados.

O campo visualiza testes indica se o aluno poderá receber os arquivos de entrada que não geraram as saídas esperadas quando submetidos ao programa enviado, se este campo estiver preenchido com “N” o aluno apenas receberá a informação do número de erros e do números de acertos que obteve.

Através do botão “Testes” disponível nesta tela o professor deverá cadastrar os casos de testes para cada exercício cadastrado, conforme a Figura 22.

Figura 22 – Cadastro de Testes



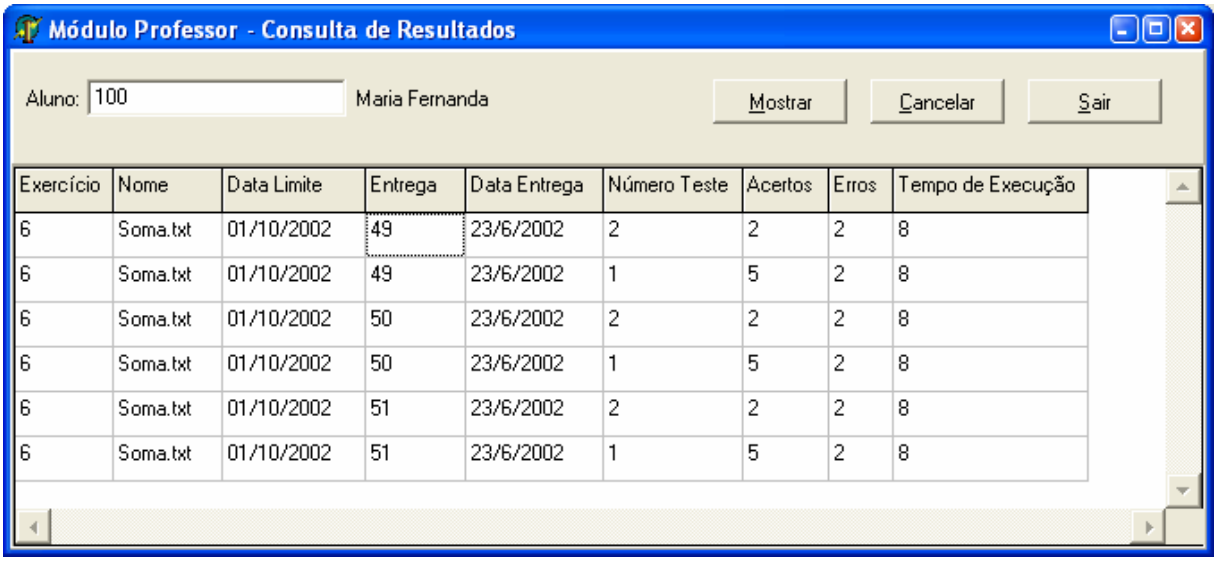
Todas as informações desta tela também são utilizadas pela ferramenta para a correção dos exercícios submetidos pelos alunos. Para cada exercício cadastrado é possível relacionar vários arquivos de casos de testes (entradas e saídas).

5.2.4.2.3 VERIFICAR CORREÇÕES

Nesta tela o professor poderá verificar as entregas de exercícios de cada um dos alunos cadastrados. Para isso ele deverá informar o código do aluno e pressionar o botão “Mostrar”. O sistema irá mostrar a lista de todas as entregas feitas pelo aluno selecionado, como mostra a

Figura 23.

Figura 23 – Tela de Consulta de Resultados



As informações disponíveis nessa tela são:

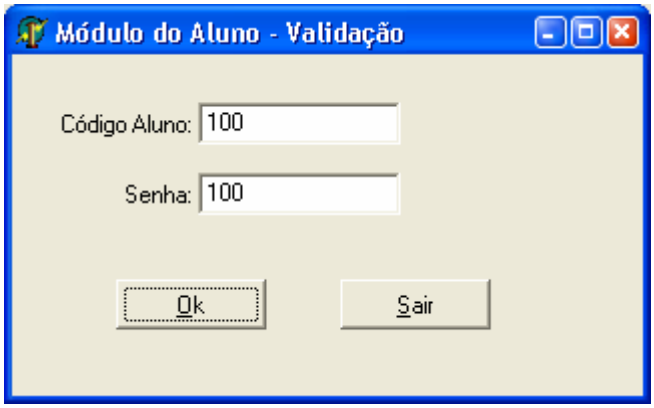
a) número do exercício: é o número do cadastro do exercício;
b) nome do exercício: é o nome do documento onde está o enunciado do exercício;
c) data limite: é a data limite para a entrega do exercício;
d) entrega : é o número da entrega feita pelo aluno;
e) data de entrega: é a data que o aluno fez a entrega do exercício;
f) número teste: é o número do arquivo de teste que foi submetido ao exercício;
g) acertos: é o número de acertos que o aluno obteve com a entrega;
h) erros: é o número de erros que o aluno obteve com a entrega;
i) tempo de execução: é o tempo, em segundos, que o programa do aluno demorou para executar.

5.2.4.3 MÓDULO DO ALUNO – ALUNOCLX

Neste módulo foram implementadas as funcionalidades disponíveis para os alunos das disciplinas de programação, tais como a consulta de exercícios disponíveis para resolução, o *download* do enunciado do exercício e o envio do exercício resolvido.

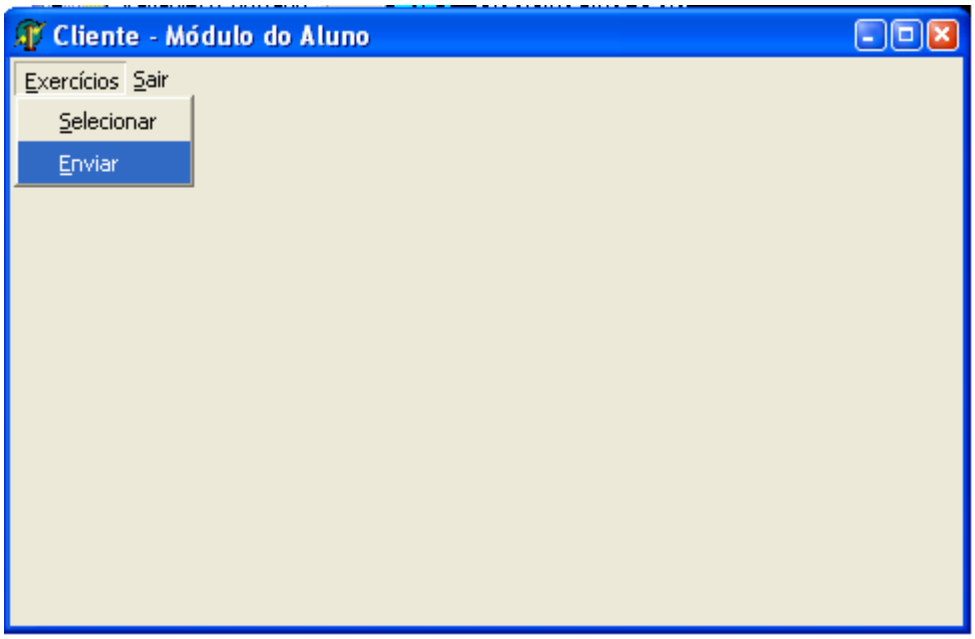
Ao acessar o módulo, o aluno deverá informar o seu código e senha para que a ferramenta possa verificar se o aluno está autorizado a acessar suas funcionalidades (Figura 24).

Figura 24 – Tela de Validação dos Alunos



Se o aluno estiver cadastrado e informar a sua senha corretamente o sistema apresentará a tela principal do módulo que é o menu, disponibilizando o acesso a todas as demais funções do AlunoCLX, conforme a Figura 25.

Figura 25 – Menu principal do módulo do aluno

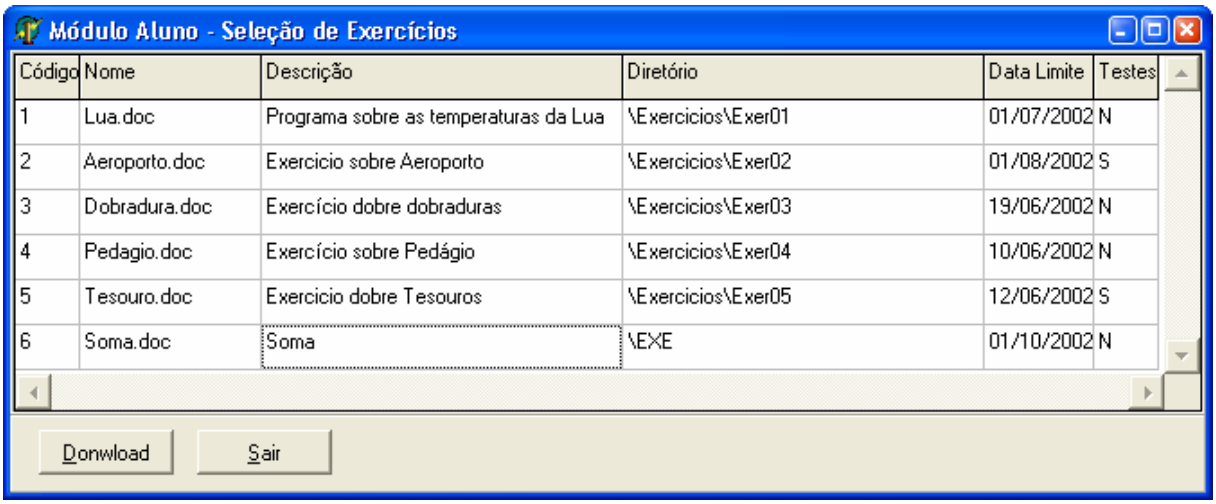


As funcionalidades disponíveis no módulo do aluno serão descritas a seguir.

5.2.4.3.1 CARREGAR EXERCÍCIO

Ao acessar essa tela a ferramenta já carrega uma lista com todos os exercícios cadastrados pelo módulo do professor, conforme a Figura 26.

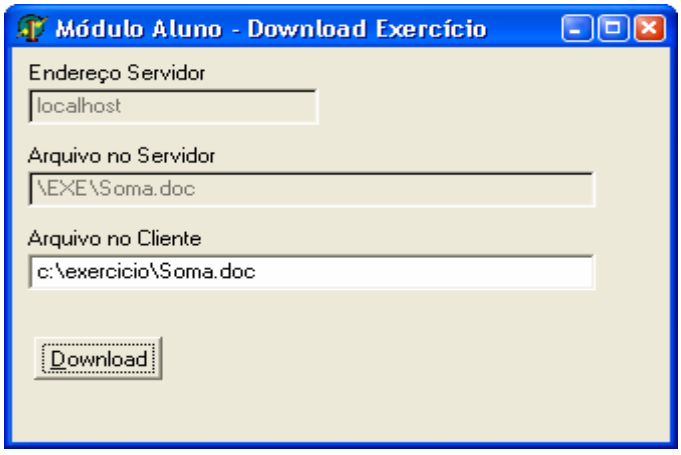
Figura 26 – Tela de Seleção de Exercícios



Nesta tela o aluno poderá verificar todas as informações sobre os exercícios cadastrados pelo professor e selecionar qual o exercício deseja fazer o *download* do enunciado.

Para fazer o *download* basta posicionar o cursor sobre o exercício desejado na lista e clicar no botão “Download”. O sistema apresentará a tela de *download* como mostra a Figura 27.

Figura 27 – Download do exercício



O sistema já carrega as informações do endereço do servidor e o caminho e nome do arquivo onde está o enunciado do exercício selecionado. O aluno apenas deverá informar o caminho e o nome do arquivo que deverá ser gerado no seu computador.

Ao clicar no botão “Download” o módulo do aluno irá conectar-se ao módulo ServerCLX através de um componente FTP para realizar a transferência do arquivo “Soma.doc” que está no servidor para o diretório informado no campo “Arquivo no Cliente”.

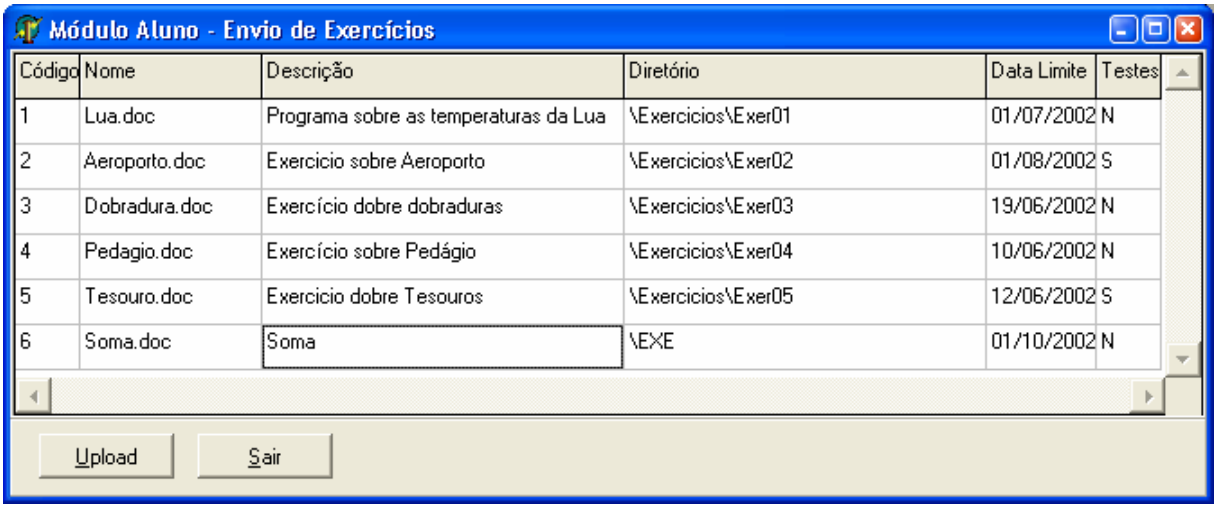
No anexo 2 desse trabalho pode-se verificar uma sugestão de padrão para o enunciado dos exercícios que estarão disponíveis para *download*.

5.2.4.3.2 CORRIGIR EXERCÍCIO

Após fazer o *download* do enunciado do exercício, o aluno deverá resolvê-lo utilizando a linguagem de programação especificada no exercício ou em sala de aula pelo professor. O aluno deverá seguir atentamente as instruções do enunciado para a leitura dos dados de entrada e para a geração dos dados de saída.

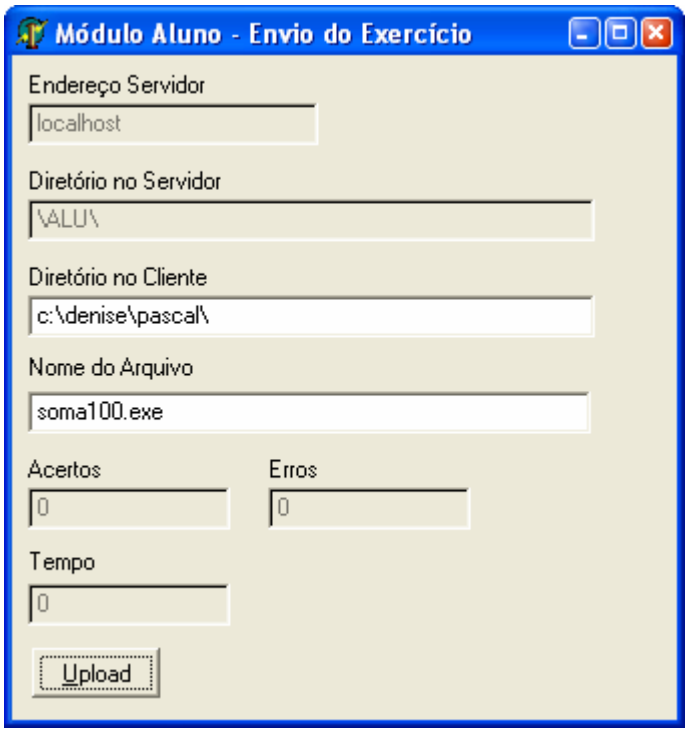
Assim que concluir o exercício o aluno deve enviar o arquivo fonte e o executável para a correção. A tela de envio é muito semelhante a tela de seleção de exercícios. O sistema também apresenta a lista com todos os exercícios cadastrados no módulo do professor para que o aluno escolha qual o exercício deseja submeter à correção (Figura 28).

Figura 28 – Envio de Exercícios para correção



Para enviar a resolução do exercício, basta posicionar o cursor sobre o mesmo e clicar no botão “Upload”. O sistema apresentará a tela a seguir, conforme a Figura 29.

Figura 29 – Envio do Exercício



A ferramenta carrega automaticamente as informações do endereço do servidor e do diretório do aluno no servidor. É neste diretório que a resolução enviada será gravada. O aluno deverá informar o diretório no seu computador onde está o exercício resolvido e o nome do arquivo, como “soma100.exe” utilizado no exemplo acima. Com todas as informações preenchidas basta clicar no botão “Upload” para transferir o arquivo.

Neste momento o ServerCLX recebe a resolução do exercício e efetua a autenticação da entrega, gravando um registro na tabela de entregas com as seguintes informações: código do aluno, código do exercício, seqüência da entrega, data de entrega e nome do exercício enviado. O Quadro 3 ostra o código implementado para o método “AutenticarEntrega”.

Quadro 3 – Método Autenticar Entrega

```
function  TEntrega.AutenticarEntrega : Boolean;
var
   QueryUpd : TQuery;
   QueryMax : TQuery;
   VarSeqEnt : Integer;
begin
result := false;
try
  QueryUpd := nil;
  QueryMax := nil;
  try
    VarSeqEnt := 0;
    QueryMax := TQuery.Create(nil);
    QueryMax.DatabaseName := 'TCCDenise';
    QueryMax.SQL.Add('SELECT MAX(SEQENT) AS SEQENT FROM ENTREGA WHERE
CODALU = :CODALU AND CODEXE = :CODEXE');
    QueryMax.ParamByName('CodAlu').AsString := _CodALu;
    QueryMax.ParamByName('CodExe').AsInteger := _CodExe;
    QueryMax.Open;
    if    not(QueryMax.EOF) then
          VarSeqEnt := QueryMax.FieldByName('SeqEnt').AsInteger + 1;

    QueryUpd := TQuery.Create(nil);
    QueryUpd.DatabaseName := 'TCCDenise';
    QueryUpd.SQL.Add('INSERT INTO ENTREGA
(CODALU,CODEXE,SEQENT,DATENT,NOMEXE) VALUES
(:CODALU,:CODEXE,:SEQENT,:DATENT,:NOMEXE)');

    QueryUpd.ParamByName('CodAlu').AsString := _CodAlu;
    QueryUpd.ParamByName('CodExe').AsInteger := _CodExe;
    QueryUpd.ParamByName('SeqEnt').AsInteger := VarSeqEnt;
    QueryUpd.ParamByName('DatEnt').AsString := DateTimeToStr(Date);
    QueryUpd.ParamByName('NomExe').AsString := _NomExe;
    QueryUpd.ExecSQL;
    _SeqEnt := VarSeqEnt;
  finally
    QueryUpd.Free;
    QueryMax.Free;
  end;
except
  raise exception.Create('Erro na gravaçao da Entrega');
end;
CorrigirExercicio;
result := true;
end;
```

Após o recebimento do arquivo pelo ServerCLX a ferramenta automaticamente executa o arquivo recebido submetendo todos os casos de testes, cadastrados para o exercício em questão, à resolução enviada pelo aluno. Automaticamente são comparados os arquivos de saída gerados pela solução do aluno com os arquivos de saídas corretos que estão

armazenados no servidor para verificar o número de erros e o número de acertos que o aluno obteve.

Para cada caso de teste submetido ao exercício o ServerCLX grava um registro na tabela de Resultados com as seguintes informações: código do aluno, código do exercício, seqüência da entrega, número do caso de teste, número de acertos e número de erros. Dessa forma a ferramenta mantém uma estatística das entregas e números de acertos e erros feitos por cada aluno, para cada exercício. No Quadro 4 é apresentada a implementação do método "CorrigirExercício".

Quadro 4 – Método Corrigir Exercício

```
function  TEntrega.CorrigirExercicio : Boolean;
var
     Query,QueryAlu,QueryUpd : tQuery;
     VarAux,Caminho1,CaminhoFinal,Arq1,Arq2 : String;
     Saida1, Saida2 : TStringList;
     i : byte;
     VNumAce, VNumErr : Integer;
begin
try
    Saida1 := TStringList.Create;
    Saida2 := TStringList.Create;
    QueryAlu := TQuery.Create(nil);
    QueryAlu.DatabaseName := 'TCCDenise';
    QueryAlu.SQL.Add('SELECT DIRALU FROM ALUNO WHERE CODALU = :CODALU');
    QueryAlu.ParamByName('CodAlu').AsString := _CodAlu;
    QueryAlu.Open;

    Query := TQuery.Create(nil);
    Query.DatabaseName := 'TCCDenise';
    Query.SQL.Add('SELECT * FROM TESTES WHERE CODEXE = :CODEXE');
    Query.ParamByName('CodExe').AsInteger := _CodExe;
    Query.Open;

    Caminho1 := 'C:' + QueryAlu.FieldByName('DirAlu').AsString + '\' + _NomExe;
    while not(Query.Eof) do
       begin
       CaminhoFinal := 'C:' + Query.FieldByName('DirTes').AsString + '\';
       VarAux := Caminho1 + ' ' + CaminhoFinal;
       winexec(PChar(VarAux),sw_hide);
              //Compara a Saida gerada com a Saida correta
       VNumAce := 0;
       VNumErr := 0;
       Arq1 := CaminhoFinal + 'Sai01.txt';
       Arq2 := CaminhoFinal + 'S' + _CodAlu + '.txt';
       Saida1.Clear;
       Saida2.Clear;
       Saida1.LoadFromFile(Arq1);
       Saida2.LoadFromFile(Arq2);
       for i:= 0 to Saida1.Count - 1 do
           begin
           if  (Saida1.Strings[i] <> Saida2.Strings[i]) then
               inc(VNumErr)
           else inc(VNumAce);
           end;
       QueryUpd := nil;
       try
         QueryUpd := TQuery.Create(nil);
         QueryUpd.DatabaseName := 'TCCDenise';
         QueryUpd.SQL.Add('INSERT INTO RESULTADOS (CODALU,CODEXE,SEQENT,NUMTES,NUMACE,NUMERR)
' +
```

```
                        'VALUES (:CODALU,:CODEXE,:SEQENT,:NUMTES,:NUMACE,:NUMERR)');
         QueryUpd.ParamByName('CodAlu').AsString  := _CodAlu;
         QueryUpd.ParamByName('CodExe').AsInteger := _CodExe;
         QueryUpd.ParamByName('SeqEnt').AsInteger := _SeqEnt;
         QueryUpd.ParamByName('NumTes').AsInteger := Query.FieldByName('NumTes').AsInteger;
         QueryUpd.ParamByName('NumAce').AsInteger := VNumAce;
         QueryUpd.ParamByName('NumErr').AsInteger := VNumErr;
         QueryUpd.ExecSQL;
       finally
         QueryUpd.Free;
       end;
       Query.Next;
       end;
finally
     Query.Free;
     QueryAlu.Free;
     Saida1.Free;
     Saida2.Free;
end;

end;
```

Após efetuar todos os testes no exercício enviado o ServerCLX devolve para o AlunoCLX a informação do tempo de execução do seu exercício, quantos erros e quantos acertos ele obteve, preenchendo os três últimos campos da tela vista na Figura 29.

5.2.5 DISCUSSÃO DOS RESULTADOS

Para um melhor entendimento do funcionamento da ferramenta e para verificar os resultados obtidos, será feita a explicação utilizando um exemplo de exercício e alguns casos de testes para que a ferramenta possa corrigi-lo. Como exemplo será utilizado o exercício “Soma.doc”, descrito no Quadro 5.

Quadro 5 – Enunciado de um exercício “Soma.doc”

Soma e Divisão de Números

1. Tarefa

Sua tarefa é escrever um programa Pascal que, receba 2 números inteiros, faça a soma desses dois números e depois faça a divisão dessa soma obtida por 2.

2. Entrada

O programa deverá receber como parâmetro o caminho onde está localizado o arquivo de entrada. A entrada deverá ser lida do arquivo EntS.TXT, através do caminho recebido. A entrada é composta de vários conjuntos de teste. Sendo que cada linha do arquivo forma um conjunto de teste. Cada linha terá 2 números inteiros separados por um espaço em branco.

Exemplo de Entrada

```
4 2
7 4
4 3
2 5
6 3
```

3. Saída

Para cada conjunto de teste da entrada seu programa deve produzir uma linha no arquivo SaiS.TXT. Cada linha corresponde ao resultado fornecido pelo seu programa, sendo o primeiro número a soma dos dois número lidos e o segundo número a divisão da soma por dois. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida

rigorosamente. Se o resultado da divisão não for um número inteiro, este deverá ser truncado, mostrando apenas a parte inteira da divisão.

Exemplo de Saída

6 3
11 5
7 3
7 3
9 4
(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

0 <= entrada <= 500

O professor deverá elaborar o enunciado do exercício, buscando esclarecer da melhor forma possível os resultados que se deseja com a execução. O próximo passo a ser executado pelo professor é o planejamento e a elaboração dos casos de teste que serão submetidos aos programas enviados pelo aluno.

Para esse exemplo foram definidos alguns casos de teste que contemplem as seguintes situações:

a) valores de entrada diferentes de zero e dentro do intervalo válido definido;
b) valores de entrada iguais ao valor mínimo definido no intervalo;
c) valores de entrada iguais ao valor máximo definido no intervalo;
d) um valor de entrada menor que o mínimo definido no intervalo e o outro valor de entrada maior que o máximo definido no intervalo;
e) um valor válido dentro do intervalo e um valor fora do intervalo.

A Tabela 4 mostra alguns casos de testes definidos para atender os requisitos especificados pelo problema.

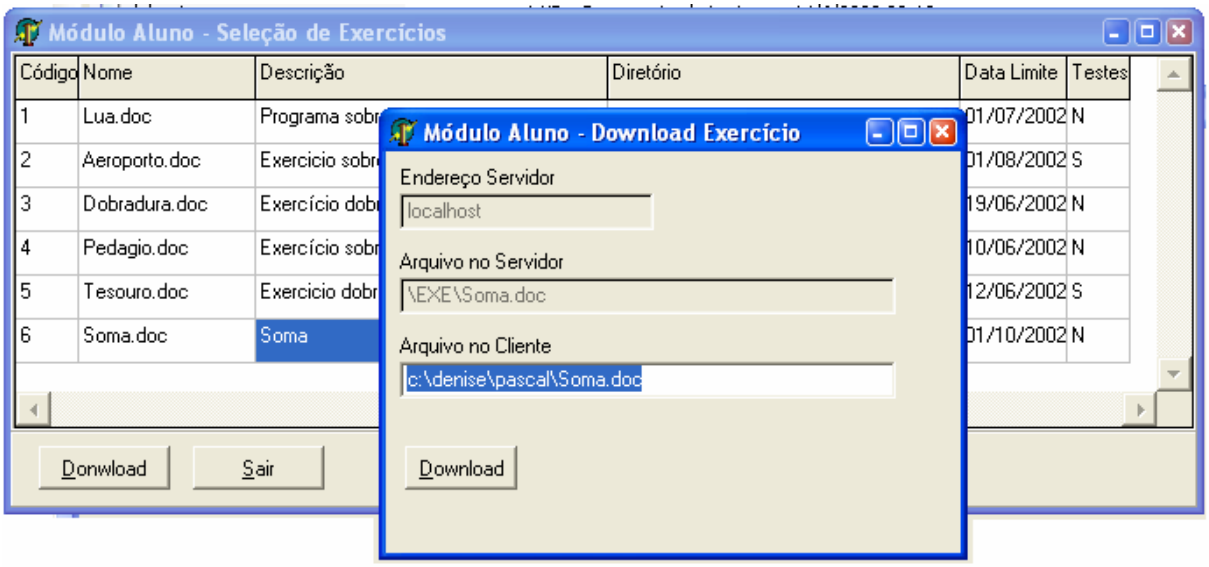
Tabela 4 - Exemplo de entradas e resultados esperados

Valores de Entrada	Saídas Esperadas
200 e 100	300 e 150
50 e 25	75 e 37
10 e 499	509 e 254
0 e 0	0 e 0
0 e 100	100 e 50
0 e 125	125 e 62
500 e 500	1000 e 500

333 e 500	833 e 416
50 e 0	50 e 25
-1 e 501	“Inválida”
105 e 505	“Inválida”

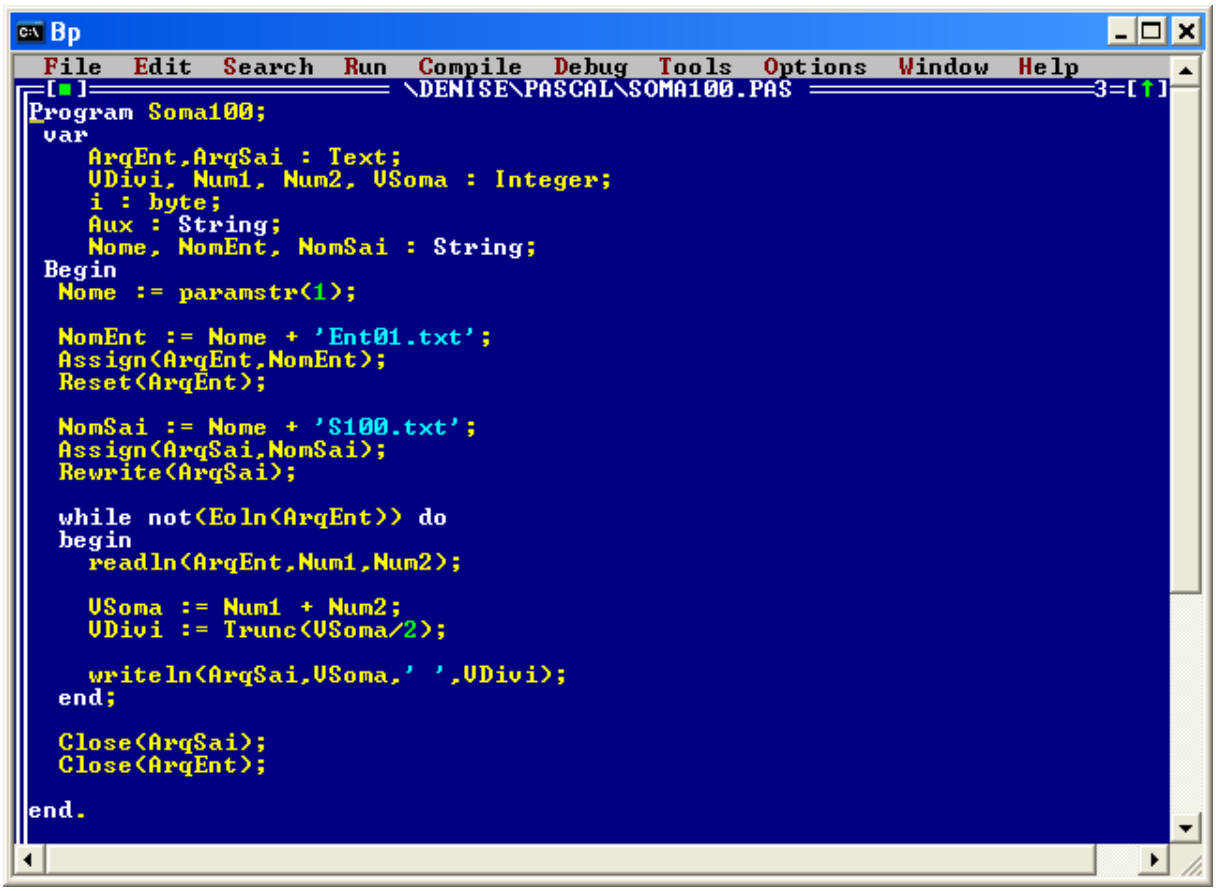
O aluno, através do módulo AlunoCLX, vai verificar quais os exercícios que o professor disponibilizou para serem implementados. Ao acessar no seu módulo o item de menu Selecionar Exercícios, o sistema chama o método “CarregarExercicio”, implementado utilizando o padrão CORBA no módulo ServerCLX. Feito isso ele seleciona o exercício desejado, para efetuar o *download* do enunciado. A Figura 30 mostra as telas acessadas no módulo do Aluno para realizar o download do exercício.

Figura 30 – Seleção e Download do exercício “Soma.doc”



O próximo passo é de responsabilidade do aluno, ele deverá resolver o exercício na linguagem de programação especificada pelo professor da disciplina, no nosso exemplo a linguagem utilizada será o Pascal. A Figura 31 mostra o código fonte do exercício “Soma.doc” resolvido.

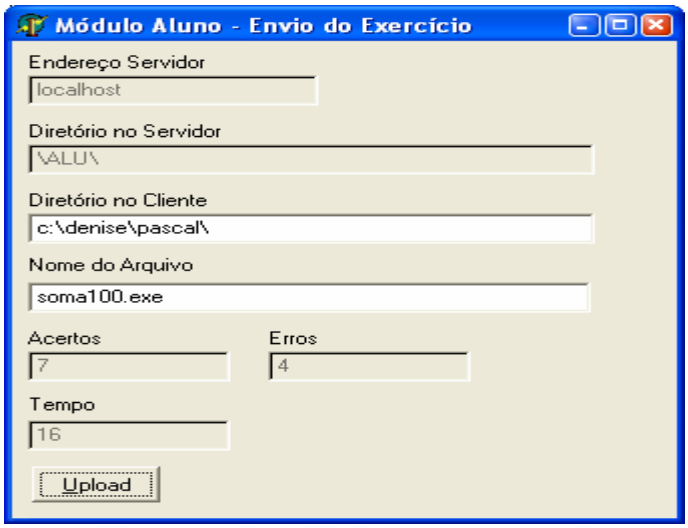
Figura 31 – Resolução do exercício “Soma.doc” utilizando a linguagem Pascal



Após resolver o exercício o aluno deverá enviar o executável para que a ferramenta possa submetê-lo a correção. Quando o ServerCLX recebe o arquivo executável ele imediatamente executa o programa uma vez para cada arquivo de casos de teste relacionado ao exercício em questão.

Depois o ServerCLX compara o arquivo de saídas gerado pelo programa do aluno com o arquivo de saídas esperadas. Para cada linha comparada o sistema registra no banco de dados se o resultado está correto ou incorreto. No final são enviados para o módulo do aluno o tempo que o seu programa levou para executar, o número de erros e o número de acertos cometidos, como mostra a Figura 32.

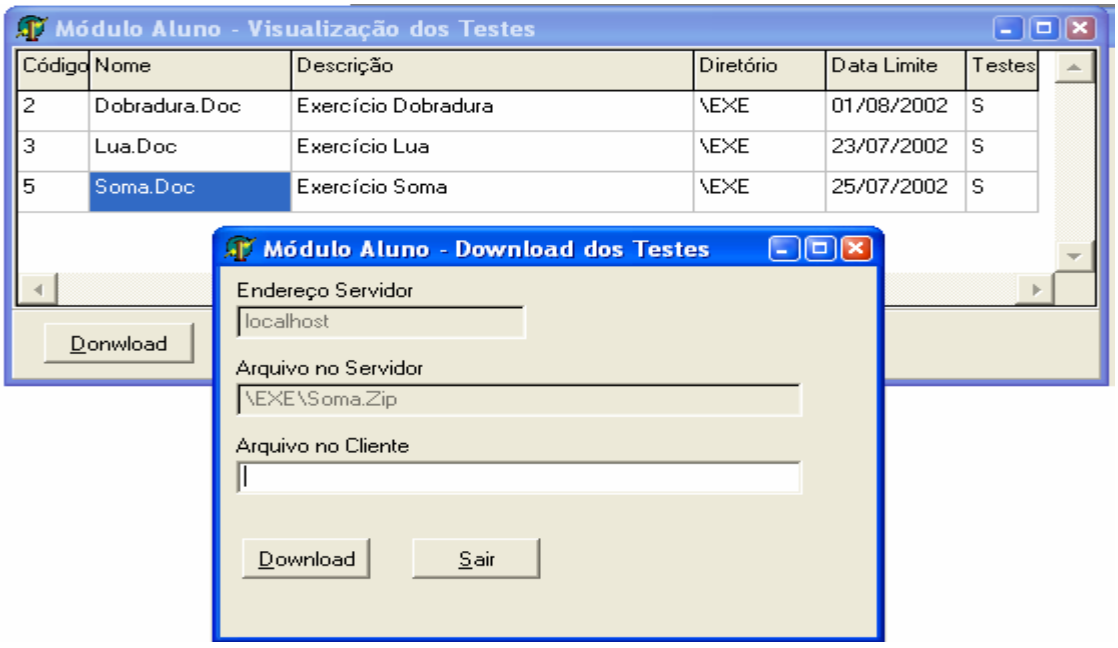
Figura 32 – Tela com o retorno enviado pelo ServerCLX



Além de verificar o número de acertos e erros, o aluno também poderá receber os casos de testes que foram submetidos ao seu programa.

Para isso, ao cadastrar o exercício o professor deverá informar o campo “Visualiza Testes” como “S”. Neste caso o aluno poderá acessar a tela de visualização de testes, como mostra a Figura 33.

Figura 33 - Tela de Visualização de Testes



Nesta tela será apresentada uma relação com os exercícios que foram cadastrados com o campo visualiza testes igual a S. O aluno seleciona o exercício que deseja visualizar os testes e pressiona o botão “Donwload” para efetuar a transferência do arquivo para um diretório a ser especificado em sua máquina.

6 CONSIDERAÇÕES FINAIS

Com este trabalho foi possível confirmar que as soluções para sistemas distribuídos surgiram para resolver os diversos desafios de integração entre as aplicações, além de facilitar o processo de comunicação entre os diversos módulos de tais tipos de sistemas.

O objetivo principal do trabalho, que era desenvolver uma ferramenta de apoio para testes de programas utilizando o teste de validação, foi alcançado com sucesso. A ferramenta foi desenvolvida e os casos de testes de validação podem ser aplicados nas correções dos programas submetidos pelos alunos. Com a disponibilização desta ferramenta em uma disciplina de programação, o professor tem melhores condições de acompanhar o desenvolvimento dos alunos em relação aos exercícios, além de propiciar que o aluno possa testar o programa com casos de testes que efetivamente permitam a validação do mesmo. Obviamente que a preparação do exercício e dos casos de teste continuam como responsabilidade do professor onde o mesmo deve observar os aspectos relevantes para os procedimentos de testes discutidos no início deste trabalho.

Sobre os objetivos específicos, pode-se dizer que também foram atingidos, pois a ferramenta foi desenvolvida utilizando os componentes da biblioteca CLX para que futuramente possa ser compilada no ambiente Kylix. Utilizou-se o padrão CORBA para a implementação dos objetos distribuídos também com o intuito de permitir a migração para outras plataformas de sistemas operacionais. Finalmente foram desenvolvidos e cadastrados no banco de dados Interbase os exercícios de programação na linguagem Pascal com seus casos de testes.

O ambiente de desenvolvimento Delphi 6 atendeu bem às expectativas em relação ao uso da tecnologia CORBA. No entanto quando utiliza-se os componentes da biblioteca CLX algumas funcionalidades ficam restringidas uma vez que muitos dos componentes CORBA, disponíveis em uma aplicação VCL, não estão disponíveis na aplicação CLX.

6.1 EXTENSÕES

Como sugestões para continuação deste trabalho destacam-se :

a) migrar a ferramenta para o ambiente operacional Linux, utilizando o ambiente de programação Kylix, para verificar a portabilidade entre o Delphi 6 e o Kylix;

b) fazer uma comparação do desempenho da ferramenta nos dois ambientes citados acima, verificando qual dos dois mostra-se mais eficiente nesse tipo de aplicação;
c) aprimorar o módulo do professor, desenvolvendo regras para calcular uma nota para o aluno baseando-se nos seus acertos, erros e número de vezes que submeteu o mesmo exercício;
d) implementar na ferramenta os testes de caixa branca;
e) desenvolver uma ferramenta para auxiliar o professor na geração dos casos de testes.

ANEXO 1 – IDLS UTILIZADAS PARA IMPLEMENTAR O SERVIDOR CORBA

IDL – ALUNO

```
interface Aluno {
attribute string CodAlu;
attribute string NomAlu;
attribute long SenAlu;
attribute string DirAlu;
boolean CadastrarAluno ();
boolean ValidarAluno ();
boolean SelecionarAluno ();
boolean ExcluirAluno ();};
```

IDL – ENTREGA

```
interface Entrega {
attribute long SeqEnt;
attribute string DatEnt;
attribute long CodExe;
attribute string CodAlu;
attribute string NomExe;
 boolean AutenticarEntrega ();
 boolean CorrigirExercicio ();
 boolean EnviarResultados ();
 boolean ReceberResultados ();
 boolean MostrarResultados ();
 boolean SelecionarTeste(); };
```

IDL – EXERCÍCIO

```
interface Exer {
attribute long CodExe;
attribute string NomExe;
attribute string DesExe;
attribute string DirExe;
attribute string DatLim;
attribute string VisTes;
 boolean CadastrarExercicio ();
 boolean SelecionarExercicio ();
 boolean CarregarExercicio (); };
```

IDL – RESULTADOS

```
interface Result {
attribute string CodAlu;
attribute long CodExe;
attribute long SeqEnt;
attribute long NumTes;
attribute long NumAce;
attribute long NumErr;
attribute long TmpExe; };
```

IDL – TESTES

```
interface Testes {
attribute Long CodExe;
attribute Long NumTes;
attribute String ArqEnt;
attribute String ArqSai;
attribute string DirTes;
 boolean CadastrarTestes (); };
```

ANEXO 2 –ENUNCIADO PARA UM EXERCÍCIO DE PROGRAMAÇÃO

Temperatura Lunar

Sem as proteções da atmosfera e do cinturão magnético que existem na Terra, a Lua fica exposta ao ataque do Sol, que é um astro em constante explosão atômica. As explosões do Sol emitem ondas letais de partículas. Uma pessoa que ficasse desprotegida na superfície da Lua, num lugar onde o Sol incidisse diretamente, sofreria um bombardeio radioativo tão intenso quanto se estivesse nas imediações da usina russa de Chernobyl no momento do acidente que matou 31 pessoas, em 1986. Além da radiação solar, outro efeito desta falta de proteção contra o Sol que existe na Lua é a enorme variação de temperatura. Nas regiões próximas do equador lunar, a variação de temperatura é brutal, passando de cerca de 130 graus positivos durante o dia a 129 graus negativos à noite.

Para estudar com mais precisão as variações de temperatura na superfície da Lua, a NASA enviou à Lua uma sonda com um sensor que mede a temperatura de 1 em 1 minuto. Um dado importante que os pesquisadores desejam descobrir é como se comporta a média da temperatura, considerada em intervalos de uma dada duração (uma hora, meia hora, oito horas, etc.). Por exemplo, para a seqüência de medições 8, 20, 30, 50, 40, 20, -10, e intervalos de quatro minutos, as médias são respectivamente 108/4=27, 140/4=35, 140/4=35 e 100/4=25.

1. Tarefa

Você foi recentemente contratado pela NASA, e sua primeira tarefa é escrever um programa que, conhecidos a seqüência de temperaturas medidas pelo sensor, e o tamanho do intervalo desejado, informe qual a maior e qual a menor temperatura média observadas, considerando o tamanho do intervalo dado.

2. Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém dois números inteiros positivos *N* e *M*, que indicam respectivamente o número total de medições de temperatura de uma seqüência obtida pelo sensor, e o tamanho dos intervalos, em minutos, em que as médias devem ser calculadas. As *N* linhas seguintes contêm um número inteiro cada, representando a seqüência de medidas do sensor. O final da entrada é indicado quando $N = M = 0$.

Exemplo de Entrada

```
4 2
-5
-12
06
7 4
35
-35
5
100
100
50
50
0 0
```

3. Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas. A primeira linha identifica o conjunto de teste, no formato “Teste *n*”, onde *n* é numerado a partir de 1. A segunda linha deve conter dois números inteiros *X* e *Y*, separados por ao menos um espaço em branco, representando respectivamente os valores da menor e da maior média de temperatura,

conforme determinado pelo seu programa. O valor da média deve ser truncado, se a média não for um número inteiro (ou seja, deve ser impressa apenas a parte inteira). A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
-8 3
Teste 2
26 75
```

(esta saída corresponde ao exemplo de entrada acima)

REFERÊNCIAS BIBLIOGRÁFICAS

AYROSO, Vanessa Dalaú. **Avaliação de métodos de teste de software em desenvolvimento de sistemas em banco de dados.** 1998. 86 f. Monografia (Curso de Pós Graduação em Nível de Especialização em Tecnologia de Desenvolvimento de Sistemas), Universidade Regional de Blumenau, Blumenau.

BORLAND, Corporation Inc. **Borland.** Disponível em: <http://www.borland.com>. Acesso em 07 Abr. 2002.

BÓSIO, Rubens. **Análise comparativa entre as especificações de objetos distribuídos DCOM e CORBA utilizando o ambiente de desenvolvimento Delphi.** 2000. 55 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

BROSE, Gerald, VOGEL Andreas, DUDDY, Keith. **Java programing with CORBA** – advanced techniques for building distributed applications. New York: John Wiley, 3ed, 2001.

CAPELETTO, Johni Jéferson. **Comunicação entre objetos distribuídos utilizando a tecnologia CORBA (Common Object Request Broker Architeture).** 1999. 60 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

COLLING, Suzete Teresinha. **Utilização da tecnologia Activex Data Objects (ADO) em um sistema com objetos distribuídos.** 2000. 77 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

FURLAN, José Davi. **Modelagem de objetos através da UML.** São Paulo: Makron Books, 1998.

HETZEL, William. **Guia completo ao teste de software.** Rio de Janeiro: Campus, 1987.

INTHURN, Cândida. **Qualidade & teste de software.** Florianópolis: Visual Books, 2001.

JACOBSEN, Douglas Thomas. **Sistema de apoio ao coordenador do simulador de empresas virtual utilizando a tecnologia CORBA.** 2000. 55 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

LEÃO, Marcelo. **Curso completo Delphi 6 & Kylix.** Rio de Janeiro: Axcel Books, 2001.

MONTEZ, Carlos. **Um modelo de programação para aplicações em tempo real em sistemas abertos.** 1997. Monografia (Exame de Qualificação de Doutorado), Universidade Federal de Santa Catarina, Florianópolis.

OSS, Fabiano. **Protótipo de software para geração de sistemas distribuídos utilizando Design Patterns.** 2001. 102 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

PAULA FILHO, Wilson de Pádua. **Engenharia de software.** Fundamentos, métodos e padrões. Rio de Janeiro: LTC, 2001.

PRESSMAN, Roger S. **Engenharia de software.** São Paulo: Makron Books, 1995.

ROCHA, Ana Regina Cavalcanti da; MALDONADO, José Carlos; WEBER, Kival Chaves. **Qualidade de Software – Teoria e Prática.** São Paulo: Prentice Hall, 2001.

SANDERINK, Gerard C.H. **Tendências no aprendizado com o auxílio do computador:** das aulas para as virtuais salas de aula. Disponível em < http://www.tpo.com.br/xijabro/sanderink-traducao.doc >. Acesso em 15 Mai. 2002.

SANTOS, Joel dos. **Fácil acesso a objetos distribuídos.** 1998. Monografia (Curso de Pós Graduação em Nível de Especialização em Tecnologia de Desenvolvimento de Sistemas), Universidade Regional de Blumenau, Blumenau.

SONNINO, Bruno. **Kylix Delphi para Linux:** guia prático de programação. São Paulo: Makron Books, 2001.

VASCONCELOS, V. **Corba O quê?** Dissettação de Mestrado. UFPA: Belém-Pa, 1998.