UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

AMBIENTE NA WEB PARA EXECUÇÃO E DEPURAÇÃO DE PROGRAMAS COM SINTAXE JAVA

SILVANO LOHN

BLUMENAU
2008

2008/1-35

SILVANO LOHN

AMBIENTE NA WEB PARA EXECUÇÃO E DEPURAÇÃO DE PROGRAMAS COM SINTAXE JAVA

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. Adilson Vahldick - Orientador

BLUMENAU
2008

2008/1-35

AMBIENTE NA WEB PARA EXECUÇÃO E DEPURAÇÃO DE PROGRAMAS COM SINTAXE JAVA

Por

SILVANO LOHN

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

	_____
Presidente:	Prof. Adilson Vahldick – Orientador, FURB
	_____
Membro:	Prof. Everaldo Artur Grahl, M.Eng. – FURB
	_____
Membro:	Prof. Dalton Solano dos Reis, M.Sc. – FURB

Blumenau, 10 de julho de 2008

Dedico este trabalho aos meus pais, meus amigos, meu orientador e especialmente para aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

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

Aos meus amigos, pelo apoio e incentivo.

Ao meu orientador, Adilson Vahldick, por ter acreditado na conclusão deste trabalho.

Para realizar grandes conquistas, devemos não apenas agir, mas também sonhar; não apenas planejar, mas também acreditar.

Anatole France

RESUMO

Este trabalho apresenta o desenvolvimento de um ambiente web para auxiliar o aprendizado de programação de algoritmos. O ambiente possui uma área administrativa, onde o professor cria os exercícios que serão resolvidos pelos alunos. O aluno desenvolve no ambiente seus algoritmos utilizando a sintaxe Java, com recursos para compilar, depurar e executar os seus programas. O ambiente também fornece uma área para um monitor realizar a correção dos exercícios, enviando dicas e comentários ao aluno caso a solução esteja incorreta. O ambiente foi desenvolvido com tecnologias Java como JSP e *Servlets*. Foi utilizada a biblioteca de *tags* JSTL para facilitar a montagem das páginas. Além disso, foi de suma importância o uso da tecnologia AJAX para oferecer no ambiente web a usabilidade parecida com ambientes *desktop*.

Palavras-chave: Ambiente web. Aprendizado de algoritmos. Java.

ABSTRACT

This paper presents the development of a web environment to help the learning of programming algorithms. The environment has an administrative area, where the teacher creates the exercises that will be solved by the students. The student develops in the environment their algorithms using the Java syntax, with resources to compile, debug and run their programmes. The environment also provides an area for a display hold the correction the exercises, sending tips and comments to the student when the solution is incorrect. The environment was developed with technologies Java and JSP Servlets. It used the library of JSTL tags to facilitate the assembly of pages. Moreover, was of paramount importance the use of Ajax in the web environment to provide the usability similar to desktop environments.

Key-words: Environment web. Learning of algorithms. Java.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

AJAX – *Asynchronous Javascript And XML*

CSS – *Cascading Style Sheets*

DOM – *Document Object Model*

DWR – *Direct Web Remoting*

HTML – *HyperText Markup Language*

JLS – *Java Language Specification*

JSP – *Java Server Pages*

JSTL – *JSP Standard Tag Library*

JVM – *Java Virtual Machine*

JVMS – *Java Virtual Machine Specification*

LCI – Laboratório de Computação e Informática

MER – Modelo Entidade Relacionamento

MVC – *Model View Controller*

RF – Requisito Funcional

RNF – Requisito Não-Funcional

SQL – *Structured Query Language*

WAE – *Web application Extension*

XML – *EXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Os algoritmos fazem parte do dia-a-dia das pessoas: instruções para o uso de medicamentos, ou indicações de como montar um aparelho eletrodoméstico são exemplos de algoritmos. Um algoritmo pode ser visto como uma seqüência de ações executáveis para a obtenção de uma solução para um determinado problema. Segundo Ziviani (2004, p. 1), um algoritmo corresponde a uma descrição de um padrão de comportamento, expresso em termos de um conjunto de ações. Ziviani (2004, p. 1) relata que o estudo dos algoritmos tem um papel decisivo para o desenvolvimento de boas aplicações e na busca da resolução de problemas.

Com a programação[1] de algoritmos cada vez maiores e mais complexos é comum o surgimento de erros de lógica. Uma maneira de encontrar e acabar com os erros é o uso de uma ferramenta chamada de depurador. Severo (2005, p. 151) define depurador como sendo uma ferramenta que permite a verificação do fluxo de execução de um programa, linha a linha, à medida que o programa é executado, na busca de possíveis erros na lógica de construção de um algoritmo. Como relatado, o depurador é uma ferramenta de grande ajuda, que possibilita facilitar e tornar mais rápido o trabalho do desenvolvedor que precisa tanto eliminar os erros do seu código fonte como também testar a sua lógica de programação.

A atividade de depuração ocorre no processo de desenvolvimento de um software em três momentos distintos: durante a codificação, depois dos testes e durante a sua manutenção. Chaim, Jino e Maldonado (2002) relatam que durante a codificação, a depuração é uma ferramenta complementar à programação. O programador codifica parte da especificação e prepara um teste para verificar o novo código. Se o resultado do teste está incorreto o novo código deve ser depurado.

A depuração que ocorre depois da atividade de teste possui características diferentes. O objeto da depuração neste momento é o software obtido depois de completada a fase de implementação e que, supostamente, já possui todas as funções estabelecidas na especificação. Além disso, a depuração recebe como entrada não somente o código e a

[1] Horstmann (2006, p. 20) relata que o programa "diz" ao computador a seqüência de etapas necessárias para realizar uma tarefa. O ato de projetar e implementar esses programas denomina-se programação.

especificação, mas também os resultados da atividade de teste (CHAIM; JINO; MALDONADO, 2002).

Chaim, Jino e Maldonado (2002) informam que durante a manutenção, novamente há a necessidade de depuração do software onde o conjunto de casos de teste disponível para depuração é definido pelo conjunto originalmente desenvolvido durante o teste.

Segundo Almeida (2003, p. 327), com a evolução da Internet e sua disseminação em todo o mundo, novas tecnologias e ferramentas de desenvolvimento foram criadas, permitindo com isso uma maior integração entre o usuário e os recursos disponíveis nos ambientes web. Entre estes recursos pode-se citar o de acesso, onde o usuário deverá possuir como requisito apenas um computador com acesso a Internet. Outro recurso muito importante é o de compartilhamento, onde vários usuários podem estar acessando o sistema ao mesmo tempo, trocando e disponibilizando informações.

As disciplinas de algoritmos e programação de computadores nos primeiros semestres apresentam alto grau de reprovação. Para o sucesso no aprendizado de algoritmos, exige-se um forte comprometimento por parte dos alunos em praticar os exercícios fornecidos pelo professor, assim como outras tarefas extras. Porém, exige-se também do professor uma atenção personalizada ao aluno para respeitar seu ritmo individual de aprendizado, seja corrigindo vários exercícios como complementando com seus comentários.

Seguindo o raciocínio apresentado, este trabalho relata o desenvolvimento de um ambiente na web onde é possível simular a programação estruturada de programas com sintaxe Java. O usuário desenvolve seu programa como está apresentado no Quadro 1. O ambiente incorpora o programa do usuário dentro do método principal de uma classe Java conforme apresentada no Quadro 2. A classe Java é abstraída do usuário, mas é utilizada pelo ambiente para compilar, executar ou depurar o programa.

```
int z = 0;
for (int x=0;x<10;x++){
   z = (z * z) + x;
}
```

Quadro 1 – Exemplo do programa desenvolvido pelo usuário

```
public class Cod1{
     public static void main(String args[]){
          int z = 0;
          for (int x=0;x<10;x++){
               z = (z * z) + x;
          }
     }
}
```

Quadro 2 – Exemplo do programa interpretado pelo ambiente

Mesmo com a linguagem Java sendo orientada a objetos, não é esse o foco do ambiente e sim a programação baseada em objetos[2]. O usuário não cria classes Java ou objetos, mas cria o seu programa de forma estruturada usando os objetos disponíveis na biblioteca `java.util` da própria plataforma Java. Também foi criada uma biblioteca de objetos para que o usuário possa realizar a entrada e a apresentação dos dados do seu programa.

Como o programa do usuário é tratado pelo ambiente como se fosse uma classe Java, foi realizada a integração com as ferramentas **java**, **javac** e **jdb** disponíveis na plataforma Java. As ferramentas citadas foram utilizadas para realizar a compilação, execução e depuração do programa do usuário. Também foi disponibilizado ao usuário recursos para que ele possa carregar, editar e salvar os seus programas.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um ambiente na web onde será possível criar, compilar e depurar algoritmos com sintaxe Java.

Os objetivos específicos do trabalho são:

a) disponibilizar um ambiente na web que facilite o aprendizado de algoritmos usando sintaxe Java;
b) desenvolver um conjunto de operações através de objetos e métodos acessíveis ao usuário, fazendo com que os algoritmos tenham recursos de entrada e

[2] Horstmann (2006, p. 48) define objetos sendo uma entidade que pode ser utilizada dentro dos programas, e geralmente utilizada para a manipulação e invocação dos métodos do próprio objeto.

apresentação dos dados no próprio ambiente web;

c) disponibilizar no ambiente a possibilidade da execução e depuração dos algoritmos em Java, apresentando os valores das variáveis contidas no algoritmo.

1.2 ESTRUTURA DO TRABALHO

A estrutura deste trabalho está dividida em capítulos que serão explicados a seguir.

O primeiro capítulo apresenta a contextualização, objetivo e justificativa para o desenvolvimento do trabalho.

O segundo capítulo apresenta a fundamentação teórica do trabalho, apresentação da biblioteca DWR, das ferramentas da plataforma Java, da linguagem JavaScript e por fim, a apresentação dos trabalhos correlatos.

O terceiro capítulo apresenta os requisitos da ferramenta, bem como sua especificação, implementação e resultados. São mostrados diagramas de casos de uso, diagrama de atividades, diagrama de classe, diagrama de navegabilidade e diagrama de seqüência. Também é apresentada a operacionalidade da ferramenta.

O quarto capítulo apresenta as conclusões, limitações e sugestões para futuros trabalhos.

2 FUNDAMENTAÇÃO TEÓRICA

A seguir são explanados os principais assuntos relacionados ao desenvolvimento do trabalho: aprendizado de algoritmos, DWR, ferramentas da plataforma Java, JavaScript e os trabalhos correlatos.

2.1 APRENDIZADO DE ALGORITMOS

A disciplina de programação é uma das disciplinas chaves na formação do profissional da computação. No entanto, por ela apresentar maior nível de dificuldade no processo de ensino/aprendizagem, a comunidade acadêmica tem se mostrado preocupada com o desempenho dos alunos nas disciplinas introdutórias de programação. Esta preocupação traduz-se em pesquisas na busca de alternativas que contribuam para resultados mais satisfatórios no processo de ensino/aprendizagem de computação (XAVIER, 2004).

Inúmeras causas são apontadas por pesquisadores para os elevados índices de reprovação nas disciplinas de algoritmos e programação como, hábitos de estudo centrados em memorização, falta de pré-requisitos em conteúdos correlatos – principalmente nos conteúdos de matemática – seqüência didática desmotivadora, dificuldades na compreensão do enunciado dos problemas, forte carga de conceitos abstratos, cultura instrucionista e prática pedagógica desmotivadora (XAVIER, 2004).

Em sua pesquisa, Pimentel (2002) destaca que diversos tipos de ferramentas e ambientes têm sido propostos com o objetivo de facilitar o aprendizado de lógica e linguagens de programação. Porém, aponta que não existem indicações de uma vasta utilização destas ferramentas.

2.2 DWR

Getahead (2004) relata que DWR é uma biblioteca Java de código aberto que permite o desenvolvimento de web sites com tecnologia AJAX.

Mozilla (2008) diz que AJAX não é uma tecnologia em si mesma, mas é um termo que define uma nova abordagem de uso conjunto de um número de tecnologias existentes, entre elas: HTML, CSS, JavaScript, DOM, XML. Quando estas tecnologias são combinadas no modelo AJAX, as aplicações web são capazes de fazer atualizações incrementais e rápidas na interface do utilizador sem ser necessário recarregar novamente toda a página. Isto faz com que a aplicação seja mais rápida às ações do utilizador.

Samir (2006, p. 20) afirma que a biblioteca DWR permite que no navegador do cliente as funções JavaScript embutidas nas páginas interajam com as funções Java no lado do servidor, construindo um fácil e transparente caminho para manipular as respostas com dados sem renderizar as páginas no cliente. O DWR consiste em duas partes principais, no lado do servidor, onde um *Java Servlet* fica em execução para receber requisições e enviar respostas ao navegador do cliente e no lado do cliente, onde executa um JavaScript que envia e recebe dados e faz dinamicamente a atualização dos dados da página.

Na seção 2.2.1 será apresentado como é feito o mapeamento e a configuração do DWR em um servidor Apache-Tomcat. Na seção 2.2.2 será apresentado um exemplo de uma requisição AJAX de um cliente web ao servidor utilizando DWR.

2.2.1 Mapeamento e configuração do DWR

Getahead (2004) ensina que para configurar uma aplicação a utilizar o DWR é necessário realizar dois passos. O primeiro passo é adicionar no arquivo WEB-INF/web.xml da aplicação o conteúdo do Quadro 3.

```
<servlet>
  <servlet-name>dwr-invoker</servlet-name>
  <display-name>DWR Servlet</display-name>
  <servlet-class>org.directwebremoting.servlet.DwrServlet</servlet-
class>
  <init-param>
     <param-name>debug</param-name>
     <param-value>true</param-value>
  </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>dwr-invoker</servlet-name>
  <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

Fonte: Getahead (2004).

Quadro 3 – Conteúdo adicionado ao arquivo WEB-INF/web.xml

Getahead (2004) diz que o segundo passo é criar um arquivo chamado dwr.xml dentro da pasta WEB-INF da aplicação com o conteúdo do Quadro 4. Esse arquivo fará o mapeamento das chamadas do cliente web para a classe Java criada pelo desenvolvedor e configurada no arquivo dwr.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dwr PUBLIC
    "-//GetAhead Limited//DTD Direct Web Remoting 2.0//EN"
    "http://getahead.org/dwr/dwr20.dtd">

<dwr>
  <allow>
    <create creator="new" javascript="Demo">
      <param name="class" value="org.getahead.dwrdemo.simpletext.Demo"/>
    </create>
  </allow>
</dwr>
```

Fonte: Getahead (2004).

Quadro 4 – Arquivo dwr.xml

2.2.2 Exemplo DWR

O exemplo a seguir foi realizado levando em consideração a configuração do DWR apresentado na seção anterior.

No exemplo Getahead (2004) relata que ao clicar sobre o botão `Send` na Quadro 5, o navegador solicita o evento `onclick` que executa a função JavaScript `update()` do Quadro 6.

Essa função realiza uma requisição AJAX ao servidor executando o método `sayHello` da classe `Demo` como apresentada no Quadro 7. O retorno da função `sayHello` irá retornar do servidor para o navegador do cliente como parâmetro da função `function(data)` apresentada no Quadro 6 que irá alterar o valor do componente HTML `span`.

```
<p>
  Name:
  <input type="text" id="demoName"/>
  <input value="Send" type="button" onclick="update()"/>
  <br/>
  Reply: <span id="demoReply"></span>
</p>
```

Fonte: Getahead (2004).

Quadro 5 – Código HTML no cliente

```
function update() {
  var name = dwr.util.getValue("demoName");
  Demo.sayHello(name, function(data) {
    dwr.util.setValue("demoReply", data);
  });
}
```

Fonte: Getahead (2004).

Quadro 6 – Código JavaScript no cliente

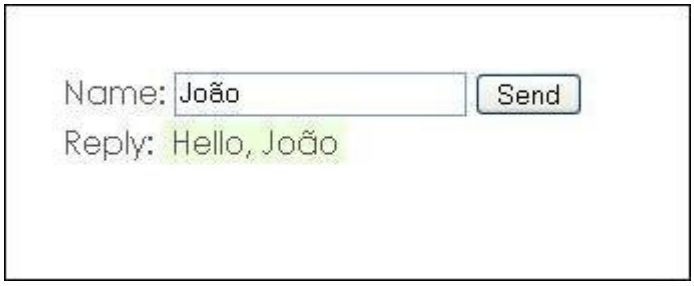
```
package org.getahead.dwrdemo.simpletext;

public class Demo {
    public String sayHello(String name) {
        return "Hello, " + name;
    }
}
```

Fonte: Getahead (2004).

Quadro 7 – Classe Java no servidor configurada no dwr.xml

Getahead (2004) informa que na Figura 1 é apresentado o término da execução, o valor "João" foi enviado ao servidor web, foi processado e teve como retorno o valor "Hello, João". Com o uso do DWR, não foi necessário recarregar toda a página, somente o que era pretendido, com isso trazendo maior velocidade para a atualização dos dados no navegador do usuário.



Fonte: Getahead (2004).

Figura 1 – Execução do exemplo

2.3 FERRAMENTAS DA PLATAFORMA JAVA

Nesta seção são apresentadas as ferramentas que acompanham a plataforma de desenvolvimento Java. Na seção 2.3.1 será apresentada a ferramenta responsável em executar, na seção 2.3.2 a ferramenta para compilar e na seção 2.3.3 a ferramenta para depurar o programa do usuário. Todas as ferramentas citadas são de uso livre e de código fonte aberto.

2.3.1 Execução

JVM é o acrônimo para Java Virtual Machine, que é o alicerce da linguagem de programação Java. JVM é uma máquina computacional abstrata formada por um interpretador de bytecodes Java e do próprio processador do sistema computacional em que está rodando. Tendo por base um arquivo com bytecodes, a JVM usa o interpretador Java chamado de **java** (SUN MICROSYSTEMS, 2006).

Sun Microsystems (2006) afirma que a ferramenta **java** tem como objetivo principal carregar um arquivo bytecode especificado pelo usuário, interpretando o código e gerando as instruções para serem executadas pelo processador.

Como nas duas ferramentas citadas anteriormente essa também é executada através de linhas de comando. O Quadro 8 mostra um exemplo de uma linha de comando utilizada para executar um código fonte Java já compilado.

```
C:\> java –cp C:\ Exemplo
```

Quadro 8 – Exemplo de uma linha de comando para execução de um programa Java

2.3.2 Compilação

Sun Microsystems (2006) relata que **javac** é considerado o compilador oficial da linguagem Java. Ele é um programa executável que possui seu funcionamento através de linhas de comando.

O compilador **javac** aceita como entrada o código fonte Java definido pela JLS[3]. Depois de compilar ele gera um arquivo bytecode. Esse arquivo possui sua estrutura definida pela JVMS[4]. Depois de gerado, esse arquivo possui um conjunto de instruções que pode ser interpretada e executada pela JVM (SUN MICROSYSTEMS, 2006).

Na linha de comando para a execução do compilador **javac** é possível incluir um conjunto de opções para executar determinadas ações durante a compilação do código fonte. Dentre essas opções será citada apenas duas, que serão de uso obrigatório no desenvolvimento desse trabalho.

O Quadro 9 apresenta o exemplo de uma linha de comando utilizada para compilar o código fonte do usuário, essa mesma linha possui os dois comandos citados anteriormente que serão explicados a seguir.

```
C:\> javac –g –cp C:\ Exemplo.java
```

Quadro 9 – Exemplo de uma linha de comando para execução do compilador **javac**

Na linha de comando do Quadro 8 tem-se a chamada do compilador através da palavra "javac". Sun Microsystems (2006) informa que a opção de compilação "–g" é necessária para gerar todas as informações que serão utilizadas para a depuração do código posteriormente. Já a opção "–cp" Sun Microsystems (2006) diz que deve ser utilizada para especificar o caminho em que o compilador irá buscar os arquivos de código fonte do usuário também como suas bibliotecas.

[3] Sun Microsystems (2006) define JLS sendo um documento que especifica a linguagem de programação Java.
[4] Sun Microsystems (2006) define JVMS sendo um documento que especifica a construção e o funcionamento da maquina virtual Java.

2.3.3 Depuração

Segundo Sun Microsystems (2006), **jdb** é um depurador de códigos fonte Java e sua execução é realizada através de linhas de comando. Para efetuar a depuração é necessário que as classes Java do usuário já tenham sido compiladas e os arquivos bytecodes gerados.

Seu principal recurso é a depuração de códigos fonte passo-a-passo. Com isso se torna mais fácil a localização de erros de lógica de programação, pois ele informa a linha de comando em execução e os valores das variáveis em uso. O **jdb** destaca-se em relação a muitos outros depuradores por efetuar a depuração tanto de modo local como remotamente (SUN MICROSYSTEMS, 2006).

O Quadro 10 apresenta um programa Java que será depurado pelo **jdb** no exemplo a seguir.

```
public class Exemplo{
   public static void main(String args[]){
      int x = 0;
      x++;
      x++;
      System.out.println("Valox x: "+x);
   }
}
```

Quadro 10 – Exemplo do programa Java que será depurado

O Quadro 11 apresenta um exemplo de uma depuração de um código fonte Java como também os principais comandos utilizados para realizar a mesma.

```
C:\>jdb
Initializing jdb ...
> stop at Exemplo:0
> run
main[1] step
>
Step completed: "thread=main", Exemplo.main(), line=3 bci=0
3               int x = 0;

main[1] step
>
Step completed: "thread=main", Exemplo.main(), line=4 bci=2
4               x++;

main[1] locals
Method arguments:
args = instance of java.lang.String[0] (id=331)
Local variables:
x = 0
main[1] step
>
Step completed: "thread=main", Exemplo.main(), line=5 bci=5
5               x++;

main[1] locals
Method arguments:
args = instance of java.lang.String[0] (id=331)
Local variables:
x = 1
main[1] step
>
Step completed: "thread=main", Exemplo.main(), line=6 bci=8
6               System.out.println("Valox x: "+x);

main[1] locals
Method arguments:
args = instance of java.lang.String[0] (id=331)
Local variables:
x = 2
main[1] step
> Valox x: 2

Step completed: "thread=main", Exemplo.main(), line=8 bci=33
8       }

main[1] locals
Method arguments:
args = instance of java.lang.String[0] (id=331)
Local variables:
x = 2
main[1] step
>
The application exited
```

Quadro 11 – Exemplo de uma depuração de um programa Java

O **jdb** é inicializado via linha de comando e sua execução é realizada através do comando “jdb”. O comando “stop at Exemplo:0” informa ao depurador em qual classe e linha do código fonte a depuração deve iniciar, nesse caso a depuração inicia na classe Exemplo na linha zero.

O comando “run” inicia a depuração do programa, e através do comando “step” que a depuração é executada. A cada comando “step” o depurador executa a próxima linha do programa. Esta execução devolve o número e o conteúdo da linha que está sendo executada pelo depurador, sendo neste caso “3 int x = 0;” ou seja a linha três e o comando “int x = 0;” foi executado.

O comando “locals” retorna ao usuário o valor de todas as variáveis utilizadas durante a depuração do programa. Depois que todas as linhas do programa foram executadas, o depurador retorna ao usuário a mensagem “The application exited” informando que a depuração do programa está finalizada.

2.4 JAVASCRIPT

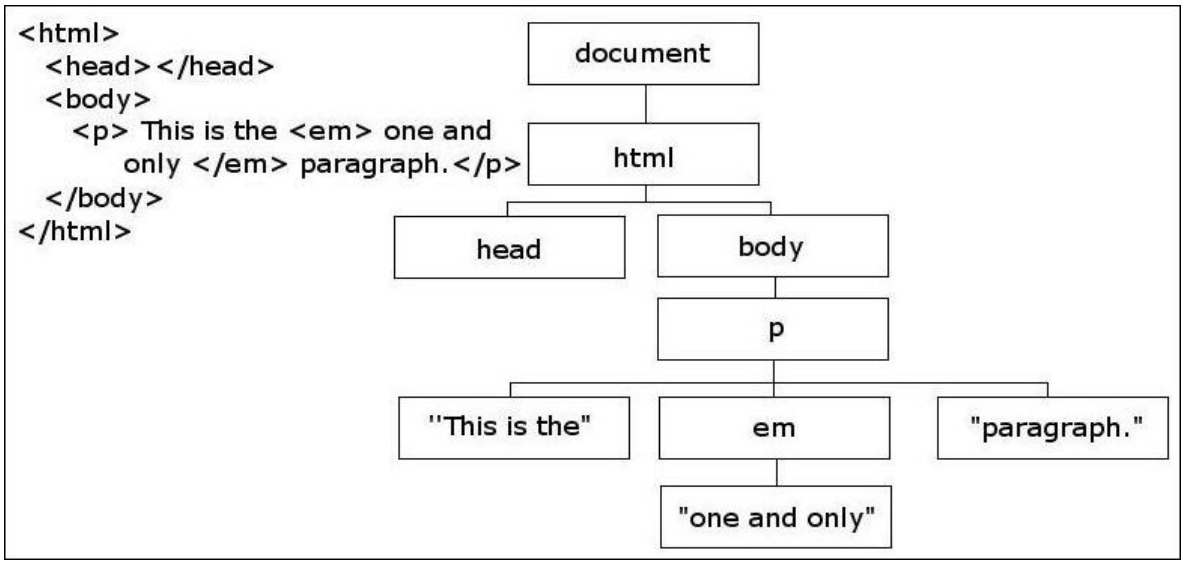
McComb (1997, p. 5) diz que o JavaScript foi desenvolvido pela Netscape e é uma linguagem *script* para o HTML, que são pequenos programas que interagem com o *browser* e o conteúdo HTML de uma página. Trabalhando em conjunto com características relacionadas ao *browser* pode ajudar a transformar uma página de conteúdo estático em um conteúdo envolvente, interativo e trazendo ótimas experiências ao usuário. Para isso se tornar possível o JavaScript utiliza o DOM para acessar e manipular os elementos de um documento HTML de uma pagina web.

Goodman e Morrison (2004, p. 177) dizem que o DOM não é uma linguagem de programação. Ele foi desenvolvido para ser independente de qualquer linguagem de programação e para acabar com as incompatibilidades dos modelos de objetos dos navegadores. Sem ele a linguagem JavaScript não teria nenhuma noção do modelo dos documentos de uma página web.

Goodman e Morrison (2004, p. 177) ainda relatam que o DOM é uma interface de

programação para documentos HTML. Ele provê uma representação estruturada do documento representado por nós e objetos que têm propriedades e métodos.

Na Figura 2 é mostrado um exemplo de uma transformação de um código HTML em uma estrutura DOM, composta dos seus respectivos nós e objetos.



Fonte: Goodman e Morrison (2004, p. 41)

Figura 2 – Exemplo de uma estrutura DOM

No Quadro 12 é apresentado um exemplo de código JavaScript utilizando DOM. A função JavaScript “upperMe()” através do método DOM “document.getElementById” acessa e altera os elementos “input”e “output” do documento HTML.

```
<html>
  <head>
    <title>Text Object Value</title>
      <script type=”text/javascript”>
        <!--
          function upperMe() {
            document.getElementById(“output”).value =
            document.getElementById(“input”).value.toUpperCase();
          }
        // -->
      </script>
    </head>
  <body>
    Enter lowercase letters for conversion to uppercase:<br>
    <form name=”converter”>
      <input type=”text” name=”input” id=”input”
        value=”sample” onchange=”upperMe()” /><br />
      <input type=”text” name=”output” id=”output” value=”” />
    </form>
  </body>
</html>
```

Fonte: Goodman e Morrison (2004, p. 41)

Quadro 12 – Exemplo programa JavaScript utilizando DOM

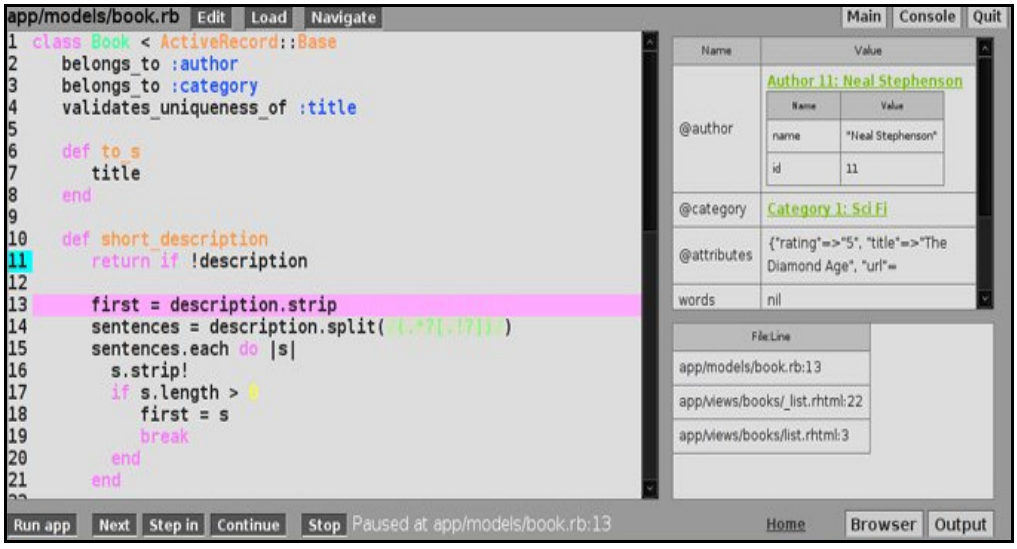
2.5 TRABALHOS CORRELATOS

Nesta seção são apresentadas duas ferramentas que desempenham um papel semelhante ao trabalho proposto. Na seção 2.5.1 será apresentado o ambiente web de desenvolvimento Gyre. Na seção 2.5.2 será apresentada um ambiente de desenvolvimento web chamado CodeIDE.

2.5.1 Gyre

Conforme Wiggins (2007), Gyre é um projeto de código aberto que no momento ainda se encontra em fase de construção. Ele é um depurador visual web para o desenvolvimento de aplicações que utilizam o framework Rails, sendo totalmente desenvolvido utilizando a linguagem Ruby.

Sua principal função é a depuração do código fonte na busca e eliminação de erros, sendo que para isso é possível visualizar as variáveis em execução e a pilha de chamadas de métodos. Possui um console para entrada de comandos específico da linguagem, como também um editor simples para o desenvolvimento dos códigos fonte (WIGGINS, 2007). A tela principal do projeto Gyre está apresentada na Figura 3.



Fonte: Gyre (2007).

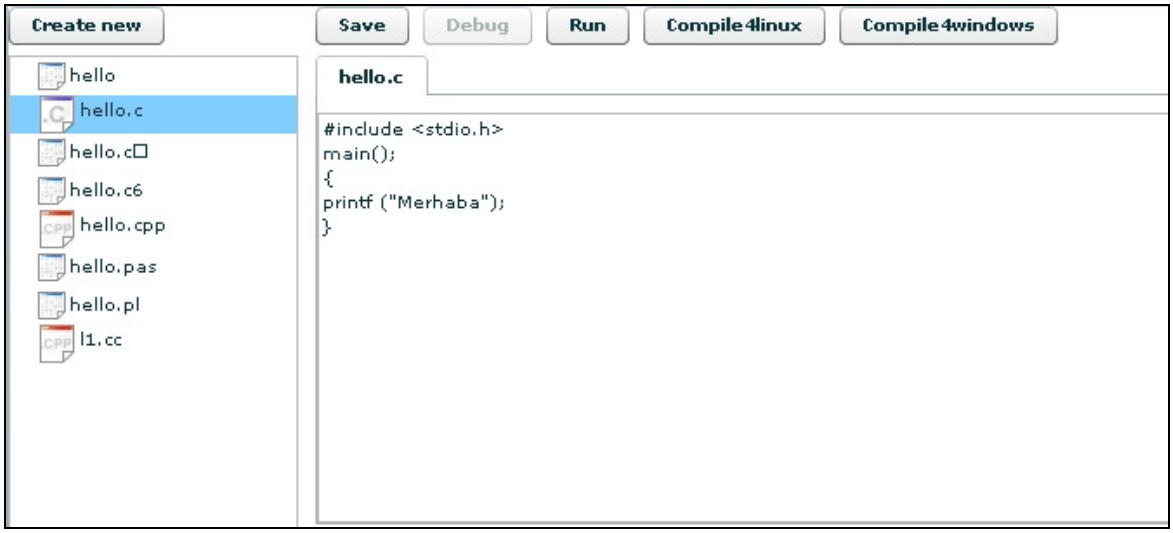
Figura 3 – Interface do ambiente Gyre

2.5.2 CodeIDE

Segundo Siteheart (2006), o CodeIDE é um ambiente de desenvolvimento via web, que fornece a possibilidade de rodar várias linguagens de programação, como o Basic, Pascal, C++, Perl, JavaScript, HTML, entre outras.

O ambiente também possui uma interface simples e amigável, um chat on-line onde os desenvolvedores podem trocar idéias, um gerenciador de projetos que possui recursos para o usuário salvar seus arquivos on-line. Isso significa que o usuário pode manter uma coleção de projetos no próprio ambiente. Além disso, o usuário pode contribuir na comunidade com algumas implementações (SITEHEART, 2006).

Siteheart (2006) informa que o CodeIDE é um ambiente de grande ajuda principalmente para pessoas que precisam fazer programas curtos de vários lugares diferentes. A tela principal do CodeIDE está apresentada na Figura 4.



Fonte: Siteheart (2006).

Figura 4 – Interface do ambiente CodeIDE

3 DESENVOLVIMENTO

Neste capítulo são apresentados os requisitos, a especificação e a implementação do trabalho. Também são mostrados os cenários, os diagramas de atividades, diagramas de classe, diagrama de navegabilidade, diagrama de seqüência e o modelo de entidades e relacionamento. Para finalizar é apresentada uma validação do trabalho utilizando códigos fonte de exemplos, também como os resultados obtidos e uma comparação com os trabalhos correlatos.

3.1 REQUISITOS

Analisando a proposta do ambiente web, têm-se os seguintes Requisitos Funcionais (RF) e Não Funcionais (RNF).

a) permitir que o programa seja carregado, editado e salvo (RF);
b) fazer a execução do programa (RF);
c) fazer a depuração do programa de modo passo-a-passo informando a linha e o valor das variáveis em execução (RF);
d) fazer a compilação do programa informando quando houver os erros de compilação (RF);
e) fazer um ambiente administrativo onde seja possível criar e corrigir os exercícios (RF);
f) permitir que os usuários se cadastrem no ambiente (RF);
g) ser implementado utilizando tecnologia JEE e AJAX (RNF);
h) ser compatível com os navegadores Internet Explorer e Firefox (RNF).

3.2 ESPECIFICAÇÃO

Para a especificação da ferramenta foram feitos os diagramas pertencentes à modelagem UML e MER, tendo como ferramentas de auxílio o Enterprise Architect e DBDesigner.

3.2.1 Diagrama de casos de uso

Para Bezerra (2002, p. 46), “um caso de uso é a especificação de uma seqüência de iterações entre um sistema e os agentes externos que utilizam esse sistema”. Segundo Bezerra (2002, p. 46), “Um caso de uso deve definir o uso de uma parte da funcionalidade de um sistema, sem revelar a estrutura e o comportamento internos desse sistema”. Os casos de usos devem ser escritos na forma narrativa das iterações que acontecem entre os elementos externos e o sistema.

Para Bezerra (2002, p. 49), os casos de usos podem ser definidos de várias maneiras. Uma delas pode ser através dos cenários. “Um cenário é a descrição de uma das maneiras pelas quais um caso de uso pode ser realizado”. Um cenário pode ser chamado de instância de um caso de uso. Conforme Bezerra (2002, p. 50), uma coleção de cenários para um caso de uso pode ser útil na fase de testes para verificar erros na implementação do sistema, esclarecimento e no entendimento dos casos de uso dos quais eles são instanciados.

A Figura 5 apresenta o diagrama de casos de uso do sistema.

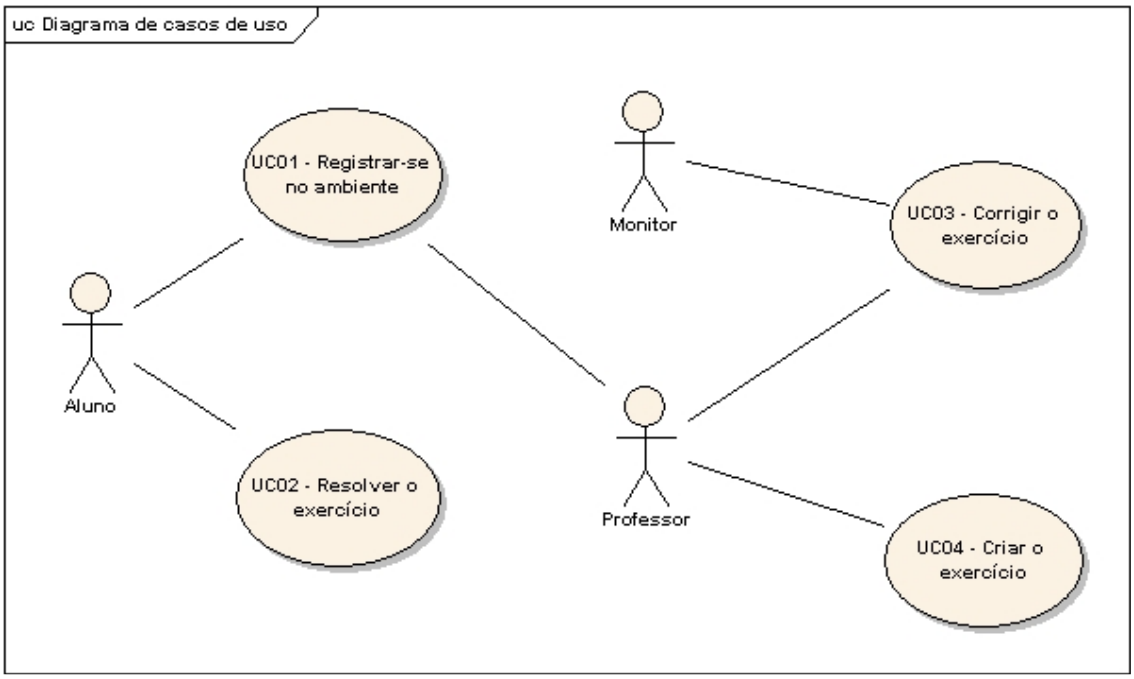


Figura 5 – Diagrama de casos de uso

No Quadro 13 é apresentado o cenário para o caso de uso registrar-se no ambiente.

UC01 – Registrar-se no ambiente

1. Descrição: O aluno deseja registrar-se no ambiente.

1.1 Atores: Aluno (Principal).
Professor (Secundário).
Sistema (Secundário).

1.2 Precondições: Nenhuma.

1.3 Fluxo de eventos

1.3.1 Fluxo Básico – Registrar-se no ambiente.

a. O aluno clica em Registrar.
b. O sistema abre a tela de cadastro.
c. O aluno preenche os campos nome, email, código de aluno e senha.
d. O aluno clica no botão Registrar.
e. O sistema registra o aluno no banco de dados.
f. O sistema envia um email para o aluno com os dados do registro.
g. O professor verifica no ambiente se existe algum registro de aluno para ser confirmado.
h. O professor confirma o registro do aluno.
i. O sistema envia um email para o aluno informando que o registro está concluído.

1.4 Pós-condição: O aluno está registrado no sistema.

Quadro 13 – Cenário do caso de uso registrar-se no ambiente

No Quadro 14 é apresentado o cenário para o caso de uso resolver o exercício.

UC02 – Resolver o exercício

1. Descrição: O aluno deseja resolver o exercício.

1.1 Atores: Aluno (Principal).
Sistema (Secundário).

1.2 Precondições: O aluno deve estar autenticado no ambiente.
Deve existir pelo menos um exercício para ser resolvido.

1.3 Fluxo de eventos

1.3.1 Fluxo Básico – Resolver o exercício.

a. O aluno clica no link Exercícios.
b. O sistema abre a tela com os exercícios para serem resolvidos.
c. O aluno escolhe um exercício para resolver clicando no link Abrir exercício.
d. O aluno desenvolve o programa para resolver o exercício.
e. O aluno finaliza o exercício clicando no link Finalizar.
f. O sistema altera o status do exercício para finalizado.

1.3.2 Fluxos Alternativos

1.3.2.1 – Compilar o programa.

a. No item d do fluxo básico o aluno clica no link Compilar.
b. O sistema compila o programa do aluno.
c. O sistema mostra as mensagens de erro de compilação ao aluno caso existir.

1.3.2.2 – Executar o programa.

a. No item d do fluxo básico o aluno clica no link Executar.
b. O sistema abre a tela de execução.
c. O sistema inicia a execução do programa.
d. O sistema apresenta a saída de dados do programa ao aluno caso existir.
e. O aluno realiza a entrada de dados no programa caso seja necessário.
f. O sistema termina a execução do programa.

1.3.2.3 – Depurar o programa.

a. No item d do fluxo básico o aluno clica no link Depurar.
b. O sistema abre a tela de depuração.
c. O aluno inicia a depuração do programa.
d. O aluno acompanha a seqüência da depuração clicando no botão Passo-a-passo.
e. O sistema apresenta a saída de dados do programa ao aluno caso existir.
f. O sistema apresenta ao aluno os valores das variáveis em uso pelo programa.
g. O aluno realiza a entrada de dados no programa caso seja necessário.
h. O aluno altera os valores das variáveis do programa caso seja necessário.
i. O aluno finaliza a depuração do programa.

1.3.2.4 – Salvar o programa.

a. No item d do fluxo básico o aluno clica no link Salvar.
b. O sistema salva o programa do aluno no banco de dados.

1.4 Pós-condição: O exercício do aluno terá seu estado alterado para finalizado.

Quadro 14 – Cenário do caso de resolver o exercício

No Quadro 15 é apresentado o cenário para o caso de uso corrigir o exercício.

UC03 – Corrigir o exercício

1. Descrição: O monitor ou professor desejam corrigir o exercício do aluno. Caso o exercício do aluno esteja incorreto, o monitor ou professor podem escrever um comentário ao aluno descrevendo eventuais erros do programa ou dando algumas dicas.

1.1 Atores: Monitor (Principal).
Professor (Principal).
Sistema (Secundário).

1.2 Precondições: O monitor ou professor deve estar autenticado no ambiente.
Deve existir pelo menos um exercício para ser corrigido.

1.3 Fluxo de eventos

1.3.1 Fluxo Básico – Corrigir o exercício.

a. O monitor ou professor clicam no link Exercícios.
b. O sistema abre uma janela contendo os exercícios de todos os alunos.
c. O monitor ou professor escolhem um exercício que possua o estado de finalizado para ser corrigido.
b. O monitor ou professor compilam e executam o programa do aluno para testá-lo.

1.3.2 Fluxos Alternativos

1.3.2.1 – Enviar Comentário.

a. No item b do fluxo básico caso o exercício não esteja correto, o monitor ou professor clicam no link Comentário.
b. O sistema abre a janela Escrever Comentário.
c. O monitor ou professor preenchem o campo de comentário.
d. O monitor ou professor clicam no botão Enviar Comentário.
e. O sistema grava o comentário para o exercício do aluno.
f. O sistema altera o estado do exercício para não finalizado.

1.4 Pós-condição: O exercício muda para o estado de corrigido.
Foi adicionado comentário no exercício do aluno.

Quadro 15 – Cenário do caso de uso corrigir o exercício

No Quadro 16 é apresentado o cenário para o caso de uso criar o exercício.

UC04 – Criar o exercício

1. Descrição: O professor deseja criar um exercício.
1.1 Atores: Professor (Principal).
Sistema (Secundário).

1.2 Precondições: O professor deve estar autenticado no ambiente.

1.3 Fluxo de eventos
1.3.1 Fluxo Básico – Criar o exercício.
a. O professor abre o ambiente administrativo.
b. O professor clica no link Exercícios.
c. O sistema abre a tela com todos os exercícios cadastrados.
d. O professor clica no link Criar Exercício.
e. O sistema abre a tela para o cadastro do exercício.
f. O professor informa o título e a descrição do exercício.
g. O professor clica no botão Criar Exercício.
h. O sistema cadastra o exercício no banco de dados.
i. O sistema atribui o exercício criado a todos os alunos do sistema.

1.4 Pós-condição: Um exercício foi criado.

Quadro 16 – Cenário do caso de uso criar o exercício

3.2.2 Diagrama de atividades

Segundo Bezerra (2002, p. 228), o diagrama de atividades é considerado um tipo especial de diagrama de estados, sendo orientado pelo fluxo de controle. Para apresentar o fluxo do processo é usado o diagrama de atividades.

A Figura 6 apresenta o diagrama de atividades onde o usuário utiliza o ambiente.

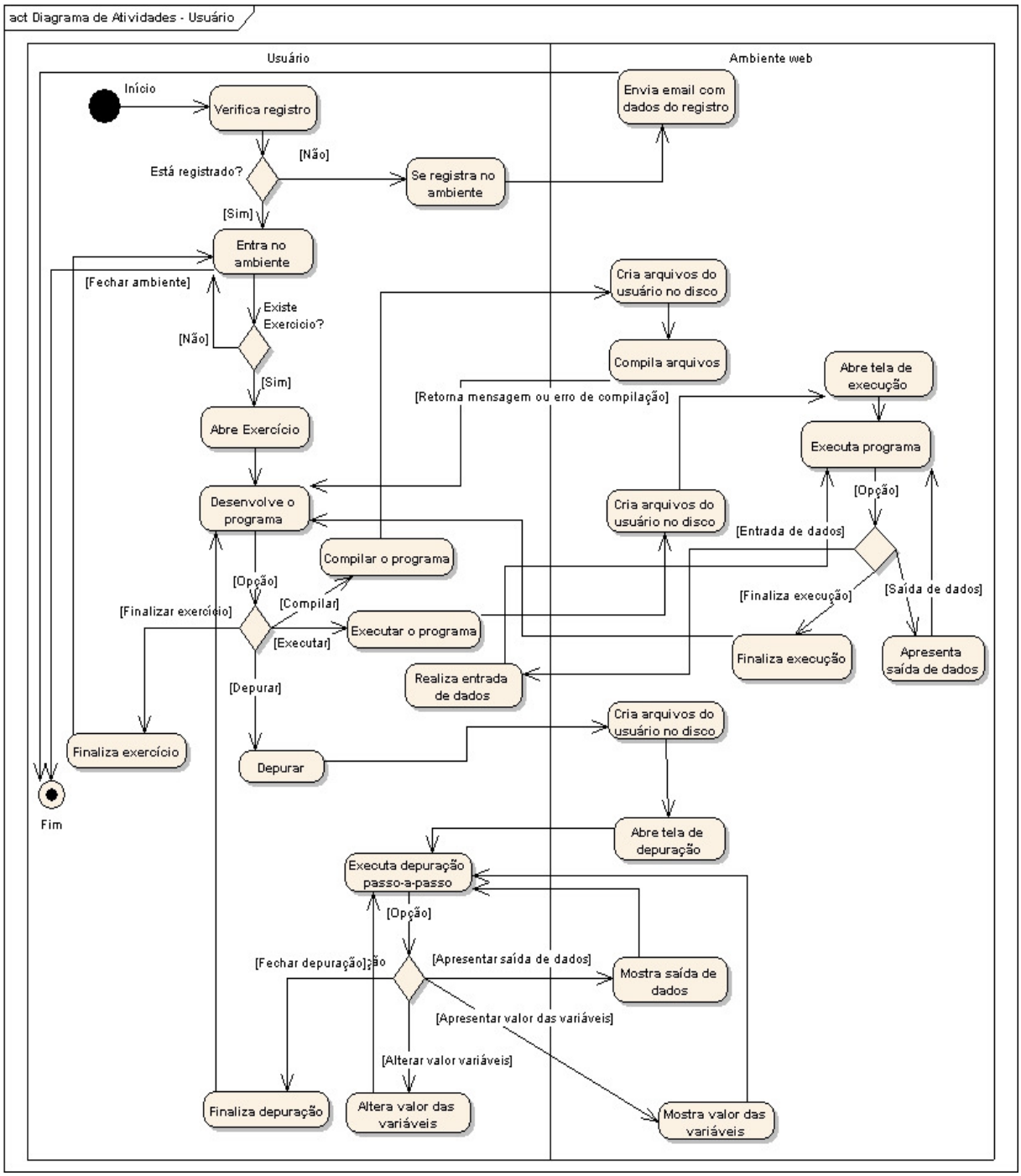


Figura 6 – Diagrama de atividades do usuário.

Como apresentado na Figura 6, o usuário é obrigado a se registrar no sistema para utilizar o ambiente web. No registro o usuário preenche um formulário contendo os campos nome, email, código de aluno e senha. Depois de efetuar o registro o sistema envia um email para o usuário contendo os dados de registro.

Agora o usuário pode entrar no ambiente web. Quando ele entra pode efetuar duas

operações: fechar o ambiente web ou então verificar se existe algum exercício para ser resolvido. Caso exista um exercício o usuário deverá abrí-lo e desenvolver o programa para resolvê-lo. No desenvolvimento do programa o usuário poderá compilar, executar, depurar, salvar e finalizar o exercício.

Na compilação, execução ou depuração o sistema no servidor cria arquivos temporários no disco contendo o programa do usuário encapsulado dentro de uma classe Java. Quando o usuário realiza a operação de compilação o sistema compila o arquivo do disco que foi criado anteriormente e retorna ao usuário os possíveis erros de compilação se caso existir.

Na operação de execução o sistema abre uma janela onde será apresentada a execução do programa. Durante a execução o sistema pode apresentar a saída de dados do programa e o usuário pode realizar a entrada de dados quando o sistema solicitar. No final dessa operação a execução será finalizada pelo sistema.

Como na execução a depuração segue da mesma forma: o sistema abre uma janela de depuração e o usuário acompanha a seqüência da depuração clicando no botão Passo-a-passo. O sistema apresenta tanto as saídas de dados como a apresentação dos valores das variáveis em uso pelo programa. O usuário poderá realizar entrada de dados caso seja necessário ou então alterar o valor das variáveis em uso pelo programa. A depuração tem o seu fim quando o usuário fecha a janela de depuração.

Caso o usuário não tenha ainda terminado o seu programa, ele pode salvá-lo para voltar a refazer em um outro momento. Mas se o programa já estiver terminado então o usuário pode finalizar o exercício. Depois do exercício finalizado o usuário pode escolher outro exercício para resolver ou então fechar o ambiente.

A Figura 7 apresenta o diagrama de atividades onde o monitor utiliza o ambiente.

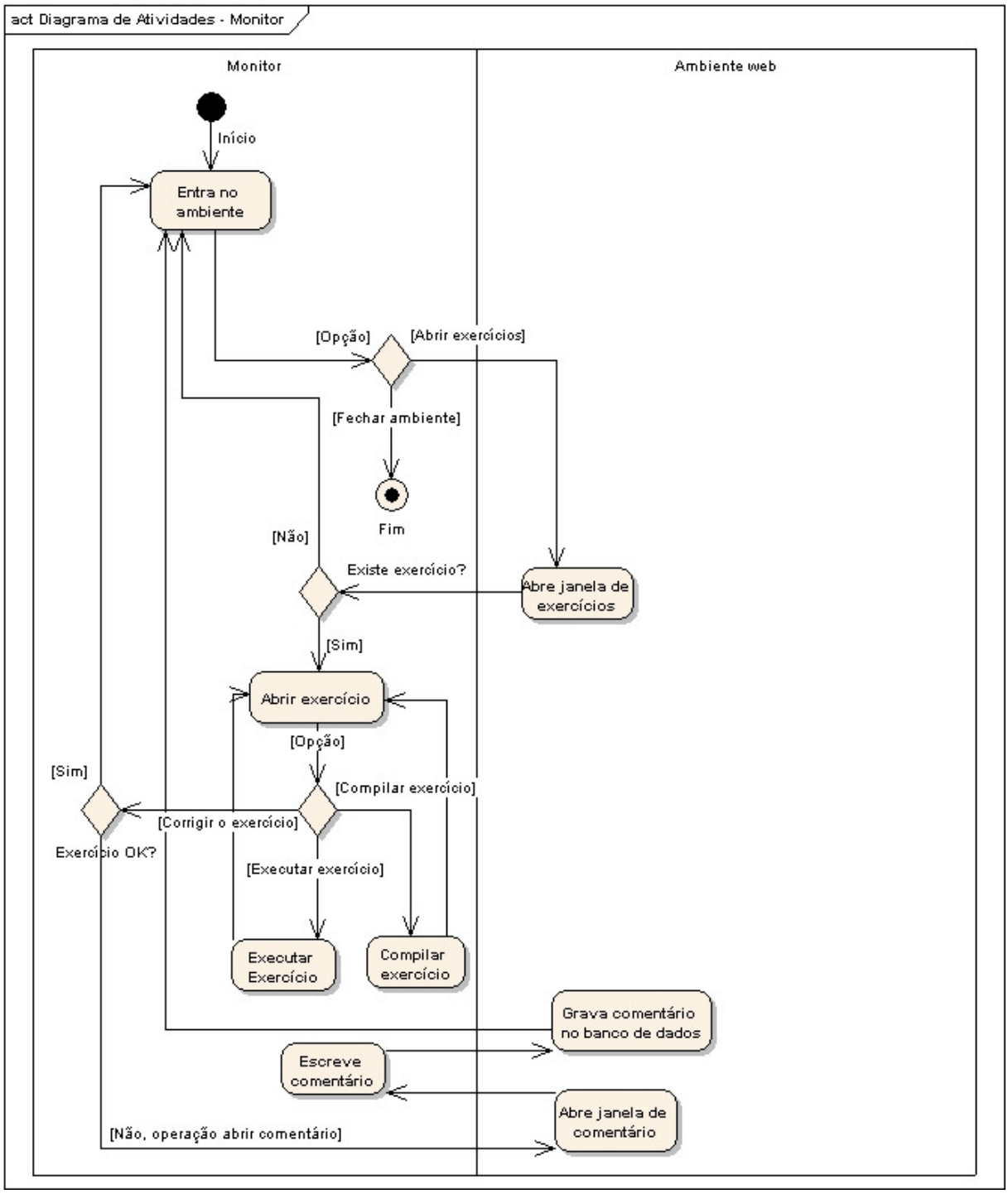


Figura 7 – Diagrama de atividades do monitor.

Como mostrado no diagrama, a interação do monitor começa com a abertura do ambiente web. Nesse momento o monitor possui duas alternativas: fechar o próprio ambiente ou abrir a janela de exercícios. Abrindo a janela de exercícios ele verifica se algum dos exercícios já foi finalizado pelo usuário, e pode abrí-lo para ser corrigido.

Depois do exercício aberto, o monitor pode testar o programa do usuário realizando a compilação e a execução do programa. Com isso o monitor irá descobrir se o programa do usuário está ou não funcionando.

Caso o programa do usuário esteja correto, a correção do exercício está terminada, o monitor pode então abrir outro exercício para corrigir ou fechar o ambiente. Mas se o programa do usuário estiver incorreto o monitor abre a janela de comentários. Nessa janela o monitor irá escrever um comentário para o exercício do usuário, onde ele informa eventuais erros ou então dar algumas dicas ao usuário.

A Figura 8 apresenta o diagrama de atividades onde o professor utiliza o ambiente.

O diagrama de atividades do professor é muito semelhante ao diagrama de atividades do monitor. A única diferença é que o professor pode criar exercícios.

O professor entra no ambiente web, onde pode corrigir ou então criar exercícios, a correção dos exercícios é idêntica à apresentada anteriormente no diagrama de atividade do monitor.

Para criar um exercício, o professor deve abrir o ambiente administrativo. Nesse ambiente o professor pode visualizar todos os exercícios existentes ou então criar um novo exercício. Para criar o exercício o professor deve preencher um formulário que possui os campos do título e da descrição do exercício. Depois disso, o ambiente irá cadastrar o exercício no banco de dados e atribuirá esse exercício a todos os usuários registrados no sistema.

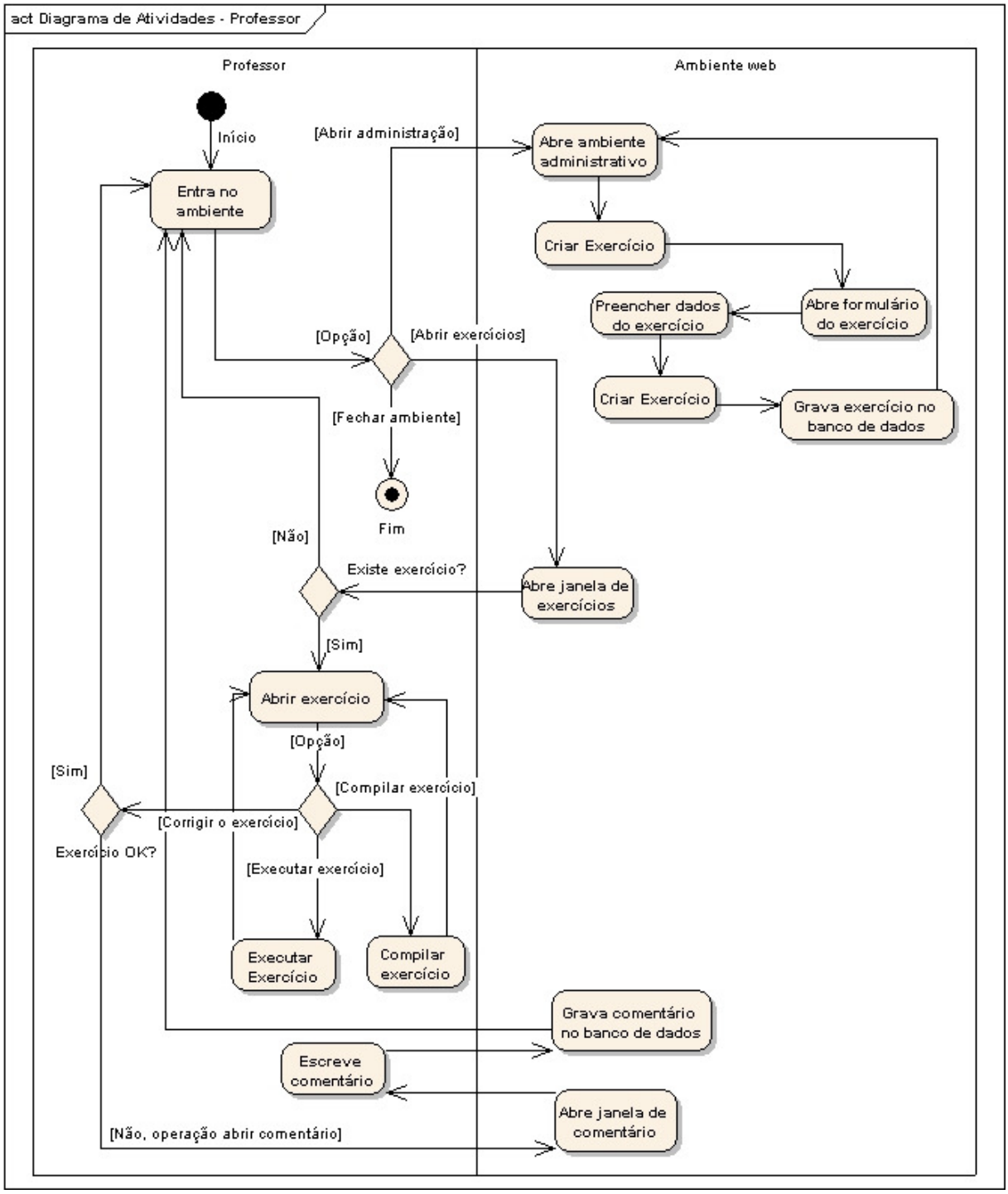


Figura 8 – Diagrama de atividades do professor.

3.2.3 Diagrama de pacotes e classes

Para Bezerra (2002, p. 95), “o aspecto estrutural estático de uma cooperação permite compreender como o sistema está estruturado internamente para que as funcionalidades externamente visíveis sejam produzidas”. Segundo Bezerra (2002, p. 95), “também é dito estrutural porque a estrutura das classes de objetos e as relações entre elas são apresentadas”.

A Figura 9 apresenta o diagrama de pacotes da camada de controle e modelo do sistema.

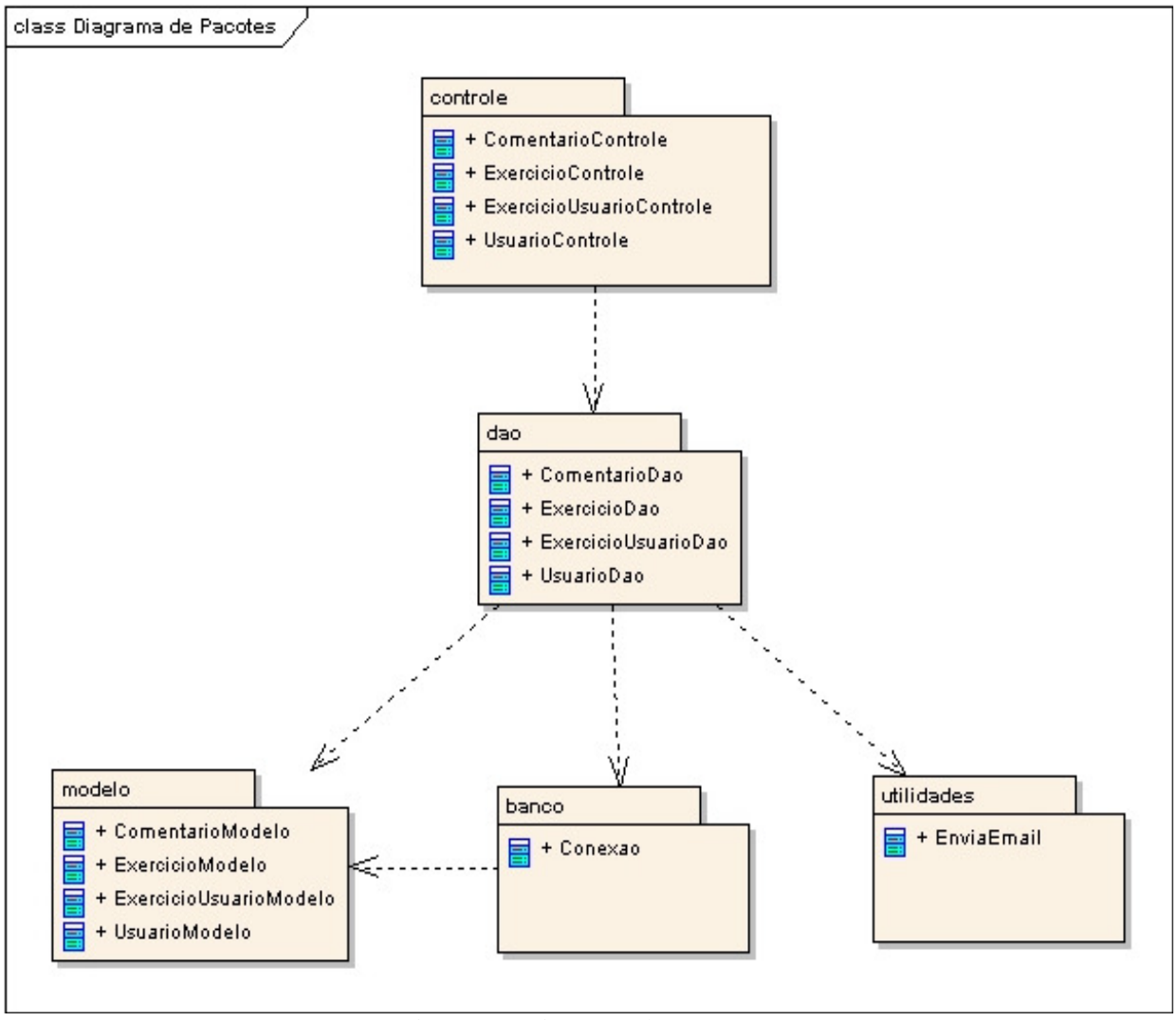


Figura 9 – Diagrama de pacotes

A camada de controle e modelo do sistema está dividida em três pacotes: `controle`, `dao` e `modelo`. O pacote controle realiza todas as regras de negócio sistema e faz a ligação do

ambiente web ao pacote `dao`. O pacote `dao` é quem realiza todas as funções da camada de dados do ambiente, como `inserir`, `buscar` e `alterar` os registro do banco de dados. O pacote `modelo` possui as classes responsáveis em fazer a representação dos dados do sistema.

Na Figura 10 são apresentadas as classes da camada de controle e modelo do usuário do sistema, o Quadro 17 mostra cada classes com suas responsabilidades.

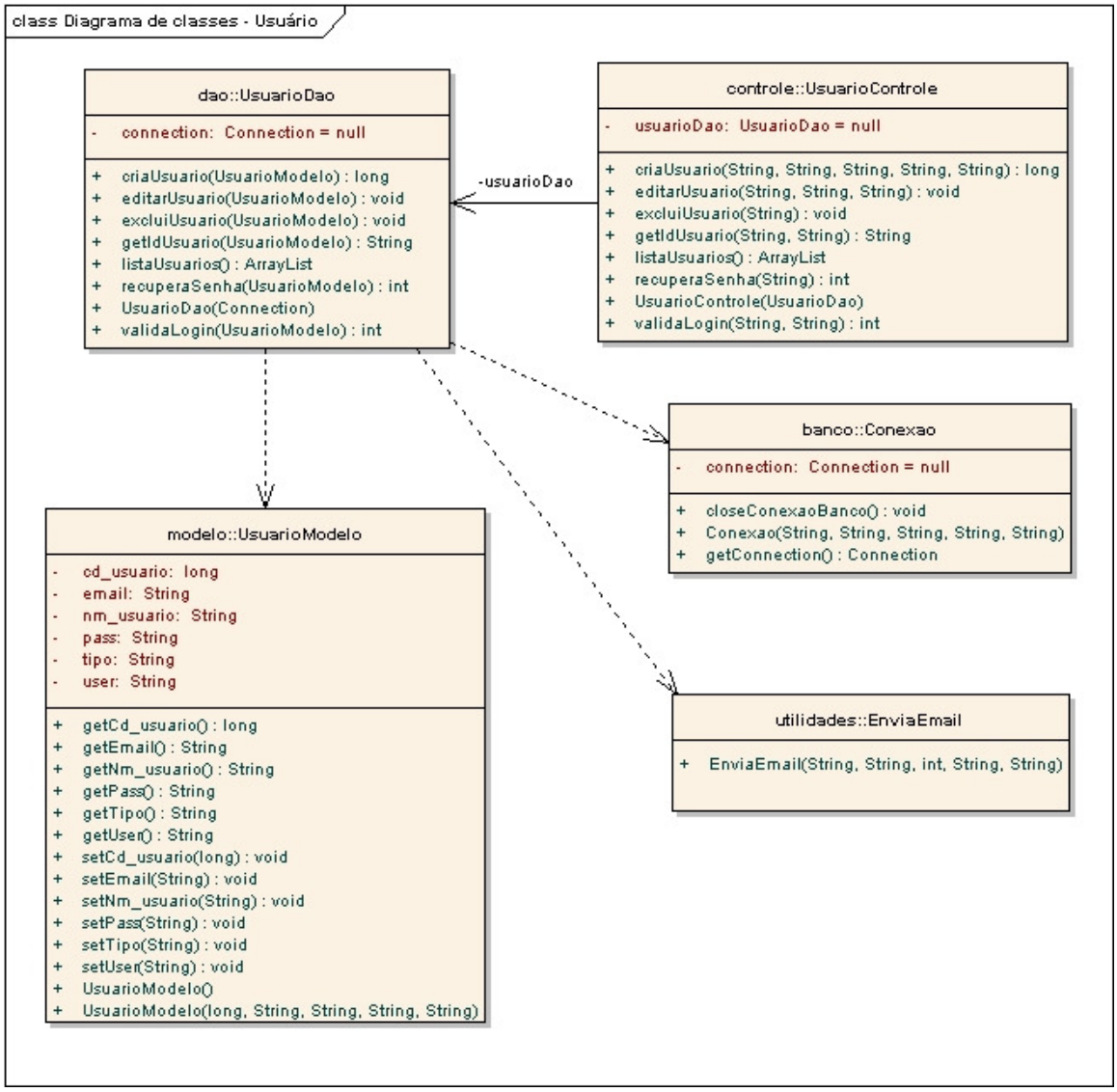


Figura 10 – Classes de controle e modelo do usuário

Classe	Descrição
UsuarioControle	Classe utilizada para realizar a integração entre a camada de visão ao objeto UsuarioDao.
UsuarioDao	Classe que executa todas as funções pertencentes ao usuário, como o seu cadastro, edição, exclusão e validação de login.
UsuarioModelo	Classe utilizada para representação dos dados do usuário.
Conexao	Classe responsável em realizar a conexão com o banco de dados.
EnviaEmail	Classe utilizada para enviar o email ao usuário quando ele efetua o registro.

Quadro 17 – Descrição das classes de controle o modelo do usuário

Na Figura 11 são apresentadas as classes da camada de controle e modelo do exercício do sistema, o Quadro 18 mostra cada classe com suas responsabilidades.

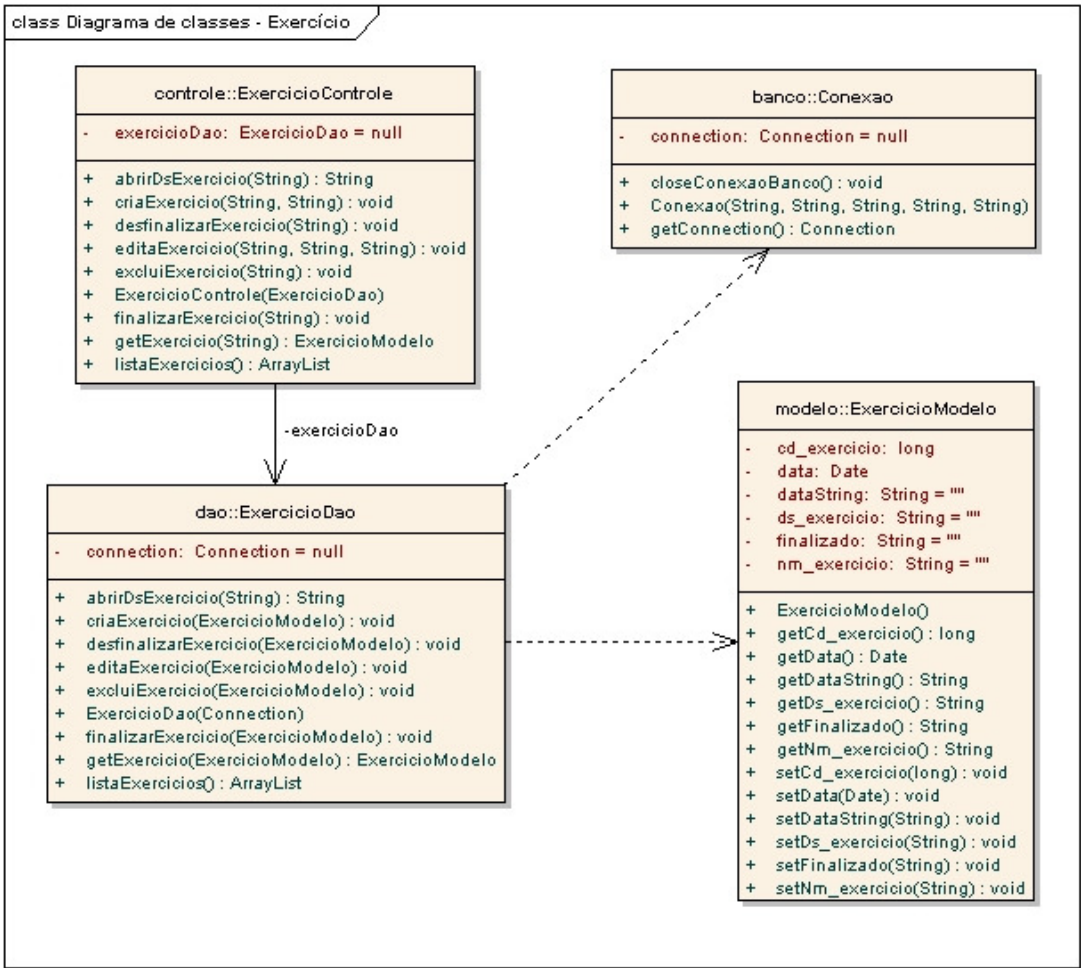


Figura 11 – Classes de controle o modelo do exercício no ambiente administrativo

Classe	Descrição
`ExercicioControle`	Classe utilizada para realizar a integração entre a camada de visão ao objeto ExercicioDao.
`ExercicioDao`	Classe utilizada para criar, editar e excluir um exercício, é utilizada no ambiente administrativo pelo professor.
`ExercicioModelo`	Classe utilizada para representação dos dados do exercício.
`Conexao`	Classe responsável em realizar a conexão com o banco de dados.

Quadro 18 – Descrição das classes de controle o modelo do exercício no ambiente administrativo

Na Figura 12 são apresentadas as classes da camada de controle e modelo dos comentários dos exercícios do sistema, o Quadro 19 mostra cada classe com suas responsabilidades.

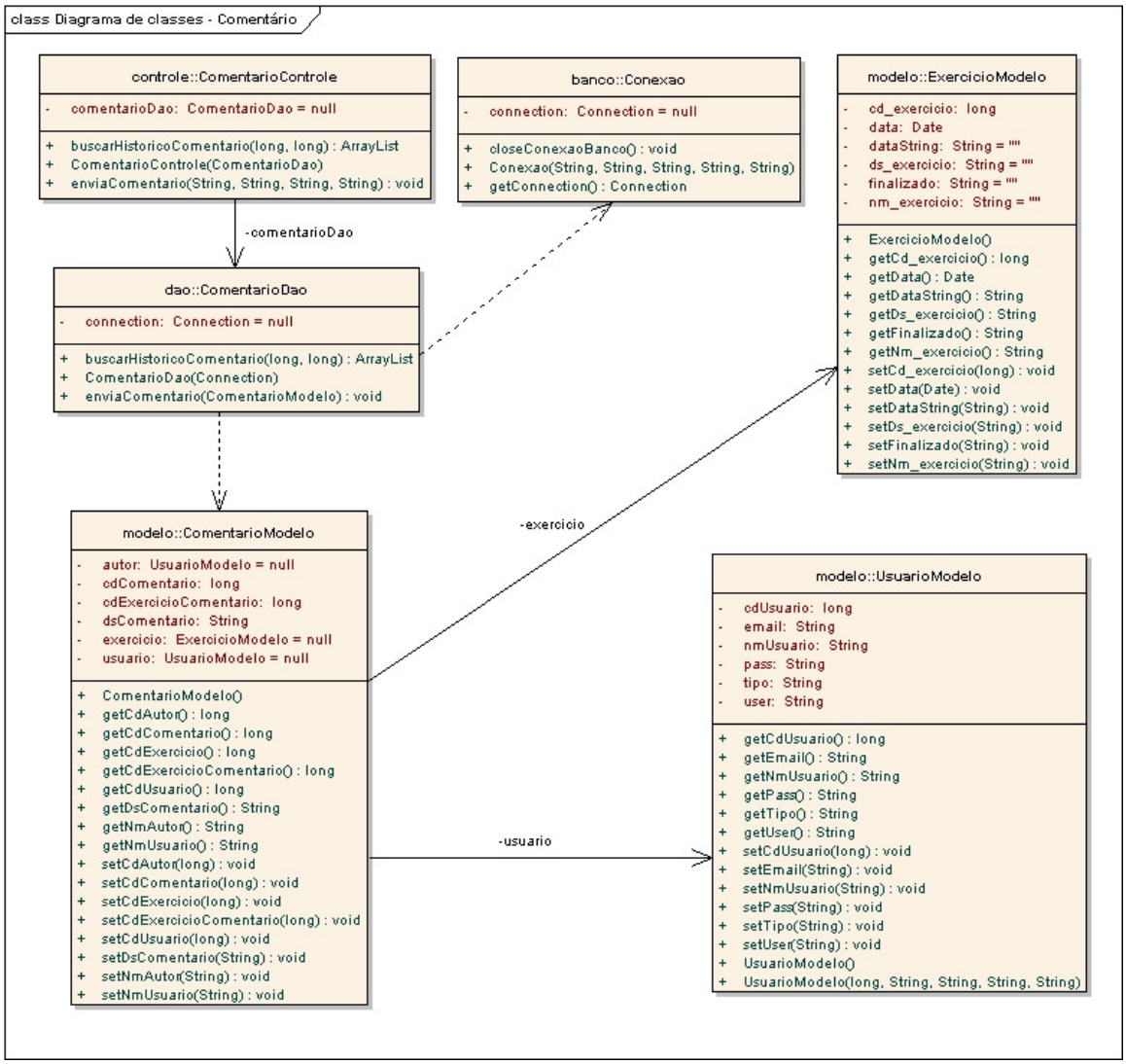


Figura 12 – Classes de controle o modelo do comentário dos exercícios

Classe	Descrição
`ComentarioControle`	Classe utilizada para realizar a integração entre camada de visão ao objeto ComentarioDao.
`ComentarioDao`	Classe utilizada para atribuir um comentário a um exercício do usuário e para visualizar o histórico de comentários.
`ComentarioModelo`	Classe utilizada para representação dos dados do comentário.
`ExercicioModelo`	Classe utilizada para representação dos dados do exercício.
`UsuarioModelo`	Classe utilizada para representação dos dados do usuário.
`Conexao`	Classe responsável em realizar a conexão com o banco de dados.

Quadro 19 – Descrição das classes de controle o modelo dos comentários dos exercícios

Na Figura 13 são apresentadas as classes da camada de controle e modelo do exercício do usuário, o Quadro 20 mostra cada classes com suas responsabilidades.

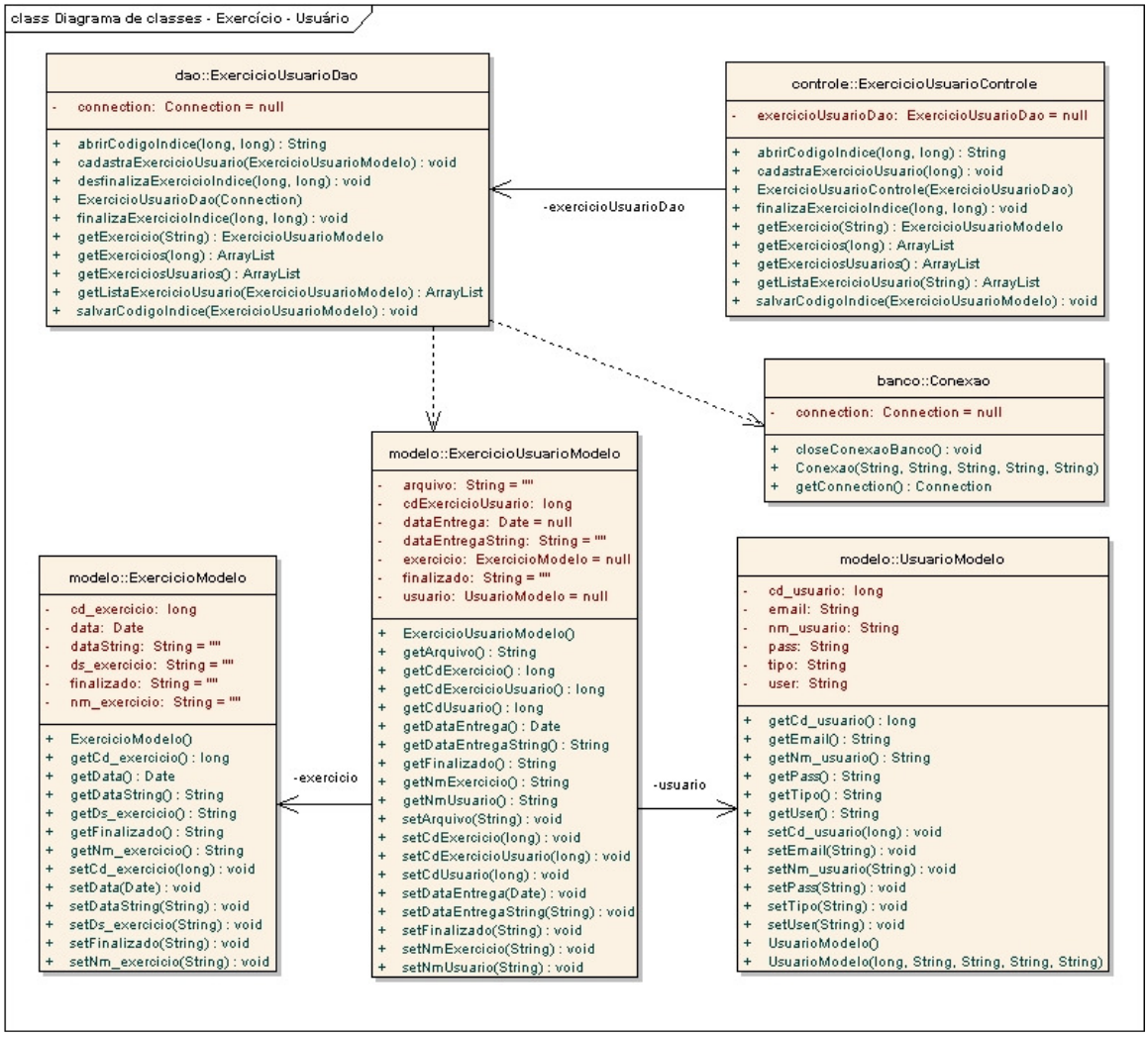


Figura 13 – Classes de controle o modelo dos exercícios do usuário.

Classe	Descrição
`ExercicioUsuarioControle`	Classe utilizada para realizar a integração entre a camada de visão e objeto ExercicioUsuarioDao.
`ExercicioUsuarioDao`	É através dessa classe que é feita a visualização dos exercícios que o usuário possui, também como salvar o programa do usuário para um determinado exercício e finalizar o exercício do usuário.
`ExercicioUsuarioModelo`	Classe utilizada para representação dos dados dos exercícios do usuário.
`UsuarioModelo`	Classe utilizada para representação dos dados do usuário.
`ExercicioModelo`	Classe utilizada para representação dos dados do exercício.
`Conexao`	Classe responsável em realizar a conexão com o banco de dados.

Quadro 20 – Descrição das classes de controle o modelo dos exercícios do usuário

3.2.4 Diagrama de navegabilidade

A Figura 14 apresenta o diagrama de navegabilidade do ambiente de programação e a Figura 15 do ambiente administrativo. Para o desenvolvimento dos diagramas foi utilizada a notação WAE, conforme apresentado em Conallen (2003, p. 329).

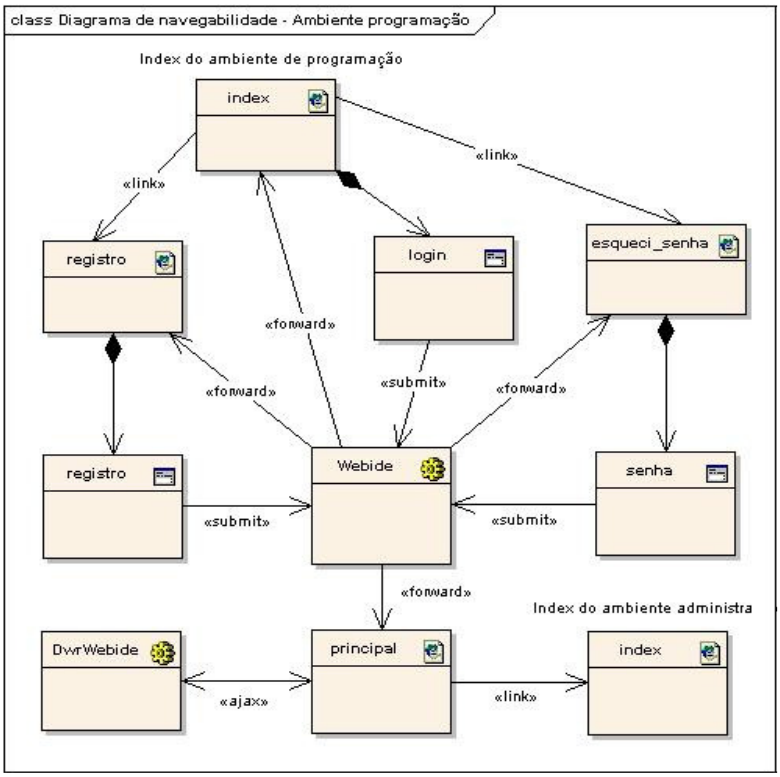


Figura 14 – Diagrama de navegabilidade do ambiente de programação

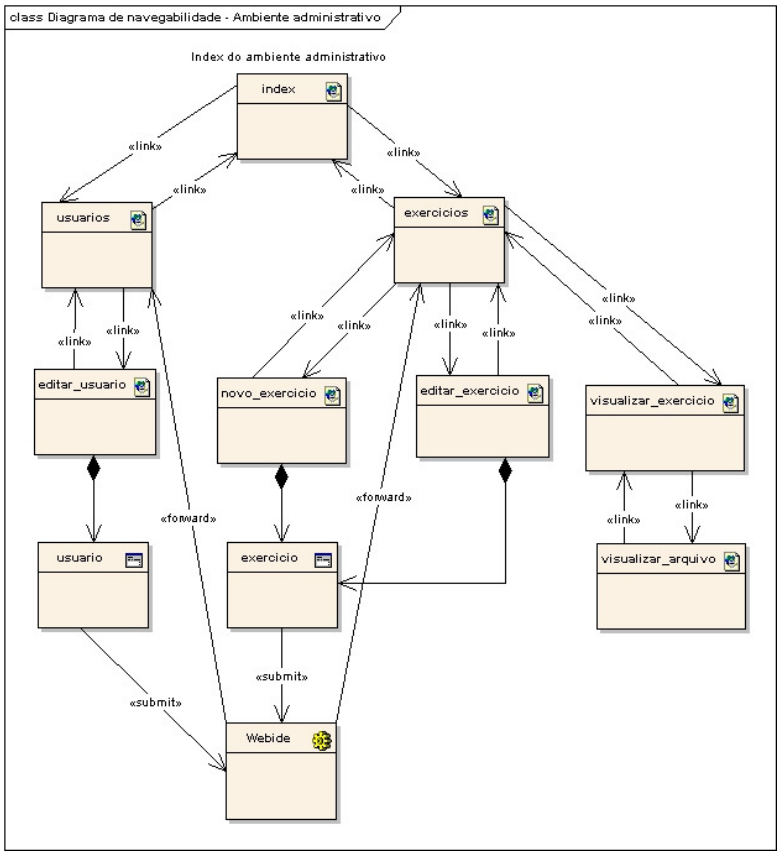


Figura 15 – Diagrama de navegabilidade do ambiente administrativo

3.2.5 Diagramas de seqüência

O diagrama de seqüência apresentado na Figura 16, mostra a seqüência dos passos realizados pelo sistema quando o usuário compila o seu programa.

Tudo começa com o usuário clicando no link Compilar no ambiente de programação onde a pagina principal do sistema faz uma chamada AJAX para a classe `DwrWebide`. Essa classe por fim cria uma instância da classe `Compilador`, passando como parâmetros o programa do usuário, código do usuário e local onde a classe `Compilador` irá criar os arquivos pertencentes ao usuário. Depois disso a classe `DwrWebide` executa o método `executaCompilacao()` da classe `Compilador` para que ela realize a compilação do programa do usuário. O retorno do método `executaCompilacao()` irá devolver à página principal e os erros de compilação do programa se caso existir.

Nas Figuras 16, 17 e 18 os métodos executados pelo objeto `DwrWebide` que contém em seu nome o texto "retornaXXX", não é o retorno das chamadas dos métodos recebidos anteriormente pelo objeto `DwrWebide`, mas sim um novo método executado, pois esse objeto é baseado numa arquitetura AJAX conforme descrito no parágrafo anterior.

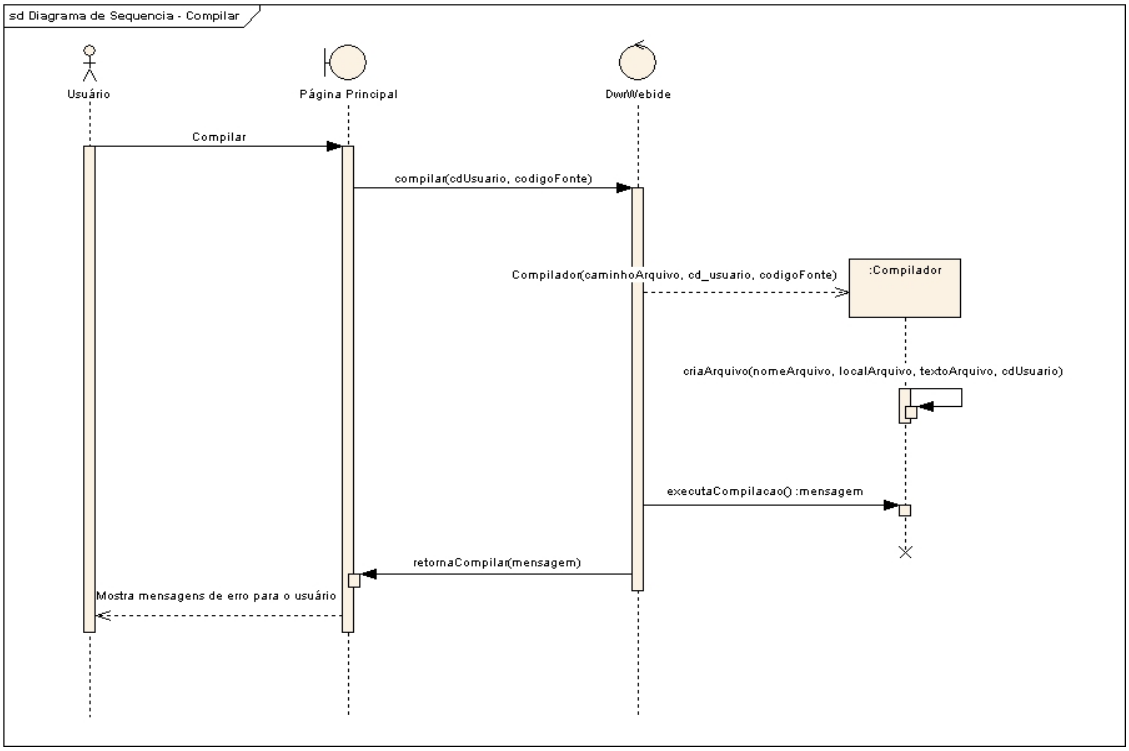


Figura 16 – Diagrama de seqüência para compilar o programa

O diagrama de seqüência apresentado na Figura 17, mostra a seqüência dos passos realizados pelo sistema quando o usuário executa o seu programa.

Para dar início à execução do programa, o usuário deve clicar no link Executar no ambiente de programação. A página principal do sistema faz uma chamada AJAX para o objeto `DwrWebide`, que por fim cria uma instância da classe `Execucao`, passando como parâmetros o código do usuário e um *array* de *String* chamado `listaSaidaDados` que irá guardar a saída de dados da execução do programa.

A partir daí a execução entra em um laço realizando dois processos. Um deles é a saída de dados, onde a página principal fará chamadas consecutivas ao objeto `DwrWebide`, que executará o método `buscaValorExecucao()` que vai buscar no *array* `listaSaidaDados` os dados de saída que foram adicionados pela classe `Execucao`.

O outro processo do laço é a entrada de dados que será realizada pelo usuário. O usuário fará a entrada dos dados na janela de execução e a página principal vai repassar ao objeto `DwrWebide` o valor da entrada que será enviada a classe `Execucao` pelo método `EnviaValorExecucao()`.

O fim da execução do programa acontece quando o usuário fecha a janela de execução, essa ação faz com que a página principal do ambiente de programação realize uma chamada AJAX ao objeto `DwrWebide` que por sua vez executa o método `fimExecucao()` do objeto `Execucao` finalizando todo o processo.

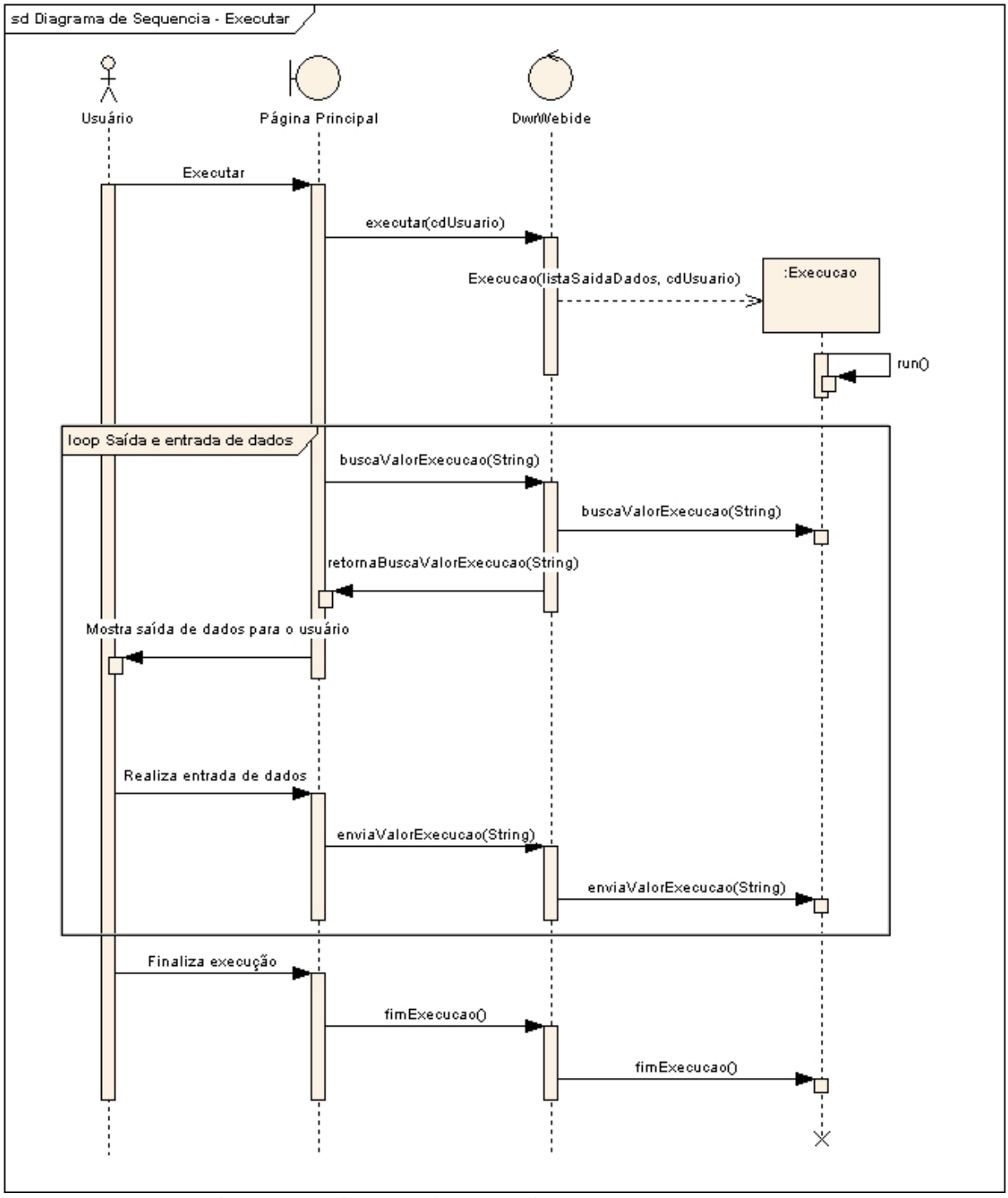


Figura 17 – Diagrama de seqüência para executar o programa

O diagrama de seqüência apresentado na Figura 18, mostra a seqüência dos passos realizados pelo sistema quando o usuário faz a depuração do seu programa.

Para dar início à depuração do programa, o usuário deve clicar no link Depurar no ambiente de programação. A página principal do sistema faz uma chamada AJAX para o

objeto `DwrWebide`, que por fim cria uma instância da classe `Depurador`, passando como parâmetros o código do usuário e os *arrays* de S*tring* chamados `listaSaidaDados`, `listaComando` e `listaVariaveis` que irá guardar a saída de dados da execução do programa, o próximo comando a ser executado pelo depurador e as variáveis utilizadas pelo programa do usuário. A depuração do programa começa quando o usuário clica no botão Passo-a-passo na janela de depuração.

A partir daí cada vez que o usuário clicar no botão Passo-a-passo a classe `DwrWebide` irá executar alguns métodos como o `buscaValorDepuracao` utilizado para buscar e apresentar ao usuário a saída de dados do programa. O método `buscaValorDepuracaoVariaveis` traz ao usuário todas as variáveis e o seu conteúdo que está sendo utilizado pelo programa. O método `buscaValorComando` traz ao usuário o próximo comando do programa que será executado pelo depurador.

Durante o laço de depuração, o usuário poderá enviar dados de entrada para o programa como também alterar os valores das variáveis utilizando o método `enviaValorDepuracao`.

O fim da depuração do programa acontece quando o usuário fecha a janela de depuração. A página principal do ambiente de programação faz uma chamada AJAX ao objeto `DwrWebide` que por sua vez vai executar o método `fimDepuracao()` do objeto `Depurador` finalizando o processo de depuração.

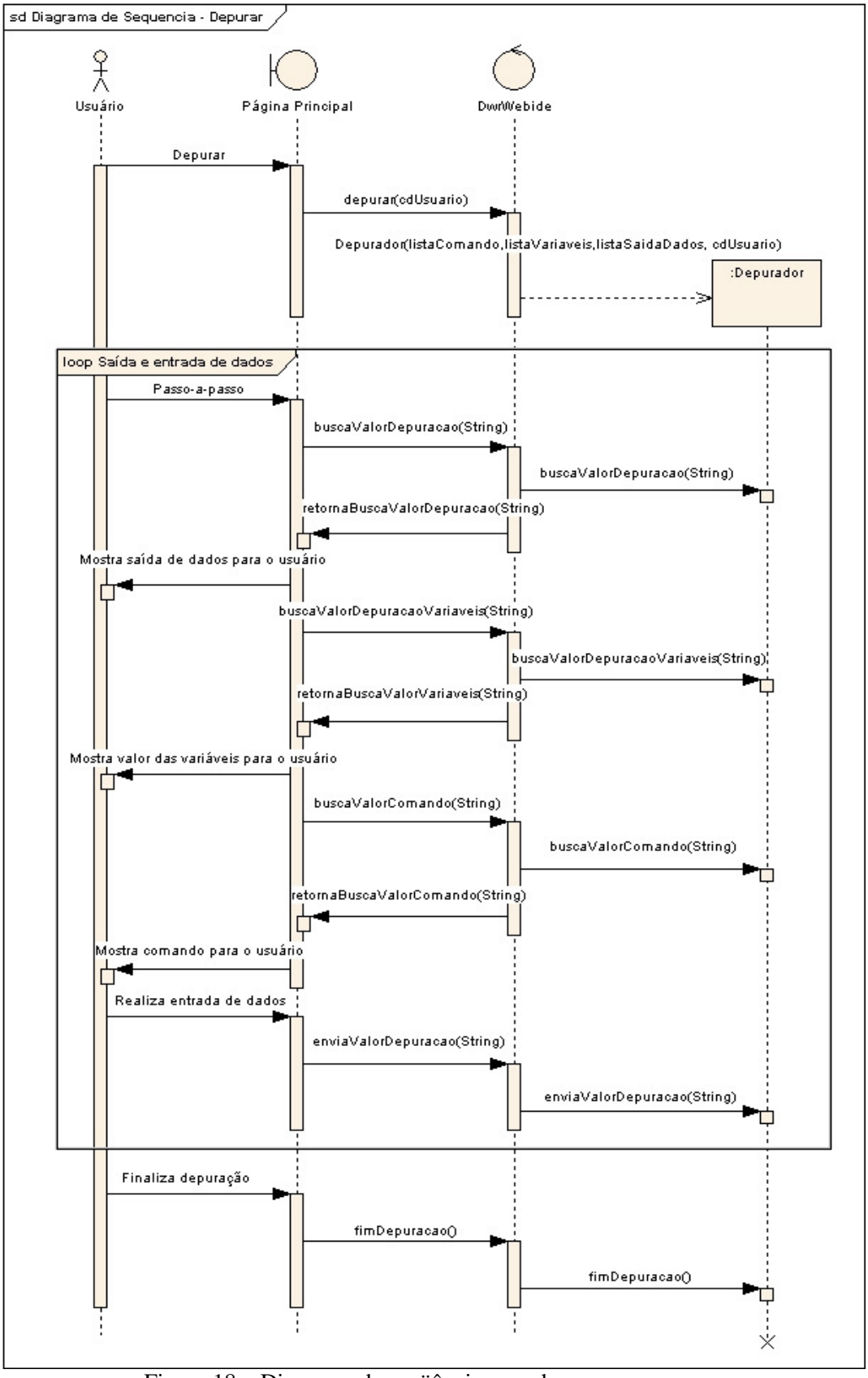


Figura 18 – Diagrama de seqüência para depurar o programa

3.2.6 Modelo entidade relacionamento

A Figura 19 apresenta o modelo entidade relacionamento utilizado pelo ambiente web.

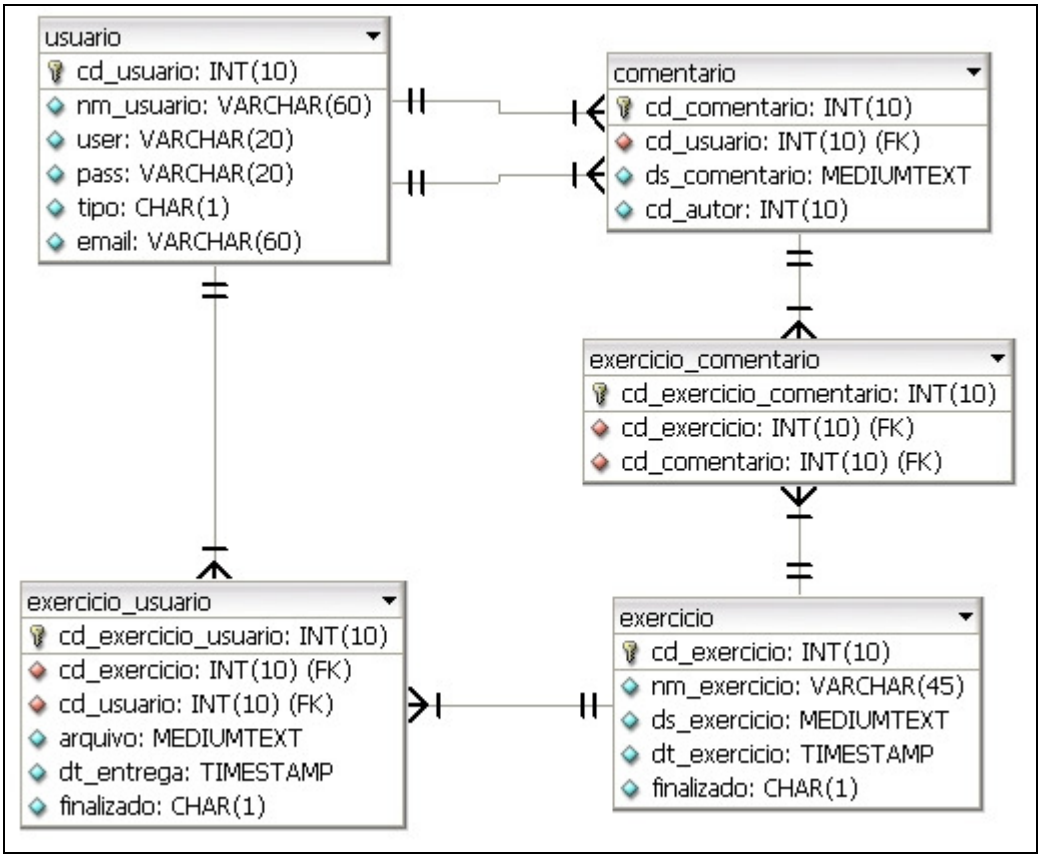


Figura 19 – Modelo entidade relacionamento

A descrição dos atributos do modelo entidade relacionamento pode ser observado no Apêndice A.

3.3 IMPLEMENTAÇÃO

Para o desenvolvimento do ambiente web foi utilizada a linguagem de programação Java JDK 1.5 através da plataforma de desenvolvimento Netbeans 6.0, para o servidor web foi utilizado o Apache-Tomcat 5.5 e o banco de dados MySQL 5.0.

3.3.1 Biblioteca de entrada e saída de dados

Foi desenvolvida uma biblioteca para o usuário realizar a entrada e a saída de dados durante a execução e a depuração do seu programa. A biblioteca é composta por duas classes Java chamadas de `WebE` e `WebD`. Essa biblioteca pode ser utilizada pelo usuário no desenvolvimento do seu programa no ambiente de programação.

Na Figura 20 são apresentadas às classes da biblioteca e o Quadro 21 mostra a descrição das classes.

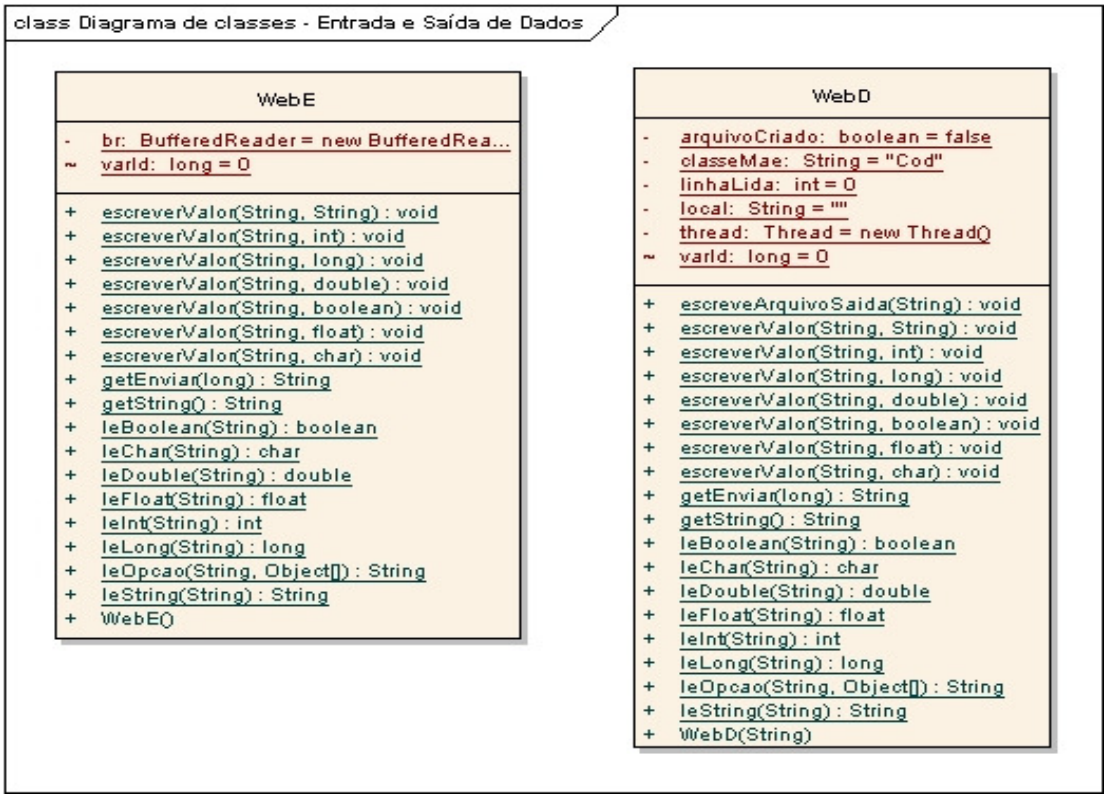


Figura 20 – Classes utilizadas para prover a entrada e saída de dados durante a execução e depuração

Classe	Descrição
`WebE`	Essa classe implementa os métodos que o usuário utilizará no seu programa para realizar os recursos de entrada e saída durante a execução.
`WebD`	Essa classe implementa os métodos que o usuário utilizará no seu programa para realizar os recursos de entrada e saída durante a depuração.

Quadro 21 – Descrição das classes utilizadas para prover os recursos de entrada e saída de dados

O Quadro 22 apresenta uma descrição dos métodos que o usuário pode utilizar para realizar a entrada e saída de dados no seu programa. Os métodos utilizados são os mesmos para as classes `WebE` e `WebD`.

Método	Tipo	Retorno	Parâmetros
`web.escreverValor`	Saída de dados	-	String, String
`web.escreverValor`	Saída de dados	-	String, int
`web.escreverValor`	Saída de dados	-	String, long
`web.escreverValor`	Saída de dados	-	String, double
`web.escreverValor`	Saída de dados	-	String, boolean
`web.escreverValor`	Saída de dados	-	String, float
`web.escreverValor`	Saída de dados	-	String, char
`web.leBoolean`	Entrada de dados	boolean	String
`web.leChar`	Entrada de dados	char	String
`web.leDouble`	Entrada de dados	double	String
`web.leFloat`	Entrada de dados	float	String
`web.leInt`	Entrada de dados	int	String
`web.leLong`	Entrada de dados	long	String
`web.leOpcao`	Entrada de dados	String	String
`web.leString`	Entrada de dados	String	String

Quadro 22 – Descrição dos métodos utilizados para prover os recursos de entrada e saída

Todos os métodos utilizados para saída e entrada de dados possuem o seu primeiro parâmetro sendo uma String, que é utilizada para escrever um título junto ao campo de entrada ou saída de dados; e o segundo parâmetro de cada método de saída de dados é utilizado para imprimir o dado que o usuário desejar.

O Quadro 23 apresenta o trecho de um programa que utiliza dois métodos, um para entrada e o outro para a saída de dados.

```
int idade = web.leInt("Digite a sua idade");
web.escreverValor("Sua idade é",idade);
String sexo = web.leOpcao("Digite o sexo",new
    String[]{"Masculino","Feminino","Outro"});
```

Quadro 23 – Exemplo de comandos para entrada e saída de dados

A Figura 21 apresenta os campos para entrada e saída de dados que foram executados pelo programa do Quadro 23. Pode ser visto o campo para a entrada de dados, a linha onde foi realizada a saída de dados, como também o comando `web.leOpcao`, que é utilizado para realizar a entrada de dados através de valores pré-definidos em seu comando.

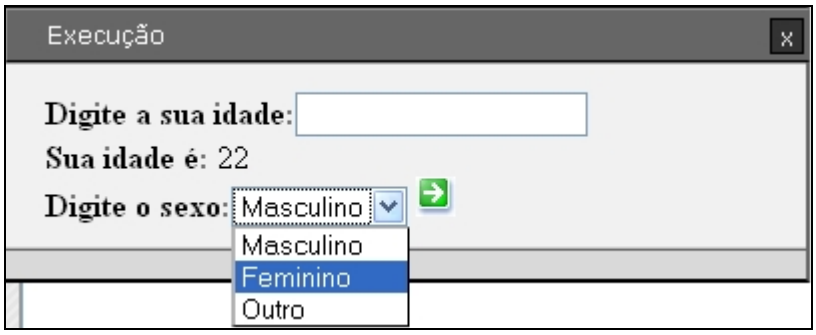


Figura 21 – Execução do comando de entrada e saída de dados

3.3.2 Ambiente de programação

Esta seção apresenta a implementação do ambiente de programação. Na seção 3.3.2.1 apresenta trechos de códigos fonte responsáveis em compilar, executar e depurar o programa do usuário. A seção 3.3.2.2 apresenta a classe `DwrWebide` que controla todas as funcionalidades e recursos do ambiente de programação.

3.3.2.1 Compilação, execução e depuração de classes

Antes de compilar, executar ou depurar, o ambiente deve criar no disco o arquivo que contém o programa do usuário. O programa do usuário será encapsulado dentro de uma classe Java. O Quadro 24 apresenta um programa do usuário como exemplo, e o Quadro 25 apresenta o arquivo criado no disco contendo o programa do usuário encapsulado dentro de uma classe Java.

```
int numero = web.leInt("Digite um numero");
if ( numero % 2 == 0 ) {
   web.escreverValor("Resposta","O número "+numero+" é par!");
}else{
   web.escreverValor("Resposta","O número "+numero+" é ímpar");
}
```

Quadro 24 – Programa do usuário

```
public class Cod11{

   private static WebE11 web;

   public static void main(String args[]){
      int numero = web.leInt("Digite um numero");
      if ( numero % 2 == 0 ) {
        web.escreverValor("Resposta","O número "+numero+" é par!");
      }else{
        web.escreverValor("Resposta","O número "+numero+" é ímpar");
      }
   }
}
```

Quadro 25 – Programa do usuário encapsulado em uma classe Java

Pode-se observar no Quadro 25 que o nome da classe que foi gravada no disco é chamada de `Cod11.P`ara diferenciar os arquivos de cada usuário foi definido que cada arquivo no disco possua seu nome composto da *String* "Cod" seguido pelo número do índice da tabela `Usuario` gravada no banco de dados. Pode-se observar também a importação da biblioteca de entrada e saída de dados na linha `private static WebE11 web;`. Cada usuário possui a sua biblioteca e para diferenciar a sua das outras é atribuído o código de usuário ao nome da classe. A importação estática da classe é realizada para que o usuário não precise instanciar nenhuma classe da biblioteca, desse modo ele somente executa os métodos do objeto `web`.

Depois do programa do usuário gravado no disco, é realizada a sua compilação. O Quadro 26 apresenta um trecho de código fonte do método `executaCompilacao` da classe `Compilador` responsável em compilar o programa do usuário.

```
Runtime rt = Runtime.getRuntime();
Process proc = rt.exec(DwrWebide.caminhoCompilacao + " -g -cp "
        + DwrWebide.caminhoArquivos + " " + this.caminhoArquivo);
erro = new BufferedReader(new
        InputStreamReader(proc.getErrorStream()));
String linhaErro = erro.readLine();
```

Quadro 26 – Código fonte utilizado para compilar o programa do usuário

O atributo `DwrWebide.caminhoCompilacao` guarda o endereço do programa **javac** no computador, o atributo `DwrWebide.caminhoArquivos` guarda o endereço da pasta onde estão localizados os programas dos usuários e as bibliotecas de entrada e saída de dados. O atributo `this.caminhoArquivo` possui o endereço e o nome da classe do programa do usuário que foi gravado no disco anteriormente. Depois que o método `rt.exec()` tenha sido

executado, os erros de compilação serão retornados pelo método `erro.readLine()`.

Caso a compilação do programa do usuário tenha sido realizada com sucesso, o usuário poderá então executar o seu programa. O Quadro 27 apresenta o início do processo de execução que é o mesmo utilizado para a depuração do programa do usuário.

```
rt = Runtime.getRuntime();
proc = rt.exec(DwrWebide.caminhoExecucao+" -cp "+
               DwrWebide.caminhoArquivos+" "+arquivo);

entrada = new BufferedReader(new InputStreamReader(
                            proc.getInputStream()));
erro = new BufferedReader(new InputStreamReader(
                         proc.getErrorStream()));
saida = new PrintWriter(proc.getOutputStream(),true);
```

Quadro 27 – Início do processo de execução e depuração do programa do usuário

O atributo `DwrWebide.caminhoExecucao` informa o endereço onde se encontra o programa **java** no computador. No caso da depuração é utilizado o atributo `Webide.caminhoDepuracao` que contém o endereço do programa **jdb**. Também são criados os *buffers* `entrada`, `erro` e `saida`. O *buffer* `entrada` é utilizado para recuperar a saída de dados realizada no programa do usuário, o *buffer* `erro` recupera possíveis erros gerados por exceções no programa do usuário e o *buffer* `saida` é utilizado para realizar a entrada de dados no programa do usuário.

O Quadro 28 apresenta um trecho de código da classe `Execucao` utilizado para ler a saída de dados realizada pelo programa durante a sua execução. Caso o método `entrada.ready()` retorne verdadeiro, então o programa do usuário realizou uma saída de dados. O *buffer* `entrada` realiza a leitura do dado enviado pelo programa utilizando o método `entrada.readLine()`, os dados lidos serão armazenados no *array* `listaSaida` que será repassado ou ambiente web e apresentado ao usuário.

```
while(linhaEntrada!=null){
   listaSaida.addLast(linhaEntrada);
   while(entrada.ready()==false && fimExecucao==false){
      if(entrada.ready()){
         linhaEntrada = entrada.readLine();
      }else{
         linhaEntrada = null;
      }
   }
}
```

Quadro 28 – Código fonte utilizado para ler a saída de dados realizado pelo programa

O Quadro 29 apresenta o método `enviaValorSaida` da classe `Execucao` utilizada para realizar a entrada de dados no programa do usuário durante a execução. A entrada de dados é realizada pelo *buffer* `saida`, através do seu método `println` é que ocorre o envio dos dados de entrada para o programa do usuário.

```
public void enviaValorSaida(String valor){
      saida.println(valor);
}
```

Quadro 29 – Código fonte utilizado para realizar a entrada de dados

O início do processo de depuração do programa do usuário já foi apresentado no Quadro 26 e segue da mesma forma como na execução. Na depuração os *buffers* `entrada` e `saida` não são utilizados para realizar a entrada e a saída de dados do programa, mas para enviar comandos e obter as respostas do programa **jdb**. Depois que o processo inicial já foi executado, o objeto `Depurador` responsável em depurar o programa do usuário envia dois comandos ao programa **jdb** apresentados no Quadro 30.

```
saida.println("stop at "+arquivo+":0");
saida.println("run "+arquivo);
```

Quadro 30 – Comandos para definir o ponto de início da depuração

O comando “stop at” define ao programa **jdb** que a depuração será iniciada em uma classe que está guardada no atributo `arquivo` e que a depuração irá iniciar na linha zero do seu método `main`. O comando “run” informa ao programa **jdb** que a depuração vai começar.

Depois que o ponto de início da depuração foi configurado, a seqüência da depuração é realizada através do envio do comando “next” para o *buffer* `saida` ao programa **jdb**. A cada comando “next” enviado, o depurador irá executar uma linha do programa do usuário e realizará uma saída de dados. Essa saída contém o próximo comando que será executado pelo depurador, as variáveis em uso pelo programa do usuário e se o programa do usuário tenha terminado.

A objeto `Depurador` realiza uma verificação desses dados através do *buffer* `entrada` como pode ser visto no Quadro 31.

Caso o programa do usuário termine, será retornado ao *buffer* `entrada` a S*tring* "The application exited" que atribuirá verdadeiro a *flag* `fimExecucao` e o depurador termina o processo de depuração. Caso seja retornado "Step completed:" a próxima linha de saída vindo

do **jdb** traz o próximo comando que será executado no programa do usuário. Se o retorno for "Local variables:" estará indicando que as próximas linhas retornadas pelo **jdb** serão as variáveis que estão em uso pelo programa do usuário.

```
linhaEntrada = entrada.readLine();
if(linhaEntrada.contains("The application exited")){
   fimExecucao=true;
}else
if(linhaEntrada.contains("Step completed:")){
   proximaLinha = true;
}else
if(linhaEntrada.contains("Local variables:")){
   variaveis = true;
}
```

Quadro 31 – Verificação da saída de dados do programa **jdb**

O Quadro 32 apresenta o trecho de código que busca da saída do programa **jdb** o próximo comando que será executado na depuração caso a *flag* `proximaLinha` seja verdadeira no Quadro 31. Nesse código obtém-se o próximo comando a ser executado e o número da sua linha no programa. A variável `vazio` é utilizada para fazer a separação da linha e do comando que foi retornado na mesma *String*. A variável `linha` armazena o número da linha e a variável `linhaEntrada` armazena o próximo comando a ser executado pelo depurador.

```
if(proximaLinha==true){
  ...
  int vazio = linhaEntrada.indexOf(" ");
  int linha = integer.parseInt(
              linhaEntrada.substring(0,vazio));
  linhaEntrada = linhaEntrada.substring(
                 vazio+4,linhaEntrada.length());
  if(!linhaEntrada.contains("}}")){
     listaComando.clear();
     listaComando.addLast(String.valueOf(linha));
  }
  ...
  proximaLinha = false;
}
```

Quadro 32 – Código fonte que busca a próxima linha que será executada

O Quadro 33 apresenta o trecho de código fonte que busca da saída do programa **jdb** as variáveis em uso pelo programa do usuário caso a *flag* `variaveis` seja verdadeira no Quadro 31. O código apresentado recebe as variáveis utilizadas pelo programa do usuário até que o programa **jdb** retorne a *String* "main[1]". Quando isso acontecer o atributo `listaVariaveis` que armazena as variáveis será limpa e estará pronta para receber os dados do próximo passo da depuração do programa do usuário.

Cada linha recebida do programa **jdb** contém uma variável que está em uso pelo programa do usuário. Essa linha contém o nome da variável e o seu conteúdo. O comando `linhaEntrada.spit(“ = “)` faz a separação dessa *String* colocando o nome da variável recebida na variável `aux[0]` e o valor da mesma na variável `aux[1]`. Depois é montada uma *String* para a apresentação desse dado e armazenada no atributo `listaVariaveis` que será repassado posteriormente para a janela de depuração e apresentado ao usuário.

```
if(variaveis==true){
   ...
   if(!linhaEntrada.contains("main[1]")){
      if(limpaVariaveis == true){
         limpaVariaveis = false;
         listaVariaveis.clear();
         listaVariaveis.add("");
         codVariaveis = 0;
      }
      String aux[] = linhaEntrada.split(" = ");
      String var = "";
      var = var.concat("<b>"+aux[0]+
                  "</b> = <input type=text id=campo"+codVariaveis+
                  " value='"+aux[1]+"'>"+
                  getEnviar(codVariaveis,aux[0])+"<br>");
      listaVariaveis.addLast(var+"<br>");
      codVariaveis++;
   }else{
      limpaVariaveis = true;
      variaveis = false;
   }
}
```

Quadro 33 – Código fonte que busca as variáveis em uso pelo programa do usuário

Para realizar os recursos de entrada e saída de dados na depuração, foram criados dois arquivos utilizados para transferir os dados entre o ambiente de programação e o processo que está realizando a depuração do programa do usuário.

Os arquivos são chamados de `Cod<cd_usuario>.saida` utilizados para o envio dos dados do ambiente de programação para a depuração do programa. O outro arquivo é chamado de `Cod<cd_usuario>.entrada`, utilizado para o envio dos dados da depuração do programa para o ambiente de programação. A cada entrada de dados que o usuário realizar na depuração, a classe `DwrWebide` estará armazenando essa entrada no arquivo `Cod<cd_usuario>.saida`, onde a biblioteca de entrada e saída de dados fará a leitura desse arquivo repassando os valores lidos para o programa do usuário.

A saída de dados realizadas no programa do usuário durante a depuração é gravada no arquivo `Cod<cd_usuario>.entrada` pela biblioteca, a leitura desse arquivo será realizada

pela classe `DwrWebide` e os dados lidos que na verdade são a saída de dados do programa do usuário serão apresentados para ele no ambiente de programação.

3.3.2.2 Classe de controle do ambiente de programação

É através da classe `DwrWebide` que todos os recursos e funcionalidades do ambiente de programação são realizados. Essa é a classe que executa todas as chamadas AJAX. O Quadro 34 apresenta o conteúdo do arquivo `dwr.xml` onde é configurada a classe `DwrWebide` que vai responder pelas chamadas AJAX realizadas pelo navegador do usuário.

```
<dwr>
  <allow>
   <create creator="new" javascript="DwrWebide" scope="session">
     <param name="class" value="DwrWebide"/>
   </create>
   <convert
       converter="bean"match="banco.modelo.ExercicioUsuarioModelo"/>
   <convert converter="bean" match="banco.modelo.ComentarioModelo"/>
   <convert converter="bean" match="banco.modelo.UsuarioModelo"/>
   <convert converter="bean" match="banco.modelo.ExercicioModelo"/>
  </allow>
</dwr>
```

Quadro 34 – Arquivo `dwr.xml` do ambiente web

Caso seja necessário retornar ao ambiente algum objeto contendo um modelo de dados, esse objeto também deverá ser configurado no arquivo `dwr.xml`. Nesse arquivo foi configurado os modelos de dados `ExercicioUsuarioModelo`, `ComentarioModelo`, `UsuarioModelo` e `ExercicioModelo`.

A página contendo o código JavaScript que será executado no navegador do usuário deve possuir uma chamada para o arquivo `DwrWebide.js` como apresentado no Quadro 35. Esse arquivo é gerado pela biblioteca DWR que faz a ligação entre o código JavaScript da página ao objeto `DwrWebide`.

```
<script type='text/javascript' src='dwr/interface/DwrWebide.js'>
...
</script>
```

Quadro 35 – Chamada do arquivo `DwrWebide.js`

O Quadro 36 apresenta um trecho de código que realiza uma chamada AJAX. Essa chamada é utilizada para abrir a janela contendo os exercícios dos usuários que serão

corrigidos pelo monitor. Pode ser visto que a função JavaScript executa o método `getExerciciosUsuarios` do objeto `DwrWebide`, passando como parâmetro a função JavaScript que tratará o retorno do método `getExerciciosUsuarios`.

```
function abrirExercicioUsuario(){
    DwrWebide.getExerciciosUsuarios(retornaAbrirExercicioUsuario);
}
```

Quadro 36 – Código fonte JavaScript para chamada de um método do objeto `DwrWebide`

O Quadro 37 apresenta o método `getExerciciosUsuarios` da classe `DwrWebide`. Esse método fará uma chamada ao objeto `exercicioUsuarioControle`, retornando um *ArrayList* contendo os exercícios dos usuários. Esse *ArrayList* será devolvido como um vetor para a função JavaScript `retornaAbrirExercicioUsuario` que foi passado como parâmetro no Quadro 36.

```
...
public ArrayList getExerciciosUsuarios(){
   return exercicioUsuarioControle.getExerciciosUsuarios();
}
...
```

Quadro 37 – Chamada da classe `DwrWebide` ao objeto `exercicioUsuarioControle`

O Quadro 38 apresenta um trecho do código fonte da função JavaScript `retornaAbrirExercicioUsuario`. Essa função abre a janela de exercícios no ambiente do monitor e também realiza a montagem de uma tabela contendo os valores obtidos pelo retorno do método apresentado no Quadro 36. Pode ser observado que o vetor que foi recebido como parâmetro na função é percorrido, e usando o comando `dwr.util.addRows` são adicionadas as linhas na tabela da janela de exercícios. O último comando apresentado é utilizado para tornar a janela de exercícios visível ao usuário.

```
function retornaAbrirExercicioUsuario(resposta){
   ...
   for(var i=0;i<resposta.length;i++){
      dwr.util.addRows('tabelaExercicioUsuario',[[
          resposta[i].nmExercicio,resposta[i].nmUsuario,
          resposta[i].finalizado,resposta[i].cdExercicioUsuario,
          resposta[i].cdUsuario,resposta[i].cdExercicio ]]
      , cellFuncs, { escapeHtml:false });
   }
document.getElementById("janelaExerciciosUsuarios")
                                    .style.visibility = "visible";
}
```

Quadro 38 – Trecho do código fonte da função JavaScript `retornaAbrirExercicioUsuario`

É desta maneira que todas as funções do ambiente de programa são executadas. A função JavaScript executa um método da classe `DwrWebide`, seu retorno é repassado novamente ao código JavaScript, que por sua vez a apresentado ao usuário.

3.3.3 Ambiente administrativo

Esta seção apresenta a implementação do ambiente administrativo utilizado pelo professor. Na seção 3.3.3.1 é apresentado o *servlet* que controla todo o ambiente administrativo. Na seção 3.3.3.2 será mostrada implementação da camada de visão do ambiente administrativo.

3.3.3.1 Classe de controle e DAO

A classe `Webide` é um *servlet* que recebe e executa todas as requisições do ambiente administrativo. O Quadro 39 apresenta um trecho do conteúdo do arquivo `web.xml` onde é configurado na aplicação o *servlet* que receberá as requisições.

```
...
<servlet>
      <servlet-name>webide</servlet-name>
      <servlet-class>Webide</servlet-class>
</servlet>
<servlet-mapping>
      <servlet-name>webide</servlet-name>
      <url-pattern>/webide</url-pattern>
</servlet-mapping>
...
```

Quadro 39 – Trecho de código da configuração do *servlet* no arquivo web.xml

O *servlet* `Webide` possui três métodos principais. Como pode ser observado no Quadro 40, o primeiro método realiza a inicialização da conexão com o banco de dados e a instância dos objetos DAO.

```
public void init() throws ServletException {
   super.init();
   ...
   conexao = new Conexao(props.getProperty("servidor"),
          props.getProperty("banco"),props.getProperty("porta"),
          props.getProperty("user"),props.getProperty("pass"));

   usuarioDao = new UsuarioDao(conexao.getConnection());
   usuarioControle = new UsuarioControle(usuarioDao);
   exercicioDao = new ExercicioDao(conexao.getConnection());
   exercicioControle = new ExercicioControle(exercicioDao);
   exercicioUsuarioDao = new
                  ExercicioUsuarioDao(conexao.getConnection());
   exercicioUsuarioControle = new
                  ExercicioUsuarioControle(exercicioUsuarioDao);
}
```

Quadro 40 – Método que inicializa o *servlet* do ambiente administrativo

O segundo método é `doGet` que é executado nas requisições do método GET. O Quadro 41 apresenta o método `doGet` com o trecho de código para efetuar o *logoff* do usuário no ambiente administrativo.

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {

   if("sair".equals(request.getParameter("form"))){
      request.getSession().invalidate();
      RequestDispatcher prox =
                           request.getRequestDispatcher("index.jsp");
      prox.forward(request, response);
   }
   ...
}
```

Quadro 41 – Método que recebe as chamadas GET do ambiente administrativo

O terceiro método é `doPost` que é executado nas requisições do método POST das páginas do ambiente administrativo. O Quadro 42 apresenta o método `doPost` e também a verificação de um formulário enviado ao *servlet* quando o professor cria um novo exercício. Pode-se observar que é passado como parâmetro no formulário o título e a descrição do exercício. O *servlet* repassa esses dados nos objetos DAO que farão a gravação do exercício no banco de dados. O *servlet* ainda realiza a atualização da lista de exercícios e redireciona para a página de exercícios cadastrados.

```
protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {

if("novo_exercicio".equals(request.getParameter("form"))){
   this.exercicioControle.criaExercicio(
                                request.getParameter("titulo"),
                                request.getParameter("descricao"));

   request.setAttribute("listaExercicios",
                          this.exercicioControle.listaExercicios());
   RequestDispatcher prox =
        request.getRequestDispatcher("administracao/exercicios.jsp");
   prox.forward(request, response);
}
   ...
```

Quadro 42 – Método que recebe as chamadas POST do ambiente administrativo

3.3.3.2 Camada de visão

A apresentação dos dados ao usuário na camada de visão do ambiente administrativo é realizada através da biblioteca JSTL[5]. A seguir será apresentado trechos de códigos fonte que darão um exemplo de como é utilizada a biblioteca JSTL no ambiente administrativo.

O Quadro 43 apresenta o código fonte JavaScript executado quando o usuário clica no link Exercícios na tela do ambiente administrativo. Esse código realiza uma chamada GET ao *servlet* `Webide` passando no parâmetro `form` o conteúdo “exercicios”.

```
onclick="window.location.href = '../webide?form=exercicios'"
```

Quadro 43 – Código JavaScript do link exercícios no ambiente administrativo

O Quadro 44 apresenta o trecho de código que tratará a chamada do comando que foi executado no Quadro 43. Como pode ser observado, o código faz uma verificação se o parâmetro possui o valor “exercicios”, nesse caso o objeto `exercicioControle` executa o método `listaExercicios()` que retorna um *arrayList* contendo todos os exercícios cadastrados no sistema que é atribuído para a página `exercicios.jsp` a ser processada pelo sistema para montar a visualização do conteúdo para o usuário.

[5] Sun Microsystems (2008) define JSTL sendo uma biblioteca que possui um conjunto de *tags* que permitem escrever páginas JSP sem código Java, aumentando assim a legibilidade do código e a interação entre desenvolvedor e web designers.

```
if("exercicios".equals(request.getParameter("form"))){
   request.setAttribute("listaExercicios",
   this.exercicioControle.listaExercicios());
   RequestDispatcher prox = request.getRequestDispatcher
                              ("administracao/exercicios.jsp");
   prox.forward(request, response);
}
```

Quadro 44 – Código da classe `Webide` que executa a chamada do link Exercícios

O Quadro 45 apresenta um trecho do código da página `exercicios.jsp`, que apresenta algumas *tags* JSTL utilizadas para a exibição dos dados na camada de visão do sistema administrativo. A *tag* `<c:forEach items="${listaExercicios}" var="exercicio">` realiza uma repetição para percorrer o atributo `listaExercicios`, que foi inicializado anteriormente conforme demonstrado no Quadro 43. Em cada repetição a variável `exercicio` recebe o exercício daquele passo. A *tag* `<c:out value='${exercicio.dataString}'/>` realiza a saída de dados imprimindo na página a variável `exercicio.dataString`.

```
<c:forEach items="${listaExercicios}" var="exercicio">
  <tr>
    <td bgcolor="#EBEBEB" class="style22" valign="top">
      <c:out value='${exercicio.dataString}'/> <br>
    </td>
    <td bgcolor="#EBEBEB" class="style22" valign="top">
      <c:out value='${exercicio.nmExercicio}'/> <br>
    </td>
    <c:if test='${exercicio.finalizado=="N"}'>
      <td bgcolor="#EBEBEB" class="style22" valign="top">
        Não <br>
      </td>
    </c:if>
    <c:if test='${exercicio.finalizado=="S"}'>
      <td bgcolor="#EBEBEB" class="style22" valign="top">
        Sim <br>
      </td>
    </c:if>
  </tr>
  ...
</c:forEach>
```

Quadro 45 – Código fonte JSTL da página exercícios.jsp

3.4 OPERACIONALIDADE

Nesta seção é apresentada a operacionalidade do trabalho. Na seção 3.4.1 será

apresentada a criação de um exercício pelo professor. Na seção 3.4.2 será mostrada a resolução de um exercício pelo aluno. Na seção 3.4.3 será apresentada a correção de um exercício pelo monitor.

3.4.1 Criando o exercício

A Figura 22 apresenta o ambiente administrativo do sistema que é utilizado pelo professor.

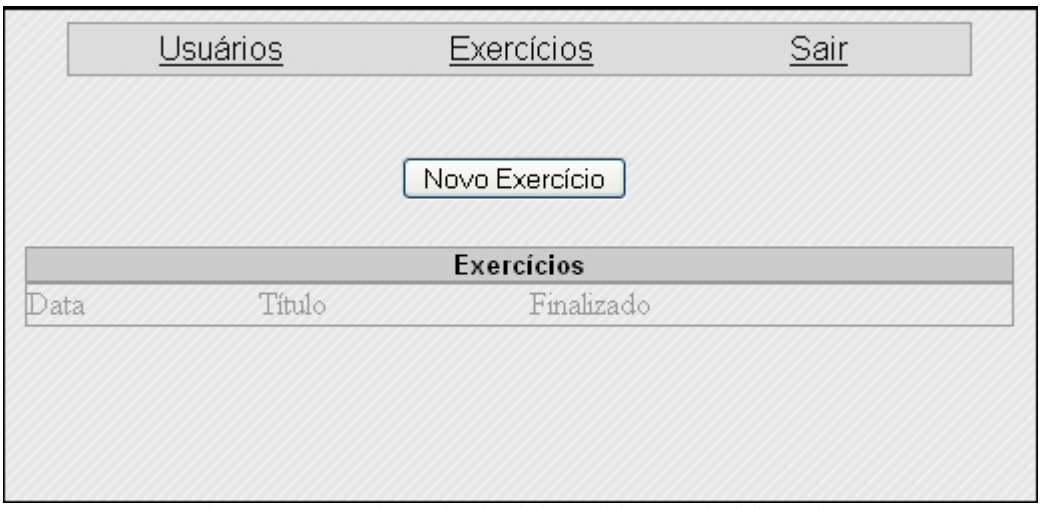


Figura 22 – Página principal do ambiente administrativo

Quando o professor seleciona o botão `Novo Exercício` abre um formulário para a criação do exercício conforme apresentado na Figura 23.

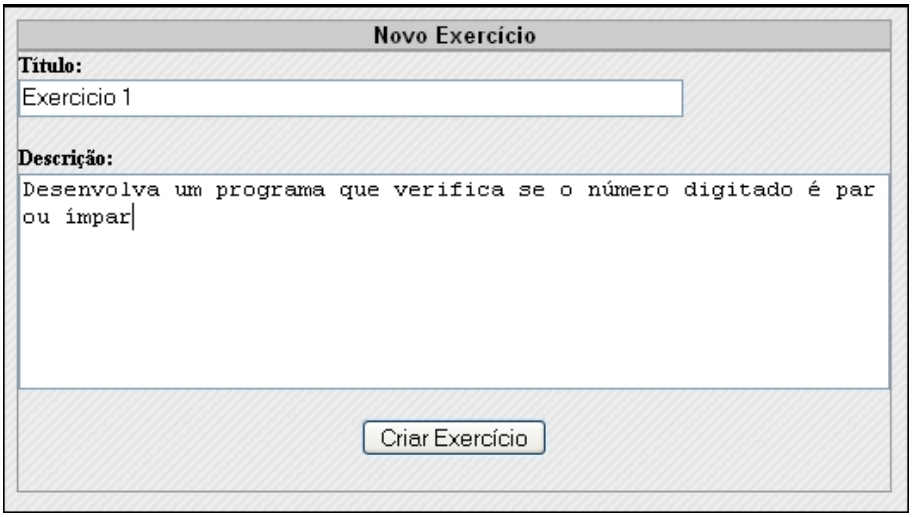


Figura 23 – Formulário para criar um novo exercício

Depois do formulário preenchido, como exemplificado na Figura 23, o professor pode selecionar o botão `Criar Exercício`. Esse exercício será criado no banco de dados e atribuído a todos os usuários registrados no sistema.

A Figura 24 apresenta a janela de exercícios depois que um exercício foi criado. O professor pode excluir o exercício (1), editar o exercício (2), visualizar os alunos que estão realizando o exercício (3) e pode finalizar o exercício (4). Neste caso com o exercício finalizado pelo professor nenhum dos alunos poderá resolver o exercício.

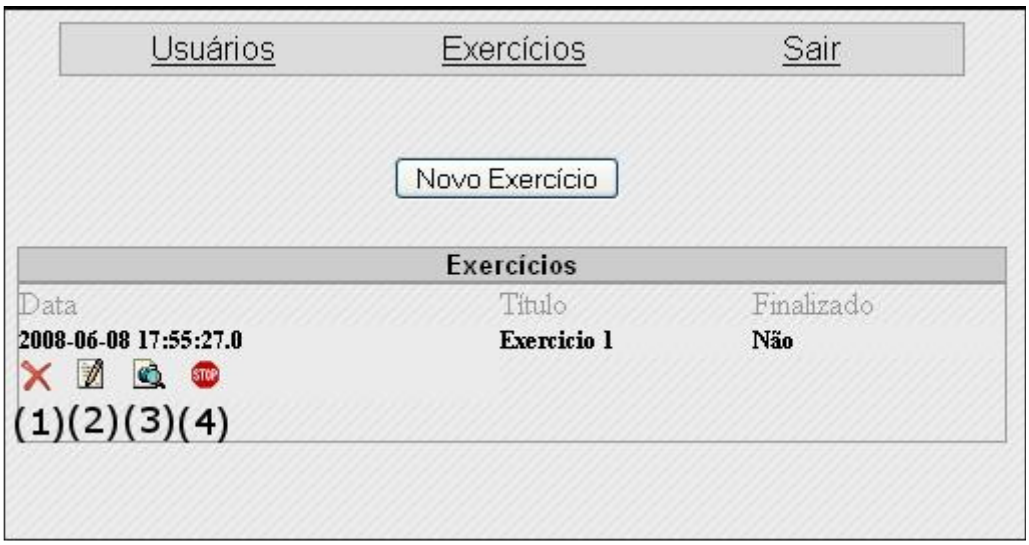


Figura 24 – Janela dos exercícios cadastrados no sistema

3.4.2 Resolvendo o exercício

A Figura 25 apresenta a tela principal do ambiente de programação utilizado pelo aluno.

Figura 25 – Ambiente de programação

Nesse ambiente estão disponíveis os links de `Compilar`, `Depurar` e `Executar` o programa, além do link `Exercícios` para abrir uma janela contendo todos os exercícios cadastrados no sistema e os links `Salvar` e `Finalizar` o exercício.

O usuário inicia a resolução de um exercício selecionando o link `Exercícios`, que abre a janela de `exercícios` como pode ser vista na Figura 26.

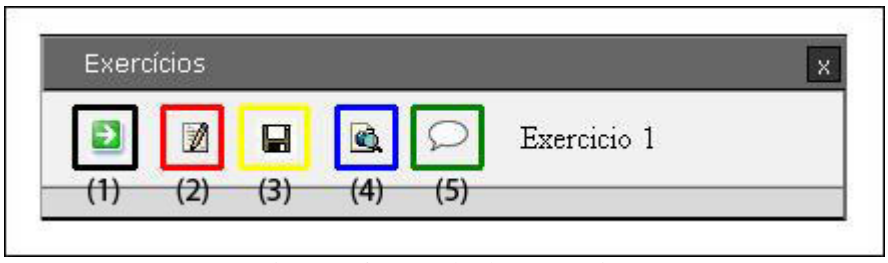


Figura 26 – Janela de exercícios

A janela `Exercícios` possui os botões para finalizar o exercício (1), abrir o exercício (2), salvar o exercício (3), visualizar o exercício (4) e visualizar os comentários (5). O usuário ao selecionar o botão abrir o exercício, é carregado o código fonte se caso o usuário já havia digitado e salvo anteriormente. A Figura 27 apresenta a interface do ambiente depois que o

exercício foi aberto e um programa possuindo um erro já foi implementado e compilado.

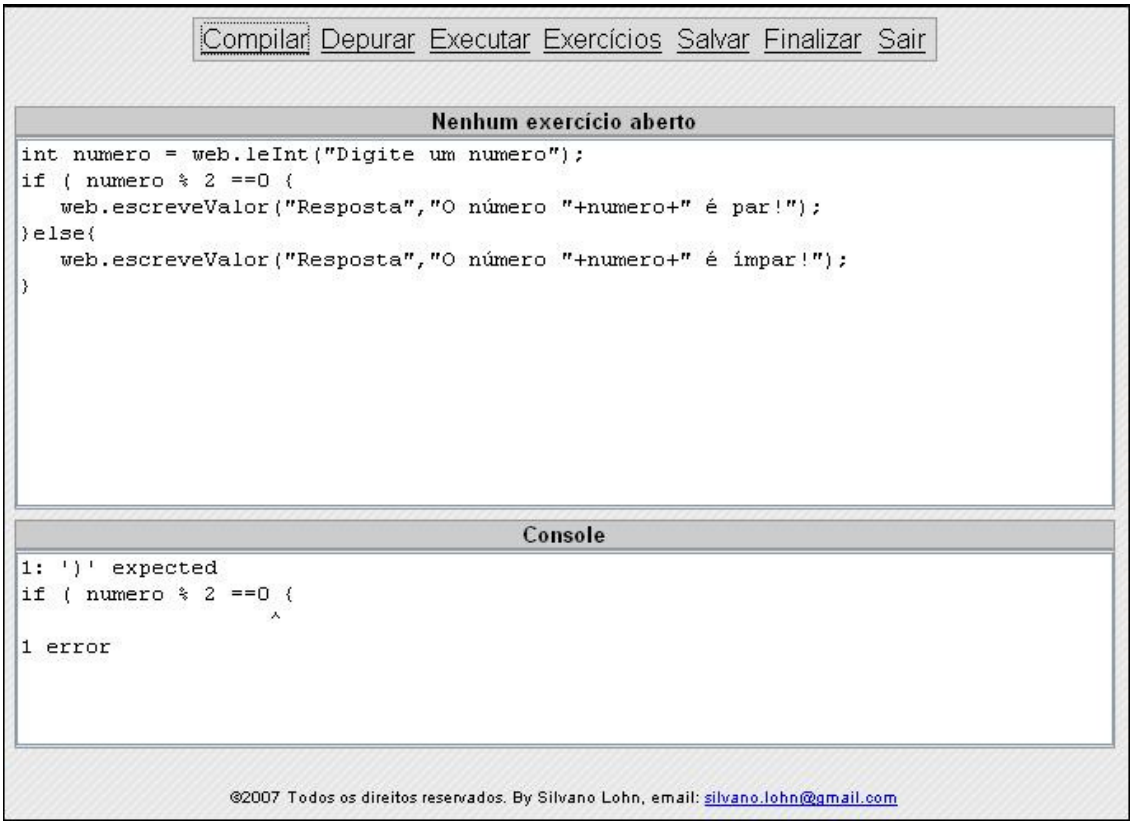


Figura 27 – Compilação do programa com erros

Na Console do ambiente pode-se ver as mensagens de erros vindas da compilação do programa. As mensagens de compilação informam a linha do programa que ocorreu o erro e uma pequena descrição, essas são mensagens de erros padrão do compilador Java. Depois de compilado o usuário pode realizar a execução do programa. A Figura 28 apresenta os dois passos utilizados para a execução do programa que foi implementado.

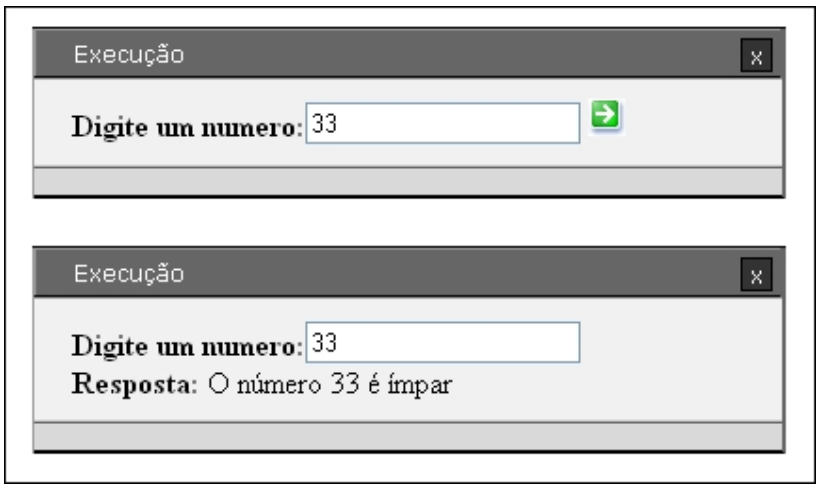


Figura 28 – Execução do programa

A Figura 29 apresenta a janela de depuração do programa e o seu conteúdo. O botão `Passo-a-passo` (1) é onde o usuário executa a depuração, no código fonte do usuário (2), é apresentada uma linha destacando o próximo comando que será executado pelo depurador, o usuário poderá realizar a entrada e saída de dados (3) e as variáveis (4) que estão sendo utilizadas pelo programa do usuário, podem ter seus valores alterados durante a execução do programa.

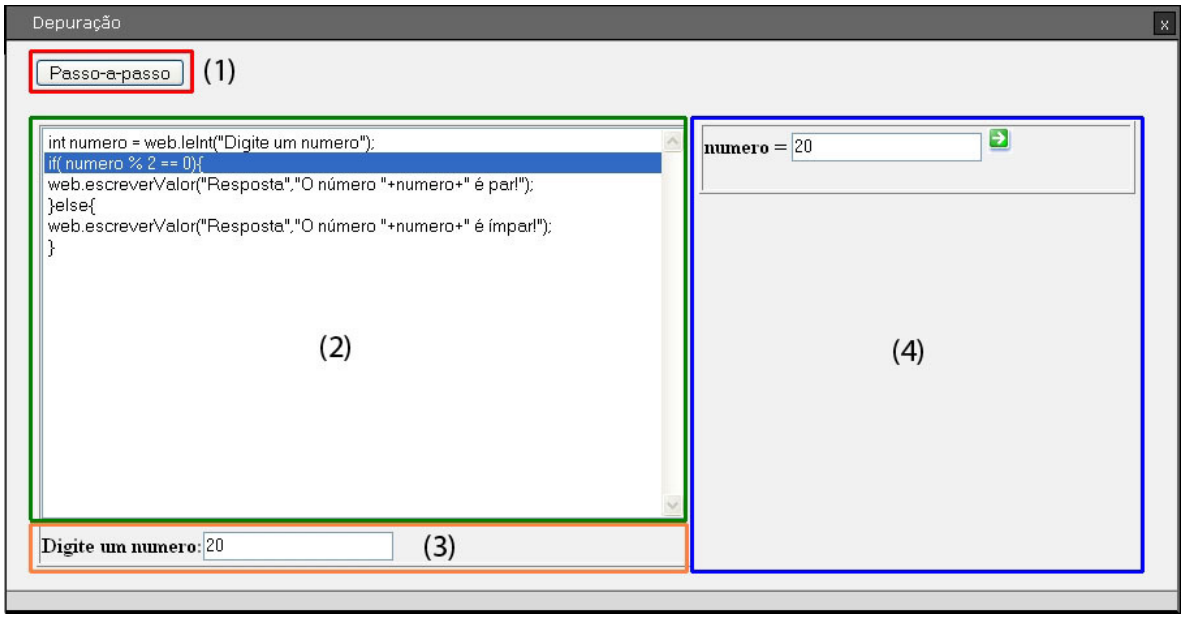


Figura 29 – Depuração do programa do usuário

A Figura 30 apresenta a saída de dados do programa realizada na janela de depuração.

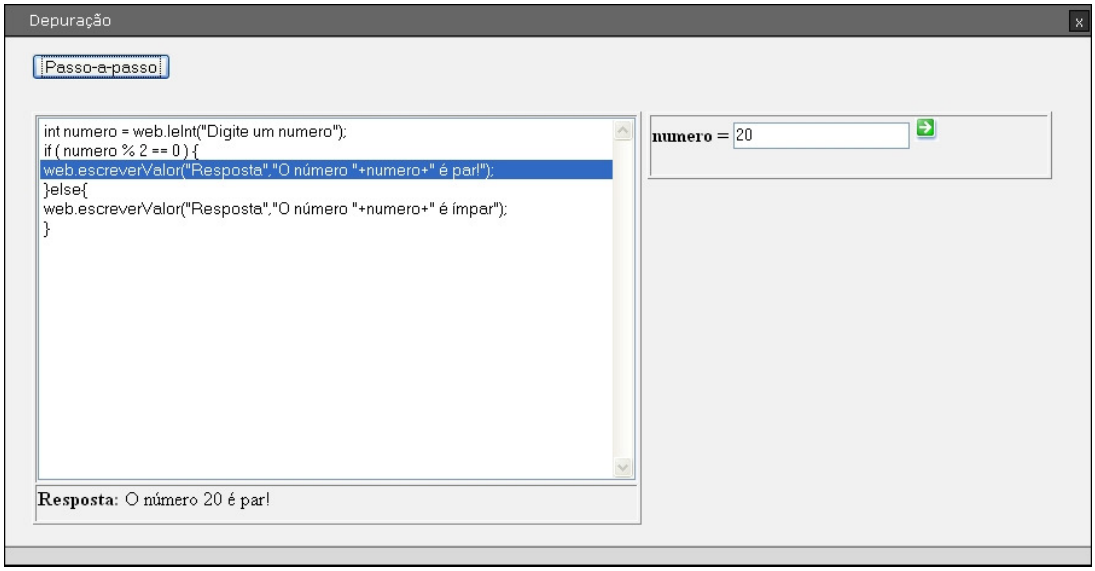


Figura 30 – Saída de dados na depuração do programa do usuário

Depois que o exercício tenha sido resolvido pelo aluno, ele deverá selecionar o link

`Finalizar` na página principal. Isso fará com que o exercício possa ser corrigido pelo monitor, e o aluno não poderá mais alterá-lo.

3.4.3 Corrigindo o exercício

A Figura 31 apresenta o ambiente de programação utilizado pelo monitor.

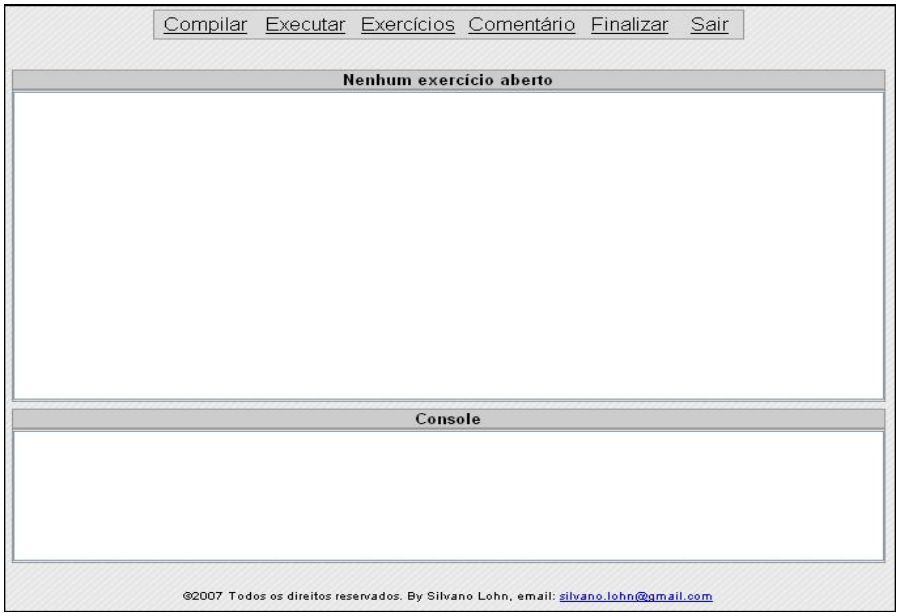


Figura 31 – Ambiente de programação utilizado pelo monitor

A figura 32 apresenta a janela de exercícios dos alunos no ambiente do monitor. Nessa janela podem ser vistos todos os exercícios já finalizados pelos alunos, é apresentado na janela o título do exercício, o nome do aluno que resolveu o exercício, e o botão que abre o exercício para ser corrigido.

Figura 32 – Janela de exercícios dos usuários no ambiente do monitor

Depois que o monitor já tenha compilado e executado o exercício do aluno, ou seja, se o exercício esteja correto o monitor deve selecionar o link `Finalizar` e o exercício estará

corrigido, mas se o exercício estiver incorreto, o monitor enviará um comentário ao exercício do usuário. A janela onde são enviados os comentários pode ser vista na Figura 33.

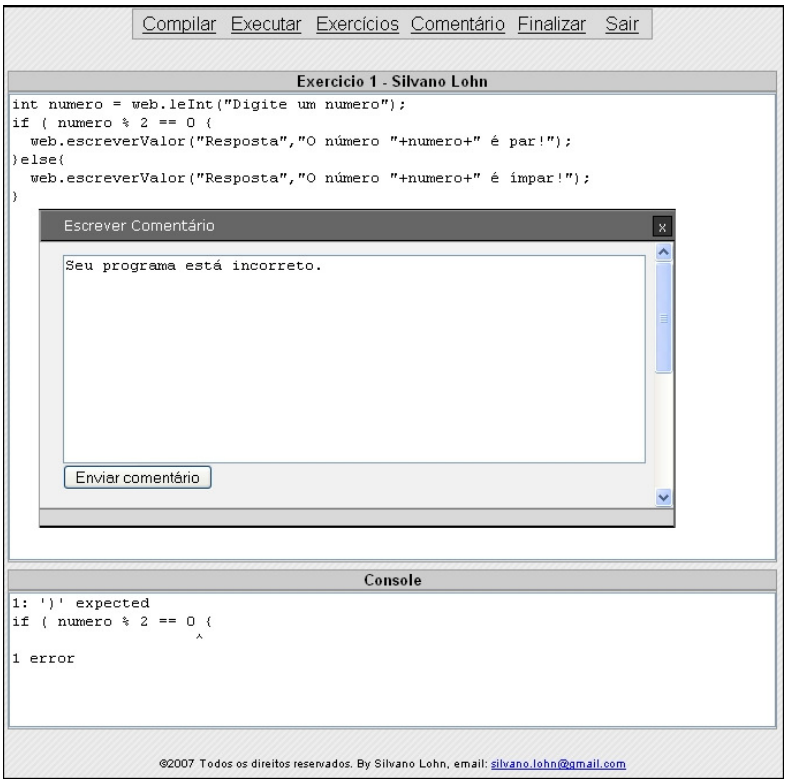


Figura 33 – Janela onde o monitor envia os comentários para o exercício do aluno

A Figura 34 apresenta a janela de comentários do aluno no ambiente de programação.

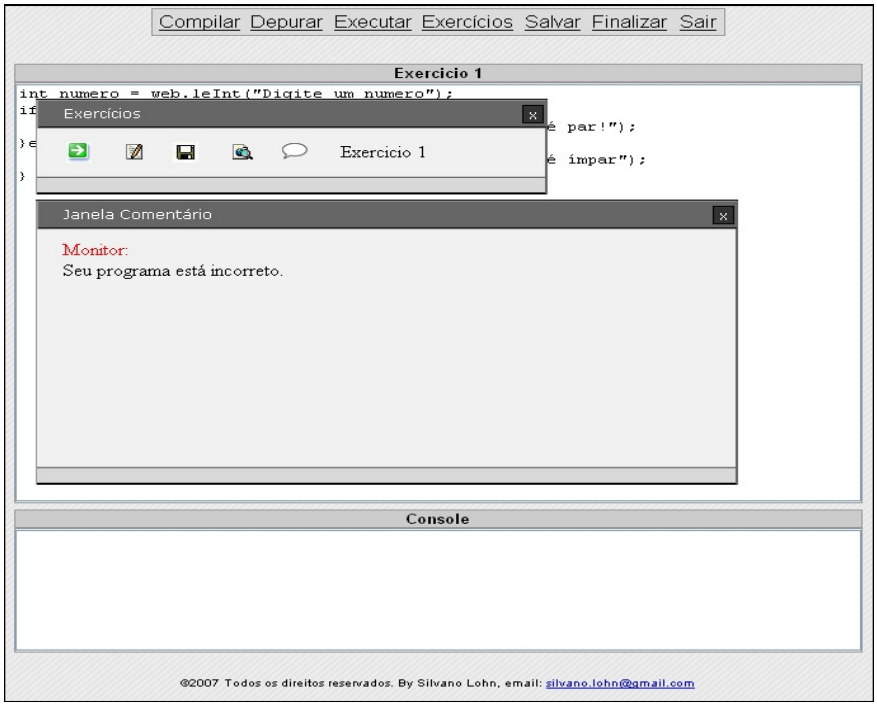


Figura 34 – Usuário visualizando o comentário enviado pelo monitor

A janela de comentários apresenta o autor e o seu comentário. Neste caso o aluno deverá refazer o seu programa e finalizar o exercício para o que monitor possa corrigir novamente.

3.5 IMPLANTAÇÃO

Em junho de dois mil e oito foi realizada a implantação do sistema no servidor campeche do LCI. Com a ajuda de um funcionário foram realizadas as seguintes ações para a configuração e implantação do sistema:

a) criação de uma base de dados no banco MySQL e a execução do *script* SQL;
b) criação de uma pasta no disco do servidor campeche que o sistema utilizará para criar os arquivos que contém os programas dos alunos;
c) implantação do sistema no servidor de aplicação Apache/Tomcat;
d) configuração do banco de dados e dos programas **java**, **javac** e **jdb** no sistema.

Durante a implantação do sistema, surgiram dois problemas:

a) como todo o sistema foi desenvolvido utilizando uma plataforma Microsoft e o servidor que foi implantado o sistema utiliza uma plataforma Linux, o sistema continha erros de *case sensitive* no acesso aos campos das tabelas do banco de dados. A solução foi alterar o *script* SQL do banco como também a compilação de uma nova versão do sistema. Fazendo assim com que o sistema execute em ambas as plataformas;
b) o servidor campeche possui um *firewall* web chamado de ModSecuriry. Esse programa estava bloqueando as requisições DWR utilizadas pelo sistema. A solução foi alterar as regras do *firewall* para que as requisições fossem completadas.

Depois da implantação realizada, o sistema estava disponível em Lohn (2008).

3.6 VALIDAÇÃO

No mesmo dia da implantação do sistema foi realizada uma validação parcial do sistema no laboratório D006 do Campus IV da FURB.

Essa validação parcial tem como objetivo:

a) realizar registros de usuários em todos os níveis (aluno, monitor e professor);
b) realizar testes de compatibilidade entre os navegadores Firefox e Microsoft Internet Explorer;
c) realizar testes de compilação, depuração e execução de programas;
d) realizar testes de performance do sistema;
e) realizar testes dos recursos do sistema com mais de um usuário logado.

Para a validação foram registrados quatro usuários no sistema: um professor, um monitor e dois alunos chamados de Adilson e Silvano. O usuário professor fez o cadastro de dois exercícios para serem resolvidos pelos alunos. A Figura 35 apresenta os dois exercícios cadastrados no sistema.

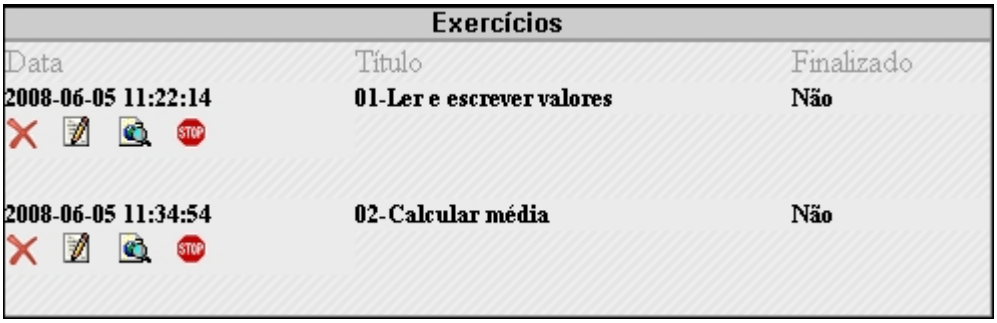


Figura 35 – Exercícios do estudo de caso cadastrados no sistema

O Quadro 46 apresenta o enunciado do exercício "01-Ler e escrever valores".

```
Faça um programa que solicite ao usuário o seu nome (como String),
sua idade (como int), sua renda (como double), se mora sozinho (como
boolean) e estado civil (solteiro, casado, divorciado ou viúvo).

Em seguida apresente esses dados.
```

Quadro 46 – Enunciado do exercício 01-Ler e escrever valores

O Quadro 47 apresenta o enunciado do exercício "02-Calcular média".

```
Faça um programa que solicite 4 números inteiros e no
final apresente a média deles.
```

Quadro 47 – Enunciado do exercício 02-Calcular média

O usuário Adilson utilizou o navegador Microsoft Internet Explorer, já o usuário Silvano utilizou o navegador Firefox. Nos testes os dois usuários realizaram a resolução dos exercícios compilando programas corretos e incorretos, realizando tanto a depuração como a execução dos mesmos. Os exercícios foram finalizados pelos usuários Silvano e Adilson e foram corrigidos pelo usuário monitor. O monitor realizou a correção dos exercícios, como também fez o envio de um comentário ao usuário Silvano, informando que o seu programa estava incorreto.

Dessa maneira foi realizada a validação parcial do sistema, chegando a conclusão que o sistema poderia ser utilizado por um número maior de usuários. Sendo assim, foi realizada uma validação completa do sistema. A turma matutina da disciplina de Programação de Computadores do curso de Bacharel em Ciências da Computação composta por vinte e dois alunos, utilizaram o sistema no mesmo dia da sua implantação.

Foi realizada uma breve apresentação do sistema e do seu funcionamento aos alunos. Depois os alunos realizaram o seu registro no sistema e tinham como trabalho resolver os dois exercícios apresentados na Figura 35. A Figura 36 apresenta uma foto da sala de aula durante a resolução dos exercícios pelos alunos.

Figura 36 – Turma da disciplina de Programação de Computadores utilizando o sistema

Depois da resolução dos exercícios pelos alunos, foi aplicado a eles um questionário com perguntas objetivas sobre os recursos e funcionalidades do sistema. O questionário é apresentado no Anexo A. Os resultados obtidos na validação do sistema, como também o resultado do questionário, são apresentados na seção 3.7.

3.7 RESULTADOS E DISCUSSÃO

Mesmo com os alunos não tendo nenhum contato anterior com o sistema, através de conversas pessoais e do questionário aplicado aos alunos, constatou-se que o sistema teve uma ótima aceitação.

Durante a validação foram encontrados os seguintes problemas:

a) apesar do navegador Firefox realizar todos os recursos e funcionalidades do sistema, ele apresenta um problema com o cursor no componente HTML `textarea` utilizado para a edição do programa. Com isso se torna impossível realizar a edição de um programa no navegador Firefox;
b) o navegador Microsoft Internet Explorer apresenta um problema com o redimensionamento das janelas do sistema. Algumas janelas são exibidas maiores e outras menores do que realmente deveriam ser;
c) durante a validação do sistema, um aluno desenvolveu códigos "maliciosos", esses códigos possuíam comandos de repetição e uma grande quantidade de comandos de saída de dados, com isso o servidor de aplicação Apache/Tomcat apresentou uma grande lentidão, deixando inutilizável o sistema até que o servidor de aplicação fosse reinicializado.

Os alunos não só utilizaram o sistema, como também apresentaram algumas sugestões de melhorias, como segue:

a) quando o exercício for finalizado pelo aluno, o sistema deve salvar o seu programa automaticamente;
b) o sistema não pode permitir que o aluno digite um programa sem que exista um exercício aberto;
c) o sistema deveria salvar o programa do aluno automaticamente de tempos em tempos.

A Figura 37 apresenta graficamente o resultado das cinco questões aplicadas aos alunos pelo questionário apresentado no Anexo A. Já o Quadro 48 apresenta o resultado das questões de forma tabular.

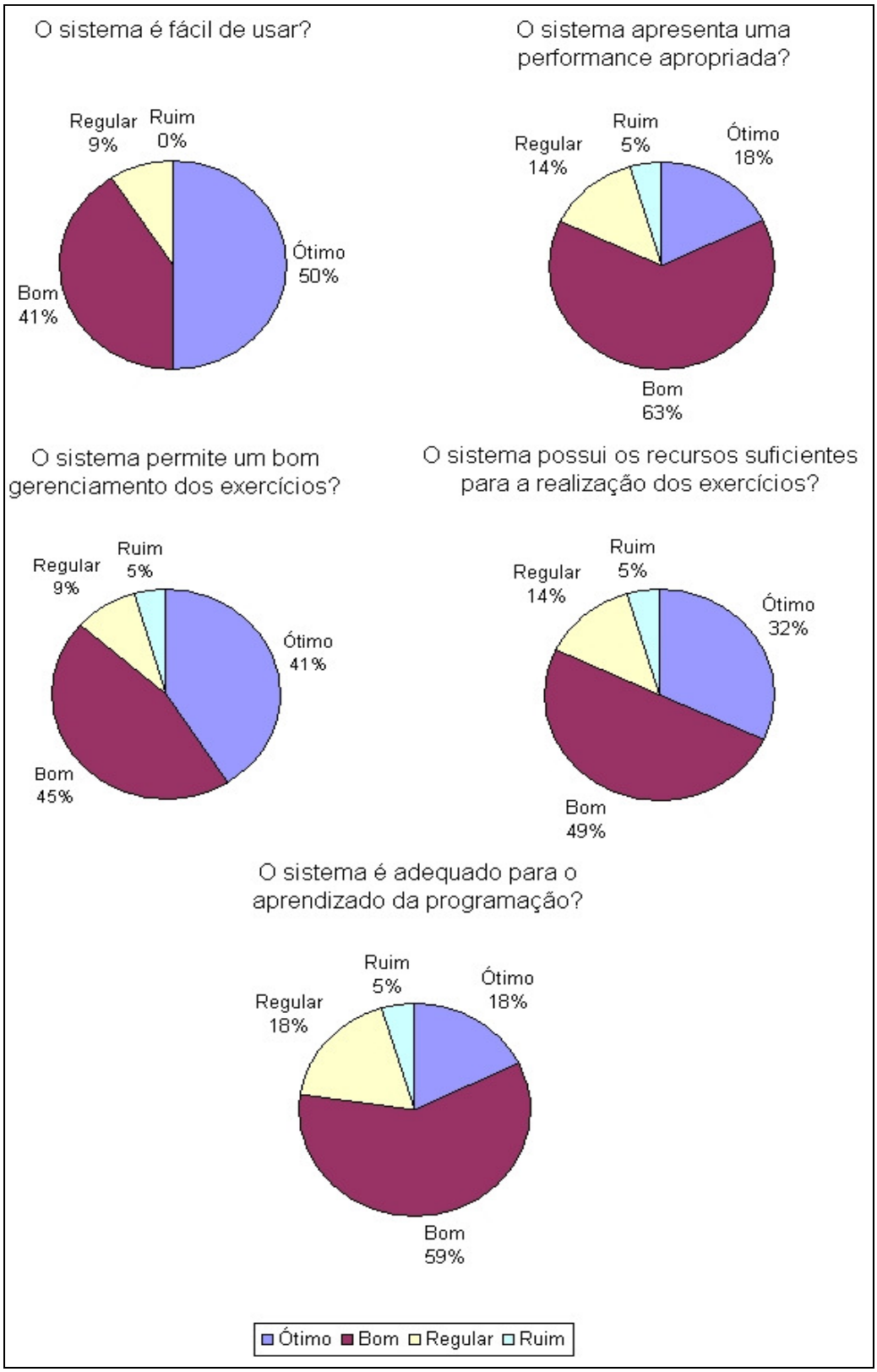


Figura 37 – Gráficos dos resultados do questionário aplicado aos alunos

Pergunta	Ótimo	Bom	Regular	Ruim
1. O sistema é fácil de usar?	50%	41%	9%	0%
2. O sistema apresenta uma performance apropriada?	18%	63%	14%	5%
3. O sistema permite um bom gerenciamento dos exercícios (Abrir, Editar, Salvar, Finalizar)?	41%	45%	9%	5%
4. O sistema possui os recursos suficientes para a realização dos exercícios?	32%	49%	14%	5%
5. O sistema é adequado para o aprendizado da programação?	18%	59%	18%	5%

Quadro 48 – Resultado tabular do questionário aplicado

No Quadro 49 é apresentada uma comparação dos principais recursos do ambiente desenvolvido em relação aos trabalhos correlatos.

Como pode ser visto no Quadro 48, todos os trabalhos realizam a compilação e a execução de programas, mas somente o ambiente Webide e o Gyre realizam a depuração. Também pode ser visto que somente o trabalho desenvolvido fornece ao usuário a programação com a linguagem Java, como também somente ele possui um ambiente integrado para a resolução e a correção de exercícios.

Comparação			
	Webide	Gyre	CodeIDE
Compilação de programas	X	X	X
Execução de programas	X	X	X
Depuração de programas	X	X	
Programação em Java	X		
Ambiente integrado para criação, resolução e correção de exercícios.	X		

Quadro 49 – Comparação do sistema com os trabalhos correlatos

4 CONCLUSÕES

O desenvolvimento do sistema web foi voltado para o auxílio do aprendizado da programação, com objetivos de facilitar o trabalho do professor na criação e correção dos exercícios. O ambiente permite que tanto o professor como o monitor possa estar corrigindo os exercícios em um mesmo momento, trazendo assim um ganho na velocidade das correções dos exercícios.

O ambiente traz aos alunos uma ferramenta de fácil utilização que possui recursos fundamentais para que aprenda e desenvolva os seus programas. Um dos maiores benefícios do ambiente é fazer com que o aluno possa resolver os exercícios em qualquer lugar que possui um computador com acesso a internet, seja em casa, no trabalho ou na própria instituição de ensino, podendo começar o exercício em um local e terminar em outro. Isso mostra que o ambiente pode ser utilizado durante as aulas da disciplina ou como uma ferramenta extra-classe. Dessa maneira o ambiente também pode ser utilizado como uma ferramenta de ensino a distância, onde o professor e os alunos não estarão juntos em um mesmo local, mas estarão interligados pela internet. Em relação a uma IDE, o ambiente desenvolvido permite o acompanhamento da resolução dos exercícios do aluno por parte do professor e do monitor, pois todos os dados relacionados aos exercícios do alunos estão armazenados junto ao servidor onde o ambiente foi implantado.

Todos os objetivos do trabalho foram alcançados com êxito. Apesar de ser utilizado por enquanto com poucos alunos, durante a validação o trabalho mostrou ser um sistema com grande potencial na área do ensino, podendo trazer benefícios aos seus usuários e ser facilmente implantado e adequado para a utilização. O trabalho teve uma ótima aceitação por parte dos usuários, podendo no futuro ser utilizado como uma ferramenta didática nas disciplinas de introdução a programação, programação de computadores e algoritmos.

O estudo das ferramentas e bibliotecas apresentadas foi de suma importância para o desenvolvimento do trabalho. Elas não apresentaram nenhum problema ou dificuldade para a implementação do sistema. Os recursos principais do sistema como a compilação, execução e depuração funcionaram de forma esperada e satisfatória, fazendo com que o trabalho proposto pudesse ser desenvolvido e implantado para a utilização.

4.1 EXTENSÕES

Como extensões da ferramenta sugerem-se:

a) desenvolver uma biblioteca para que o usuário possa criar interfaces gráficas para o seu programa;
b) desenvolver um editor de programas que possui recursos de auto-completar, numeração das linhas, realce de código sintático e semântico através de cores;
c) desenvolver um novo depurador onde o usuário possa escolher pontos de parada no programa para a depuração, visualizar a pilha das chamadas dos métodos e visualizar o tipo das variáveis utilizadas pelo programa;
d) desenvolver maneiras para o professor e o monitor reutilizarem os seus comentários para problemas comuns encontrados nos exercícios dos alunos;
e) desenvolver recursos para o sistema criar *logs* de acompanhamento das atividades realizadas pelos alunos, monitores e professores como também uma ferramenta para o monitoramento do próprio sistema e do uso rede;
f) realizar uma análise das sugestões e as possíveis implementações sugeridas pelos alunos na seção 3.7;
g) Disponibilizar o sistema em um repositório de códigos fonte *open sources*, podendo com isso ser desenvolvido por outras pessoas e ganhando novos recursos e funcionalidades.

REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, Maria E. B. **Educação a distância na internet**: abordagens e contribuições dos ambientes digitais de aprendizagem. São Paulo, 2003. Disponível em: <http://www.scielo.br/pdf/ep/v29n2/a10v29n2.pdf>. Acesso em: 20 maio 2008.

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML.** Rio de Janeiro: Campus, 2002.

CHAIM, Marcos L.; JINO, Mario; MALDONADO, José C. **Processo de depuração depois do teste**: definição e análise. [S.l], 2002. Disponível em: <www.cnptia.embrapa.br/modules/tinycontent3/content/2002/bolpesq5.pdf>. Acesso em: 20 maio 2008.

CONALLEN, Jim. **Desenvolvendo aplicações web com UML**. São Paulo: Campus, 2003.

GETAHEAD. **DWR:** easy AJAX for Java. [S.l], 2004. Disponível em: <http://getahead.org/dwr/overview/dwr>. Acesso em: 20 maio 2008.

GYRE. **Web-based IDE and debugger for rails**. [S.l], 2007. Disponível em: <http://gyre.bitscribe.net>. Acesso em: 20 maio 2008.

GOODMAN, Danny; MORRISON, Michael. **JavaScript bible**. 5. ed. Indiana: Wiley Publishing, 2004.

HORSTMANN, Cay. **Big Java**. São Paulo: Bookman, 2006.

LOHN, Silvano. **WebIde 1.0**. [S.l], 2008. Disponível em: <http://campeche.inf.furb.br/tchuky>. Acesso em: 5 jun. 2008.

MCCOMB, Gordon. **JavaScript sourcebook.** São Paulo: Makron, 1997.

MOZILLA, **Ajax**. [S.l], 2008. Disponível em: <http://developer.mozilla.org/pt/docs/ajax>. Acesso em: 20 maio 2008.

PIMENTEL, Edson P. et al.. Avaliação Contínua da Aprendizagem, das Competências e Habilidades em Programação de Computadores. In: CONGRESSO NACIONAL DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 23., 2003, Campinas. Anais...Campinas: UNICAMP, 2003.

SAMIR, Franklin. **AJAX e DWR**. Porto Alegre, 2006. Disponível em: <http://portaljava.com/franklin/artigos/ajax_e_dwr.pdf >. Acesso em: 20 maio. 2008.

SEVERO, Carlos E. P. **NetBeans IDE 4.1**: para desenvolvedores que utilizam a tecnologia Java. Rio de Janeiro: Brasport, 2005.

SITEHEART. **CodeIde**. [S.l], 2006. Disponível em: <http://www.codeide.com>. Acesso em: 20 maio 2008.

SUN MICROSYSTEMS. **JavaTM Platform Standard Edition 6**. [S.l], 2006. Disponível em: < http://java.sun.com/javase/6/docs/technotes/guides>. Acesso em: 20 maio 2008.

SUN MICROSYSTEMS. **JavaServer Pages**. [S.l], 2008. Disponível em: <http://java.sun.com/products/jsp/jstl/reference/docs/index.html>. Acesso em: 10 jun. 2008.

WIGGINS, Adam. **Gyre, web-based debugger for rails**. [S.l], 2007. Disponível em: <http://adam.blogs.bitscribe.net/category/gyre>. Acesso em: 20 maio 2008.

XAVIER, Gláucia M. C. **Estudo dos Fatores que Influenciam a Aprendizagem Introdutória de Programação**. [S.l], 2004. Disponível em: <http://www.uefs.br/erbase2004/documentos/weibase/Weibase2004Artigo002.pdf>. Acesso em: 20 jul. 2008.

ZIVIANI, Nivio. **Projeto de algoritmos**. 2. ed. São Paulo: Thomson, 2004.

APÊNDICE A – Descrição dos atributos do modelo entidade e relacionamento

No Quadro 50 – É apresentada a descrição dos atributos do modelo entidade relacionamento.

Tabela usuario	
cd_usuario	Código do usuário no sistema
nm_usuario	Nome do usuário
user	Nome do usuário para login
pass	Senha do usuário para login
tipo	Tipo do usuário (aluno, monitor, professor)
email	Email do usuário

Tabela comentario	
cd_comentario	Código do comentário
cd_usuario	Código do usuário que recebeu o comentário
ds_comentario	Comentário enviado ao usuário
cd_autor	Código do usuário que enviou o comentário

Tabela exercício_comentario	
cd_exercicio_comentario	Chave primaria utilizada como índice da tabela
cd_exercicio	Código do exercício
cd_comentario	Código do comentário

Tabela exercicio	
cd_exercicio	Código do exercício
nm_exercicio	Nome do exercício
ds_exercicio	Descrição do exercício
dt_exercicio	Data da criação do exercício
finalizado	Indicador de exercício finalizado

Tabela exercício_usuario	
cd_exercicio_usuario	Chave primaria utilizada como índice da tabela
cd_exercicio	Código do exercício
cd_usuario	Código do usuário
arquivo	Exercício do usuário
dt_entrega	Data da entrega do exercício
finalizado	Indicador de exercício do usuário finalizado

Quadro 50 – Descrição dos atributos do modelo entidade relacionamento

ANEXO A – Questionário aplicado aos alunos para a avaliação do sistema

No Quadro 51 – É apresentado o questionário aplicado aos alunos.

FURB – UNIVERSIDADE REGIONAL DE BLUMENAU
DSC – DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO
BCC – BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

AMBIENTE NA WEB PARA EXECUÇÃO E DEPURAÇÃO DE PROGRAMAS COM SINTAXE JAVA

QUESTIONÁRIO DE AVALIAÇÃO DO TRABALHO DE TCC PELO ALUNO

Nome da Disciplina: _____

Data: ___ / ___ / _____

O objetivo deste questionário é coletar as opiniões dos alunos sobre a funcionalidade e os recursos disponíveis no trabalho de TCC apresentado. Portanto, a seriedade nas respostas às questões é de suma importância para a avaliação do trabalho e de possíveis implementações futuras do mesmo.

Para responder às questões, utilize a seguinte escala de valores, marcando com X o conceito devido.

Pergunta	Ótimo	Bom	Regular	Ruim
1. O sistema é fácil de usar?				
2. O sistema apresenta uma performance apropriada?				
3. O sistema permite um bom gerenciamento dos exercícios (Abrir, Editar, Salvar, Finalizar)?				
4. O sistema possui os recursos suficientes para a realização dos exercícios?				
5. O sistema é adequado para o aprendizado da programação?				

Sugestões: _____

Orientando: SILVANO LOHN

Orientador: ADILSON VAHLDICK

Quadro 51 – Questionário aplicado aos alunos