UNIVERSIDADE FEDERAL DO RIO GRANDE - FURG
CENTRO DE CIÊNCIAS COMPUTACIONAIS – C3
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO
CURSO DE MESTRADO EM ENGENHARIA DE COMPUTAÇÃO

Dissertação de Mestrado

CODERUNNER: PROPOSTA DE INSERÇÃO DE BALANCEAMENTO DE CARGA DE SERVIDORES NO CONTEXTO DA FERRAMENTA EDUCACIONAL

Leonardo Santana Benevides

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia de Computação da Universidade Federal do Rio Grande. Como requisito parcial para obtenção do Grau de Mestre em Engenharia de Computação.

Orientador: Prof. Dr. André Luis Castro de Freitas

Rio Grande, 2019.

Ficha catalográfica

B465c Benevides, Leonardo Santana.
CodeRunner: proposta de inserção de balanceamento de carga de servidores no contexto da ferramenta educacional / Leonardo Santana Benevides. – 2019.
61 f.

Dissertação (mestrado) – Universidade Federal do Rio Grande – FURG, Programa de Pós-Graduação em Engenharia da Computação, Rio Grande/RS, 2019.
Orientador: Dr. André Luis Castro de Freitas.

1. CodeRunner 2. Ferramentas de Ensino de Programação 3. Pensamento Computacional 4. Balanceamento de Carga 5. Replicação de Servidores I. Freitas, André Luis Castro de II. Título.

CDU 004:37

Catalogação na Fonte: Bibliotecário José Paulo dos Santos CRB 10/2344

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO RIO GRANDE
CENTRO DE CIÊNCIAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO
CURSO DE MESTRADO EM ENGENHARIA DE COMPUTAÇÃO

DISSERTAÇÃO DE MESTRADO

CODERUNNER: PROPOSTA DE INSERÇÃO DE BALANCEAMENTO DE CARGA DE SERVIDORES NO CONTEXTO DA FERRAMENTA EDUCACIONAL

LEONARDO SANTANA BENEVIDES

Banca examinadora:

Profa. Dra. Diana Francisca Adamatti

Prof. Dr. Luis Otoni Ribeiro

Profa. Dra. Regina Barwaldt

Prof. Dr. André Luis Castro de Freitas
Orientador

ABSTRACT

BENEVIDES, Leonardo Santana. **Coderunner: Proposal to Insert Server Load Balancing in the Context of Educational Tool**. 2019. Dissertação (Mestrado) – Programa de Pós-Graduação em Engenharia de Computação. Universidade Federal do Rio Grande, Rio Grande.

We conducted a qualitative bibliographic research on CodeRunner, in order to analyze the tool from its architecture to how the communication processes CodeRunner - MOODLE - Jobesandbox. Thus, we identified that although there are several educational tools for the purpose of autonomous correction of computational exercises CodeRunner proved to be much more flexible and can be used in different computing contexts, from the simplest to the most complex as finite state machines.

From this opinion, we identified that although the tool has a complex architecture, it becomes limited as to the server responsible for the correction of computational exercises. Since CodeRunner receives only one result table, it is up to an external server to perform this correction, making it subject to failures such as overload. Thus, we researched authors who entered the distributed systems environment with a focus on process balancing, seeking to understand the mechanisms of these systems and thus develop a load balancing algorithm to manage the communication between CodeRunner and the external server.

Keywords: CodeRunner, Programming Teaching Tools, Computational Thinking, Load Balancing, Server Replication.

RESUMO

BENEVIDES, Leonardo Santana. **Coderunner: Proposta de Inserção de Balanceamento de Carga de Servidores no Contexto da Ferramenta Educacional**. 2019. Dissertação (Mestrado) – Programa de Pós-Graduação em Engenharia de Computação. Universidade Federal do Rio Grande, Rio Grande.

Realizamos uma pesquisa qualitativa de base bibliográfica sobre o CodeRunner, a fim de analisar a ferramenta desde a sua arquitetura até como se dá os processos de comunicação CodeRunner - MOODLE - Jobesandbox. Dessa maneira, identificamos que apesar de existirem várias ferramentas educacionais com o propósito de correção autônoma de exercícios computacionais o CodeRunner se mostrou muito mais flexível, podendo ser utilizado em diferentes contextos na computação, desde os mais simples até os mais complexos como máquinas de estados finitos.

A partir desse parecer, identificamos que apesar de a ferramenta possuir uma complexa arquitetura ela torna-se limitada quanto ao servidor responsável pela correção dos exercícios computacionais. Já que o CodeRunner recebe apenas uma tabela de resultados, ficando a cargo de um servidor externo realizar essa correção, tornando-o sujeito a falhas como sobrecarga. Dessa forma, pesquisamos autores que adentaram ao ambiente de sistemas distribuídos com enfoque em balanceamento de processos, buscando entender os mecanismos desses sistemas e assim desenvolver um algoritmo de balanceamento de carga para gerenciar a comunicação entre o CodeRunner e o servidor externo.

Palavras-Chave: CodeRunner, Ferramentas de Ensino de Programação, Pensamento Computacional, Balanceamento de Carga, Replicação de Servidores.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CPU	Central Processor Unit
PHP	Hypertext Preprocessor
HTML	HyperText Markup Language
MATLAB	MATrix LABoratory
Web	World Wide Web
URL	Uniform Resource Locator
GUI	Graphical User Interface
MOJO	Módulo de Integração com os Juízes Online
MOODLE	Modular Object Oriented Dynamic Learning Environment
TDIC	Tecnologias Digitais da Informação e Comunicação
AVA	Ambiente Virtual de Aprendizagem

SUMÁRIO

1. INTRODUÇÃO

No ensino das disciplinas introdutórias de programação, normalmente nota-se uma dificuldade inerente por grande parte dos alunos, pois essas disciplinas constituídas de conceitos normalmente intangíveis e até certo ponto inesperados, como um profundo pensamento abstrato necessário para se ter um real entendimento de como um laço "loop for" ou uma "chamada recursiva" de fato funciona. Soma-se ainda a isso, a metodologia tradicional empregada por cursos e disciplinas com o uso apenas de exposições orais e escritas em slides.

Nesse sentido, o trabalho aqui desenvolvido apresenta inicialmente algumas análises do que seria o pensamento computacional pela visão de alguns pesquisadores. Realiza a análise do CodeRunner e de algumas ferramentas similares pois tratam de plataformas voltadas ao ensino-aprendizagem da lógica de programação. Busca entender todos os mecanismos inseridos em suas arquiteturas e de como é realizado os processos de envio do exercício pelo aluno até o retorno automático da resposta.

O CodeRunner por ser altamente escalável e flexível torna-se mais uma forma de incentivar os alunos em busca do conhecimento, pois como é uma ferramenta totalmente autônoma e colaborativa os alunos podem até trocar informações e dicas entre si, incluindo professores. Dessa forma, acrescentar conceitos de sistemas distribuídos para tornar a ferramenta mais robusta e eficiente vem ao encontro do que inicialmente propõe o CodeRunner, que é a melhoria do ensino computacional a qualquer momento e horário.

De acordo com o avanço do trabalho, foi proposto o uso de técnicas de balanceamento de carga em servidores, a fim de minimizar o risco da ferramenta ficar inacessível durante seu uso no ambiente educacional. Para tal, sendo necessário realizar pesquisas e o estudo sobre algoritmos de balanceamentos de processos e técnicas de replicação de servidores.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Analisar a ferramenta computacional MOODLE CodeRunner e como se dá à troca de requisições entre a ferramenta e o servidor externo. A partir dessa análise propor a inserção de técnicas de balanceamento de carga para minimizar falhas decorrentes de erro de comunicação ou evitar sobrecarga no servidor.

1.1.2 Objetivos específicos

As etapas da construção desse trabalho contemplam os seguintes objetivos específicos:

1. Pesquisar o pensamento computacional e seus diferentes entendimentos;
2. Realizar uma pesquisa de base bibliográfica sobre o CodeRunner;
3. Analisar as trocas de informações entre a plataforma MOODLE e o servidor utilizado para compilar e executar os códigos.
4. Demonstrar como o MOODLE recebe e exibe os resultados gerados pela ferramenta.
5. Propor o conceito de replicação de servidores para a ferramenta CodeRunner, usando para isso técnicas de balanceamento de carga em servidores.

1.2 PROCEDIMENTOS METODOLÓGICOS

Foi realizada neste trabalho uma pesquisa qualitativa de base bibliográfica, tal que, foi realizado a revisão de alguns artigos relacionados ao pensamento computacional e seus diferentes entendimento, sobre a ferramenta de ensino-aprendizagem de lógica de programação CodeRunner e ferramentas similares.

A pesquisa incluiu ainda técnicas de balanceamento de cargas de servidores e seus conceitos, a fim de tornar o CodeRunner tolerante a falhas

de sobrecarga, sendo necessário analisar as características da arquitetura e suas funcionalidades.

No primeiro momento, foram definidas as palavras-chave para efetuar a busca de artigos nos repositórios de dados científicos. Dessa maneira, para realizar as pesquisas usamos as expressões CodeRunner, Ferramentas de Ensino de Programação, Pensamento Computacional, Balanceamento de Carga e Replicação de Servidores em bases de dados nacionais, já nas bases de nível internacional foram usados termos como CodeRunner, Programming Teaching Tools, Computational Thinking, Load Balancing, Server Replication.

Em todos os repositórios de dados foram feitas buscas utilizando combinações com o conector AND a fim de obter mais resultados positivos, como por exemplo, Ambiente Virtuais de Aprendizagem AND CodeRunner.

Por fim, tais informações foram filtradas e coletadas para uma posterior análise dos dados que culminou na elaboração deste trabalho.

2. EMBASAMENTO TEÓRICO

2.1 Pensamento computacional

A concepção de que a programação de computadores reflete diretamente em nossa mente, nos ajudando a pensar não é recente. Quando a linguagem Logo foi desenvolvida nos anos 1960, Papert vislumbrava o quão importante seria essa atividade para a construção de conhecimento e o desenvolver do pensamento.

Assim, em 1971, a argumentação de que a computação é capaz de ter "um impacto profundo por concretizar e elucidar muitos conceitos anteriormente sutis em psicologia, linguística, biologia, e os fundamentos da lógica e da matemática" (PAPERT, 1971, p. 2). Assim, existe essa possibilidade, pois oportuniza de forma geral a criança a ter uma capacidade "de articular o trabalho de sua própria mente e, particularmente, a interação entre ela e a realidade no decurso da aprendizagem e do pensamento" (p. 3).

O pensamento da criança auxiliado pela computação evidenciou-se no livro Mindstorms, PAPERT (1980) em que ele propôs que o ato de programar na linguagem Logo incentivaria o que ele denominou de “Powerful ideas” e “Procedural knowledge”. Então, na visão desse pesquisador, as pessoas deveriam utilizar os computadores como um suporte técnico a fim de ajudá-las a “pensar com” o mundo digital e “pensar sobre” o seu próprio pensar.

A expressão “pensamento computacional” ou “computational thinking” tornou-se conhecido com o artigo de Jeannette M. Wing, em 2006, de acordo com o entendimento de Jeannette o “pensamento computacional se baseia no poder e nos limites de processos de computação, quer eles sejam executados por um ser humano ou por uma máquina” WING (2006, p. 33)

A pesquisadora ainda diz que é fundamental a todos possuírem a habilidades e competências do pensamento computacional. Segundo ela, à leitura, escrita e aritmética, é preciso acrescentar o pensamento computacional à capacidade analítica de cada criança WING (2006).

Nas palavras de WING (2016):

> Pensamento computacional é usar raciocínio heurístico na descoberta de uma solução. É planejar, aprender e agendar na presença da incerteza. É pesquisar, pesquisar e pesquisar mais, resultando em uma lista de páginas da web, uma estratégia para vencer um jogo ou um contraexemplo. Pensamento computacional é usar quantidades imensas de dados para aumentar a velocidade da computação. É fazer concessões entre tempo e espaço e entre poder de processamento e capacidade de armazenamento. (Wing, 2016, p. 3)

2.2 AVAs e ferramentas para o ensino da lógica computacional

Ambiente Virtual de Aprendizagem (AVA) constitui-se em uma alternativa de mídia que está sendo empregada para realizar a mediação do processo ensino-aprendizagem. A democratização da Internet, na década de 90, propiciou o desenvolvimento de ambientes virtuais de aprendizagem por meio dos quais a troca de informação entre os partícipes poderia ocorrer em qualquer lugar, a qualquer momento na categoria de um para um, um para muitos, muitos para um e muitos para muitos Moraes (2004).

Ambientes Virtuais de Aprendizagem (AVA) são softwares educacionais via internet, destinados a apoiar as atividades de educação tanto a distância como presenciais. Estes softwares oferecem um conjunto de tecnologias de informação e comunicação, que permitem desenvolver as atividades no tempo, espaço e ritmo de cada participante.

Os ambientes virtuais de aprendizagem podem ser utilizados em:

- Atividades presenciais: possibilitando aumentar as interações para além da sala de aula;
- Atividades semipresenciais: nos encontros presenciais;
- Atividades à distância: oferecendo suporte para a comunicação e troca de informações e interação entre os participantes.

De acordo com Souza (2006), conforme citado por Ribeiro et al. (2007), ‘o advento tecnológico proporcionou as pessoas muitas plataformas de comunicação acessíveis na internet. Em determinados sistemas locados na web, obtemos funcionalidades diferentes encapsuladas em um único ambiente virtual, proporcionando um espaço participativo e apropriado para o desenvolvimento do conhecimento e compartilhamento de informações”.

Os ambientes virtuais de aprendizagem agregam várias tecnologias encontradas na Web para provê a comunicação, disponibilização de materiais e administração do curso. O conjunto de funcionalidades que cada ambiente possui é estabelecido pelos requisitos definidos em cada ambiente. Segundo Gonzáles (2005), as funcionalidades dos ambientes virtuais de aprendizagem podem ser organizadas em quatro grupos de ferramentas: de Coordenação, de Comunicação, de Produção dos Alunos ou de Cooperação e de Administração.

Ferramentas de coordenação servem de suporte para a organização de um curso são utilizadas pelo professor para disponibilizar informações aos alunos, tanto informações das metodologias do curso (procedimento, duração, objetivos, expectativa, avaliação) e estrutura do ambiente (descrição dos recursos, dinâmica do curso, agenda, etc.), quanto informações pedagógicas: material de apoio (guias, tutoriais), material de leitura (textos de referência, links interessantes, bibliografia e etc.) e recurso de perguntas frequentes (reúne as perguntas mais comuns dos alunos e as respostas correspondentes do professor).

- Ferramentas de Comunicação, que englobam fóruns de discussão, bate-papo, correio eletrônicos e conferência entre os participantes do ambiente têm o objetivo de facilitar o processo de ensino-aprendizagem e estimular a colaboração e interação entre os participantes e o aprendizado contínuo.
- Ferramentas de Produção dos Alunos ou de Cooperação oferece o espaço de publicação e organização do trabalho dos alunos ou grupos, através do portfólio, diário, mural e perfil (de alunos e/ou grupos).
- Ferramentas de Administração oferecem recursos de gerenciamento, do curso (cronograma, ferramentas disponibilizadas, inscrições, etc.), de alunos (relatórios de acesso, frequência no ambiente, utilização de ferramentas, etc.) e de apoio a tutoria (inserir material didático, atualizar agenda, habilitar ferramentas do ambiente, etc.).

Através delas é possível fornecer ao professor formador informações sobre a participação e progresso dos alunos no decorrer do curso, apoiando-os

e motivando-os durante o processo de construção e compartilhamento do conhecimento.

Durante o ensino da lógica computacional é possível observar o quanto se torna difícil para a maioria dos ingressantes em cursos da área, aprender conteúdos de Ciência da Computação principalmente disciplinas que envolvem algoritmos, lógica de programação e cálculo. Essas disciplinas exigem muitas vezes uma nova forma de pensar, requerem habilidades que quase sempre não foram desenvolvidas no ensino regular, levando o aprendiz a ter grandes dificuldades e consequentemente um mal desempenho durante o curso Barbosa (2011).

Do mesmo modo Kelleher et al. (2005), conforme citado por Goulart (2019), ‘afirmam que além dos desafios de aprender a formar soluções, programadores iniciantes também têm que aprender uma sintaxe rígida, o que pode ser muitas vezes desanimador. Logo, observa-se a necessidade de não apenas oportunizar o uso de equipamentos que a escola possua ou venha a adquirir, mas o de ensinar como se processam aquilo do qual fazemos uso, como por exemplo a aprendizagem através de Quiz, desafios através dos jogos, passeios interativos dentro da história, museus, tours geográficos, entre outros”.

Dessa forma, os avanços tecnológicos e a democratização do acesso à internet propiciaram que os cursos de tecnologia da informação utilizassem os ambientes virtuais de aprendizagem como mais uma forma de suporte ao processo de ensino aprendizagem para a área computacional. Dentre os AVA's mais conhecidos podemos citar o MOODLE.

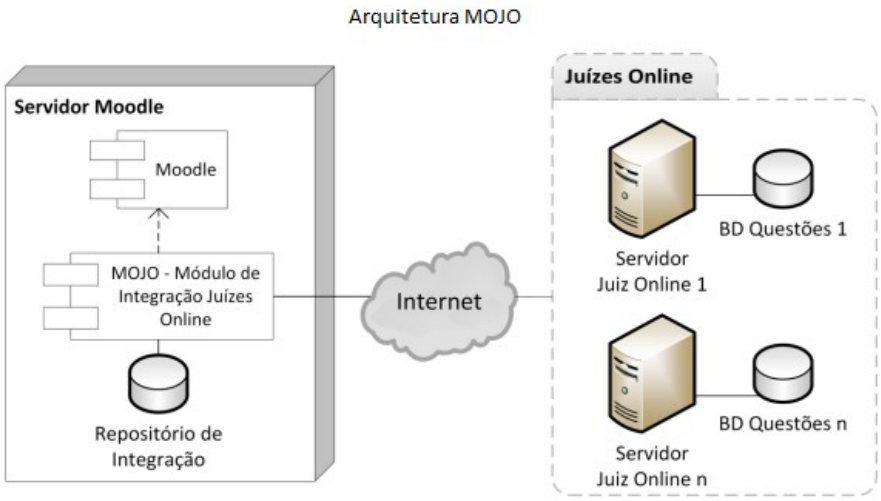
O MOODLE é um ambiente para EAD, mas que também é largamente utilizado em cursos presenciais. Ele foi desenvolvido com o enfoque pedagógico baseado na construção contextualizada do conhecimento, ou seja, o aprendizado se desenvolve através do compartilhamento colaborativo entre os indivíduos envolvidos. Ele armazena os conteúdos, os acessos e interações realizado durante todo o curso, e permite que o tutor gere o material a ser disponibilizado aos alunos de maneira livre, podendo ser apresentado em forma de arquivo ou em páginas de hipertexto.

Diante da consolidação destes tipos de plataformas no ensino educacional, surgiram algumas ideias de mesclar os ambientes virtuais de aprendizagem com ferramentas específicas para o ensino da lógica computacional. Assim, uniria as potencialidades dos AVA's em cursos tanto a distância como em cursos presenciais com a praticidade e eficiência dessas ferramentas.

Dentre algumas ferramentas utilizadas para o suporte ao ensino da lógica de programação, que usam um ambiente virtual de aprendizagem, podemos citar o MOJO a qual é uma ferramenta que engloba as técnicas dos Juízes Online ao MOODLE. Essa associação permitiu aos docentes manipular os recursos educacionais necessários, o qual pode citar notas, materiais e atividades. Ainda tem a vantagem de automatizar o processo de elaboração, submissão e avaliação de atividades, possibilitando assim que o professor consiga acompanhar com mais atenção e com mais eficiência os seus alunos.

Segundo Chaves et.al (2014), a ferramenta é a responsável pela comunicação e interação que irá ocorrer entre o MOODLE e os Juízes Online envolvidos nas operações. Na arquitetura geral de integração entre os ambientes, apresentada abaixo, na Figura 1, o MOODLE fornece a interface e o conjunto de funcionalidades necessárias à gestão e ao acompanhamento das atividades de programação, e o MOJO fornece a interação com os Juízes Online.

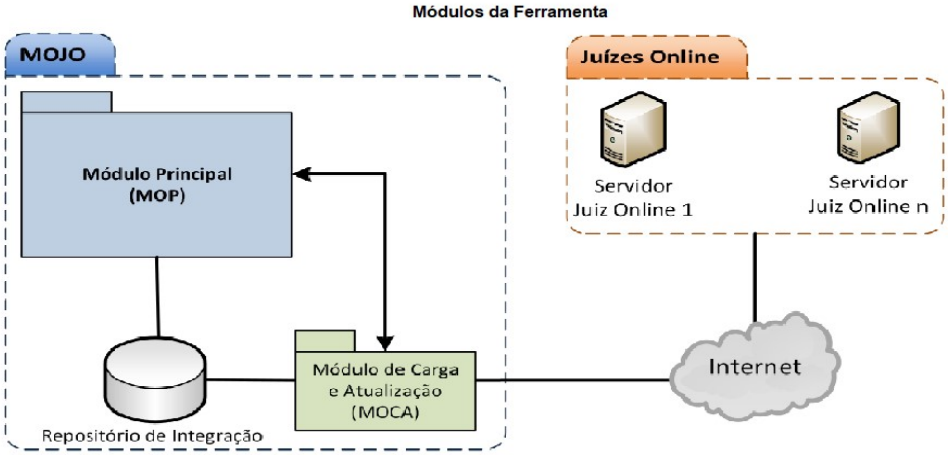
Figura 1. Arquitetura MOJO



Fonte: Mota et al. 2008

De acordo com Chaves et.al (2014), para a integração entre os dois ambientes, MOODLE e Juízes Online, a arquitetura da ferramenta é composta por dois módulos que se comunicam entre si. Tem-se o módulo principal e o módulo de carga e atualização, somado a isso, ainda, um repositório de integração, como mostra a figura 2.

Figura 2. Módulos de funcionamento MOJO



Fonte: Chaves et.al (2014)

O módulo principal é a parte que fica responsável por gerenciar, controlar e fornecer todas as funcionalidades essenciais para o funcionamento da ferramenta, como a disponibilização de questões e resultados. Já o modulo de carga e atualização fica responsável pelo carregamento e atualização de questões no repositório local da ferramenta. A MOJO incialmente suporta para a submissão de códigos-fontes as linguagens em C, C++, C#, Java e PHP. As principais funcionalidades da ferramenta MOJO são:

Importação de questões – É a função que está relacionada ao módulo de carga e atualização, sendo o procedimento responsável por realizar a importação das questões do repositório de questões de cada um dos Juízes Online. Ainda identifica e sinaliza quando o processo foi bem-sucedido ou não. As informações obtidas no processo de importação de cada questão são:

- Código da questão;
- Juiz responsável pela questão;
- Título da questão;
- Descrição da questão e;
- Linguagem de programação permitida.

Atualização de questões – Esta operação é responsável por verificar se houve a adição de novas questões no repositório de questões dos juízes, sendo também executada no modulo de carga e atualização. Existindo novas questões no repositório é feita a atualização no repositório local do MOJO. O processo da busca por novas questões considera o número de questões que havia anteriormente no repositório e verifica também o código da questão, para que não ocorra questões com códigos duplicados. O modulo ainda verifica e sinaliza se o processo de atualização foi bem-sucedido ou não.

Submissão de questões para resolução – Esse é o processo onde o professor poderá examinar e escolher, dentre várias listas de questões, aquela que ele julgar em conformidade com o nível e objetivo pretendido de avaliação do aluno. Escolhida a questão e feita a submissão para a possível resolução, a

questão ficará disponível e visível para os alunos o tempo que o professor julgar necessário para a resolução da mesma.

Submissão de código-fonte – Os dados do código-fonte para o processo de avaliação automática nos Juízes Online são enviados nesta operação. Quando submetido os dados do código-fonte, a ferramenta armazena essas informações em sua base de dados. O processamento desses códigos é realizado nos servidores dos juízes, e após esse processo, o feedback retornado é armazenado na base de dados para posteriormente ser utilizado para visualização. O método e forma que o processamento do código acontece em cada um dos juízes é de responsabilidade dos mesmos, sendo assim, o MOJO funciona apenas como uma rota de envio e recebimento do código para avaliação, recebendo uma resposta (feedback) do processo realizado por cada juiz.

Analisamos, ainda, a ferramenta JavaTool, que segundo Mota et.al (2008), também serve para auxiliar no processo de ensino-aprendizagem nos primeiros anos dos cursos de programação, usando a plataforma MOODLE, como um ambiente de ensino Web ou AVA.

Nessa ferramenta, o estudante poderá ver uma animação como exemplo de algoritmo presente no conteúdo dado pelo professor, assim como selecionar um exercício proposto pelo professor e desenvolver uma solução. A edição de código, compilação, depuração e visualização da animação do código são as opções presentes nessa ferramenta.

De acordo com Motta et.al (2008), o objetivo principal do ambiente é animar códigos de algoritmos desenvolvidos com uma sintaxe reduzida em um curso inicial de programação e desenvolvimento de algoritmos, sem abordar conhecimentos relacionados a programação orientada a objetos. Sendo assim, a ferramenta funcionaria como um suporte aos estudantes de maneira didática durante o início da aprendizagem de programação, utilizando Java como linguagem.

Isto posto, a ferramenta tende a ser mais uma forma de inserir os alunos que tem mais dificuldade em se relacionar com o pensamento abstrato, já que a abstração é profundamente necessária na atividade de um

programador, assim a funcionalidade de visualização dos programas, em formas de animações 2D, traduz esse pensamento abstrato para algo visível ao aluno.

Segundo Motta Et. Al. (2008), para a ferramenta Javatool atingir o objetivo proposto foi implementado um editor, o qual reconhece a sintaxe Java e assim, facilita a correção de erros de sintaxe. Logo após, dá a opção de visualizar o código em uma animação, por meio de um interpretador. A ferramenta permite à animação da maior parte dos conceitos que são aplicados em disciplinas de programação, permitindo que os dados sejam representados em vários formatos de exibição (binário, hexadecimal, etc.), dando suporte a visualização de um histórico da execução do programa de forma gráfica e também na forma de texto.

Na figura 3 abaixo é apresentada a interface do JavaTool que como a maioria dos aplicativos, contém um menu com funções básicas como: criar novo arquivo, abrir arquivo, salvar arquivo, editar arquivo, imprimir, um menu de ajuda, entre outras funções. O código é escrito na área de "edição" e os resultados de saída do programa, incluindo possíveis erros que serão exibidos no console. A partir do momento que o usuário clica o botão play no menu da área de "animação" o código é animado. A área designada "histórico" corresponde a uma descrição textual do que aconteceu durante cada passo da animação. A área "animação" também exibe o histórico na forma de imagens, podendo ser visualizadas por meio dos botões no menu de animação: seta para direita e seta para esquerda.

Figura 3. Interface Javatool

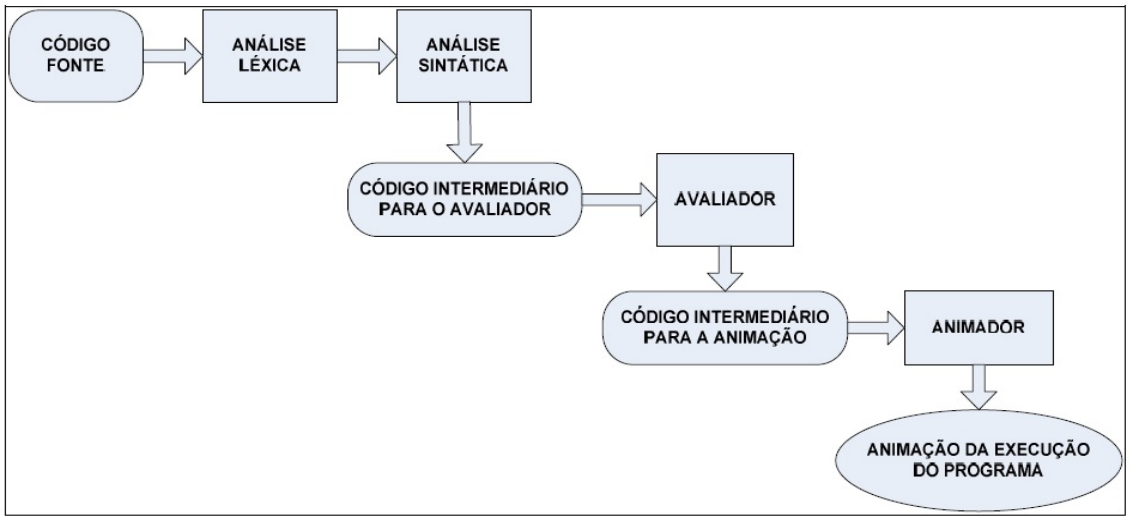
Fonte: Adaptado de Mota et al. 2008

Segundo Mota et.al (2008), para desenvolver essa ferramenta foi implementado um compilador próprio, afim de alcançar um resultado mais preciso nas análises dos códigos, composto de três partes: analisador léxico, analisador sintático e analisador semântico.

O analisador léxico foi gerado por meio de um gerador de analisador léxico e sintático (JavaCC) e foi remodelado para o sistema. Já os analisadores sintático e semântico foram desenvolvidos de modo que todos os parâmetros pudessem ser controlados para que se pudesse extrair o máximo de informação do código fonte para a posterior animação da execução. O desenvolvimento dos analisadores sintático e semântico foi importante também no que diz respeito à depuração do código fonte e tratamento de erros, para a

exibição de mensagens de erro de forma que o usuário fosse melhor direcionado na sua melhor correção.

Figura 4. Etapas de Processamento



Fonte: Mota et al. 2008

De acordo com Motta et.al (2008), a figura 4 acima, mostra as etapas de processamento que acontecem durante o funcionamento da ferramenta e em cada parte de seu processamento. Após a edição do código implementado pelo aluno, o código fonte passa pela análise léxica, onde é transformado em uma “fita de tokens”, que serve de entrada para a análise sintática.

Na análise sintática são reconhecidos erros da sintaxe da linguagem, nesse momento de transição é gerado um código intermediário para o avaliador. Então o avaliador é quem realiza as operações e atribuições especificadas no código fonte e ao mesmo tempo gera um código intermediário para a animação.

As principais funcionalidades da ferramenta JavaTool são:

Animação do código: ao clicar no botão play, após a edição do código fonte, o código é compilado e, caso não existam erros de compilação, a animação é exibida no painel de animação. A interface possui quatro áreas, baseando-se no trabalho de Moreno et.al (2004):

a) área para variáveis do tipo String e tipos primitivos, onde são declaradas as variáveis;

b) área para vetores, onde são declarados e inicializados os vetores;

c) área para constantes, de onde saem as constantes que são utilizadas no código fonte;

d) área para avaliação: onde são avaliadas as expressões e ficam registradas graficamente e textualmente as operações que são realizadas durante a animação (avaliação de expressões, entrada de dados, chamadas de métodos, etc.), a linha do código fonte que está sendo animada é destacada com a cor laranja.

Entrada de dados: durante a operação de leitura de dados via teclado, exibe-se uma janela para que o usuário entre com o valor. Ao terminar de digitar o valor o usuário pressiona a tecla Enter do teclado ou clica com o mouse sobre o botão OK para que a animação prossiga.

Visualização dos dados: durante a animação ou após, ao clicar em cima de uma variável no painel de animação, o usuário poderá visualizar o valor desta em diversos formatos (decimal, binário, hexadecimal ou octal), dependendo do tipo da variável. Esta funcionalidade serve, por exemplo, para que o aluno visualize operações com bits (operações de deslocamento de bits, etc.). Outro recurso é a utilização de cores diferentes para cada tipo (int, double, etc) e também para valores constantes, valores provindos da entrada de dados, valores resultantes da avaliação de uma expressão e valores das variáveis.

Histórico da execução: com essa função, o usuário não precisa executar a animação novamente caso queira rever algum detalhe, ficando todas as informações registradas visualmente e também na forma de texto. O usuário pode visualizar o histórico passo a passo, bastando clicar nos botões seta para direita e seta para esquerda.

Mensagens de erro: O JavaTool exibe, no painel de animação, os erros de compilação e de execução, caso existam no código fonte. São exibidas mensagens que contém uma descrição do erro (por exemplo: “divisão por

zero”), a linha que contém o erro e o caractere onde possivelmente se encontra o erro.

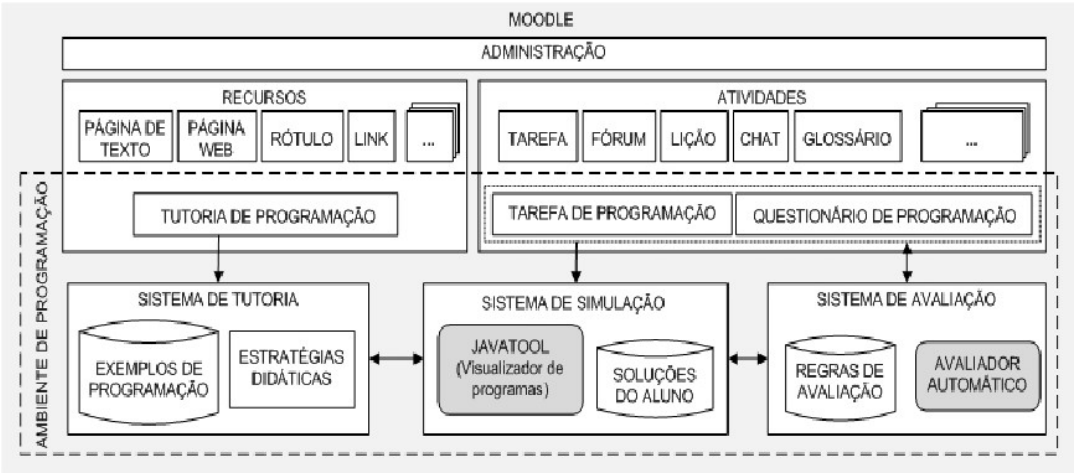
Segundo Mota et.al (2008), a ferramenta JavaTool suporta alguns recursos da linguagem Java, os recursos que são possíveis de se animar no JavaTool são:

- Todos os tipos primitivos(byte, short, int,long, float, double, char, boolean) e String;
- Arrays do tipo unidimensional.
- Estruturas de seleção if-then, if-then-else e switch.
- Estruturas de repetição while, do-while e for.
- Chamada de métodos.
- Alguns métodos da classe Math(ceil, floor, max, min, sqrt, pow, random, round) e da classe String(length, charAt, toUpperCase, toLowerCase, substring, trim, replace, valueOf).

De acordo com Mota et.al (2009), a junção do MOODLE e a ferramenta Javatool, possibilita que o espaço de desenvolvimento esteja acessível por meio da plataforma MOODLE. A possibilidade de inserir recursos e tarefas no MOODLE garantem que o material disponibilizado ao aluno promovam essa interação entre o usuário e as atividades propostas.

Como um recurso da ferramenta Javatool podemos citar o sistema de Tutoria como mostra abaixo a figura 5 junto com o sistema de Simulação e Avaliação de Programas pois são comparáveis tais quais as atividades do MOODLE.

Figura 5. Módulo de Integração com o MOODLE



Fonte: Mota Et. Al. (2009)

Segundo Mota et.al (2009):

> Foi desenvolvido um novo tipo de recurso denominado tutoria de programação e dois novos tipos de atividades (tarefa e questionário de programação). Nas tutorias de programação, o professor armazena o código exemplo e o estudante visualiza a animação do código exemplo. A tarefa de programação é incluída pelo professor como um problema que deve ser resolvido pelo estudante. (Mota et.al 2009, p. 6)

Nas tutorias de programação, o professor armazena o código exemplo e o estudante visualiza a animação do código exemplo. A tarefa de programação é incluída pelo professor como um problema que deve ser resolvido pelo estudante.

De acordo com Mota et. al (2009), o histórico das questões assim como um relatório contendo tentativas e notas pode ser observado na figura 6. A nota recebida pelo aluno ao fim de suas tentativas é equivalente a uma média entre as maiores notas obtidas ao longo de suas tentativas.

Figura 6. Questionário de programação

Pergunta	Texto da pergunta	Histórico das respostas	Melhor nota	
1	Descubra o maior de três números.	#1(8), #2(7),	8	Responder
2	faça o fatorial de n.	#1(4),	4	Responder
3	Imprima um numero aleatorio entre 1 e 10.	#1(3), #2(5), #3(9),	9	Responder

Fonte: Mota Et. Al. (2009)

De acordo com Mota et.al (2009), durante um curso é importante que o estudante possa acessar materiais didáticos básicos como conceitos, teorias e exemplos práticos. Esse sistema de Tutoria gerencia os exemplos com códigos-fonte de programas armazenados na base de exemplos, que são exibidos como recursos dentro da plataforma. Os exemplos são classificados de acordo com estratégias didáticas pré-estabelecidas formando uma base de exemplos de programação. Esse sistema tem função semelhante ao módulo tutor apresentado por Branco Neto e Schuvartz (2007), onde o estudante participa do ambiente visualizando exemplos que estão disponíveis de acordo com o seu grau de conhecimento e habilidade em programação.

O professor cadastra no sistema os exemplos ou pode salvar uma solução desenvolvida pelo estudante na base de exemplos. Desse modo, os exemplos estão disponíveis como conteúdos e podem ser utilizados de acordo com os critérios do professor ou por livre exploração do estudante. As questões de programação podem ser relacionadas aos exemplos, conduzindo o estudante a correlacionar os problemas e suas soluções. Assim, se o estudante percebe, na simulação, um erro de sintaxe na declaração de uma estrutura de repetição do-while, pode buscar um exemplo relacionado àquela estrutura de repetição na base de exemplos. Outras hipóteses podem ser viabilizadas, pois o módulo é flexível e adaptável, já que segue as orientações de desenvolvimento para o MOODLE (2009).

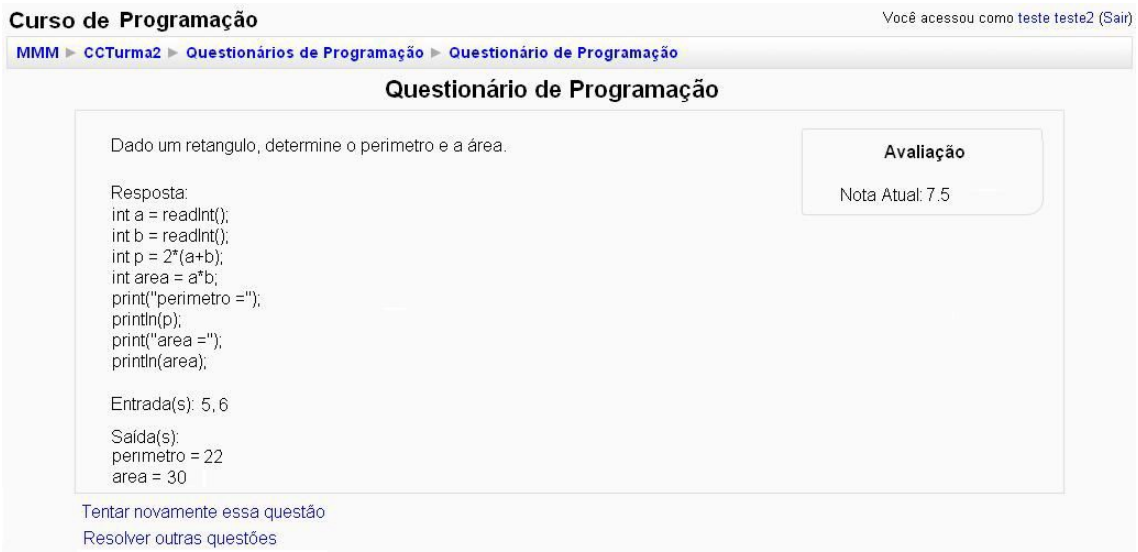
O sistema de avaliação permite a avaliação automática ou manual, com lançamento de notas e comentários, conforme os recursos do MOODLE. O modelo de avaliação automática implementado no ambiente de programação foi proposto por Moreira e Fávero (2009).

Esse modelo combina o uso de uma avaliação da complexidade do código através da técnica estatística de Regressão Linear Múltipla com indicadores de complexidade, aliada a um testador de código por entrada/saída.

Os indicadores aplicados pelo avaliador por complexidade de código são métricos da Engenharia de Software, a técnica consiste em extrair da solução do estudante o valor obtido com a aplicação de cada uma das métricas descritas por Moreira e Fávero (2009). Esses valores são submetidos ao modelo de Regressão Linear Múltipla, que compreende em uma equação linear obtida através de treinamento em base de testes.

Neste caso, as métricas são as entradas da fórmula que calcula diretamente a nota do estudante. Em seguida, é verificado se a solução retorna o resultado esperado para as entradas previamente informadas. Assim, para o correto funcionamento do avaliador, o professor deve cadastrar uma solução considerada “resposta-modelo” para o problema. Assim, se não houver erro de compilação, o sistema avalia por complexidade e obtém um valor para a nota do estudante, que pode ser modificada dependendo do resultado da avaliação por testes de entrada/saída. A nota final é então exibida na tela figura 7.

Figura 7. Nota final do Programa



Fonte: Mota Et. Al. (2009)

Mesmo no modo automático de correção, os resultados são visualizados pelo professor, dessa maneira se pode aferir o nível de conhecimento dos alunos e até mesmo se é conveniente interferir no processo

figura 8. Com a rápida resposta empregada pela ferramenta, proporciona que o professor organize as tarefas compartilhadas e utilize de forma mais eficaz os seus monitores, de acordo com o progresso dos alunos no decorrer das aulas no laboratório.

Figura 8. Resultado Automático

Pergunta				
1	Descubra o maior de três números.			
	Nome		Histórico das respostas	Melhor nota
		Aluno A	#1(8), #2(7)	8
		Aluno B	#1(3), #2(2), #3(2), #4(1), #5(2)	3
		Aluno C	#1(6)	6
		Aluno D	#1(6), #2(2), #3(9), #4(7)	9

Fonte: Mota Et. Al. (2009)

2.3 Análise da Ferramenta Educacional CodeRunner

Segundo Lobb (2016), o CodeRunner é um plug-in gratuito e de código aberto do MOODLE que permite que os professores criem exercícios/questões, onde a resposta do aluno é um código de algoritmo. Ele foi projetado para poder suportar qualquer linguagem de programação baseada em texto. Dessa forma, os tipos de questões integradas estão disponíveis para HTML, C, Java, Python, PHP, JavaScript e Octave (Matlab), entre outras.

Com essa ferramenta os alunos conseguem desenvolver e testar seu código usando um ambiente de desenvolvimento virtual e enviam o código para o CodeRunner através de um navegador da Web somente quando acreditam que estejam corretos. Um pressuposto é que o questionário em que as perguntas aparecem está sendo executado no modo adaptativo do MOODLE, o que dá aos alunos um feedback imediato sobre a correção de sua resposta e permite reenviá-la mediante uma penalidade.

O CodeRunner fornece um grupo de penalidades flexíveis que permite aos autores adaptar a classificação delas ao contexto. Por exemplo, pode ter algumas questões que não acarreta penalidades para o reenvio, algumas permitem um ou dois envios seguidos de uma penalidade crescente de 10%

por cada submissão errada e outras que incidem uma penalidade de 100% para cada submissão errada.

Ele é altamente escalável, ou seja, as questões podem variar desde o preenchimento simples para questões de codificação em branco, até as atribuições bastante complexas. Como qualquer outro tipo de pergunta do MOODLE, testes ou quizzes, os questionários podem misturar as perguntas do CodeRunner com outros tipos de perguntas, como classificação de computador, múltiplas escolhas, numérico, resposta curta ou correspondência e ainda se pode até incorporar questões de ensaio classificadas por professores.

Em um contexto de ensino-aprendizagem, como as salas de aulas, os questionários podem incluir questões de descrição, que são, na verdade, apenas o material didático, por exemplo, uma introdução a “if-statements” ou “loops”. Indo até se tornar o principal meio de aprendizagem. Por serem baseados na web, eles podem ser feitos pelos estudantes em suas casas ou em qualquer ambiente com acesso a internet, liberando o curso de muitas das restrições impostas pelos laboratórios, que agora podem passar a ser um auxilio extra para os estudantes que necessitarem.

O CodeRunner é particularmente adequado aos cursos de programação introdutória, para os quais os alunos precisam de prática com problemas de programação que ensinam as diferentes construções e técnicas da linguagem.

No entanto, o CodeRunner também pode ser usado em níveis mais altos, por exemplo em documentos teóricos de ciência da computação para testar coisas como autômatos de estados finitos e construção de compiladores, em um curso de inteligência artificial para a programação de Clojure e em um curso de programação web para avaliação de sites de autoria de estudantes.

2.3.1 Arquitetura do CodeRunner

Segundo Lobb (2016), a arquitetura do CodeRunner é relativamente simples e foi desenvolvida usando o Twig modelo, o qual separa a arquitetura

da ferramenta em blocos independentes, facilitando assim o seu entendimento e futuras manutenções. Dessa forma, Lobb ainda afirma que o CodeRunner atualmente suporta HTML, Python2, Python3, C, C++, Java, PHP, JavaScript (NodeJS), Octave e Matlab, porém por ter uma arquitetura flexível, qualquer desenvolvedor pode acrescentar outras linguagens de programação facilmente.

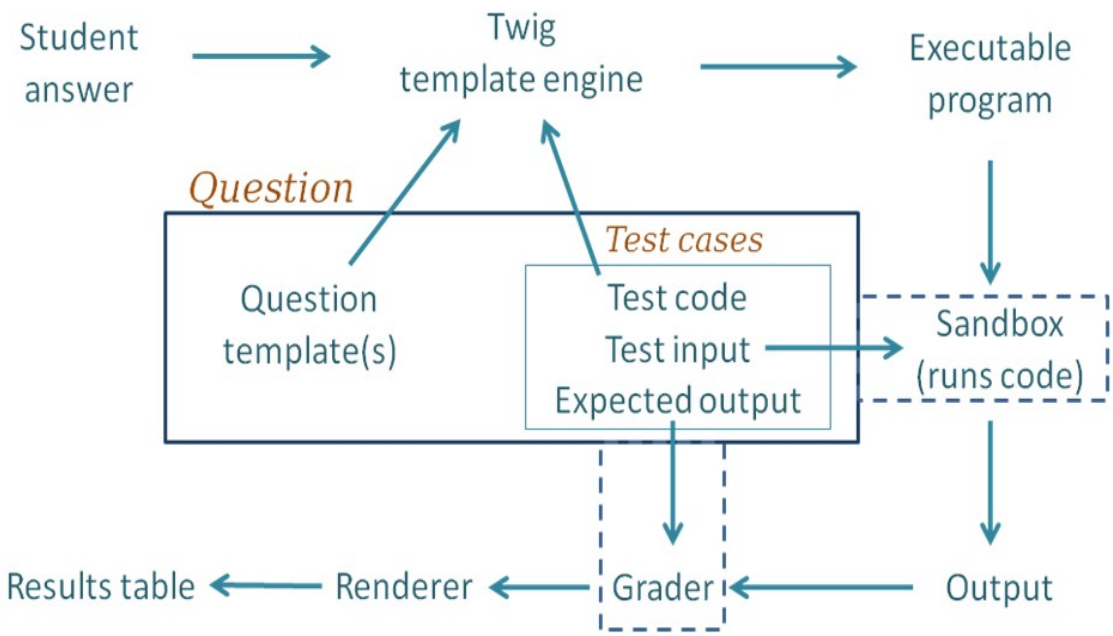
Embora o CodeRunner tenha uma arquitetura totalmente versátil, que suporte várias maneiras diferentes de executar a tarefa do aluno em um ambiente protegido ("Sandboxed"), apenas a Sandbox Jobe é suportada pela versão atual do CodeRunner. Esta Sandbox faz uso de um servidor separado, desenvolvido especificamente para uso pelo CodeRunner e por padrão na instalação no MOODLE a configuração usa o servidor Jobe da Universidade de Canterbury, e isso não é adequado para uso na produção em ambientes reais, nesse caso o mais recomendável seria a instalação própria de um Sandbox Jobe.

De acordo com o Lobb (2016), é altamente recomendável que se instale o Sandbox em um servidor separado ao MOODLE, pois assim ele irá isolar completamente o código do aluno do servidor MOODLE. No entanto, o Jobe pode ser instalado no próprio servidor do MOODLE, em vez de em uma máquina completamente diferente, isso funcionará normalmente, mas é muito menos seguro do que executar o Jobe em uma máquina completamente independente.

Já que se um programa desenvolvido por algum estudante como resposta a alguma questão elaborada conseguir sair do Sandbox quando estiver executando em uma máquina separada, o pior que pode fazer é derrubar o servidor do Sandbox, enquanto uma violação de segurança no servidor MOODLE pode ser usada para invadir o banco de dados, que contém resultados de códigos rodados pelos estudantes e notas dos alunos.

Na figura 9, podemos verificar melhor o que acontece quando um aluno submete sua resposta ao CodeRunner, assim para cada um dos casos de teste, o modelo Twig combina a resposta enviada pelo aluno com o modelo da pergunta juntamente com o código para este caso de teste específico, para assim produzir um programa executável.

Figura 9. Arquitetura CodeRunner



Fonte: CodeRunner.org

O programa executável é passado para qualquer Sandbox configurada no momento da instalação do plug-in, portanto é a Sandbox a qual fica responsável pela compilação do programa (se necessário) e o executa, usando a entrada padrão fornecida pela testcase. A saída da execução é passada para qualquer componente classificador que esteja configurado, assim como a saída esperada especificada para o caso de teste. O classificador mais comum é o grader da"correspondência exata", mas outros tipos estão disponíveis para uso.

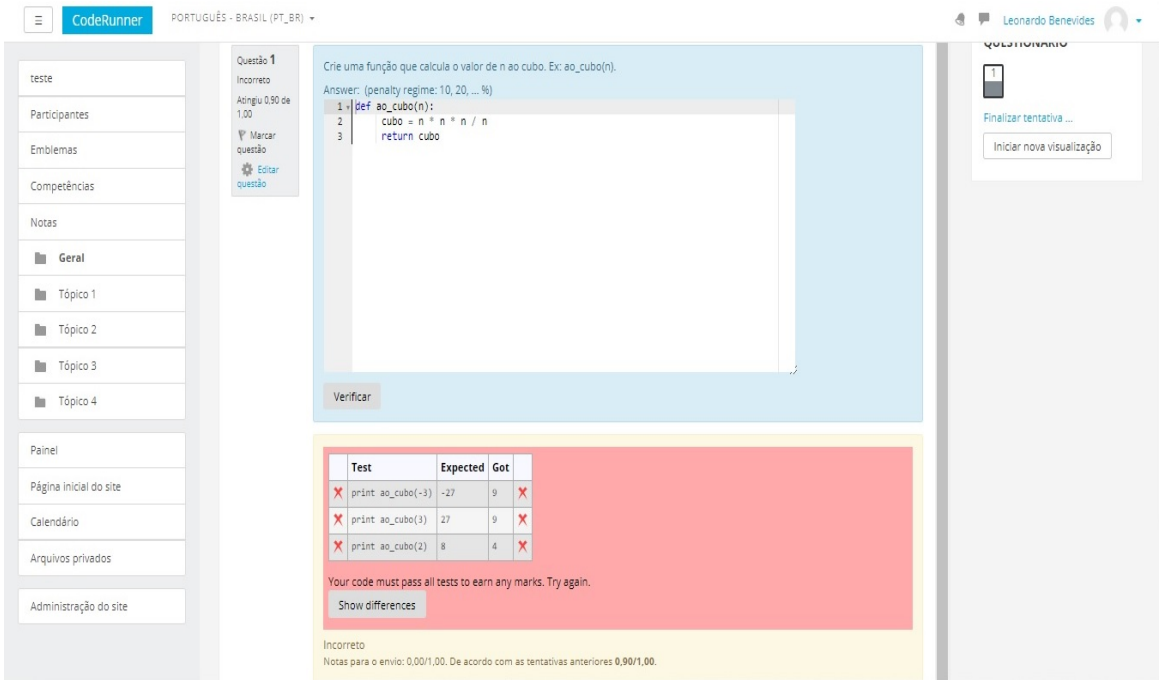
O resultado do grader é um "objeto de resultado de teste" que contém (entre outras coisas) atributos "Esperados" e "Obtidos", assim os passos anteriores serão repetidos para todos os testcases, dando uma série de objetos de resultados de teste. Todos os resultados do teste são passados para o processador de perguntas do CodeRunner, que os apresenta ao usuário como a tabela de resultados. Dessa forma, os testes que foram desenvolvidos

corretamente são mostrados com um "tique" verde e os que estão errados mostrados com uma cruz vermelha. Normalmente, a tabela inteira é colorida vermelha se algum teste falhar ou verde se todos os testes passarem.

2.3.2 Interface do CodeRunner

Iremos mostrar como são as interfaces do CodeRunner, tanto na visão do aluno, como na visão dos professores desenvolvedores de questões. Segundo Lobb (2016), a interface do CodeRunner foi desenvolvida para ser extremamente simples e consecutivamente bem intuitiva. Na visão do aluno, o CodeRunner basicamente se resume há dois blocos principais, o primeiro bloco é onde deve ser desenvolvida a resposta proposta pelo exercício e na parte superior se encontra uma pequena descrição que foi elaborada pelo desenvolvedor da questão para ajudar o aluno a entender o que deve ser implementado.

Figura 10. Resposta Incorreta

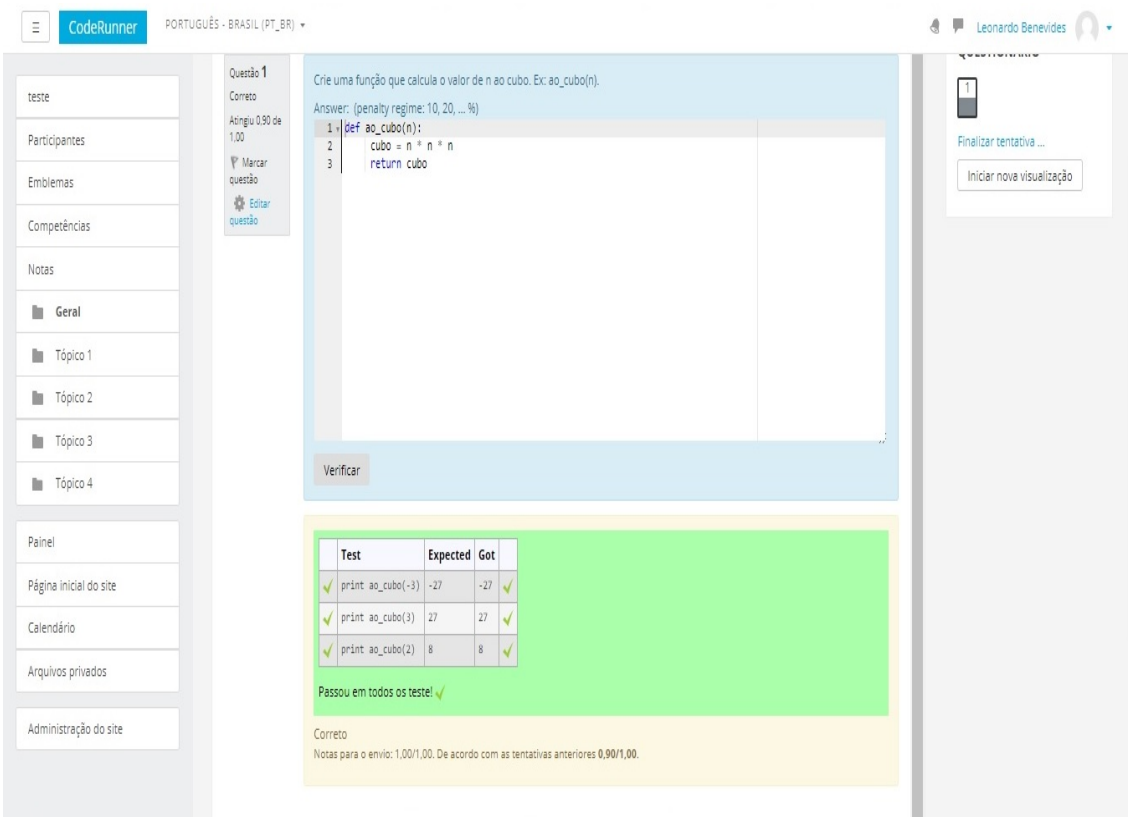


Fonte: Benevides, L.

A figura 10 mostra uma questão em Python no CodeRunner que pede aos alunos que escrevam uma função chamada ao cubo (n), que retorna o cubo de qualquer valor inserido.

Na figura 11, podemos verificar que o segundo bloco é onde o aluno consegue visualizar as informações referentes ao desenvolvimento do seu código, ou seja, ali será exibido todo e qualquer erro possível que a ferramenta detectou no momento da compilação. Além disso, neste bloco do CodeRunner contém uma pequena tabela de resultados esperados, que o desenvolvedor propôs no momento da construção da questão, e possui também os resultados obtidos de acordo com o código implementado pelo aluno. Dessa forma, ele realiza uma comparação entre os resultados esperados e obtidos.

Figura 11. Resposta Correta



Fonte: Benevides,L.S.

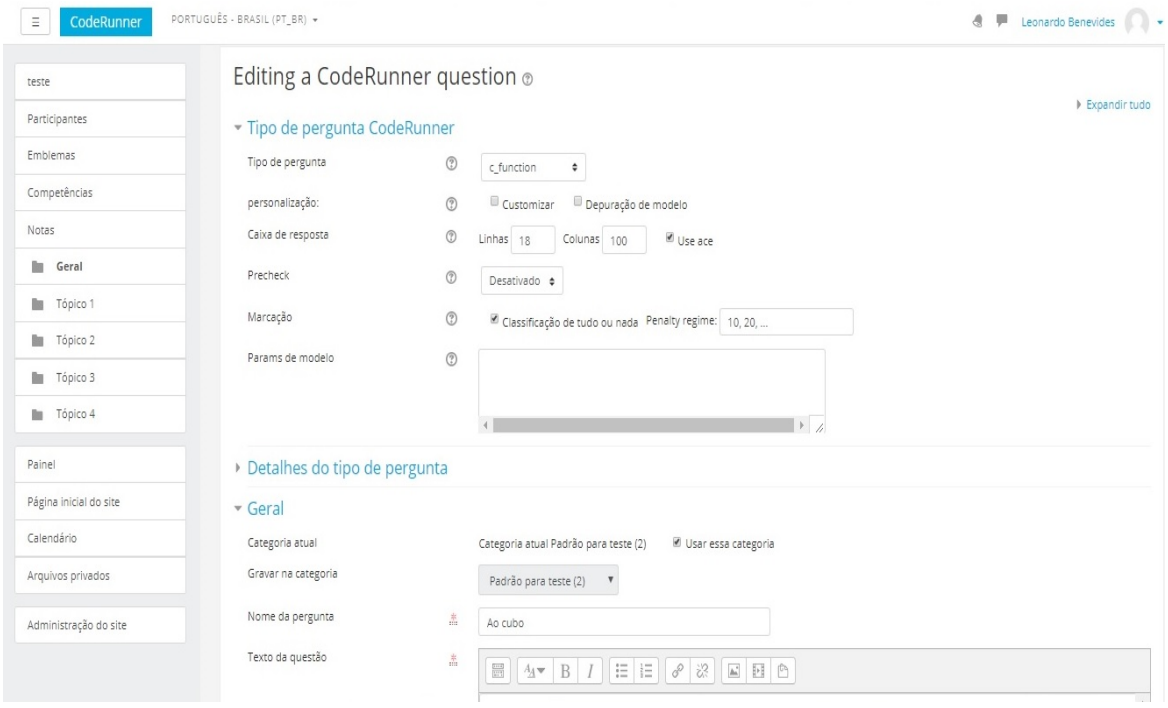
A figura 10 mostra ainda o que o aluno vê após um retorno incorreto de alguma parte do código por ele submetido e a figura 11 mostra o estado da

tabela de resultado após uma submissão totalmente correta. Algumas coisas a serem observadas sobre o feedback retornado pelo CodeRunner.

O feedback é uma tabela que mostra os testes que foram usados, o resultado esperado de cada teste da função do aluno e a saída real, os tiques verdes indicam saídas corretas, o vermelho indica que houve alguma parte do código que eventualmente tenha sido enviado de maneira errada. Quando o aluno submete uma resposta correta, figura 3, todo o painel fica verde e uma marca de 100% é concedida, menos qualquer penalidade acumulada, que por ventura tenha sido programada pelo professor para tal, nesse caso, 10%.

Na figura 12, podemos ver a interface vista pelos desenvolvedores de questões, é nesta parte que os professores necessariamente elaboram os exercícios que serão propostos aos seus alunos.

Figura 12. Edição de Questão



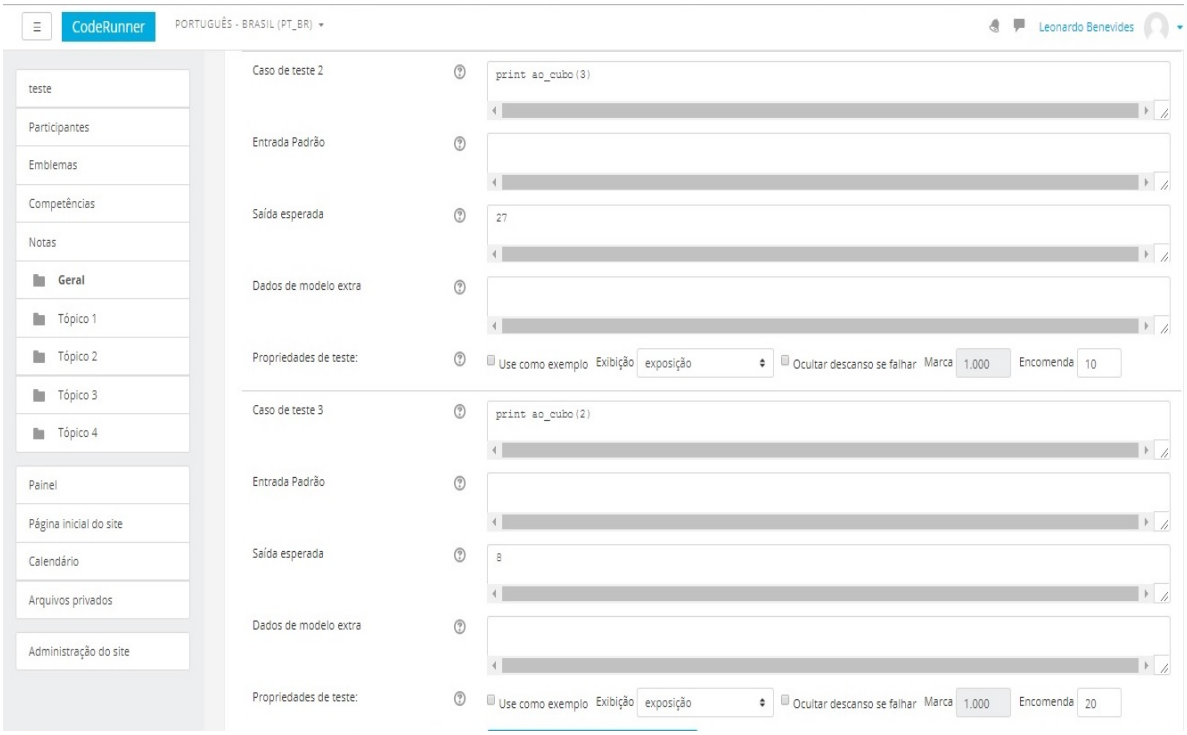
Fonte: Benevides, L.

Aqui se deve escolher qual a linguagem de programação será utilizada para elaboração da questão, impor algum limite de espaço que o aluno deve

ser restringido, colocar algum tipo de penalidade para algum eventual erro de código submetido pelo aluno, como é aqui que o desenvolvedor da questão coloca a descrição da questão entre outras muitas possibilidades de manipulação e métodos que o CordeRunner permite ser inserido em uma questão.

Já na figura 13, podemos observar onde os professores devem inserir os casos de testes, entrada padrão, saída esperada o qual será comparado com o resultado real obtido com base na resposta do aluno, assim exibindo o resultado na tabela resultado.

Figura 13. Edição de Questão



Fonte: Benevides, L.

2.3.3 Tipos de Questões suportadas pelo CodeRunner

Segundo Lobb (2016), toda questão realizada no CodeRunner é uma instanciação de um protótipo de questão, que essencialmente define o "tipo" da questão. Sendo que um conjunto de protótipos internos definem uma gama de questões do CodeRunner, como "escrever uma função", "escrever um programa" ou "escrever uma classe" para as linguagens padrões, mas os desenvolvedores das questões podem definir seus próprios protótipos para fornecer funcionalidades extras.

O uso de um modelo customizável para definir o programa a ser executado oferece grande flexibilidade, permitindo que o autor da questão use a resposta do aluno de muitas maneiras diferentes. Uma aplicação importante desta flexibilidade é alguma forma de pré-processamento que valida a submissão do aluno antes de executá-la. Isso pode ser ilustrado de duas maneiras importantes, como a verificação de estilo das submissões dos alunos que em Python, podemos impor a conformidade com o verificador de estilo pylint, antes que o código seja aceito para execução, e para os tipos de perguntas de Matlab, podemos escrever modelos que aplicam regras de estilo local e podem impor limites no tamanho do programa e da função.

E a segunda maneira seria aplicar e/ou restringir o uso de construções de programação específicas. Por exemplo, em questões de qualquer linguagem podemos pedir para o aluno reescrever o seguinte programa usando um "loop while" em vez de um "loop for", assim o código do aluno é automaticamente rejeitado sem ser executado se o pré-processador detectar qualquer "loop for" presente no código.

A linguagem usada para o protótipo pode ser diferente da usada para executar a submissão do aluno, ou seja, um dos verificadores de estilo do Octave usa um modelo em Python para verificar o código dos alunos antes de enviá-lo para Octave para execução, essa distinção entre a linguagem do modelo e o que está sendo verificado é usada na maioria dos casos no CodeRunner, todos os quais exploram a flexibilidade dos modelos.

E ainda tem as questões em que a resposta do aluno é apenas uma URL que faz referência a uma página da Web que foi criada em um servidor separado, o código do modelo executa testes na página da Web referenciada,

afim de procurar qualquer tipo de erro. Outro tipo de questão é aquele em que um aluno submete uma descrição textual de uma máquina de estado finito e do código da questão (escrito em Python), para validar e classificar o comportamento da máquina.

Normalmente, a correção da submissão de um aluno é validada comparando o resultado real da execução com a saída esperada para cada teste, mas também é possível incorporar o processo de classificação no próprio modelo, como é feito no exemplo da maquina de estado finito acima. Em princípio, qualquer questão que pode ser classificada por um computador pode ser colocada como uma questão do CodeRunner, embora seja mais adequado lidar com exercícios onde os testes podem ser apresentados aos alunos em forma de tabela.

3. COMPARATIVO DAS FERRAMENTAS ABORDADAS

Diante do exposto anteriormente em relação as características das ferramentas, pudemos analisar e compreender as arquiteturas, as interfaces, o meio em que elas são instaladas, a complexidade em usa-las e a forma como elas executam o procedimento de correção dos exercícios. Por conseguinte, compreendemos que o CodeRunner se mostrou em alguns aspectos melhor do que as demais ferramentas, como ser mais flexível quanto a linguagens de programação disponíveis e quanto a inserção de níveis de dificuldade nos exercícios, entre outros aspectos, como podemos ver no quadro 1 abaixo.

Diante dessa análise, tornou-se evidente o seu grande potencial para a área educacional, pois o CodeRunner tanto pode ser utilizado em cursos computacionais iniciais básicos como em cursos mais avançados. Devido a sua flexibilidade podemos até incluir em seu âmbito exercícios de máquinas de estados finitos, fato inalcançado pelas outras ferramentas.

Visando seu potencial desenvolvimento no ambiente educacional, o trabalho aqui proposto visa em aplicar algumas técnicas de balanceamento de carga no contexto da ferramenta, aplicando ao CodeRunner certos aspectos pertencentes aos sistemas distribuídos e assim tornando-o tolerante a falhas de comunicação ou a sobrecarga de sistemas.

Quadro 1. Comparativo das ferramentas

Características	**Ferramentas Analisadas**		
	MOJO	JavaTool	CodeRunner
Linguagens suportadas	C, C++, C#, Java e PHP	Java	HTML, C, C++, Java(NodeJS), Python, PHP, JavaScript e Matlab
AVA	MOODLE	MOODLE	MOODLE
Possui avaliação automática	✓	✓	✓

Definição de nível de dificuldade das Questões			
Casos de testes			
Regras de Pontuação			
Feedback colaborativo			

Fonte: Benevides, L.

4. COMUNICAÇÃO CODERRUNER - SANDBOX

4.1 Servidores web Apache

Um dos mais utilizados servidores Web do mundo é o Apache (Apache, 2010; NETCRAFT, 2010). Ser open source, código aberto, possibilitou o acesso mundial ao sistema pelos usuários já que qualquer programador experiente pode realizar aperfeiçoamentos e correções em sua arquitetura. Outra ponto a observar é o fator que o servidor Apache possui várias versões, dessa forma viabiliza o seu funcionamento em inúmeros sistemas operacionais, totalmente confiável ao usuário.

O servidor é um ambiente formado por um conjunto de módulos, onde cada módulo tem uma função específica que funcionam independetes um do outro. Dentre alguns, existe uns que tem como função configurar a maneira que o Apache recebe e lida com as requisições do cliente web, havendo mudança devido os sistemas operacionais em que Apache está trabalhando. Tais conjuntos de módulos são denominados de Multi-processamento - MPMs.

4.2 Sandbox

Um programa é hospedado no computador e um ambiente de execução é configurado para ele com certas restrições para a execução. É dado ao programa, o direito de acesso a certos recursos do sistema. Esta idéia faz com que o programa fique confinado no Sandbox. Assim, parâmetros para o Sandbox podem ser modificados e o programa executará com certas permissões.

Caso se modifique os indicadores do sandbox acarretará em mudanças também nas políticas utilizadas por ela. Em certas ocasiões, isso não é permitido se um determinado aplicativo queira usar um arquivo, primeiro o usuário é obrigado alterar as políticas de segurança da sua máquina e então rodar o programa. Usando uma analogia, é a mesma coisa de aumentar os limites de uma caixa de areia para uma criança brincar, oferecendo a essa criança uma quantidade maior ou menor de brinquedos, variando de acordo com o seu comportamento.

A arquitetura de uma Sandbox (caixa de areia) é composta por cinco elementos básicos: Permissões de acesso, Fontes de Código, Domínios de Proteção, Arquivos de Políticas e Banco de chaves.

4.3 Comunição CodeRunner e Sandbox

Embora o CodeRunner tenha uma arquitetura flexível que suporte várias maneiras diferentes de executar tarefas de alunos em um ambiente protegido Sandbox ("Caixa de areia"), apenas uma Sandbox é suportada pela versão atual da ferramenta. Esta Sandbox faz uso de um servidor separado, desenvolvido especificamente para uso pelo CodeRunner, chamado Jobe.

Segundo Lobb (2016), a Jobesandbox é um Sandbox que suporta a execução de pequenos trabalhos de compilação e execução em uma variedade de linguagens de programação. Ele foi desenvolvido como uma Sandbox remota para uso da ferramenta CodeRunner, a qual é um plugin do MOODLE que pede aos alunos que escrevam códigos de programação para alguma especificação proposta pelo professor. No entanto, os servidores da Jobe podem ser úteis em uma variedade de outros contextos, particularmente na educação.

A ferramenta CodeRunner se comunica com o servidor, o qual o Jobesandbox foi instalado, através da função “e*xecute”*, a qual vemos abaixo na figura 14 e recebe como parâmetros os seguintes campos: *sourcecode*, *language*, *input*, *files* e *params*.

Figura 14. Função Execute

```
public function execute($sourcecode, $language, $input, $files=null, $params=null) {
    $language = strtolower($language);
    if (!in_array($language, $this->languages)) { // This shouldn't be possible.
        return (object) array('error' => self::UNKNOWN_SERVER_ERROR);
    }

    if ($input !== '' && substr($input, -1) != "\n") {
        $input .= "\n";  // Force newline on the end if necessary.
    }

    $filelist = array();
    if ($files !== null) {
        foreach ($files as $filename => $contents) {
            $id = md5($contents);
            $filelist[] = array($id, $filename);
        }
    }

    $progname = "prog.$language";

    $runspec = array(
            'language_id'       => $language,
            'sourcecode'        => $sourcecode,
            'sourcefilename'    => $progname,
            'input'             => $input,
            'file_list'         => $filelist
        );

    if (self::DEBUGGING) {
        $runspec['debug'] = 1;
    }

    if ($params !== null) {
        // Process any given sandbox parameters.
        $runspec['parameters'] = $params;
        if (isset($params['debug']) && $params['debug']) {
            $runspec['debug'] = 1;
        }
        if (isset($params['sourcefilename'])) {
            $runspec['sourcefilename'] = $params['sourcefilename'];
        }
    }
```

Fonte: Benevides, L.

O campo *sourcecode* armazena todos os comandos inseridos pelo aluno e que serão testados, o campo *language* armazena qual linguagem de programação foi escolhida pelo professor para o aluno desenvolver o programa a ser testado.

Já o campo *input* representa uma string para ser usada como entrada padrão durante a execução, *files* representa um array associativo ou um mapa de nomes de arquivos para o conteúdo do arquivo, assim definindo um contexto de arquivo no tempo de execução.

Por fim o campo *params* depende da Sandbox específica, mas a maioria das Sandbox deve reconhecer pelo menos *cputime* (segundos), *memorylimit* (Megabytes) e *files* (um nome de arquivo de mapeamento de array associativo para string filecontents). Se o array *params* for nulo, os padrões da Sandbox serão usados.

Figura 15. Função http_request

```
private function http_request($resource, $method, $body=null) {
        list($url, $headers) = $this->get_jobe_connection_info($resource);
        $curl = new curl();
        $curl->setHeader($headers);
        if ($method === self::HTTP_GET) {
            if (!empty($body)) {
                throw new coding_exception("Illegal HTTP GET: non-empty body");
            }
            $response = $curl->get($url);
        } else if ($method === self::HTTP_POST) {
            if (empty($body)) {
                throw new coding_exception("Illegal HTTP POST: empty body");
            }
            $bodyjson = json_encode($body);
            $response = $curl->post($url, $bodyjson);
        } else {
            throw new coding_exception('Invalid method passed to http_request');
        }
        if ($response !== false) {
            $returncode = $curl->info['http_code'];
            $responsebody = $response === '' ? '' : json_decode($response);
        } else {
            $returncode = -1;
            $responsebody = '';
        }
        return array($returncode, $responsebody);
    }
```

Fonte: Benevides, L.

Já a conexão como vemos na figura 15, é através da função privada "*http_request*" onde ela faz uso da biblioteca Curl do PHP. A Curl é um recurso integrante da linguagem PHP para possibilitar a conexão com URLs externas, por meio dessa interação PHP/Curl com o Sandbox, alguns códigos são retornados e a seguir são testados a fim de verificar se a conexão foi bem-sucedida ou algum tipo de inconsistência foi gerada.

Como vemos na figura 16 abaixo, de acordo com o código retornado se faz uma reavaliação e uma possível nova tentativa de conexão com o Jobesandbox. Então, temos os seguintes códigos de retorno: para a conexão bem-sucedida temos os códigos 200, 202 e o 204, quando o código de retorno for 400 temos um erro no servidor Jobesandbox, se por ventura o código for o 401 temos como diagnostico o erro tempo limite de submissão excedida e por fim temos a possibilidade do código 403 o qual significará um erro de autenticação.

Figura 16. Códigos de retorno

```
private function get_error_code($httpcode) {
        $codemap = array(
            '200' => self::OK,
            '202' => self::OK,
            '204' => self::OK,
            '400' => self::JOBE_400_ERROR,
            '401' => self::SUBMISSION_LIMIT_EXCEEDED,
            '403' => self::AUTH_ERROR
        );
        if (isset($codemap[$httpcode])) {
            return $codemap[$httpcode];
        } else {
            return self::UNKNOWN_SERVER_ERROR;
        }
    }
```

Fonte: Benevides, L.

4.4 CodeRunner recebe e exibe os resultados

Após a conexão ser bem-sucedida, o trabalho do aluno é enviado ao servidor Jobesandbox. O servidor é encarregado de receber apenas o código do aluno o qual já foi mesclado através do mecanismo de modelo Twig com o modelo da pergunta, ou seja, o mecanismo faz uma associação entre a reposta submetida pelo aluno com o modelo previamente realizado pelo Professor.

Ele compila o código-fonte e o executa com os dados de entrada fornecidos. Assim, tem-se um programa executável que é executado no Jobesandbox, logo após a execução o objeto executado é enviado de volta para o CodeRunner, onde passa pelo graduador, que é um instrumento a qual a arquitetura do CodeRunner permite realizar comparações entre saída da execução do programa com a saída esperada inicialmente proposta pelo professor autor da questão.

Nesse exato momento podem ser feitas vários tipos de comparações e associações de acordo com o que foi proposto pelo Professor, temos o “*EqualityGrader*” , um “*NearEqualityGrader*” ou “*RegexGrader*”.

O “*EqualityGrader*” espera que a saída da execução do teste corresponda exatamente à saída esperada para o testcase. O “*NearEqualityGrader*” é semelhante, mas é insensível a maiúsculas e minúsculas e tolera variações na quantidade de espaço em branco (por exemplo, linhas em branco ou faltando, ou vários espaços onde apenas um era esperado). O “*RegexGrader*” espera uma correspondência de expressão regular. O “*EqualityGrader*” é recomendado para todos os usos normais, pois incentiva os alunos a obterem a saída correta, eles devem ser capazes de reenviar respostas quase corretas para uma pequena penalidade.

A saída do graduador é um objeto contendo várias informações de status, além da saída e saída de erro da execução. Todos os resultados do teste são passados para o renderizador do CodeRunner, que os apresenta ao usuário como a tabela de resultados.

Testes que passam são mostrados com um  "tic" verde e os que não passam são mostrados com um  “X” vermelho. Normalmente, toda a tabela é colorida em vermelho se algum teste falhar ou em verde se todos os testes forem aprovados.

5. REPLICAÇÃO DO SERVIDOR JOBESANDBOX

Depois de realizar a análise da conexão do CodeRunner com o servidor, ficou evidenciada uma possível restrição técnica. A qual a ferramenta fica dependente do servidor para realizar a comunicação, execução do código do aluno e verificar sua correção no Jobesandbox.

Nesse sentido, pensou-se em ampliar o campo de atuação da ferramenta, para que ela possa ser usada de maneira distribuída, já que a manutenção de cópias dos servidores Jobesandbox, replicação de servidores, é um dos quesitos que tornam os sistemas distribuídos tão eficazes. Dessa forma, as réplicas deixariam a ferramenta CodeRunner com melhor desempenho, alta disponibilidade e tolerante a falhas.

Na melhoria do desempenho da ferramenta com o uso de réplicas Jobesandbox, poderíamos citar a própria escolha do servidor que ficará responsável pela execução do código do aluno. Visto que, poderemos ter vários servidores Jobesandbox para o uso da ferramenta, mas nem todos necessariamente com as mesmas configurações e aptos a receber qualquer código de alunos, realizando assim sua execução e correção com a mesma eficiência e velocidade. Sendo assim, o CodeRunner de acordo com a complexidade e/ou objetivo proposto pela questão inserida pelo professor, remeteria o código do aluno ao servidor que melhor se enquadre, assim, visando sempre uma resposta mais rápida e eficiente do servidor em questão.

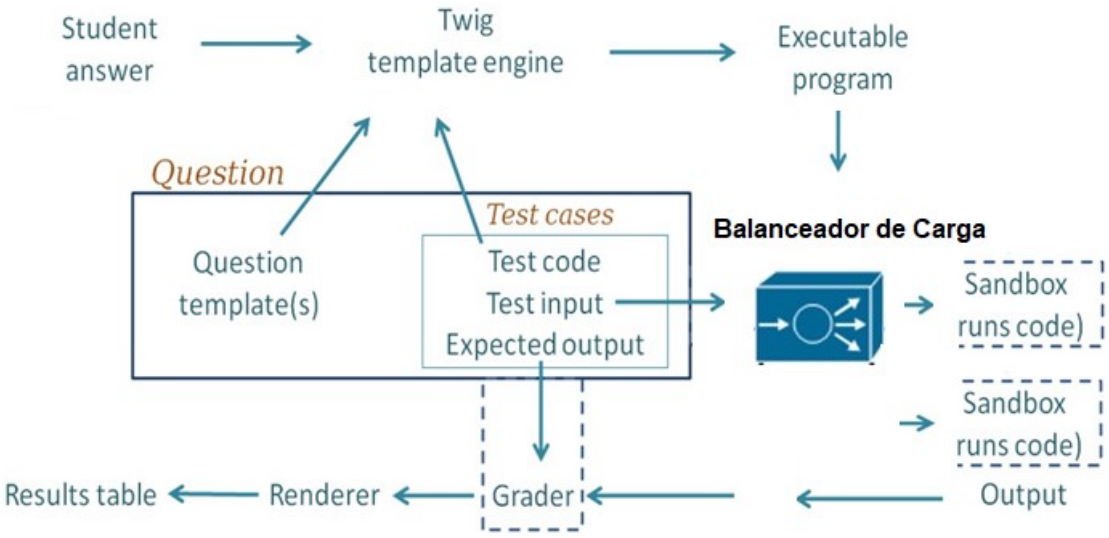
Já no quesito de maior disponibilidade do serviço no servidor, podemos dizer que quanto maior for o uso da ferramenta instalada no MOODLE, maior será a exigência por parte dos usuários que o serviço esteja disponível o máximo de tempo possível, assim evitando falhas de serviço ocasionadas pelo não acesso ou falhas de comunicação.

A técnica de replicação de servidores é uma forma para manter automaticamente a disponibilidade dos dados, em nosso caso disponibilidade do serviço, apesar de falhas no servidor. Com os servidores Jobesandbox replicados em dois ou mais servidores que vierem a falhar independentemente, a ferramenta poderá acessar algum servidor alternativamente, caso o servidor projetado como padrão falhe ou se torne inacessível naquele momento.

Dessa maneira, poderemos elevar a taxa de disponibilidade do servidor com uso de réplicas Jobesandbox, o qual é essencial para a utilização da ferramenta CodeRunner.

Sendo assim, propomos realizar algumas mudanças no código fonte da ferramenta na função responsável por realizar a conexão do CodeRunner com o servidor em questão. Sendo acrescentados algumas linhas de códigos que se encarregará de realizar uma espécie de roteamento como visto na figura 17 abaixo, assim se terá a funcionalidade das réplicas dos servidores adicionada a ferramenta e para que se possa verificar qual servidor é o mais indicado a receber aquele código em questão.

Figura 17. Arquitetura com balanceamento de carga



Fonte: Adaptado de CodeRunner.org

Diante de cada tentativa de conexão com o servidor, utilizando a biblioteca do PHP Curl, são gerados códigos de retorno, mediante isso podemos diagnosticar se a tentativa de conexão que resultou em falha foi mediante a erros ocasionados pela perda de Internet, em caso positivo podemos adicionar um comando de seleção na referida função para que a ferramenta ao invés de tentar conectar com o servidor via Internet tente usar o

servidor localmente instalado, assim sanando possíveis perdas de conexão com a rede.

5.1 Balanceamento de carga de servidores

5.1.1 Definição e tipos de balanceamento

De acordo com (Karimi et al., 2009):

> Balanceamento de carga é a técnica de divisão de trabalho entre dois ou mais computadores, *links* de rede, CPU's, discos rígidos, ou outros recursos, a fim de obter a melhor utilização dos recursos, vazão (i.e. *throughput*) ou tempo de resposta.

Como o autor citou, o termo recurso pode possuir diferentes representações, onde definição de balanceamento de carga seria passível de aplicação. Assim, pode-se utilizar como exemplo, a distribuição de carga entre trabalhadores de uma empresa, mas as formas destes balanceamentos contêm peculiaridades, a seguir diferenças de duas das principais:

Balanceamento de carga de CPU: Em um sistema com multiprocessamento, as CPU's são capazes de estarem em uma mesma máquina ou distribuídas, o intuito deste balanceamento de carga é distribuir o trabalho, mas este trabalho deve estar dividido em partes, onde os processadores atuam paralelamente (As instruções de um programa são divididas entre as CPU's).

Balanceamento de carga de servidores: Em uma arquitetura computacional existi a possibilidade de múltiplos servidores disponibilizarem o mesmo tipo de serviço e até rodarem a mesma aplicação, podendo assim distribuir em partes iguais a carga de processamento entre eles, porém não é obrigado os servidores atuarem em paralelo, provavelmente eles não possuem qualquer interligação de dados ou sabem da existência um do outro. Deste modo cada requisição ao recurso é redistribuído entre os servidores disponíveis, mas como proteção um recurso jamais será enviado para dois múltiplos servidores.

5.2 Benefícios

Com a chegada das soluções dos balanceadores de carga embasados em servidores (operam sobre máquinas servidoras com sistemas operacionais padrão) foram alcançados diversos benefícios, sendo vários deles citados diretamente ou indiretamente.

Escalabilidade: Em uma infraestrutura de balanceamento de carga, encontra-se um bom nível de escalabilidade dos serviços, pois aumentando seu poder é necessário somente adicionar novos servidores reais. Também é facilitada a remoção de servidores com defeito, ou sua desativação e ativação de acordo com a demanda. Tudo com transparência para os clientes.

Desempenho: São alcançados altos níveis de performance com um custo mais baixo do que se realizassem a compra de computadores com recursos computacionais mais potentes. Dependendo da eficiência do algoritmo utilizado para decisão de despacho das requisições, onde serão escolhidos os que estiverem menos ocupados, melhor seria seu desempenho.

Disponibilidade e confiabilidade: Caso um dos servidores falhe, o balanceador de carga é capaz de detectar e redirecionar o fluxo para demais servidores, sem que o serviço fique indisponível. Outro aspecto importante é que o balanceador de carga não deve ser um gargalo para o serviço (caso ele seja o nó a falhar), assim é necessário aplicar alguma redundância no serviço de balanceamento, possuindo um servidor para backup adicionado de um processo de failover integrado, aumentando a confiabilidade do sistema.

5.3 Algoritmos de escalonamento

Esta seção trata de um dos mais importantes aspectos do balanceamento de carga, não somente de balanceamento de cargas de servidores, mas grande parte das técnicas que serão abordadas a seguir são aplicáveis em todos os tipos de balanceamento de carga. O balanceamento de carga é um problema de escalonamento de trabalho, trabalho que originalmente deveria ser realizado por um único indivíduo poderá ser dividido

e realizado por outros *n* indivíduos, para tomar a decisão de como organizar este trabalho é necessário um algoritmo de escalonamento.

Casavant e Kuhl (1988) definiram uma taxonomia para caracterização de algoritmos de balanceamento de carga, a seguir algumas das categorias componentes dessa taxonomia:

Estático vs. Dinâmico: Um algoritmo estático tem como premissa o desempenho dos indivíduos que irão realizar o trabalho, ou seja, antes de se iniciar o escalonamento carga de trabalho é definida a forma que ele será distribuído, não sendo alterado em tempo de execução. Já um algoritmo dinâmico possui poucas ou nenhuma informação sobre os indivíduos, todo a decisão de escalonamento será realizada em tempo de execução;

Cooperativo vs. Não-cooperativo: Em um algoritmo não-cooperativo os indivíduos realizam suas tarefas independente dos outros indivíduos do sistema, já em um cooperativo um indivíduo pode alterar a forma de realização de sua tarefa devido a uma possível cooperação com outro indivíduo (neste caso os indivíduos realmente se conhecem).

Adaptável vs. Não-adaptável: Em uma solução adaptável a forma de escalonamento da carga de trabalho poderá ser modificada em tempo de execução, isso ocorre principalmente devido a análise de um comportamento anterior do sistema, onde se descobre que uma possível alteração no escalonamento pode aumentar o desempenho do sistema. Já em um não adaptável não existe modificação na forma de escalonamento em tempo de execução.

Foram omitidas algumas outras categorias dessa taxonomia, pois as selecionadas são a que tem maior importância quando se trata especificamente do balanceamento de carga em servidores. A seguir serão descritos um total de cinco algoritmos de balanceamento de carga de servidores, três estáticos e dois dinâmicos.

5.3.1 Round-Robin e aleatório

É um dos mais antigos algoritmos na área da computação, aplicado originalmente em escalonamento de processos. É um algoritmo estático, não-cooperativo e não-adaptável que consiste em uma simples técnica que distribui de forma homogênea as requisições aos servidores que foram previamente ordenados, similar a um sentido horário. Este algoritmo trata todos os servidores de forma igualitária, como se tivessem a mesma configuração de hardware, não diferenciado-os, caso isso seja a realidade do *cluster* este algoritmo tende a trabalhar bem (MADHURAVANI; SUMALATHA; SHANMUKHI, 2012).

Um cenário explicativo desta situação a seguir: Existem três servidores (Server1, Server2 e Server3) e foram enviadas em sequência quatro requisições (Requisição1, Requisição2, Requisição3 e Requisição4) ao sistema, usando o algoritmo round-robin, Requisição1 seria destinada ao Server1, Requisição2 ao Server2, Requisição3 ao Server3 e Requisição4 ao Server1.

A única diferença do algoritmo round-robin com o aleatório é que ao invés da requisição ser distribuída seguindo a ordem dos servidores (1, 2, 3, ..., *n*), ela é distribuída de forma aleatória, mas sem atribuição de duas requisições ao mesmo servidor antes que um ciclo completo de requisições tenha sido distribuído aos outros servidores.

5.3.2 Ratio

O algoritmo ratio, também chamado de round-robin com pesos em algumas soluções de balanceamento de carga, é outro algoritmo estático, não cooperativo e não-adaptável. O round-robin tradicional é aconselhado para clusters com servidores homogêneos, mas isso nem sempre é a realidade de uma organização, podendo existir servidores mais antigos e com configuração mais básica atuando em conjunto com servidores mais novos com maior poder computacional, nestes casos o round-robin tradicional pode sobrecarregar alguns servidores enquanto outros estão ociosos (JAYASWAL, 2006, p. 224).

Esse algoritmo é indicado para clusters com servidores heterogêneos, onde são distribuídos pesos entre eles, os que tiverem maior poder

computacional possuem maior peso e os com menor poder um peso menor. Um cenário explicativo poderia ser o seguinte: Quatro servidores reais (S1, S2, S3 e S4) com os pesos 10, 5, 5 e 7, respectivamente. Com a soma dos pesos sendo 27, para o servidor S1 seriam direcionadas 37% das requisições, para o S2 e S3 aproximadamente 18% cada, já pro S4 cerca de 26% das requisições.

5.3.3 Menos conexões

Como explicado anteriormente, os algoritmos estáticos tem o seu comportamento determinado previamente, não possuindo mudanças em tempo de execução. Já os algoritmos dinâmicos podem tomar as decisões baseadas em novas informações do sistema (MADHURAVANI; SUMALATHA; SHANMUKHI, 2012), estas podendo ser coletadas via monitoramento dos servidores ou via análise de informações mantidas pelo próprio balanceador.

Uma informação que o balanceador pode e deve armazenar é o número de conexões que cada servidor está processando. Uma conexão aqui pode ser tratada como o processamento de requisições de um único cliente, por exemplo, caso um servidor esteja tratando 3 requisições, sendo que duas destas pertencem a um mesmo cliente, o servidor possuí um total de 2 conexões então. A partir do número de conexões de cada servidor, o balanceador poderá dirigir as novas requisições ao servidor com o menor número de conexões no momento (JAYASWAL, 2006, p. 224). Este algoritmo é chamado de “Menos conexões”, ele é um algoritmo dinâmico, não-cooperativo e adaptável.

5.3.4 Menor latência

Outro algoritmo dinâmico, não-cooperativo e adaptável, é o “Menor latência”, semelhante a estratégia do ultimo algoritmo apresentado, onde é tomada a decisão de balanceamento em cima de informações do sistema, só que neste caso ela é coletada a partir do monitoramento do cluster. A latência é a quantidade de tempo que um dispositivo de rede deve esperar pela resposta de outro (JAYASWAL, 2006, p. 131).

Em um período de tempo pré-determinado é feita a coleta do tempo de resposta destes servidores, onde normalmente se utiliza o protocolo ICPM para tal. Os servidores com menor latência serão os escolhidos para receber requisições. Um problema deste protocolo é que ele está fazendo uma medida relacionada a máquina servidor, mas o que realmente importa no ponto de vista de um cliente é a performance do serviço que este servidor está provendo.

Algo ainda pior pode ocorrer, o servidor pode estar em pleno funcionamento, mas os serviços dele podem se encontrar inoperantes ou sobrecarregados, neste cenário caso a latência do servidor ainda esteja mais baixa em relação a outros, ele ainda sim poderia ser selecionado para receber novas requisições (que seriam frustradas). Uma possível solução para o problema seria checar os tempos de resposta dos serviços ao invés de ser capturada a latência do servidor, e usá-los para o os valores coletados para escalonamento.

6. ALGORITMO SELECIONADO

De acordo com a pesquisa aqui apresentada, poderíamos adequar muitos, senão todos, os algoritmos mencionados. Ainda assim, o algoritmo selecionado foi o Round-Robin, que apesar de sua simplicidade tende a ter bons resultados em um cluster homogêneo de servidores com aplicações de propósito específico como é o caso do Jobesandbox (MADHURAVANI; SUMALATHA; SHANMUKHI, 2012).

Nesse sentido propomos, então, implementar uma solução conceitual de introdução de conceitos de balanceamento de carga, inserindo para tal, o algoritmo Round-Robin a fim de realizar esse balanceamento de maneira autônoma.

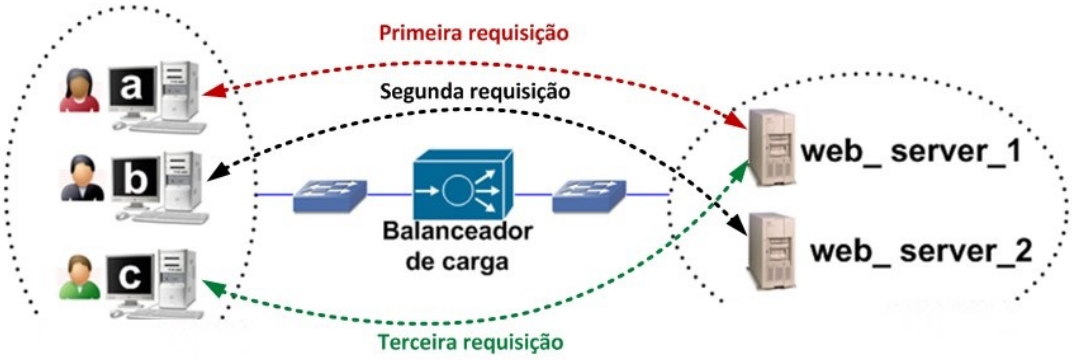
6.1 Implementação do algoritmo Round-Robin

O algoritmo Round-Robin aqui proposto visa receber as requisições da ferramenta CodeRunner, buscando em um determinado local o IP do servidor em uma lista de servidores. Dessa forma, ele realizará a função de balancear as cargas dos servidores, pois não permitirá que o mesmo servidor receba requisições de maneira consecutiva.

Como o algoritmo aqui desenvolvido não possui prioridades ou tempo de burst, ele trata todas as requisições de maneira igual sem se focar na complexidade do exercício a ser corrigido ou se o servidor da vez é o mais adequado a receber determinada requisição. Dessa maneira, o ideal é ter um cluster de servidores Jobesandbox homogêneos, a fim de manter padronizada a correção dos exercícios dos alunos.

De acordo com a figura 18 abaixo, podemos ver de forma conceitual o comportamento final da estrutura CodeRunner – Balanceador – Jobesandbox. Assim, fica explicitado que o CodeRunner através dos exercícios de cada aluno gera uma requisição para cada exercício, onde a ferramenta por meio de uma função associa cada requisição a um número de IP válido pertencente a fila de servidores Jobesandbox.

Figura 18. Balanceador de Carga



Fonte: Adaptado de http://www.site1377606654.provisorio.ws/blog-agility/author/enterprise/

A linguagem de programação escolhida para a implementação do algoritmo Round-Robin foi a PHP, já que o CodeRunner também é todo implementado nesta linguagem, assim permitindo uma fácil comunicação entre a ferramenta e o balanceador.

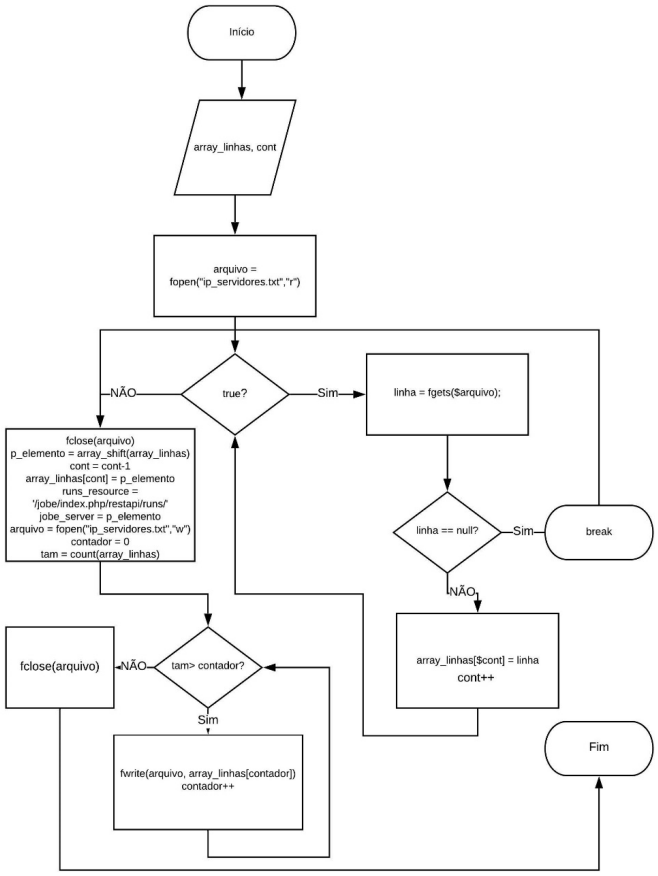
O algoritmo foi desenvolvido usando algumas funcionalidades da linguagem PHP, como a manipulação de arquivos de textos, a fim de armazenar de forma permanente a lista de servidores. Usa-se também o conceito de manipular *arrays* com o objetivo de realizar toda a operação de pôr os IPs dos servidores em ordem correta e assim permitir um correto balanceamento.

Na figura 19 abaixo, vemos com detalhes o algoritmo desenvolvido em fluxograma, tornando mais fácil a compreensão do seu funcionamento e todos os passos necessários para realizar o balanceamento aqui proposto. Nesta figura vemos que a implementação de fato é relativamente simples, como dito antes, porém eficaz.

Sendo necessário no início ler o arquivo texto, inserir cada linha deste mesmo arquivo em um espaço de memória de um *array*, em seguida

realizamos a troca de ordem através da funcionalidade *array_shift,* pois essa função além de retirar o primeiro elemento da lista ela o retorna, assim realizamos a inserção deste ao final do mesmo *array* faltando apenas salvar este no arquivo de texto incialmente aberto.

Figura 19: Fluxograma Round-Robin



Fonte: Benevides L.

Figura 20. Algoritmo Round-Robin

```
167
168  #ROUND ROBIN
169
170  $array_linhas = [];
171  $arquivo = fopen("ip_servidores.txt","r");
172  $cont = 0;
173  while(true) {
174      $linha = fgets($arquivo);
175      if ($linha==null) break;
176      $array_linhas[$cont] = $linha;
177      $cont++;
178  }
179  fclose($arquivo);
180  $p_elemento = array_shift($array_linhas);
181  $cont = $cont-1;
182  $array_linhas[$cont] = $p_elemento;
183  $RUNS_RESOURCE = '/jobe/index.php/restapi/runs/';
184  $JOBE_SERVER = $p_elemento;
185  $arquivo = fopen("ip_servidores.txt","w");
186  $contador = 0;
187  $tam = count($array_linhas);
188  while($tam > $contador) {
189      fwrite($arquivo, $array_linhas[$contador]);
190      $contador++;
191  }
192  fclose($arquivo);
```

Fonte: Benevides L.

Na figura 20 acima, no algoritmo implementado podemos ver todas as estruturas que necessárias para o seu correto funcionamento. Além das que já tinham sido mencionadas anteriormente, temos duas estruturas de repetição *while* onde na primeira ela é usada para buscar e armazenar todas as linhas do arquivo texto dentro do *array*, já na segunda o seu propósito visa armazenar o conteúdo de cada posição do array no mesmo arquivo texto.

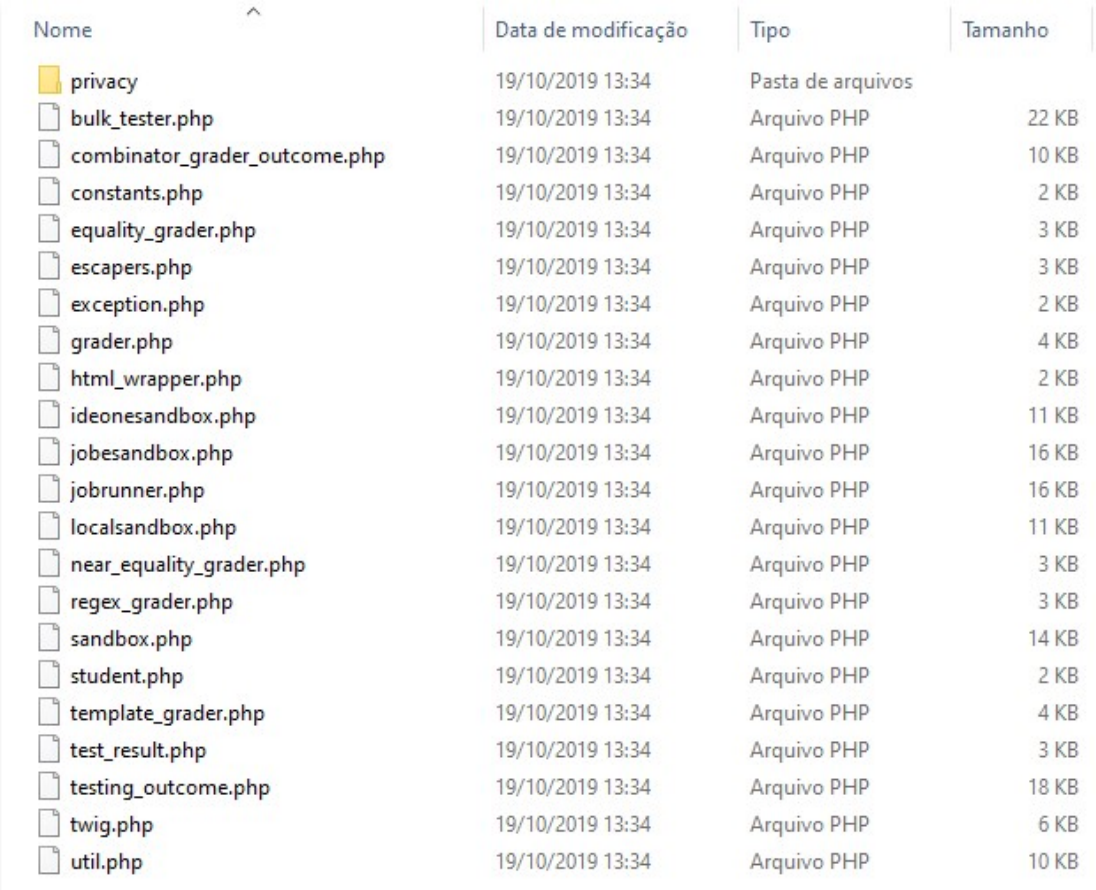
Ao final da execução do algoritmo temos um arquivo de texto contendo várias linhas reordenadas, onde cada linha é um IP válido do Jobesandbox e dentro da variável *jobe_server* fica armazenado o primeiro IP. Este por sua vez será usado primeiramente para receber o redirecionamento da requisição atual, e em seguida será realocado para o último da fila, assim realizando a intercalabilidade e por conseguinte o balanceamento de requisições.

O algoritmo parte da premissa que o arquivo de texto inicialmente já armazena os IPs dos servidores Jobesandbox, sem limite de IPs e consecutivamente de servidores, tornando a proposta de implementar um distribuidor de requisições mais robusta e tolerante a falhas.

6.2 Inserção do algoritmo no CodeRunner

Realizada a implementação do algoritmo foi necessário identificar qual dos arquivos abaixo como mostra a figura 21 seria o responsável pela comunicação entre o CodeRunner e o servidor Jobesandbox. Logo após essa pesquisa, tornou-se possível realizar a identificação do arquivo *sandbox.php* como o arquivo que informa ao CodeRunner qual IP do servidor Jobesandbox responsável pela correção dos exercícios enviados pelos alunos.

Figura 21. Arquivos de Classes do CodeRunner



Nome	Data de modificação	Tipo	Tamanho
privacy	19/10/2019 13:34	Pasta de arquivos	
bulk_tester.php	19/10/2019 13:34	Arquivo PHP	22 KB
combinator_grader_outcome.php	19/10/2019 13:34	Arquivo PHP	10 KB
constants.php	19/10/2019 13:34	Arquivo PHP	2 KB
equality_grader.php	19/10/2019 13:34	Arquivo PHP	3 KB
escapers.php	19/10/2019 13:34	Arquivo PHP	3 KB
exception.php	19/10/2019 13:34	Arquivo PHP	2 KB
grader.php	19/10/2019 13:34	Arquivo PHP	4 KB
html_wrapper.php	19/10/2019 13:34	Arquivo PHP	2 KB
ideonesandbox.php	19/10/2019 13:34	Arquivo PHP	11 KB
jobesandbox.php	19/10/2019 13:34	Arquivo PHP	16 KB
jobrunner.php	19/10/2019 13:34	Arquivo PHP	16 KB
localsandbox.php	19/10/2019 13:34	Arquivo PHP	11 KB
near_equality_grader.php	19/10/2019 13:34	Arquivo PHP	3 KB
regex_grader.php	19/10/2019 13:34	Arquivo PHP	3 KB
sandbox.php	19/10/2019 13:34	Arquivo PHP	14 KB
student.php	19/10/2019 13:34	Arquivo PHP	2 KB
template_grader.php	19/10/2019 13:34	Arquivo PHP	4 KB
test_result.php	19/10/2019 13:34	Arquivo PHP	3 KB
testing_outcome.php	19/10/2019 13:34	Arquivo PHP	18 KB
twig.php	19/10/2019 13:34	Arquivo PHP	6 KB
util.php	19/10/2019 13:34	Arquivo PHP	10 KB

Fonte: Benevides L.

A inserção do algoritmo desenvolvido foi acima da função *available_sandboxes* como mostra a figura 23, pois essa função retorna um

array associativo contendo todos as sandboxes disponiveis. Entretanto, o algoritmo Round-Robin aqui proposto informa para este array apenas um IP por vez, alternando-os a cada requisição.

Figura 22. Função Sandboxes

```
192     public static function available_sandboxes() {
193         global $JOBE_SERVER;
194         return array($JOBE_SERVER      => 'qtype_coderunner_jobesandbox'
195                     );
196     }
```

Fonte: Benevides L.

6.3 Replicação do servidor Jobesandbox

A replicação do servidor responsável por executar remotamente os códigos dos alunos é relativamente bem intuitiva, mas exige alguns itens necessários a seguir, como a obrigatoriedade de ser instalado no sistema operacional Linux, de ter o servidor Apache instalado e em execução. O PHP deve ter sido compilado com o System V Semaphores e funções de memória compartilhada ativadas. Logo após se certificar que essas exigências foram atendidas, devemos executar alguns comandos no “Terminal” do Linux:

- *sudo apt-get install apache2 php libapache2-mod-php php-cli*
- *php-mbstring octave nodejs git python3 build-essential default-jdk*
- *python3-pip fp-compiler pylint3 acl sudo sqlite3*

Esses comandos são responsáveis por instalarem e configurarem via terminal todas as ferramentas necessárias acrescidas de todas as linguagens de programação suportados no momento. Entretanto, como falamos anteriormente, nada impede de a posterior ser instaladas novas dependências dando a possibilidade de correção de exercícios em novas linguagens.

A instalação do servidor Jobe se dá a partir da cópia do projeto que se encontra no diretório “*https://github.com/trampgeek/jobe.git”* e

inserindo-o na raiz do *webroot* que geralmente é em */ var / www / html.* Em seguida, o comando abaixo deve ser executado como root no terminal, logo após, o servidor Jobesandbox já está apto a receber requisições do CodeRunner para possíveis correções de exercícios:

- *cd WEBROOT/jobe*
- *sudo ./install*

7. ANÁLISE DOS RESULTADOS

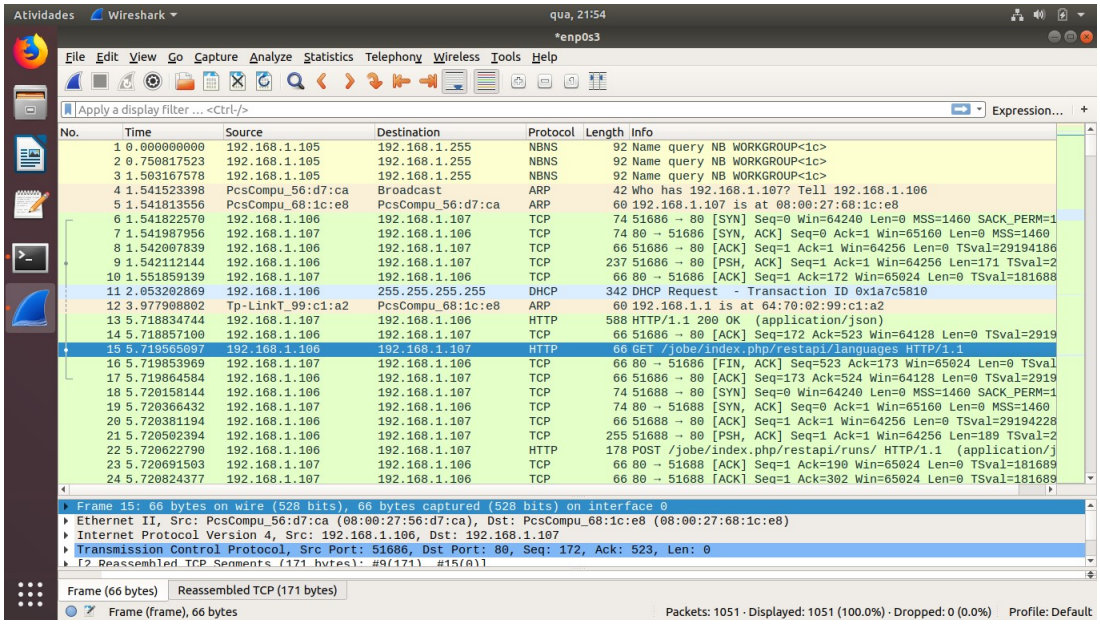
Mediante a dificuldade do ensino-aprendizagem da lógica de programação proveniente dos complexos pensamentos abstratos exigidos e somado a uma forma desatualizada de expor o conteúdo programático no contexto educacional, se faz necessário buscar maneiras mais atraentes, ilustrativas e dinâmicas de passagem do referido conteúdo. Diante disso, as ferramentas de ensino da lógica de programação surgem como uma tentativa de atenuar essa problemática, tornando a tarefa em questão mais tangível para o aluno, visando assim proporcionar um melhor aproveitamento nas disciplinas constituídas de conceitos computacionais.

Dessa forma, realizou-se uma pesquisa qualitativa de base bibliográfica, tal que, foi feito uma revisão de alguns artigos relacionados à ferramenta de ensino-aprendizagem de lógica de programação CodeRunner visando expor todas as características, funcionalidades, arquitetura, interfaces e a forma de como a aplicação se comunica com o servidor externo, Jobesandbox.

Diante do trabalho aqui desenvolvido pudemos compreender melhor todo o processo que ocorre na ferramenta CodeRunner, desde quando o aluno envia um exercício a ser corrigido até o retorno do Sandbox com a resposta. Tal análise, nos mostrou que a ferramenta apesar de ser bem elaborada torna-se susceptível a falhas de sobrecarga ou de processamento, pois como vimos a comunicação se dá de maneira não distribuída.

Dessa forma, realizamos o estudo de algoritmos de balanceamentos de processos e introduzimos no CodeRunner alguns conceitos de sistemas distribuídos, a fim de minimizar tais sobrecargas. Como podemos ver na figura 23 abaixo o IP de origem 192.168.1.106 é o IP correspondente que envia o exercício, ou seja, é o computador que o aluno está realizando a tarefa no CodeRunner e o IP 192.168.1.107 é o Sandbox selecionado pelo algoritmo de Round-Robin aqui desenvolvido a receber o exercício para correção.

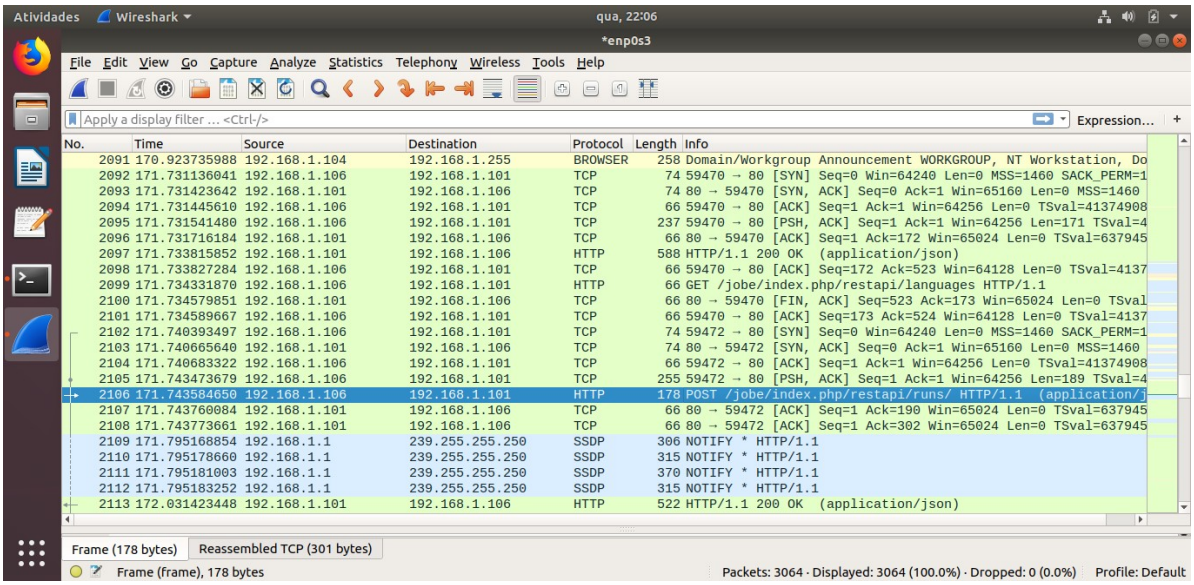
Figura 23. Teste do algoritmo de balanceamento



Fonte: Benevides L.

Já na figura 24, podemos ver a requisição sendo enviada pelo mesmo IP de origem o 192.168.1.106, porém dessa vez o Sandbox selecionado a receber o exercício desenvolvido correspondeu ao IP 192.168.1.101. Neste caso foram configuradas apenas duas Sandbox para teste, por tanto a cada requisição as Sandbox ficam se alternando.

Figura 24. Teste do algoritmo de balanceamento



Fonte: Benevides L.

8. REFERÊNCIAS

BARBOSA, L. S. **Aprendizado Significativo Aplicado ao Ensino de Algoritmos**. Dissertação (Pós-Graduação em Sistemas de Computação) - Departamento de Informática e Matemática Aplicada. Universidade Federal do Rio Grande do Norte, 2011.

BRANCO N. W. C. SCHUVARTZ, A. A. **Ferramenta Computacional de Apoio ao Processo de Ensino-Aprendizagem dos Fundamentos de Programação de Computadores.** Simpósio Brasileiro de Informática na Educação. São Paulo, 2008.

CHAVES, J. O. M. CASTRO, A. F. LIMA, R. W, et.al**. MOJO: Uma Ferramenta para integrar juízes online ao Moodle no apoio ao ensino e aprendizagem de programação.** HOLOS, vol. 5, 2014, pp. 237-251 Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte Natal, Brasil.

GOULART, R.. **SCRATCH: PRODUZINDO E APRENDENDO COM ALUNOS DO ENSINO FUNDAMENTAL I.** Porto Alegre: Especialização em Informática Instrumental pela Universidade Federal do Rio Grande do Sul. 2019.

JAYASWAL, K. **Administering Data Centers: Servers, Storage, and Voice over IP**. Indianapolis, EUA: Wiley, 2006.

MADHURAVANI, G. SUMALATHA, M. SHANMUKHI, " **AN EMPERICAL STUDY OF STATIC AND DYNAMIC LOAD BALANCING ALGORITHMS IN PARALLEL AND DISTRIBUTED COMPUTING ENVIRONMENT**" International Journal of Emerging trends in Engineering and Development ISSN 2249-6149 Issue 2, Vol.3 (April-2012)

MANNILA, L. et al. **Computational Thinking in K-9 Education**. ITiCSE-WGR'14, Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference. June 21-25, 2014, Uppsala, Sweden, p. 1-29. Disponível em: <http://dx.doi.org/10.1145/2713609.2713610>. Acesso em: 13 ago. 2019.

MORAES, M. **A monitoria como serviços de apoio ao aluno na educação a distância.** Florianópolis: Tese (Doutorado em Engenharia de Produção) pelo Departamento de Engenharia de Produção da UFSC. Florianópolis, 2004.

MOREIRA, M. P.; FAVERO, E. L. **Um Ambiente Para Ensino De Programação Com Feedback Automático De Exercícios**. Workshop Sobre Educação em Computação, 2009.

MOTA, M. P.; PEREIRA, L. W. K; FAVERO, E. L. **JavaTool: Uma Ferramenta para o Ensino de Programação**. In: Congresso da Sociedade Brasileira de Computação. Belém. XXVIII Congresso da Sociedade Brasileira de Computação. 2008.

MOTA, M. P. PEREIRA, L. W. K. FAVERO, E. L. JavaTool: **Uma Ferramenta Para Ensino de Programação.** In: Simpósio Brasileiro de Informática na Educação (SBIE 2009), 20, 2009, Florianópolis.

KELLEHER, C. ANDPAUSCH, R. **Loweringthebarrierstoprogramming: A taxonomyofprogrammingenvironmentsandlanguages for noviceprogrammers**, ACM ComputingSurveys (CSUR). 2005.

LOBB, R. **CodeRunner.** Disponível em: http://coderunner.org.nz/. Acesso em: 14 de outubro de 2018.

NETCRAFT. Disponível em: http://news.netcraft.com/archives/2018/03/index.html. Acesso em: 30 de março de 2018.

PAPERT, S. (1971) **Teaching Children Thinking**, Logo Memo nº 2, 1971. Disponível em: <https://archive.org/stream/bitsavers_mitaiaimAI_471587/AIM-247_djvu.txt>. Acesso em: 25 jun. 2019.

PAPERT, S. (1980) **Mindstorms. Children, computer and powerful ideas**. New York: Basic Books. Traduzido como Logo: Computadores e Educação. São Paulo: Brasiliense, 1985.

PEREIRA JÚNIOR, J. C. R. et al. **Ensino de Algoritmos e Programação: uma experiência no nível médio**. In: Congresso da Sociedade Brasileira de Computação; Workshop de informática na escola, 11. 2005, São Leopoldo, RS. Anais... São Leopoldo, RS, 2005. p. 2351-2362

PHP. Disponível em:<http://php.net/manual/pt_BR/book.curl.php>. Acesso em: 10 de julho de 2019.

RIBEIRO, N. E. et al. **A Importância dos Ambientes Virtuais de Aprendizagem na Busca de Novos Domínios da EAD**. In Congresso da Associação Brasileira de Educação a Distância – ABED, 04. 2007, Curitiba, PR.

SANDBOX. Disponível em: https://github.com/openjudge/sandbox/. Acesso em: 21 de janeiro de 2018.

SOUZA, M. **Produção do conhecimento em ead: um elo entre professor – curso – aluno**. In Proceedings CINFORM - Encontro Nacional de Ciência da Informação V, Salvador/Bahia, 2004.

WING, J. M. **Computational thinking**. Communications of the ACM, v. 49, n. 3, p. 33-35, 2006.

WING, J. M. **Computational Thinking: what and why**. Thelink, 2011. Disponível em: http://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why. Acesso em: 19 mai. 2019.

WING, J. **PENSAMENTO COMPUTACIONAL. Um conjunto de atitudes e habilidades que todos, não só cientistas da computação, ficaram ansiosos para aprender e usar**. Revista Brasileira de Ensino de Ciência e Tecnologia , v. 9, n. 2, 2016. Disponível em: <https://per iodicos.utfpr.edu.br/rbect/article/view/4711>. Acesso em: 28 mai. 2019.

ANEXO A - Código fonte comunicação com o servidor

```
public function execute($sourcecode, $language, $input, $files=null, $params=null) {
    $language = strtolower($language);
    if (!in_array($language, $this->languages)) {          return (object) array('error' =>
self::UNKNOWN_SERVER_ERROR);
    }

    if ($input !== '' && substr($input, -1) != "\n") {
        $input .= "\n";
    }

    $filelist = array();
    if ($files !== null) {
        foreach ($files as $filename => $contents) {
            $id = md5($contents);
            $filelist[] = array($id, $filename);
        }
    }

    $progname = "prog.$language";

    $runspec = array(
        'language_id'      => $language,
        'sourcecode'       => $sourcecode,
        'sourcefilename'   => $progname,
        'input'            => $input,
        'file_list'        => $filelist
    );

    if (self::DEBUGGING) {
        $runspec['debug'] = 1;
    }

    if ($params !== null) {
            $runspec['parameters'] = $params;
        if (isset($params['debug']) && $params['debug']) {
            $runspec['debug'] = 1;
        }
        if (isset($params['sourcefilename'])) {
            $runspec['sourcefilename'] = $params['sourcefilename'];
        }
    }

    $postbody = array('run_spec' => $runspec);
    $this->currentjobid = sprintf('%08x', mt_rand());

    $httpcode = $this->submit($postbody);
    if ($httpcode == 404) {
        foreach ($files as $filename => $contents) {
```

```
            if (($httpcode = $this->put_file($contents)) != 204) {
                break;
            }
        }
        if ($httpcode == 204) {

            $httpcode = $this->submit($postbody);
        }
    }

    if ($httpcode != 200 || !is_object($this->response) || !isset($this-
>response->outcome)) {  $errorcode = $httpcode == 200 ?
self::UNKNOWN_SERVER_ERROR : $this->get_error_code($httpcode);
        $this->currentjobid = null;
        return (object) array('error' => $errorcode, 'stderr' => $this->response);
    } else if ($this->response->outcome == self::RESULT_SERVER_OVERLOAD) {
        return (object) array('error' => self::SERVER_OVERLOAD);
    } else {
        $stderr = $this->filter_file_path($this->response->stderr);
        if (trim($stderr) !== '') {
            $this->response->outcome = self::RESULT_RUNTIME_ERROR;
        }
        $this->currentjobid = null;
        return (object) array(
          'error'   => self::OK,
          'result'  => $this->response->outcome,
          'signal'  => 0  'cmpinfo' => $this->response->cmpinfo,
          'output'  => $this->filter_file_path($this->response->stdout),
          'stderr'  => $stderr
        );
    }
  }


  private function get_error_code($httpcode) {
    $codemap = array(
        '200' => self::OK,
        '202' => self::OK,
        '204' => self::OK,
        '400' => self::JOBE_400_ERROR,
        '401' => self::SUBMISSION_LIMIT_EXCEEDED,
        '403' => self::AUTH_ERROR
    );
    if (isset($codemap[$httpcode])) {
        return $codemap[$httpcode];
    } else {
        return self::UNKNOWN_SERVER_ERROR;
    }
  }


  private function put_file($contents) {
    $id = md5($contents);
    $contentsb64 = base64_encode($contents);
    $resource = "files/$id";

    list($url, $headers) = $this->get_jobe_connection_info($resource);
```

```
    $body = array('file_contents' => $contentsb64);
    $curl = curl_init();
    curl_setopt($curl, CURLOPT_HTTPHEADER, $headers);
    curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "PUT");
    curl_setopt($curl, CURLOPT_URL, $url);
    curl_setopt($curl, CURLOPT_POSTFIELDS, json_encode($body));
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
    $result = curl_exec($curl);
    $info = curl_getinfo($curl);
    curl_close($curl);
    return $result === false ? -1 : $info['http_code'];
}

private function get_jobe_connection_info($resource) {
    global $CFG;
    if ($CFG->prefix == "b_") {
        $jobe = "localhost";
    } else {
        $jobe = get_config('qtype_coderunner', 'jobe_host');
    }
    $url = "http://$jobe/jobe/index.php/restapi/$resource";

    $headers = array(
            'User-Agent: CodeRunner',
            'Content-Type: application/json; charset=utf-8',
            'Accept-Charset: utf-8',
            'Accept: application/json',
    );
    $apikey = get_config('qtype_coderunner', 'jobe_apikey');
    if (!empty($apikey)) {
        $headers[] = "X-API-KEY: $apikey";
    }
    if (!empty($this->currentjobid)) {
        $headers[] = "X-CodeRunner-Job-Id: $this->currentjobid";
    }

    return array($url, $headers);
}

private function submit($job) {
    list($returncode, $response) = $this->http_request('runs', self::HTTP_POST, $job);
    $this->response = $response;
    return $returncode;

}

private function http_request($resource, $method, $body=null) {
    list($url, $headers) = $this->get_jobe_connection_info($resource);

    $curl = new curl();
    $curl->setHeader($headers);

    if ($method === self::HTTP_GET) {
        if (!empty($body)) {
            throw new coding_exception("Illegal HTTP GET: non-empty body");
        }
```

```
            $response = $curl->get($url);
        } else if ($method === self::HTTP_POST) {
            if (empty($body)) {
                throw new coding_exception("Illegal HTTP POST: empty body");
            }
            $bodyjson = json_encode($body);
            $response = $curl->post($url, $bodyjson);
        } else {
            throw new coding_exception('Invalid method passed to http_request');
        }

        if ($response !== false) {
            $returncode = $curl->info['http_code'];
            $responsebody = $response === '' ? '' : json_decode($response);
        } else {
            $returncode = -1;
            $responsebody = '';
        }

        return array($returncode, $responsebody);
    }


    /home/jobe/runs/<directory>/filename
        private function filter_file_path($s) {
        return preg_replace('|(/home/jobe/runs/jobe_[a-zA-Z0-9_]+/)([a-zA-Z0-9_]+)|', '$2', $s);
    }
}
```